Rational software

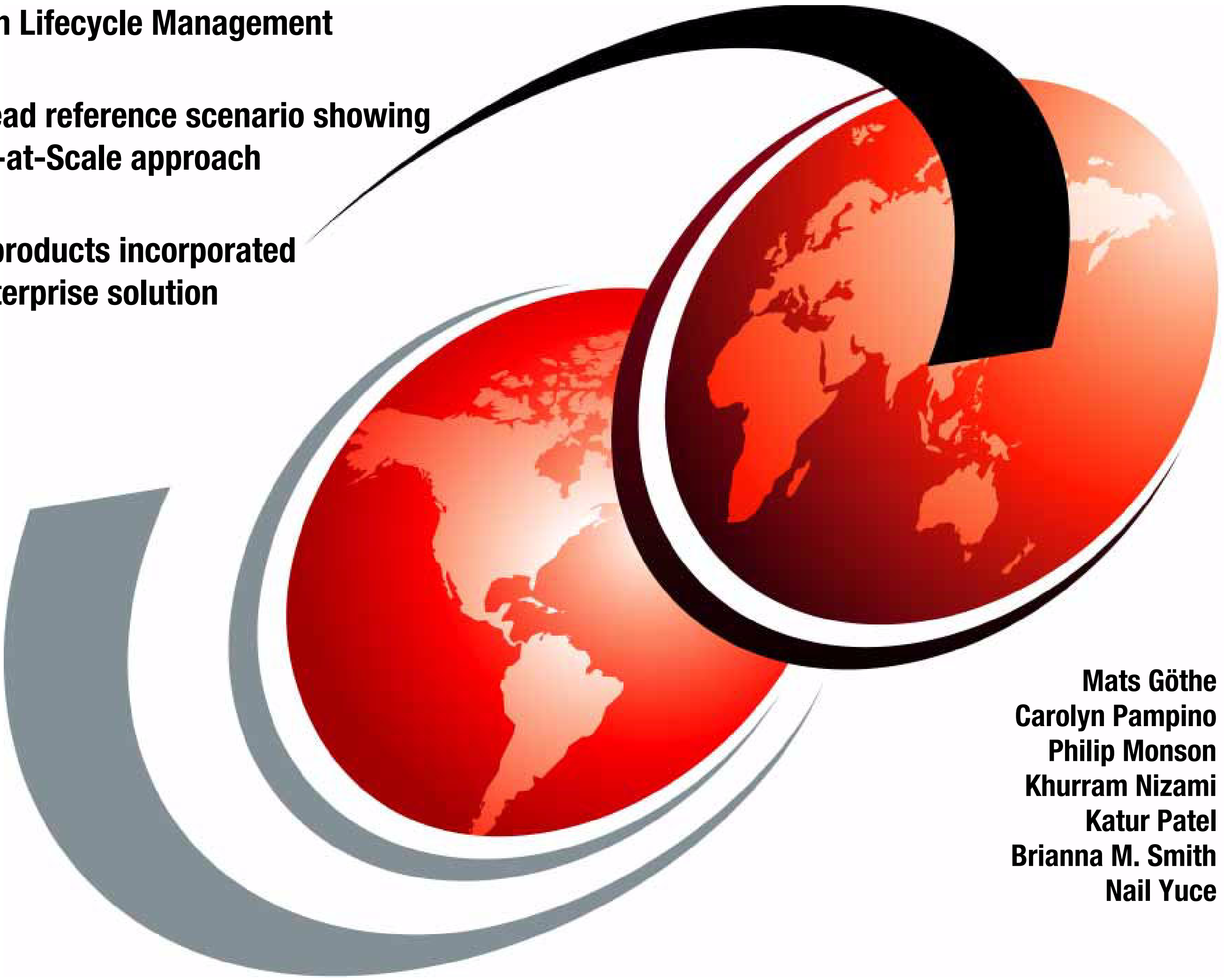
IBM

Collaborative Application Lifecycle Management with IBM Rational Products

An IBM blueprint for Collaborative Application Lifecycle Management

Green-thread reference scenario showing the Agility-at-Scale approach

IBM Jazz products incorporated into an enterprise solution



Mats Göthe
Carolyn Pampino
Philip Monson
Khurram Nizami
Katur Patel
Brianna M. Smith
Nail Yuce

Redbooks

ibm.com/redbooks

International Technical Support Organization

Collaborative Application Lifecycle Management with IBM Rational Products

December 2008

SG24-7622-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (December 2008)

This edition applies to IBM Rational Build Forge Enterprise Edition 7.1, Rational ClearCase 7.1, Rational ClearQuest 7.1, Rational RequisitePro 7.1, Rational Quality Manager 1.0, Rational Requirements Composer 7.1, Rational Software Analyzer 7.0, and Rational Team Concert 1.0.

Contents

Notices

Trademarks

Preface

Holistic, Collaborative, Community-based, End-to-End Management of all Developed Software Assets (HCCEEMDSA) is not as succinct a term as *Application Lifecycle Management* (ALM), but it might be a better way to describe what ALM means. Numerous vendors and analysts have made several attempts to define ALM. Inevitably, ALM either describes the technical details of individual assets all working well together across an entire development life cycle, or it proposes a higher level of manager-oriented benefits that one never seems to obtain. In the end, ALM is the ability for a development team (the key being a *team*, not an individual) to continuously be able to answer a set of hard questions, such as the following questions, regardless of your location, position, or project:

- How are my teams in India, Brazil, and San Jose progressing against the project plan?
- Can we enforce a development process that spans across my life-cycle activities?
- What requirements and defect fixes are implemented in this build?
- Have we implemented all requirements with sufficient quality?
- How do I best organize and trace my assets so that I can respond to a regulatory audit?

Successful IT organizations are finding that the complex ALM implementations they have attempted over the years are brittle in their touch points and tend to be focused on individual vendor solutions. For example, software is the result of many ever-changing conversations, across disparate teams and geographies. It is almost humorous to think that we believed that those brittle integrations might endure as those conversations became more rapid across multiple languages, time zones, technologies, and iterations.

New technologies, such as XML and Web 2.0, offer flexible new capabilities that make robust, long-lasting ALM integrations possible. Meanwhile, community-based development models that were borrowed from the best open-source projects can help ensure that ALM solutions are not “single-vendor” focused. To distinguish these capabilities from those of the past, we call this new capability *Collaborative Application Lifecycle Management* (CALM).

Regardless, of your definition, or the questions for which you need answers, CALM is quickly becoming a strategic necessity in today’s business-driven world. The Internet has changed everything, making software business critical. Teams in business, development, and operations must collaborate in an unprecedented way to provide high-impact software and services to their customers. Gone are the days of one tool per role with many silos per organization. Teams must access each other’s work to be successful.

With the advent of the agile and Extreme Programming development methodologies, individuals are performing more than one traditional role. In this evermore competitive world, companies do not have the time, nor the budget for specialists. Individuals must become collaborators to be effective, and teams must have transparency to work well together. For instance, successful, distributed development requires a high level of collaboration and governed process so that teams do not overspend time and resource on micromanaging training, handoffs, and deliveries.

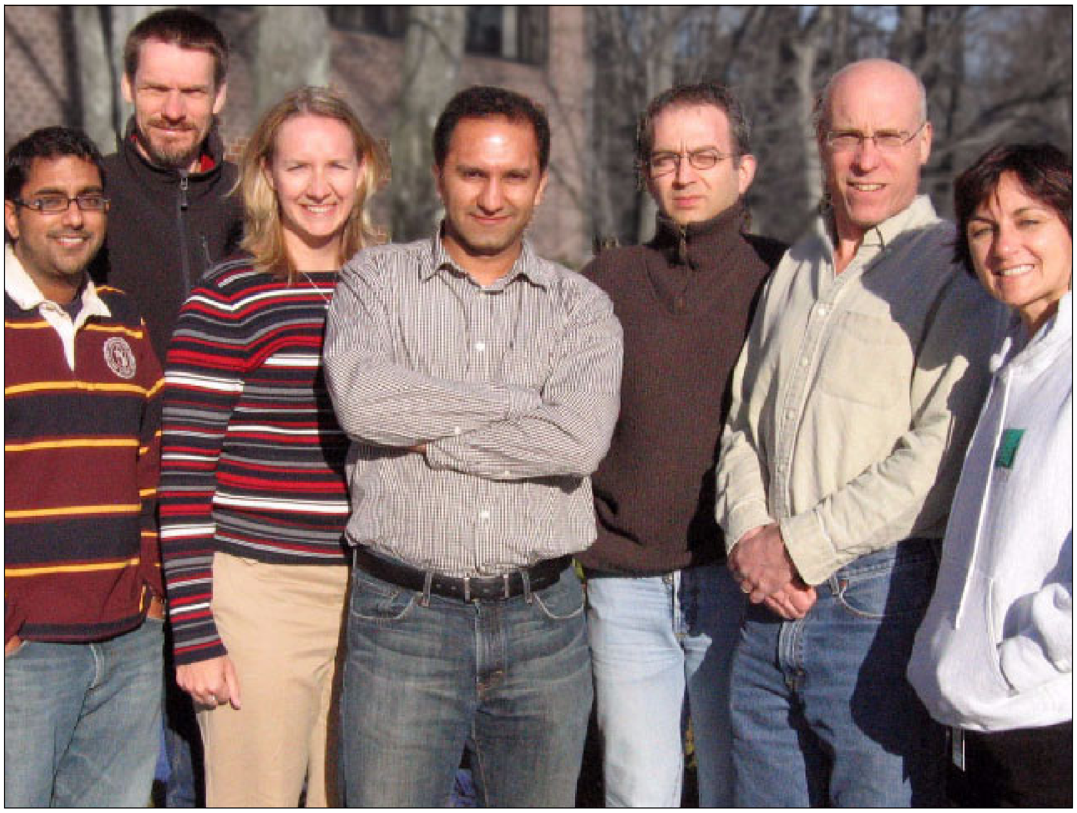
Regulatory bodies around the world continue to force development organizations to provide traceability details about artifacts throughout their application teams, not just from a single development silo. Organizations are pinched for resources and hardware. The ability to reuse assets, share resources, and have a common model for integrating functionality becomes critical to staying competitive. Therefore, coordinating these development life-cycle activities and managing the relationships between development artifacts is a mandate for success.

In this IBM® Redbooks® publication, we examine Collaborative Application Lifecycle Management from the inside out by leveraging these state-of-the-art capabilities. We define a blueprint for CALM and provide details about the handoffs that occur in an iteration of a sample software development project. We provide a reference scenario and architecture that characterizes proof points for CALM business value. We offer a deployment model for IT organizations that are looking to evolve toward a successful, proven CALM model. The definitions, handoffs, and solutions that are described in this Redbooks publication are vendor neutral. However, we concentrate on products that are developed and delivered by the IBM Rational® brand in order to show a reference architecture.

Mike O'Rourke, Vice President, Rational Software Development

The team that wrote this book

This book was produced by a team of specialists from around the world working with the International Technical Support Organization (ITSO) in Lexington, Massachusetts.



The team from left to right: Katur Patel, Mats Göthe, Brianna Smith, Khurram Nizami, Nail Yuce, Phil Monson, and Carolyn Pampino

Mats Göthe is a Solution Architect on the Rational Cross-Product Green Threads team. He is co-leading the Green Threads on Geographically Distributed Application Lifecycle Management. Mats joined Rational in 1991 and has held various positions, which include development manager for Rational Rose® development in Sweden and Rational technical sales and service manager on the Ericsson Corporate Account team and in the Nordic region. Mats has a doctorate (PhD.) in physics from Uppsala University and is based in Kista, Sweden.

Carolyn Pampino is a Solution Architect on the Rational cross-product Green Threads team. She co-leads the geographically distributed Application Lifecycle Management green thread and was a co-lead for this Redbooks publication. Carolyn joined Rational in 2002 and has held various positions that influence brand strategy. These positions include serving as product manager for the Rational ClearQuest® 7.1 ALM solution, transition manager for the Rational Build Forge® acquisition, product manager for identifying and driving integration strategies with the Tivoli® portfolio, and product manager for launching the Rational Professional Bundle. Prior to IBM, Carolyn was Director of Product Management, Development, and Competitive Intelligence at BroadVision, Inc. Prior to BroadVision, she was a Director of Development at Interleaf and contributed to the acquisition of Interleaf by BroadVision. Carolyn received her degree with University Honors from Carnegie-Mellon University.

Philip Monson is a Business Development Project Leader for the ITSO. Phil has been with Lotus® and IBM for 18 years, joining the company when the early versions of Lotus Notes® were rolled out for internal use. He has served in management, technical, and consulting roles in the IT, Sales, and Development organizations.

Khurram Nizami is responsible for worldwide enablement of Rational products. His expertise is in helping customers and IBM by applying practical solutions to challenges faced by the software delivery community today. Khurram's specialty and area of interest is Geographically Distributed Development (GDD). Prior to working for IBM, he was a manager for Cap Gemini Ernst & Young in the Systems Development and Integration practice, where he was a project manager and technical lead for a number of global software delivery projects. Khurram holds a Master of Science degree in Management of Technology from the University of Minnesota and is a PMI-certified Project Management Professional.

Katur Patel is a Staff Software Engineer at for IBM Rational Software. He has held positions in the server technology group, working with TSO/E and Java™ on S/390®, and was a developer in the 300mm Fab Test Solutions Center. He supported Rational ClearQuest as a technical support engineer before moving to his current role in the Customer Advocacy Group for the same product. As a Customer Advocacy Group engineer, Katur has developed expertise with the Rational ClearQuest schema design and integration with Rational Test Manager, MultiSite, and Web applications. He holds a Master of Science degree in Computer Science and a Graduate Certificate in Human Computer Interaction from Tufts University in Medford, Massachusetts.

Brianna M. Smith has worked with Rational and IBM for nine years in various capacities as a Staff Software Engineer for the Requirements Customer Advocacy Group and a Delivery Engagement Manager. She has developed expertise in the Requirements Definition and Management discipline as it integrates across the CALM. She has aided organizations both within and outside IBM in end-to-end requirements process transformation and leveraged Rational tools to support these engagements. Brianna has also aided the RequisitePro® product group in creating Rational RequisitePro project templates to support the Rational Unified Process (RUP®), and Business Modeling Integration.

Nail Yuce is a Rational Advisory Accredited IT Specialist in Turkey. He has been working for Rational for over four years. He has focused on Rational products, particularly Rational Software Architect, Rational Application Developer, Rational Build Forge, Rational ClearCase®, Rational ClearQuest, Rational Performance Tester, and Rational Functional Tester. He has also worked with them in Proof of Concept, Proof of Technology, and support at customer sites in Turkey. He has a degree in Computer Science and Engineering from the University of Hacettepe, in Ankara, Turkey.

Thanks to Scott Ambler, Practice Leader Agile Development, for IBM Rational Canada, for his contributions to this project. We also thank the following members of the IBM Software Group for their contributions to this book:

- Bruce Baron, Marketing Manager - Rational Project and Portfolio Management
- Douglas Bush, Information Developer, Requirements Management
- David Colasurdo, Senior Development Manager, Rational Quality Manager
- Dwayne Dreakford, Product Manager - Requirements Definition and Management
- Tim Feeney, TechWorks Practice Lead - Change & Release Management
- Carlos Ferreira, Rational Asset Manager Product Manager
- Richard Forziati, Staff Software Engineer
- Kevin Haaland, Jazz Project Development Lead
- Bryan Hogan, Software Developer, Rational Enterprise Tools
- Chris Kau, Lead Architect Lotus Workforce Management
- Mathieu Lapensee, Software Quality Advisor
- Grant Larsen, Senior Technical Staff Member - Chief Architect Asset Management
- Mario Maldari, RequisitePro Composer System Test Lead
- Brian Massey, Product Manager, Automated Software Quality
- Gili Mendel, Senior Technical Staff Member - Application Architect
- Elizabeth Miller, Senior Engineering Manager, Jazz Interop
- Daniel Moul, IBM Rational Offering Management
- Douglas Mueller, Manager, Rational Automated Software Quality Products
- Lynn Mueller, Senior Consultant, IBM Certified Executive IT Specialist
- Bob Myers, Rational Internal Deployment Architect
- Chibuzo Obi, Rational Engineering Services Tools
- Stuart W. Poulin, Customer/Partner Engagement Leader, Rational Enterprise Tools
- Sreedhar Rella, ClearQuest Software Developer
- Michael Saylor, Software Architect, Rational Software Configuration Management
- Nirav Sheth, Software Developer, Rational Software Configuration Management
- Dominique Simoneau-Ritchie, Software Developer, Java
- Paul Tasillo, Rational Test Lab Manager Chief Designer
- John Vasta, Team lead, ClearQuest Connectors
- Rick Weaver, Senior Manager, Rational Cross Product Green Threads
- Carol Yutkowitz, Senior Software Engineer, IBM Jazz™ Development Team

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks® in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part A

Collaborative Application Lifecycle Management defined

In this part, we describe the context for this Redbooks publication. The term *Application Lifecycle Management* has existed in the software industry for many years. Over the past several years, the definition has blurred. In this part, we provide a new view that we call *Collaborative Application Lifecycle Management* (CALM). We assert that, by following a single iteration in a development project, the value of CALM becomes apparent.

This book breaks with the traditional format of most Redbooks publications by using a storyboard to define the structure for the entire book. An overview of the storyboard, or scenario, is described in this part, which contains the following chapters:

- Chapter 1, "Introduction to Application Lifecycle Management and this book" on page 3
- Chapter 2, "Collaborative Application Lifecycle Management" on page 15
- Chapter 3, "A scenario for CALM" on page 47

Role-based guide: To understand how the content in this part applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

1

Introduction to Application Lifecycle Management and this book

In this chapter, we introduce the topic of Application Lifecycle Management (ALM) and define the areas of ALM that we discuss in this book.

Specifically, we include the following sections in this chapter:

- 1.1, “The scope of Application Lifecycle Management” on page 4
- 1.2, “Using this book” on page 11

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

1.1 The scope of Application Lifecycle Management

Application Lifecycle Management is a key enabler for streamlining a team's ability to produce a release of software. ALM consists of the core disciplines of requirements definition and management, asset management, development, build, test, and release that are all planned by project management and orchestrated by using some form of process. The development tool repositories store the assets and their relationships. Charts and reports gain visibility into a team's progress. ALM involves the coordination of software development activities and assets to produce and manage software applications throughout their life cycle.

1.1.1 People, process, information, and tools that drive the life cycle

In today's world, teams of people distributed around the world must collaborate on producing sophisticated business-critical applications in compressed time frames, while, in some cases, adhering to government regulations. The need to streamline a team's efficiency and govern without impeding progress is greater than at any point in the history of software development. Products must be introduced to market to respond to a changing competitive landscape, and Web sites must be updated constantly to remain fresh and interesting. The life span of software applications has shortened, and demand has dramatically increased. Software development teams must produce meaningful and competitive products at an unprecedented rate.

In this environment, teams can no longer act in silos where one team throws a solution "over the wall" for the next team to integrate, test, or deploy. Nor can the disciplines of software development be treated as silos where one team member only manages requirements, while another only develops the source code or only tests the software. Business demands software that meets the needs of the user. Operations teams must know how to manage these applications and respond to problems. Business analysts are more involved in the development process. Architects need to understand the architectural principles, patterns, and requirements set forth by the enterprise architecture. Developers must understand the stakeholders' needs, the business processes, design, and reuse assets, as well as implement and test the system. Testers are closely aligned with business analysts to ensure high quality, customer-driven software applications.

As a result, teams that develop internal IT systems and Internet applications take a much different approach to software development by using techniques for agile development, mock-ups, fast paced iterations, and increased interactions with stakeholders. Software development teams must be aware of and respond to customer demands. In addition, teams no longer have the luxury of 18- to 24-month development cycles where every detail is deeply analyzed and designed. Waterfall approaches to software development have given way to agile approaches with short iterations. Built-in feedback loops are necessary for producing software for the target users.

ALM is the vehicle that not only brings these disciplines and team members together, it drives the transparency of their progress and governs how the team works. Today's software development teams cannot function without it.

At the heart of ALM lies a coordinated set of contributions between team members. This coordination involves people, process, information, and tools with the underlying aspects of transparency and shared responsibility for success. Within this context, teams are self-configuring. The simplest view of this definition, as illustrated in Figure 1-1 on page 5, involves a series of contributions between team members. While this might be an overly simplified view, it gets to the heart of the matter: What one team member produces, another team member consumes. Understanding these dependencies helps a project team to organize their efforts, streamline their ability to produce software, and produce a healthy software development environment.

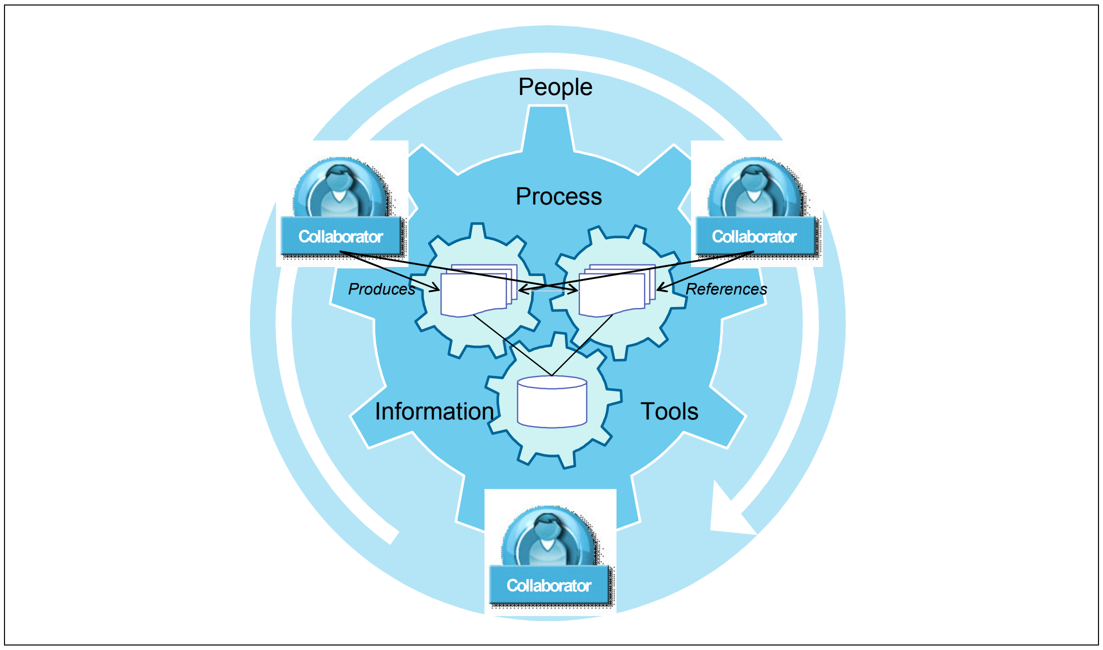


Figure 1-1   The coordination of people, process, information, and tools in ALM

If we agree that at the heart of ALM lies a coordinated set of handoffs between team members, then the importance of the key fundamentals becomes abundantly clear:

- People collaborate. Teams self-organize.

  Collaboration between team members is critical to successful contributions. Many times one team member has work to complete that requires referencing or reusing the work of another team member. By collaborating, team members gain better insight and understanding into how to approach completing or refining their own work, and the entire team stays current. Collaboration, however, requires trust, and trust develops through experience. Teams that self-organize leverage their trust to build a culture where they can flourish. In this environment, collaboration comes naturally.

- Process facilitates the flow of events or timing of what work is completed when.

  A group of musicians does not play their instruments all at once. Rather, their parts are orchestrated so that the combination of events produces a melody. Process enactment provides the cue to each team member of when it is time to play their part.

- Plans organize the scope of work to be addressed and by whom.

  A concert does not involve all work being performed at once. Rather a concert program is defined with an ordered set of songs to produce the most effective overall experience, and all musicians work from the same program.

  In software development, iterations set the cadence. Plans are used to scope the level of effort and to determine what work will be completed when, with a focus on achieving a specific goal or goals. However, project planning is a continuous effort that depends on transparency and involves evaluating the current situation and making course corrections.

  Therefore, transparency into real-time performance of the team is for creating plans to respond to current events. Transparency leads to health when the lead and team respond to current events by balancing the workload or changing the plan accordingly. A skeleton plan can, in some cases, come from “the top down,” but the reality comes from the bottom up as each team member assumes ownership and contributes their talent to filling out the plan.

- Information is shared in repositories.

  Repositories for storing the team's assets are shared across the team. It does not do any good for a team member to produce a critical piece of information and store it on their personal computer. Nor does it do any good to store that information in a repository to which other team members do not have access.

- Tools are specific to the user's task.

  A team member who produces a rich requirements storyboard is different from the developer who produces the source code to implement the story. Each user requires a tool that suits their need.

These basics can be confusing when the word *life cycle* is brought into the picture. After all, the team, the process, and what constitutes the beginning and end of the life cycle inherently drive a level of detail that differs from project to project and organization to organization. These details can be determined only when the boundaries and context for the life cycle are well understood. Herein lies the challenge. The definition of the life cycle depends on the nature and the context of its use. For example, *application lifecycle* is sometimes used inter-changeably with *software development lifecycle* (SDLC), whose acronym can also be defined as *systems development lifecycle*. As new business needs, technologies, and approaches have surfaced, the needs and expectations in the ALM market have changed.

Recently the term *IT Lifecycle Management* entered the market to describe the alignment of business, development, and IT operations. In IT Lifecycle Management, the cycle begins with business needs and extends to the operations team monitoring an application in production.

Some use the term *Application Lifecycle Management* to define this same IT cycle. *Product Lifecycle Management* addresses the complexities that are involved in managing physical products that contain software in addition to the physical parts that comprise the product. Portfolio management, enterprise architecture, and governance have also broadened the scope of the software development market with needs specific to each domain. With our view of software development expanding to include these new domains, some boundaries must be put in place to create a meaningful book. Regardless of the size of definition of the life cycle, the fundamental need for a coordinated set of handoffs between team members remains the same.

1.1.2 The scope of this book

By examining the handoffs that occur in a single iteration of a development project, we assert that the value of ALM is understood well enough to scale to a larger life cycle. To clarify the value of ALM, we focus on a single iteration in the Construction phase of a software development project, as illustrated in Figure 1-2.

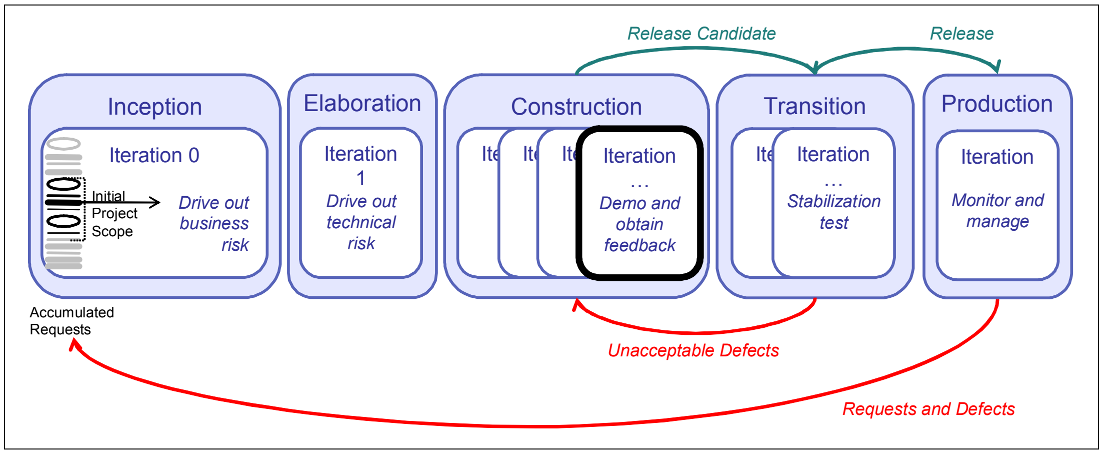


Figure 1-2 A single iteration revealing the value of ALM

Even within a single iteration, it is easy to get lost in the details of requirements management, change management, asset management collaborative development, build management, and quality management. Therefore, we selected a scenario that touches on each of these disciplines in the simplest of terms, with the intent of highlighting the relationships across disciplines, rather than highlighting the details within a single discipline.

For example, rather than provide details about every aspect of change management, we demonstrate how a single change request impacts the activity of the rest of the team, from iteration planning, requirements definition, asset reuse, development, build, and quality management to final delivery. Our exploration of each discipline is enough to highlight the interaction points across the disciplines. The intent of this book is to show how ALM helps to streamline a team's ability to produce a release of software by examining a single iteration.

By limiting the scope of the discussion, you can focus on the core value, rather than become lost or overwhelmed by the complexities that are involved in every discipline that participates in the life cycle. Clearly every organization defines the cycle differently. Different vocabularies are used. Some organizations apply more process, while others are more lenient. Some use sophisticated asset management techniques to perform impact analysis, where others reuse assets in an ad hoc manner. Some use agile techniques, while others follow traditional development patterns. It is impossible to cover every aspect of the application lifecycle. To produce this book, we have limited the scope of the discussion to a single path through an iteration, which is referred to as the "scenario" throughout the remainder of this book.

Figure 1-3 uses heavy black ellipses to indicate how a single change request impacts each of the software disciplines.

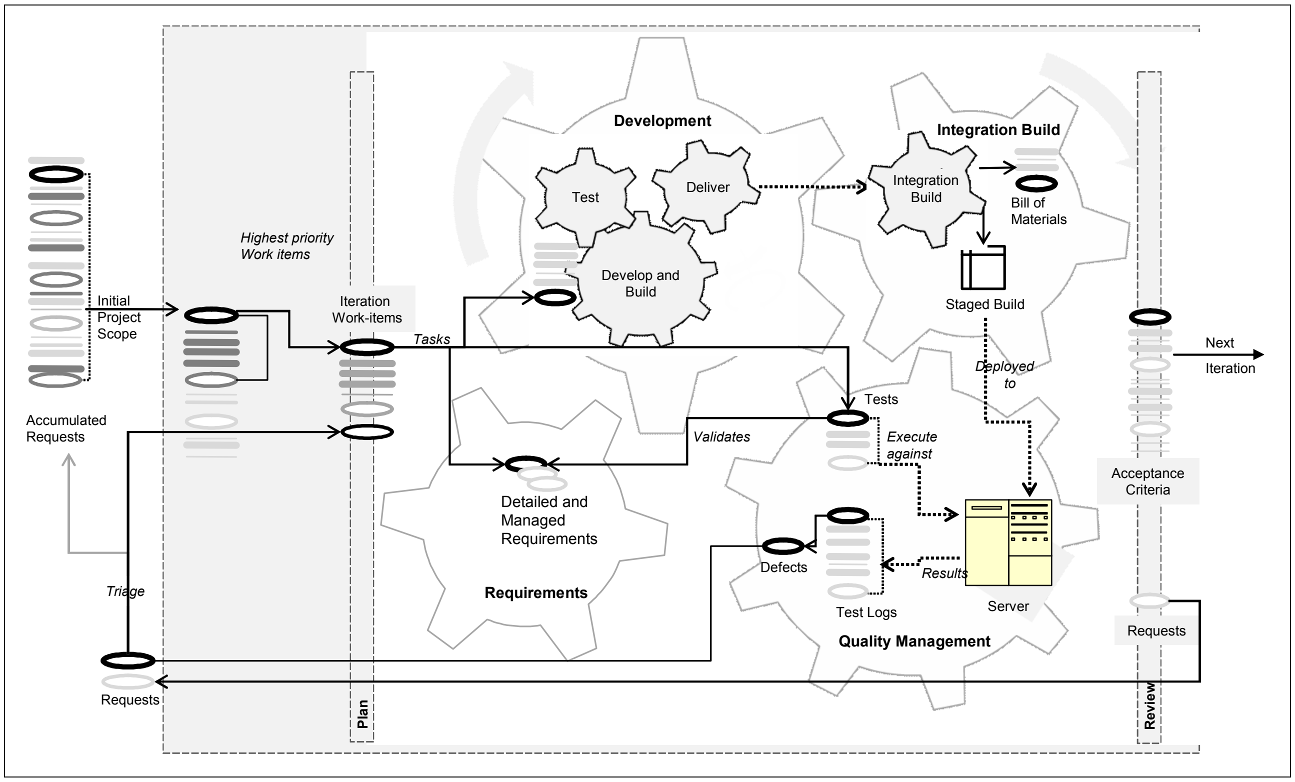


Figure 1-3 Tracking the impact of a single request through an iteration

Figure 1-3 shows the following sequence of actions:

1. On the far left, the product owner prioritizes the accumulated requests for the new project. The dark black circle represents the request that we trace through the iteration.
2. The requests are added to the project stack as work items and prioritized.
3. The highest priority work items are added to the iteration plan. In this case, the request of interest is the highest priority and, therefore, at the top of the stack.
4. One work item requires work from several areas to be completed:
   a. Detailed requirements are provided to clarify the request.
   b. The development team collaborates on understanding the requirements, discovers a reusable asset, and then develops, tests, and delivers the source code to the integration build.
   c. The integration build integrates the changes of several component teams into a single solution build. The bill of materials lists the changes that are delivered in the build.
   a. The test team manages the quality of the integrated solution. The test plan includes the requirement or requirements that must be verified. Test cases are written to satisfy the requirement or requirements. These test cases are executed against a build running in a specific hardware configuration. The test results are evaluated.
5. A defect is submitted and treated as a new request. If the team plans to fix the defect in the current iteration, the defect is added to the iteration work-item stack, as illustrated in Figure 1-3. Defects that will be fixed in a future iteration are added to the project stack or a future iteration plan. Otherwise, the defect is added to the stack of accumulated requests to be addressed in a future project.

6. At the end of an iteration, the team meets the acceptance criteria and delivers a working solution. Subsequent iterations build on the functionality until the final solution is delivered.

The choice of tools to implement the scenario largely depends on the size of the team, the existing tools investment, development processes, and the complexity of the solution under development. As shown in Figure 1-4, the team size can range from a small startup team to an interconnected team of teams in a global enterprise.

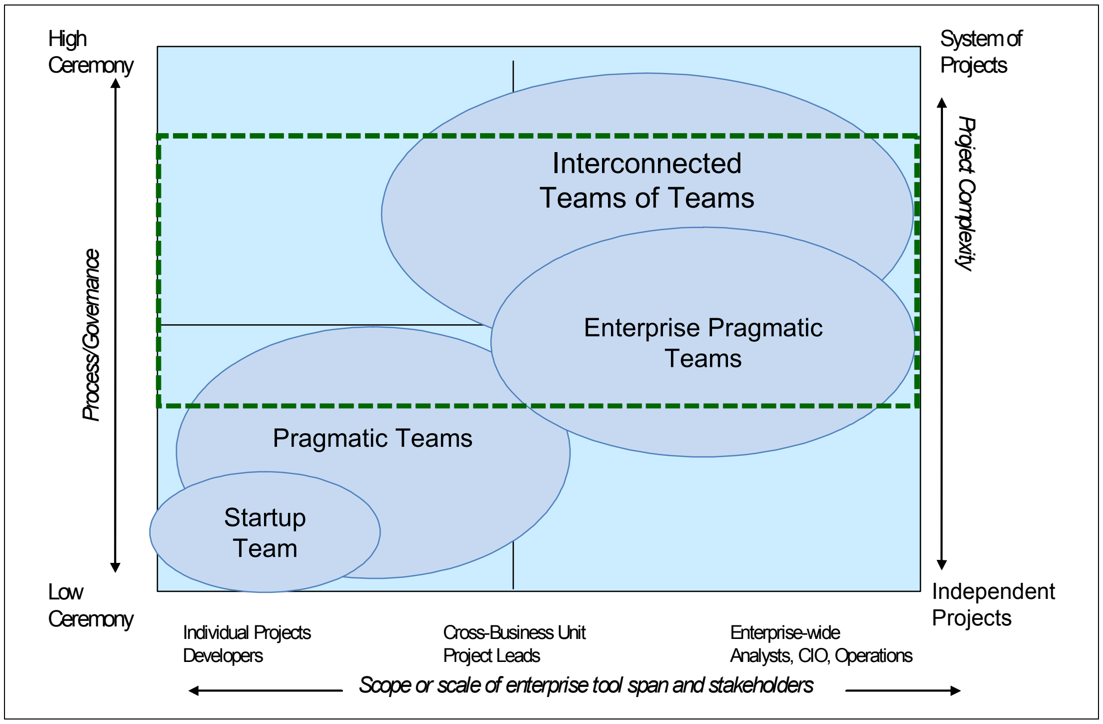

High Ceremony
Process/Governance
Low Ceremony
System of Projects
Project Complexity
Independent Projects
Interconnected Teams of Teams
Enterprise Pragmatic Teams
Pragmatic Teams
Startup Team
Individual Projects Developers
Cross-Business Unit Project Leads
Enterprise-wide Analysts, CIO, Operations
Scope or scale of enterprise tool span and stakeholders

Figure 1-4 Target team size for this book

The set of tools that is selected to support the reference scenario supports an interconnected team of teams in a global enterprise, where project complexity and process ceremony tend toward the high side. Figure 1-5 shows the products that are used in the reference scenario.

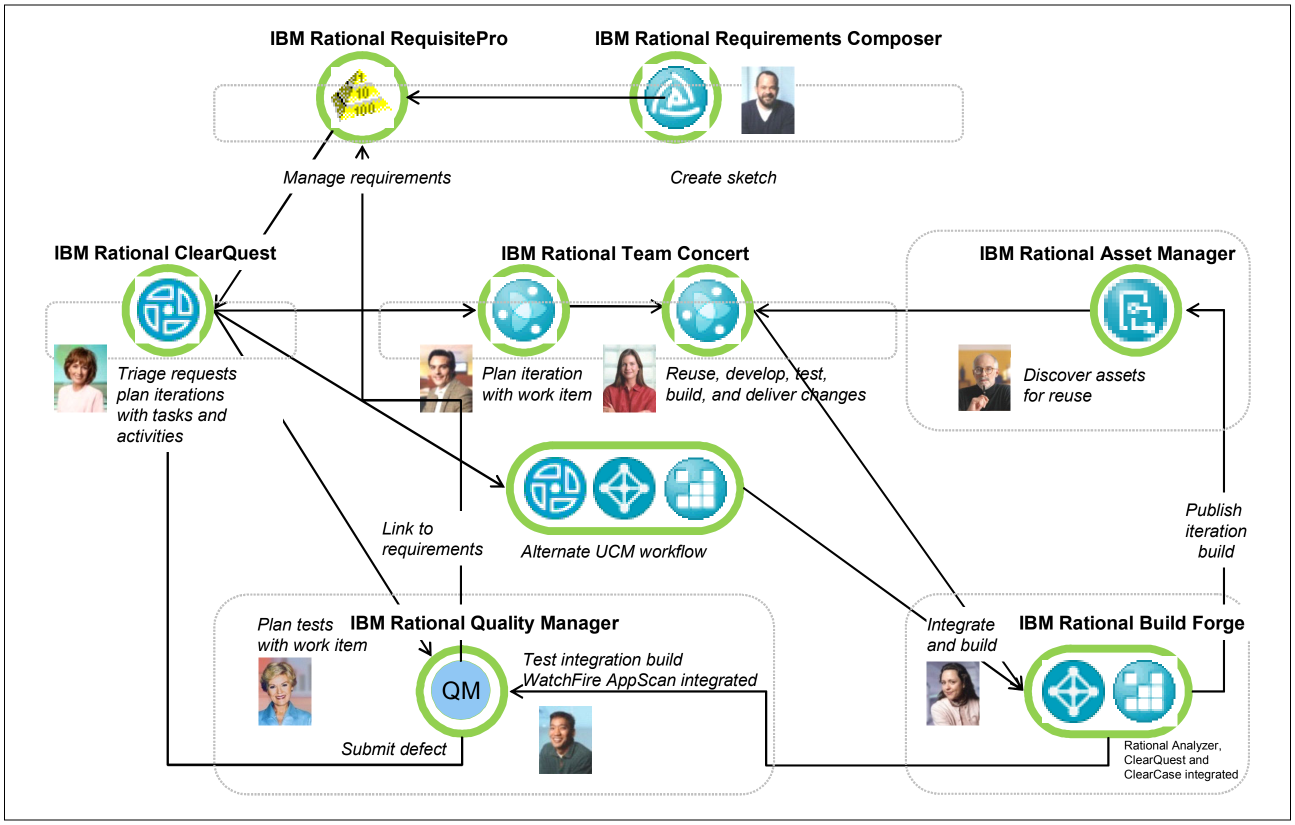

IBM Rational RequisitePro
IBM Rational Requirements Composer
Manage requirements
Create sketch
IBM Rational ClearQuest
IBM Rational Team Concert
IBM Rational Asset Manager
Triage requests plan iterations with tasks and activities
Plan iteration with work item
Reuse, develop, test, build, and deliver changes
Discover assets for reuse
Link to requirements
Alternate UCM workflow
Publish iteration build
Plan tests with work item
IBM Rational Quality Manager
Test integration build WatchFire AppScan integrated
QM
Submit defect
Integrate and build
IBM Rational Build Forge
Rational Analyzer, ClearQuest and ClearCase integrated

Figure 1-5 The products used in the reference scenario

The reference scenario shown in Figure 1-5 uses the following products:

- The project manager uses Rational ClearQuest 7.1.0.0 and a new “ALM Schema” with integrations to Rational Team Concert 1.0 and Rational Quality Manager 8.0.0.0. Rational ClearQuest is used as an information hub in this scenario and is the repository where new requests are submitted.
- The product owner uses Rational Requirements Composer 7.1.0.0 to define and illustrate storyboard requirements. The product owner also uses Rational RequisitePro 7.1.0.0 to manage and trace requirements. The requirements in RequisitePro are linked back to Rational ClearQuest to establish a traceability link between the request, the work, and the detailed requirements.
- The solution architect uses Rational Asset Manager 7.1.0.1 to discover and reuse an existing asset. The agile development team uses this asset as a base for the source code that is required to develop the change request.
- An agile development team uses Rational Team Concert to develop, unit test, and build the software component. Rational Team Concert is integrated with ClearQuest and ClearCase. An integration with Rational Asset Manager is also provided, although it is not shown in Figure 1-5.
- The release engineer uses IBM Rational Build Forge Enterprise Edition 7.1.0.0 with adapters to IBM Rational ClearCase, ClearQuest, and Rational Software Analyzer 7.0.0.0 to automate the entire build process. Rational ClearCase uses the ClearCase Connector to Rational Team Concert and manages the source for the entire solution. Rational

ClearCase manages the solution source code. Source code from Rational Team Concert is copied to Rational ClearCase by using the ClearCase Connector.

- Rational Quality Manager with integrations to ClearQuest, RequisitePro, and Rational AppScan (version to be determined (TBD)) is used by the test team to plan, manage, construct, execute, and analyze the testing effort.
- The following products are referenced, but no detailed information is provided:
  - Rational Functional Tester 8.0.0.0
  - Rational Performance Tester 8.0.0.0
  - Rational Services Tester 8.0.0.0
  - Rational Policy Tester (version TBD)

It is our hope and intention to cover a scenario that the majority of you can relate to, with full knowledge that we are unable to cover every case.

1.2 Using this book

In this section, we guide you through the contents of the book to help you form a strategy for reading this book.

1.2.1 Goals and objectives

This book has the following goals and objectives:

- Provide a blueprint for ALM and each of the disciplines with the development life cycle.
- Elucidate the value of ALM by demonstrating the handoff points that occur across disciplines in a single iteration of a software development project.
- Provide a reference scenario that characterizes the key value points for ALM.
- Provide a reference architecture for ALM deployments in enterprises that seek to support large teams.
- Document the usage of the Rational products in support of the reference scenario.
- Demonstrate how to deploy configurations for the IBM Jazz technology-based tools planned for the market in 2008.
- Demonstrate how the Rational ALM tools can support an Agility-at-Scale scenario.

1.2.2 How this book is organized

We have organized this book by using the reference scenario shown in Figure 1-3 on page 8. We describe the scenario in detail in Chapter 3, “A scenario for CALM” on page 47.

The scenario is presented in the form of a storyboard. The storyboard is divided into five major acts, each with one or more scenes. This division is similar to that of a movie script or play. Imagine the curtain being drawn and lights going on in the theater at the end of each act.

Each of the major acts stands as a milestone in the overall life cycle and covers at least one discipline in software development. Each scene demonstrates how one or more of the characters completes a specific task that contributes to the act. For example, as shown in Figure 1-6 on page 12, Act 2: Collaborative Development has five scenes. They start with the “Marco monitors component health” and end with “Diedrie conducts a team build and delivers for integration build.”

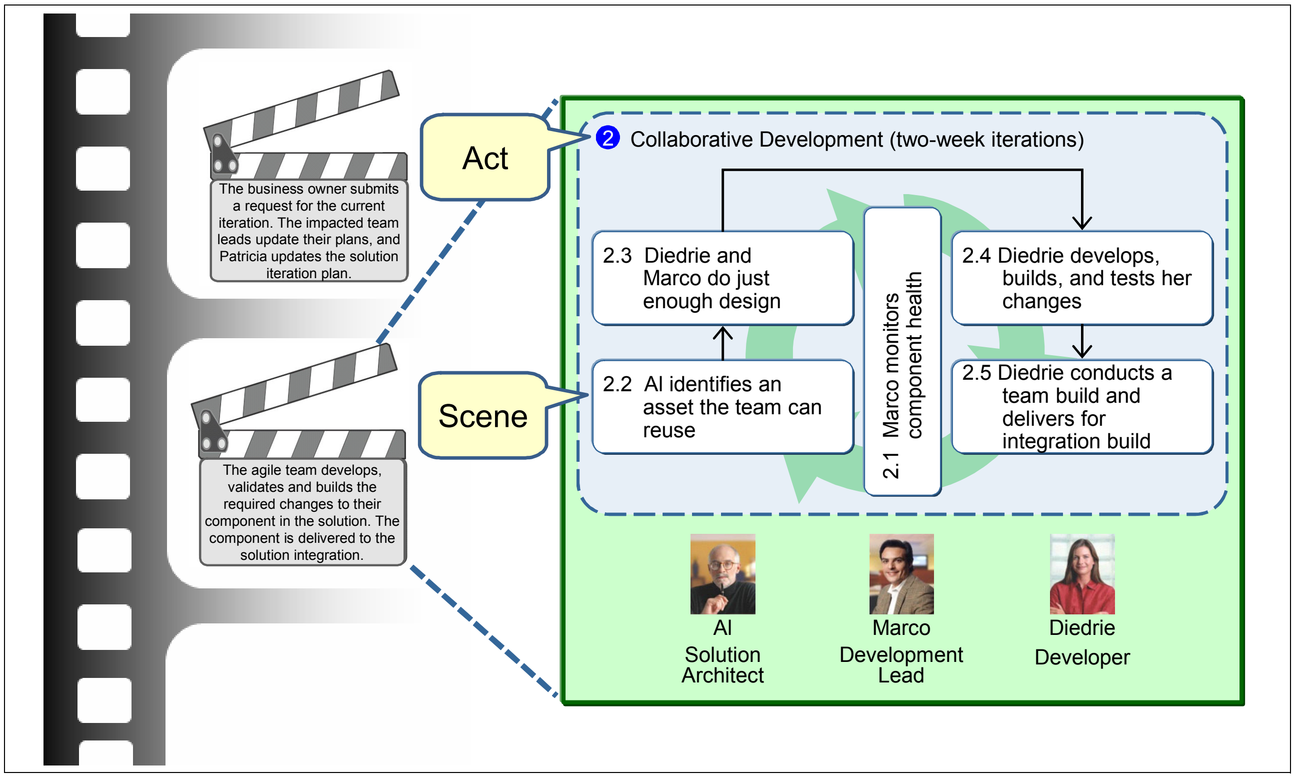

The business owner submits a request for the current iteration. The impacted team leads update their plans, and Patricia updates the solution iteration plan.
Act
The agile team develops, validates and builds the required changes to their component in the solution. The component is delivered to the solution integration.
Scene
2 Collaborative Development (two-week iterations)
2.3 Diedrie and Marco do just enough design
2.2 Al identifies an asset the team can reuse
2.1 Marco monitors component health
2.4 Diedrie develops, builds, and tests her changes
2.5 Diedrie conducts a team build and delivers for integration build
Al Solution Architect
Marco Development Lead
Diedrie Developer

Figure 1-6 A story told in acts and scenes

Each part of this book corresponds to one of the acts in the storyboard. The following parts provide a synopsis of each of the acts as illustrated in Figure 1-6:

- Part B, “Act 1: Responding to a change request” on page 77
- Part C, “Act 2: Collaborative development” on page 211
- Part D, “Act 3: Enterprise integration builds” on page 313
- Part E, “Act 4: Managing quality” on page 387
- Part F, “Act 5: Delivering the solution” on page 479

Each of these parts comprises two chapters. The first chapter in Parts B through F, which we refer to as the “about” chapter, introduces the discipline or disciplines that come into play in the act. It discusses the current market forces and presents a blueprint for that discipline. The reference scenario is introduced along with additional considerations for that discipline.

The second chapter in Parts B through F, which we refer to as the “using” chapter, describes the tools that are used, followed by a set of instructions for using the tool to complete the scenario. As stated earlier, ALM is all about the handoffs that occur across the disciplines. Therefore, a summary section called “Lifecycle collaboration” is provided to help you understand the assets that are created in the act and how they relate to assets that are referenced in the other acts. We also provide sections on measuring success and problem determination.

1.2.3 Target audience and chapters of interest

Because ALM includes all software development disciplines, this book is written with many readers in mind, each with a different level of influence and decision making power.

Project managers or *team leads* who have ownership in the delivery of complete application solutions, or components into larger solution, will find value in understanding the overall scenario provided in this book. The book clarifies the needs and benefits within ALM that can help to identify areas of improvement across a team. It also addresses the typical patterns in adopting ALM principles.

Tool administrators who have ownership of the deployment and maintenance of development platforms will find the reference architecture and problem determination sections of value. The book clarifies the deployment best practices and configurations. It also enumerates alternative approaches and suggestions on trouble resolution.

Individuals who specialize in a specific discipline, such as requirements management, development, release engineering, and testing, and who seek an understanding of how their discipline fits into overall project effort will find value in a specific part of this book.

This book can also be a source for inspiration and reference for *senior line of business* and *development managers* for deepening their understanding of ALM and the capabilities that it can unlock in the IT organization.

We consider this book an important guide for teams who are maturing their approaches to software delivery, by stepping up from source code management to change, quality, and release management. We also consider this book to be an important complement to information about the Jazz products by illustrating how a team already on the Rational Delivery Platform, including Rational ClearCase and ClearQuest, can adopt Rational Team Concert. To learn more about the Jazz products, see the following Web site:

`http://jazz.net`

To further facilitate the reading of the book, we provide a role-based guide in Table 1-1 on page 14, based on the key shown in Figure 1-7, to the sections of interest to those of you who perform a specific role. We use the smaller icon, which is shown in the lower right corner in Figure 1-7 in the role map in Table 1-1 on page 14, to indicate the topics of interest.

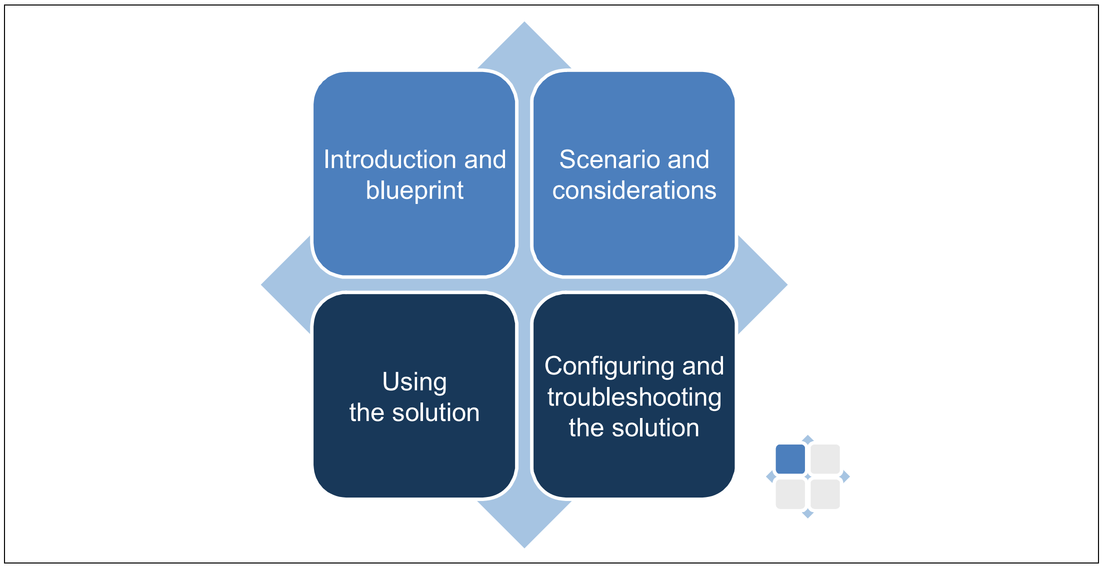

Introduction and blueprint
Scenario and considerations
Using the solution
Configuring and troubleshooting the solution

Figure 1-7 A guide to the topics of interest in the 'about' and 'using' chapters in Parts B to F

The first chapter of each part (the “about” chapter) is indicated in light blue (upper half of the map). It includes a square for the “Introduction and bueprint” section and a square for the “Scenario and considerations” section.

The second chapter of each part (the “using” chapter) is indicated in dark blue (lower half of the map). It includes a square for the “Using the solution” section and a square for the “Configuring and troubleshooting the solution” section.

For each role, the icon demonstrates which chapter (indicated by light or dark blue) and which topics (indicated by a left or right square) are suggested reading material. For example, the small icon on the right in Figure 1-7 suggests that the “Introduction and blueprint” section in the “About” chapter is of interest. The topics of less interest are shaded in light gray.

Table 1-1 provides a map for each role to the suggested sections of interest. Refer back to this table to help you understand how the content of each chapter applies to your role.

Table 1-1 Suggested reading by role

Role	**Part A** **Introduction**	**Part B** **Change management**	**Part C** **Collaborative development**	**Part D** **Enterprise iIntegration**	**Part E** **Quality management**	**Part F** **Delivery**
C*x*O	🟦🟦 ⬛⬛	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜
Product owners	🟦🟦 ⬛⬛	🟦🟦 ⬛⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦🟦 ⬛⬜
Project lead	🟦🟦 ⬛⬛	🟦🟦 ⬛⬜	🟦🟦 ⬛⬜	🟦🟦 ⬛⬜	🟦🟦 ⬛⬜	🟦🟦 ⬛⬛
Architect	🟦🟦 ⬛⬛	🟦🟦 ⬛⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜
Developers	🟦🟦 ⬛⬛	🟦🟦 ⬛⬜	🟦🟦 ⬛⬛	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜	🟦⬜ ⬜⬜
Release engineers	🟦🟦 ⬛⬛	🟦🟦 ⬛⬜	🟦⬜ ⬜⬜	🟦🟦 ⬛⬛	🟦🟦 ⬛⬜	🟦🟦 ⬛⬛
Testers	🟦🟦 ⬛⬛	🟦🟦 ⬛⬜	🟦⬜ ⬜⬜	🟦🟦 ⬛⬜	🟦🟦 ⬛⬛	🟦⬜ ⬜⬜
Tool administrators	🟦🟦 ⬛⬛	⬜⬜ ⬛⬛	⬜⬜ ⬛⬛	⬜⬜ ⬛⬛	⬜⬜ ⬛⬛	⬜⬜ ⬛⬛

2

Collaborative Application Lifecycle Management

In this chapter, we discuss Collaborative Application Lifecycle Management (CALM) and set forth some fundamental definitions that we use throughout the book. We explain how the Application Lifecycle Management (ALM) market is changing and discuss a blueprint for CALM. We also explain how collaboration occurs across the life cycle.

Specifically, this chapter includes the following main sections:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

2.1 Understanding Collaborative Application Lifecycle Management

As discussed in Chapter 1, "Introduction to Application Lifecycle Management and this book" on page 3, at the heart of CALM lies a coordinated set of handoffs between team members. This coordination involves people, process, information, and tools as the team moves a project through its life cycle. In this section, we place more emphasis on defining the life cycle and explaining how the ALM market has changed. We provide a blueprint for ALM along with a set of integration themes for tracing relationships through the cycle. We also provide additional considerations.

2.1.1 Changes in the ALM market

To understand the scope for this book, it helps to understand the context from which it is derived. The business context and priorities have changed, and as such, a new emphasis is placed on development team productivity and efficiency.

The changing business context

Before the Internet blossomed, the focus for most software development teams was either to produce a software product that was delivered on CD-ROM or embedded into system. The time to market was reasonably long, with teams enjoying 12-34 month long development cycles. In addition, the business that funded development efforts was once viewed as an afterthought, or in some cases, a necessary "evil" to the more meaningful act of developing software. The software produced by these teams was a luxury item, limited to businesses with access to computers.

However, business analysts were held accountable for launching successful new software products. They spoke with the users and listened to their needs. This constant contact with the users inspired a new vocabulary that the business analysts attempted to convey to the development teams. This void of contact between the users and the developers created a cultural gap that psychologically placed the business and the development teams at odds.

At the same time, the test team set out to find defects. While the need to find and fix defects is an important part of managing quality, this too placed the test and development teams at psychological odds. Because the developers took pride in not having defects in their source code, while the testers' job was to find them, a natural tension arose between the two organizations when their function was viewed from this perspective. The business, development, and test teams often worked in silos unaware of, or at odds with, the others' activities.

In this environment, the software development market matured down a path of providing feature-deep applications that were targeted at solving the needs of a single discipline. There are sophisticated tools for managing requirements, defects, source code, builds and test cases, where each tool targets an individual discipline, often with each having its own repository for managing the assets. For many years, software development tools vendors marketed a tool or tools for each role. As these tools were adopted, team members realized they needed to share work with each other.

The term *Application Lifecycle Management* surfaced to define the market need for a suite of integrated tools to help teams manage all of the assets in a software development project. At this time, ALM was defined by a set of roles that perform a specialized discipline with a specialized tool or set of tools. With this definition, analysts, architects, testers, build engineers, deployers, and project managers were all considered first-class members of the software development team. While this approach was successful at broadening the definition of the software development team, a side-effect of this role-based approach led to silos of capability. As such, development managers invested in the best-of-breed solution for their particular discipline.

Then the shift occurred. The Internet blossomed, and software became a household norm. It was no longer restricted to the few who had access. Software was everywhere. Schools, healthcare facilities, nonprofit organizations, households, and others all had access to the Internet. In addition to critical IT applications, software proliferated into common household devices, mobile phones, appliances, machines, and complex systems.

Software has become a part of daily life, and as such is driving revenue for the business. The connectivity provided by the Internet has had a direct impact on business-to-business and business-to-consumer transactions and processes. This has driven a quantum leap on software demands to automate, implement, integrate, and manage these new business processes. The demand to realize these processes, required for the competitiveness in the new online economy, has created a fundamental shift in the business context for creating software. The context for conducting business has changed in several dimensions:

- The focus is on business outcomes.

  The business, development, and operations teams must now collaborate to produce and manage software solutions that will keep them competitive in a global market that is open and active 24 hours a day, every day of the week. The software services provided by the IT organization must align with business needs while conforming to a larger IT strategy. The development team provides the service to the business.

- Distributed development is commonplace.

  Teams are distributed across many geographic regions, from people who work from home offices to offshore and outsource partners. Additionally leadership is also distributed with organizationally distributed teams. Co-location is rare, and team distribution is the now the norm.

  Teams must learn new strategies for working across geographic boundaries. For example, some teams have learned to follow the sun, where work completed by a team in one time zone is handed off to team members in the next time zone, where there are several hours left to the work day. An example of the extended work day is where the work starts in China, is handed to India, is then given to the eastern United States, is sent next to the western United States, and starts again in China. In each transition, several hours are added to the work day. This is an extreme case, but in some situations, such time-zone crossing activities, when used wisely, can considerably decrease the time to market.

- Platforms and application families or suites are the norm.

  ALM solutions have evolved from point products to application suites, and many companies have invested in these suites. Intellectual property is managed by these repositories. ALM solutions must work with these existing repositories while introducing new approaches to managing the software assets. In addition, strategies for software development governance and enterprise architectures place additional emphasis on cross-repository integration and sharing.

- Packaged applications and existing application integration are the norm.

  Solutions must be produced and deployed quickly. The days of home-grown solutions are quickly vanishing. While software solutions might be driving revenue, software is not necessarily a core competency of the business developing that software.

  For example, a pharmaceutical company is interested in investing in and developing solutions that are pharmaceutically related such as developing a new drug or medical device. The product is the drug or medical device, not the software that helps get them there. Therefore, enterprises are increasingly reaching for packaged applications that can be configured and deployed quickly without having a vast team of developers writing thousands of lines of code.

- Audit and compliance are business imperatives.

  Regulatory compliance issues impact many enterprises around the world. Teams must understand which regulations, if any, they must comply with before digging into the details of knowing what it means to be compliant. When understood, proper measures can be put in place to ensure that software development practices do not result in regulatory penalties.

  Regulatory compliance is a reality that requires organizations to govern how they run their business with an unprecedented audit trail. Organizations must have policies and be able to prove that the policies are enforced. For development organizations, this means that they must be able to prove what changes went into a release of software, how it was tested and with what result, how it was built, and where it was deployed. They might even be asked to reproduce a version of software that was deployed several years ago. It is no longer good enough to provide a tool for each role. Now the assets created by the team must be traceable with each new release.

Business goals with a spotlight on development efficiency

Caught in a cross-fire of these escalating industry trends, IT development organizations are being asked to meet goals that seem mutually exclusive. Teams are asked to increase product quality and accelerate time to market, even as software solutions and software development environments both become rapidly more complex.

Many business drivers compound these challenges. The key challenges are as follows:

- Lower development costs.

  Development teams are asked to effectively manage and control staff development resources by taking advantage of lower cost resources that are available through the use of on-site, off-site, and offshore software development.

- Increase staff productivity.

  Teams are expected to improve individual and project productivity to meet the backlog of business requests. This requires that they increase the current staff capability to take advantage of current and emerging technology, while at the same time quickly leverage staff across project portfolios.

- Decrease time to market.

  With reduced project delivery time, clients can bring projects online faster while incorporating more business critical features. In a 24x7 global market, agility and reducing time to market become core capabilities.

- Improve quality.

  The need for software quality has risen to the forefront of business needs. The quality of the software provided has a direct impact on the business. Standard processes, methods, and tools drive higher quality software, which in turn, drives business results.

- Increase competitive advantage.

  Software provides a critical differentiation for providing new services to customers and for opening new markets. Driving innovation requires a unique collaboration between the business, development, and operations teams.

As attractive as these benefits are, many challenges can prevent companies from recognizing these benefits. To address these challenges, managers have moved away from individual and team productivity and have focused more on organizational productivity as shown in

Figure 2-1. It is no longer good enough for a development team to "throw the solution over the wall" to the testers. Instead businesses seek to continually align software initiatives with business imperatives. At the same time, the competitive landscape demands business agility, and business agility in turn demands software development agility.

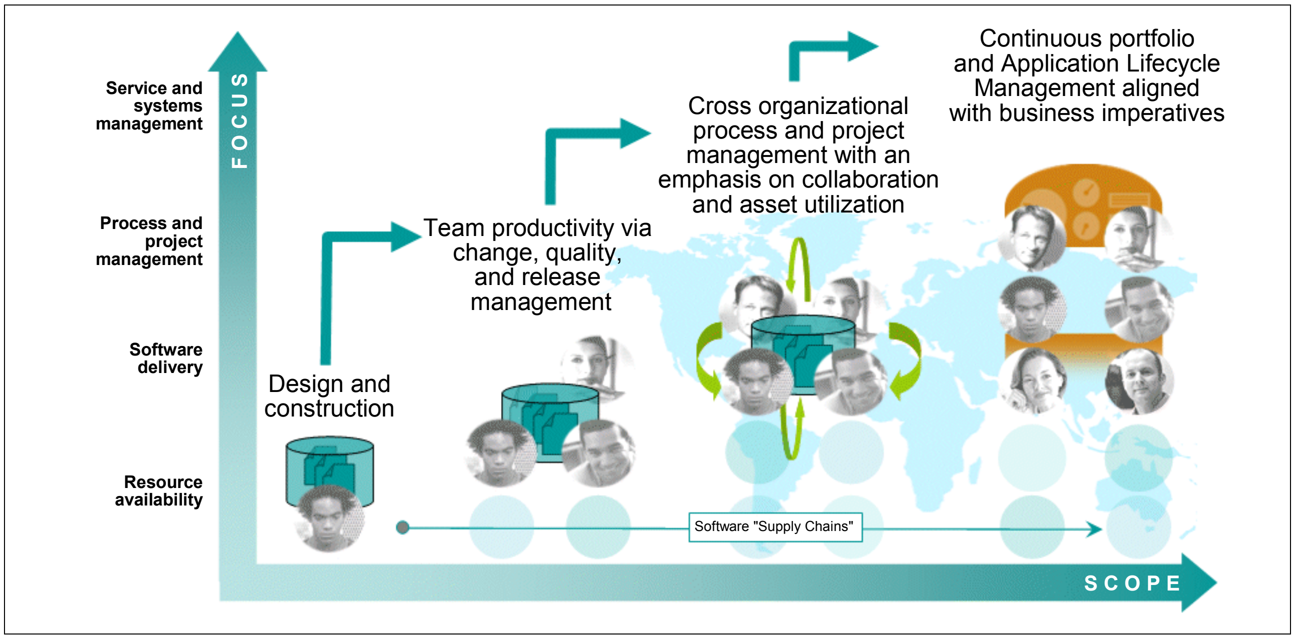

Service and systems management
Process and project management
Software delivery
Resource availability
FOCUS
Design and construction
Team productivity via change, quality, and release management
Cross organizational process and project management with an emphasis on collaboration and asset utilization
Continuous portfolio and Application Lifecycle Management aligned with business imperatives
Software "Supply Chains"
SCOPE

Figure 2-1 Business goals evolving toward the ALM of the portfolio

Software development, in effect, provides a service to the business. Successfully executing software development projects that are aligned with business imperatives provides a competitive advantage in the global marketplace. Today's software delivery challenges are mounting as the number of projects increases and the applications become more complex.

Greater challenges

While the business seeks to improve efficiency or a competitive advantage, the challenges imposed on the development team are difficult to overcome. Teams are confronted with awareness, organizational, functional, geographic, and technological barriers.

Awareness barriers

Perhaps the biggest challenge in developing software is awareness. Each discipline has its own set of tools and repositories for managing their assets. As such, it is difficult for teams to collaborate and align their efforts. Yet information about the project, the teams, and the build is critical to successfully manage the solution through the life cycle:

- Project awareness

  With project teams distributed around the world, collecting project metrics is extremely difficult. The rate of change requires the management of multiple releases and immediate re-tooling to support fixing a high priority defect in the production system. Tracking the work of all of the individuals on the team who are distributed around the world and working on concurrent multiple releases is critical to your success, yet ripe for chaos and lack of clarity.

- Team awareness

  Amidst this chaos, individual team members must learn to effectively change context as they move from one project to another, or from release to release. Individual team members need awareness about the team they are joining and its culture, policies, and best practices. They also need to know how to configure their environment for the project, how to share changes, and how the team as a whole is progressing. In essence, individuals need awareness on how they can fit in and contribute to the team's success.

- Build awareness

  The build has a tendency to be a black box. Knowing what is planned for the next build ensures that testers are ready with test cases and test scripts when the build arrives. Also it important that the team has awareness in regard to the changes that have been implemented in the build and the defects that have been fixed. This awareness helps the test team to target their test efforts to create and use meaningful test cases. When builds fail, the team must understand the context of the failure to quickly and efficiently resolve the problem.

Organizational barriers

Unfortunately organizational boundaries tend to prevent teams from collaborating as freely as they need to be successful. False boundaries tend to prevent cross-team visibility and interaction. Embedded organizational cultures, politics, and a lack of management support can impede progress. Teams must overcome these organizational barriers to produce successful solutions.

Functional barriers

Software development starts with a focus on individual roles and practitioners. As a result, there is a wide variety of tools for each discipline. As each discipline in the organization grows, tool and vendor choices are made that best serve the needs of that discipline. Managers who are responsible for the ALM solution are faced with the complex task of integrating multiple teams using different tools from multiple vendors.

Geographical barriers

With teams distributed around the world, multiple barriers come into play. Team members must collaborate to complete their work. They must have access to the work of the other team members. Wide area networks (WANs) with high bandwidth in all geographic locations are a must. Even if these barriers are overcome, team members must adjust to understanding how and when to communicate with each other and gain respect for each other's cultures.

Technology barriers

Service-oriented architectures (SOAs) demand a new approach to software development with a whole new skill set. Organizations are faced with the challenge of retraining personnel, which can be costly and time consuming. With this transition in coding practices, combined with the offshoring and outsourcing of development efforts, a wide range of coding practices and techniques is introduced. Managers are confronted with the need to institute coding best practices and source code analysis to ensure a level of consistency in the source code.

As enterprises seek to develop and implement enterprise architectures, the need for consistency across business units becomes an imperative. Having multiple teams implement the same type of service in slightly different ways is not only time consuming and inefficient, it can lead to stakeholder confusion. Users who are confronted with the multiple approaches might perceive the enterprise as being disjointed. Such perceptions can lead to a loss of business. Establishing consistency and reuse across solutions is an important barrier to overcome.

Expanded definition of life cycle

The desire to satisfy the user's needs brings the business, development, and test teams together: Everyone must understand the user in order to produce software that will keep them competitive. At the same time, this need for 24x7 continuous availability also brings the business and IT operations together. Service level agreements (SLAs) are made to ensure that the software will be available to serve the business needs.

Additionally the development teams and operations teams must be aware of each others' needs. The development teams must understand the operational constraints of the solution they are producing. They must also know the details of the solution that can help the operations team more successfully monitor and manage the solution. The business, development and operations teams must collaborate to remain competitive.

Figure 2-2 provides the building blocks for this discussion based on an IT organization. Software is developed for users and in the context of a business:

- Business owners request new solutions from the IT organization.
- IT strategy, development, and operations serve the business by producing and managing software solutions.

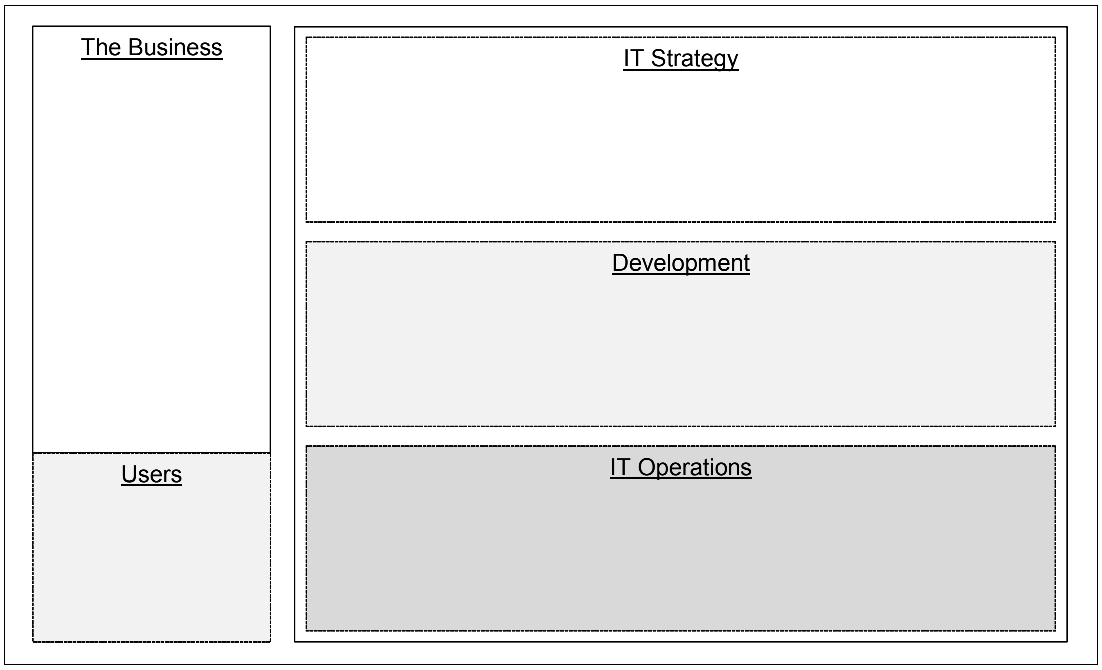


Figure 2-2 Software development in the context of the business, strategy, and operations

As shown in Figure 2-3, product owners focus on their customers' needs and driving new revenue. This figure also shows consumers of the software who are often referred to as "users" or "stakeholders." The users of the system want to complete specific tasks in the most simple, convenient, and efficient manner. A user who buys a product from a Web site has a goal of locating, purchasing, and receiving the item that they purchase. Errors, inconveniences, or flaws are reported to the product owner who is responsible for that line of business.

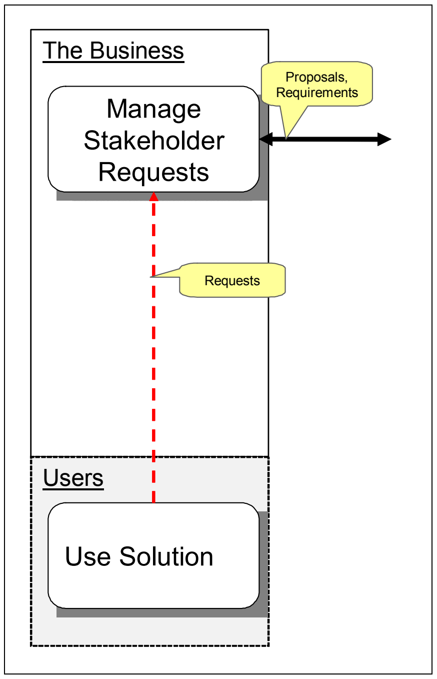


Figure 2-3 Users making requests of the business and the business owner creating proposals

The user requests are captured by the business owner. These requests and market assessments contribute to the creation of new business strategies. The business owners look for new revenue opportunities and create proposals for how software solutions can improve their revenue. This side wants to deliver value to the business. The result is the creation of a proposal (business case) for a new or upgraded business system. This occurs in the line of business. However collaboration with members of the IT team can take place.

As illustrated in Figure 2-4, the proposal is sent to a board that manages the portfolio. Here, the projects proposed by the lines of business are compared and evaluated. Proposals are reviewed against the rest of the portfolio and examined in the context of the enterprise architecture. The approved proposal that contains the approved set of requirements, along with the guidance or constraints of the enterprise architecture, are provided to a development team that is tasked with implementing a solution.

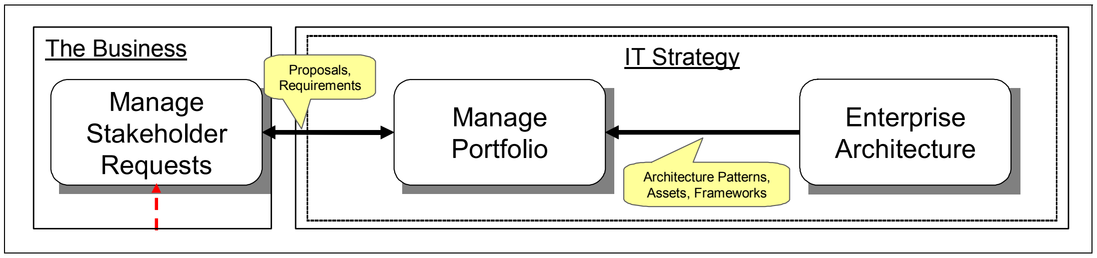


Figure 2-4 Proposals evaluated in context of the portfolio and the enterprise architecture

In Figure 2-5, funded projects move into development, where the software is designed, implemented, and tested. The development, business, and operations teams come to agreement on a set of software requirements and acceptance criteria, which relate back to the initial business proposal that was created by the line of business. Iterations involving design, implementation, and testing take place until a release candidate is deemed ready for operations. These iterations include requirements elaboration, design, development, and testing.

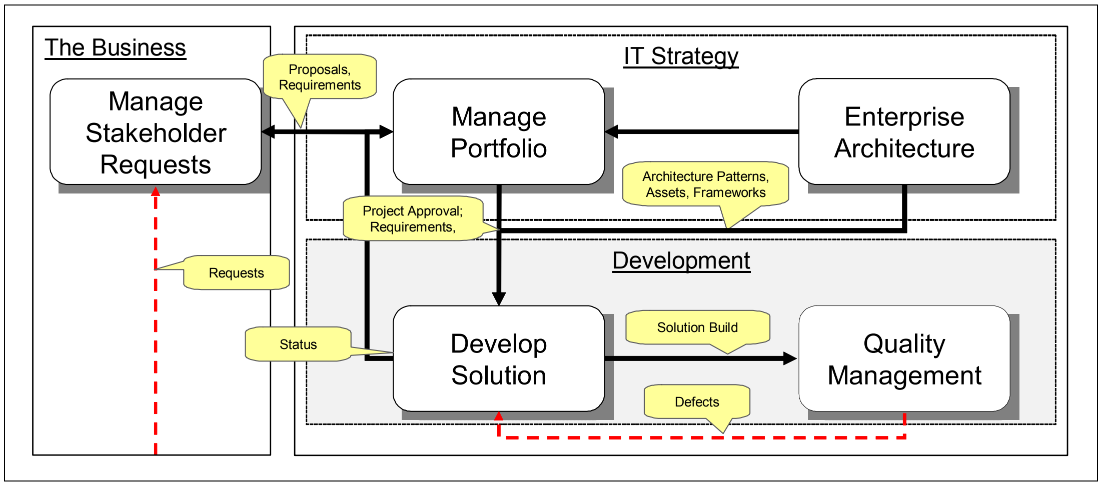


Figure 2-5 Funded projects moving into development

Feedback loops ensure continuous improvement. For example, a defect found during testing is sent back to development where it is fixed, and a new build is delivered to the test organization. Throughout this process, status reports and metrics are used to govern the release and to compare the project against the rest of the portfolio. When the solution team has implemented and tested all of the requirements, a release candidate is given to operations.

A release candidate transitions through multiple test environments, such as integration, system, security, performance, and acceptance testing, before being approved for production. In Figure 2-6, the line between quality management and transition is a gradient of ownership between development and operations from one enterprise to the next. At some point, however, the readiness of the release candidate is approved, and the solution is deployed into production. The IT Information Library defines the release management process for managing the deployment of large releases.

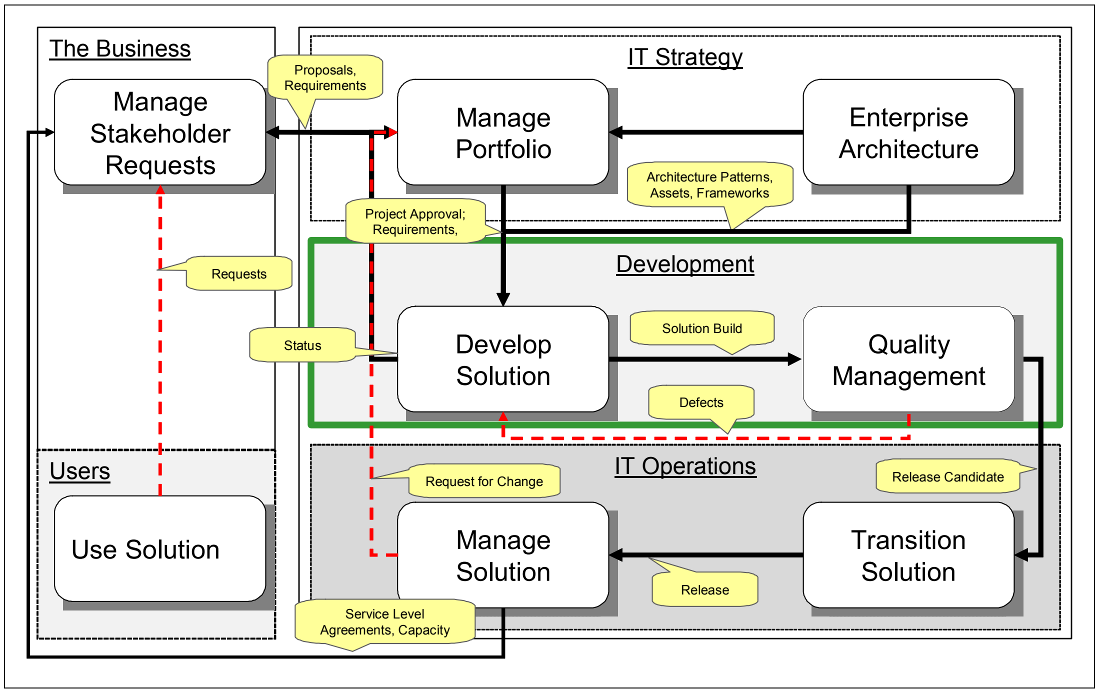


Figure 2-6 The IT life cycle

After the solution is rolled into production, it is monitored and managed by the operations team. Problems found by operations are reported, prioritized, and assessed. The IT Information Library defines the change management process for handling a request for change. The IT life cycle continues with SLAs, and capacity plans are negotiated between operations and the line of business.

Given this view of the IT life cycle, the scope of this book resides in the development domain, which is highlighted in green in Figure 2-6. Portfolio management, enterprise architecture, and IT operations are not within the scope of this book. Figure 2-6 illustrates two major organizations in an enterprise. The disciplines of change and requirements management, analysis and design, collaborative development, build and release management, and quality management are all contained within the development organization.

2.1.2 A CALM blueprint to streamline software delivery

CALM enables enterprises to effectively develop and deliver software solutions. The goal of a CALM solution is to streamline a team's ability to deliver a release of software. To address the needs of the ALM market, IBM Rational has produced and delivered a CALM blueprint, which is illustrated in Figure 2-7 on page 25.

CALM defines the need for a project dashboard to continually improve and monitor software projects along with discipline-specific modules for team members to act upon and complete their work. The Lifecycle Service Integration layer provides the platform for integrating the entire team and the various repositories.

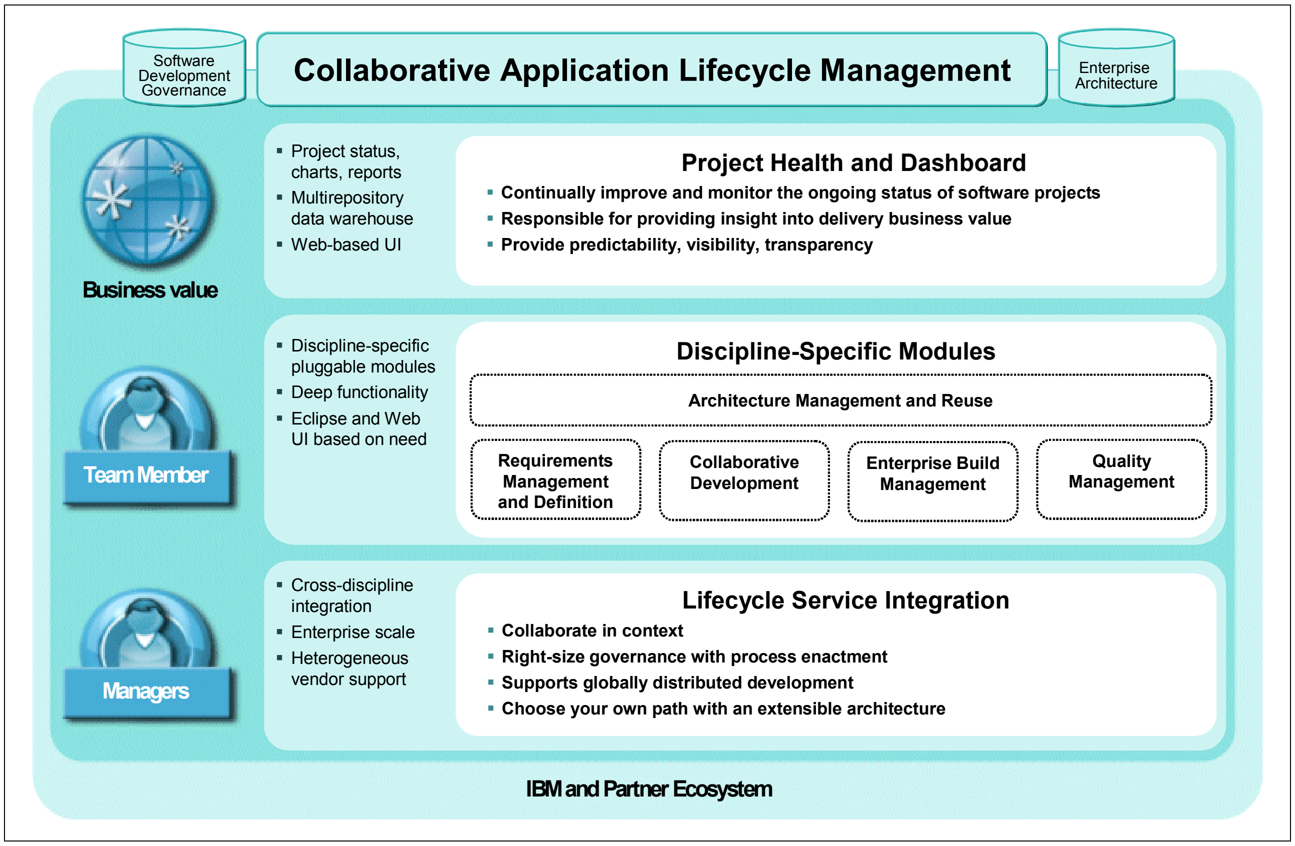

Software Development Governance
Collaborative Application Lifecycle Management
Enterprise Architecture
Business value
Project status, charts, reports
Multirepository data warehouse
Web-based UI
Project Health and Dashboard
Continually improve and monitor the ongoing status of software projects
Responsible for providing insight into delivery business value
Provide predictability, visibility, transparency
Team Member
Discipline-specific pluggable modules
Deep functionality
Eclipse and Web UI based on need
Discipline-Specific Modules
Architecture Management and Reuse
Requirements Management and Definition
Collaborative Development
Enterprise Build Management
Quality Management
Managers
Cross-discipline integration
Enterprise scale
Heterogeneous vendor support
Lifecycle Service Integration
Collaborate in context
Right-size governance with process enactment
Supports globally distributed development
Choose your own path with an extensible architecture
IBM and Partner Ecosystem

Figure 2-7 Rational Software - Collaborative Application Lifecycle Management blueprint

Project health and dashboard

We cannot control what we cannot measure. How do you know that you are done if you do not have a plan? Planning begins when the project team receives funding and continues through the development process and into the next version of the solution. In the old days of project management, inordinate amounts of time were devoted to building a complex Gantt chart that, when printed, created what seemed to be wallpaper lining the office walls. Keeping such a chart up to date with the each team member was a complex and error prone task. The information that went into the chart rarely aligned with the actual work effort of the team. Garbage in, garbage out. Gantt charts are nice, but do they or hinder your ability to execute?

Today, the act of planning is continuous and agile. In this context, just enough planning is performed to get the development team moving on an iteration. The plans are constantly assessed and updated based on the team’s progress or velocity. Planning is more like driving an automobile, where the manner of driving constantly adjusts to changes in the road. Most importantly, plans are built from the bottom up.

A cadence to the iterations might be set by some of the team leads, but the estimation of the work effort comes from the people who must do the work. This estimation of work by the individual team member then rolls up to an overall dashboard for the iteration. Any team member can look at the plan and see the health of the project. Some team members will have their work contained, while others might be overloaded. The work can be reassigned to other team members, and the plan automatically reflects the change. Doing so gives an accurate indication of the health of the project with the ability to actively manage the team’s work load.

It is one thing to have an accurate read on one project, but it is quite another to see accurate information across multiple projects that leverage many repositories for storing data. This bird's-eye view of software development helps managers to more accurately identify which projects are succeeding from those that need additional attention.

As businesses seek to streamline their development organizations and manage entire portfolios, a common reporting platform across the enterprise is critical to successful planning, execution, and governance of software development projects. Enterprises need to deploy a consistent set of reports and dashboards that measure project health and status. Such a reporting platform should leverage industry standard data access and storage methods and be extensible to include third-party data, business intelligence, and reporting tools.

Discipline-specific modules

In this new world of CALM, there is still a need for targeted functionality. The difference is that these modules plug into an integration platform, so that the assets produced by one team member can be consumed by another. In Eclipse, there are views that contain the discipline-specific user interfaces. By changing a view in Eclipse, you have access to an entirely new set of functions. In a similar vein, the management of assets requires pluggable modules for each of the disciplines. Each module provides the functionality that is needed to perform that discipline. A team can pick and choose from the modules that they want to deploy for their projects.

Architecture management and reuse

The discipline of architecture management and asset reuse includes architectural models and reusable assets. Architectural patterns from enterprise architecture are applied to specific software development projects. Assets that have been identified as reusable have versions and are managed in a common repository. Reusable assets can come in the form of source code, requirements, test cases and test scripts, build scripts, and models.

In the context of CALM, these assets are identified and incorporated into a current software development project. Additionally, the software development project might develop an asset that the team believes is valuable to the rest of the organization. New assets are submitted to the repository for review and use by the rest of the organization. These assets have their own life cycle. However the key in this context is the ability to find, consume, and contribute reusable assets to the repository.

Requirements management

The discipline of requirements management begins after the business has documented and prioritized the direction and business needs of the solution. The business provides a project proposal that captures the vision, stakeholder goals, and business plan for the solution. On the development team, the analyst uses the proposal to define the requirements in a form that the development team can act on. Where the business details the need, the analyst defines what the solution must do. In some cases, as in this book, the business owner and the analyst roles are performed by the same person. On agile teams, the product owner and developers often collaborate on analyst-related activities.

The analyst uses a battery of techniques to drive to the next level of detail for the development team, from business process models with predefined key performance indicators, to sketches, storyboards, keywords, process flows, and diagrams. Rich text and images live side by side to help analysts convey their needs by using a rich and detailed vocabulary. The analyst works closely with the business to ensure that the detail they are providing meets the initial intention of the business owner. Those details that have been confirmed by stakeholders are then organized, managed, and tracked as requirements. Requirements are assessed to ensure completion, accuracy, and traceability.

Analysts must be able to enrich requirements with sketches, storyboards, and business process flows; to manage requirements over time and across the life cycle; and to collaborate with the business stakeholders to ensure that they capture the right requirement.

Collaborative development

Software is the result of countless conversations. Development has become a team sport where teams of developers, often geographically distributed, develop great software in concert. Co-located or geographically distributed team members require a new set of collaborative development principles and tools.

A key capability in collaborative development is to open information to external contributors. Product owners, analysts, program office, and other cross-organizational stakeholders need easy access to project information to provide timely input. Development teams need a central repository that supports their geographic distribution. Source code, work item management for iteration planning, and build management are all important aspects of collaborative development. Teams need insight into the project status to enable them to respond to change and track project health.

Enterprise build management

Release engineering provides the link between the development and testing teams. This discipline involves collecting and integrating the changes made by the development team, ensuring that the solution compiles cleanly, running verification tests and staging the solution to the test team.

Producing the build is far more than just compiling an application. The build process involves an intricate set of steps that, when chained together, create an assembly line for software delivery. Some of these steps must run sequentially while others can be performed in parallel. Complex applications can take many hours to build. Having the option to run parallel tasks across a pool of available worker machines can reduce the build time significantly.

Additionally builds often occur for more than one environment, with each environment requiring a different set of variables to compile correctly. The creation of a powerful and automated assembly line requires access to a host of third-party tools such as source code control, change management, and static analysis tools. Web-based dashboards provide real-time insight into the status of a running build, along with a rich history of previously run processes. Automating this function includes the need to provide a full audit trail of what happened. Knowing what is included in the build helps testers to target their test effort. Therefore, a rich bill of materials that describes the build must be produced. Similar assembly lines can be created to deploy the solution onto a test server upon the request of the test team.

Quality management

Quality management is a shared discipline across the organization, where the development team (developers and testers alike) seeks to understand the business and delivers what the business needs. With quality management, every team member contributes to the quality of the release. The business owner tests the requirements with their stakeholders. Architects test the architecture against the requirements and the operational constraints. The developers test their change to the code before delivering their changes. The release engineering team conducts build verification tests to confirm the quality of the build before staging it for the test team. The test team conducts manual, functional, performance, security, and services tests against the solution to validate that the implementation against the requirements. The solution leads test the release against a set of exit criteria prior to approving it for operations.

Storing test assets on a file system is no longer acceptable. As the name *quality management* implies, all test assets must be managed in a meaningful and constructive manner. The test effort itself involves planning the effort, constructing the test cases that were planned, and executing test cases against a build that has been deployed in the test lab.

Testers also own a set of test servers in a test lab. Knowing which servers are in the lab, which are available, and how they are currently configured is a critical element of the test effort. Analyzing the results of the effort and identifying strategies for improvement are also involved. By having the assets managed and by executing against a plan, the test team can streamline their work, reduce redundant efforts, analyze the results, and track and report their progress.

Life-cycle service integration

CALM involves coordinating the people on the team, so that the activities they perform and assets they produce contribute to the delivery of a software application in an efficient and streamlined manner. It is much like coordinating an orchestra. If every musician plays notes of their choosing at any point during the concert, the result is a cacophony of noise. Instead, the musicians know when they are to play and which notes they are to play at specific points in time.

Well-known approaches for coordinating development activities involve process definition and enactment. The Eclipse Way,[1] the Open Unified Process,[2] and the Rational Unified Process® (RUP)[3] are three approaches for coordinating the team activities that scale from small to large teams.

Repositories are used to manage the resources that are produced by the team. These include, but are not limited to, source code control, defect, test, and requirements management systems. These systems are designed to manage the assets over time, and in many cases, through multiple revisions.

Collaboration across the disciplines is key to transforming from silo-oriented disciplines to collaborative software development teams. The life-cycle service integration layer ensures that collaboration on the server side enables cross-discipline collaboration.

IBM and the partner ecosystem

As discussed in "Expanded definition of life cycle" on page 20, the life cycle continues to expand to include disciplines beyond the traditional development team. A CALM solution must embrace third-party solutions and encourage the formation of an ecosystem. All layers of the blueprint call for and anticipate third-party participation, from an extensible reporting platform and third-party discipline modules, to extensions to the life-cycle service integration platform.

Enterprise architecture and software development governance

Enterprise architectures and software development governance influence the application lifecycle, but are not explicitly covered in this book. To understand the relationships and differences between software development governance, enterprise architecture, and CALM, IBM Rational has produced and is executing against an enterprise software development blueprint as illustrated in Figure 2-8 on page 29.

Software development governance addresses the way that the deliverables of the software development process evolve from cradle to grave. It also addresses the responsibilities, decision rights, and policies that drive development teams to effectively manage the life cycles of those deliverables. Packaging and tracking the effectiveness of governance

[1] `http://www.eclipsecon.org/2005/presentations/econ2005-eclipse-way.pdf`
[2] `http://www.eclipse.org/epf/`
[3] `http://www-306.ibm.com/software/awdtools/rup/`

solutions are key to the success of how an organization approaches their governance efforts. Software development governance solutions should also be enactable within the development environment to enable broad adoption of the solutions within the governed organization. The effectiveness of software development governance is ultimately measured in the incremental value experienced by the business that has been bolstered by the software that is being produced with the governed development process.

Enterprise architecture is a methodology to organize business processes and IT infrastructure as a way to reflect an organization's operational model. Enterprise architects play a key role in organizing and aligning IT investments with business strategy. The practice of enterprise architecture has also evolved into a broad category of activities that are designed to understand, optimize, and communicate the structure and relationships between various business entities and elements. Included in these practices are business architecture, performance management, organizational structure, and process architecture. *Actionable enterprise architecture* is a focus on using enterprise architectures to effectively drive solution implementation across an organization and improve business performance and productivity.

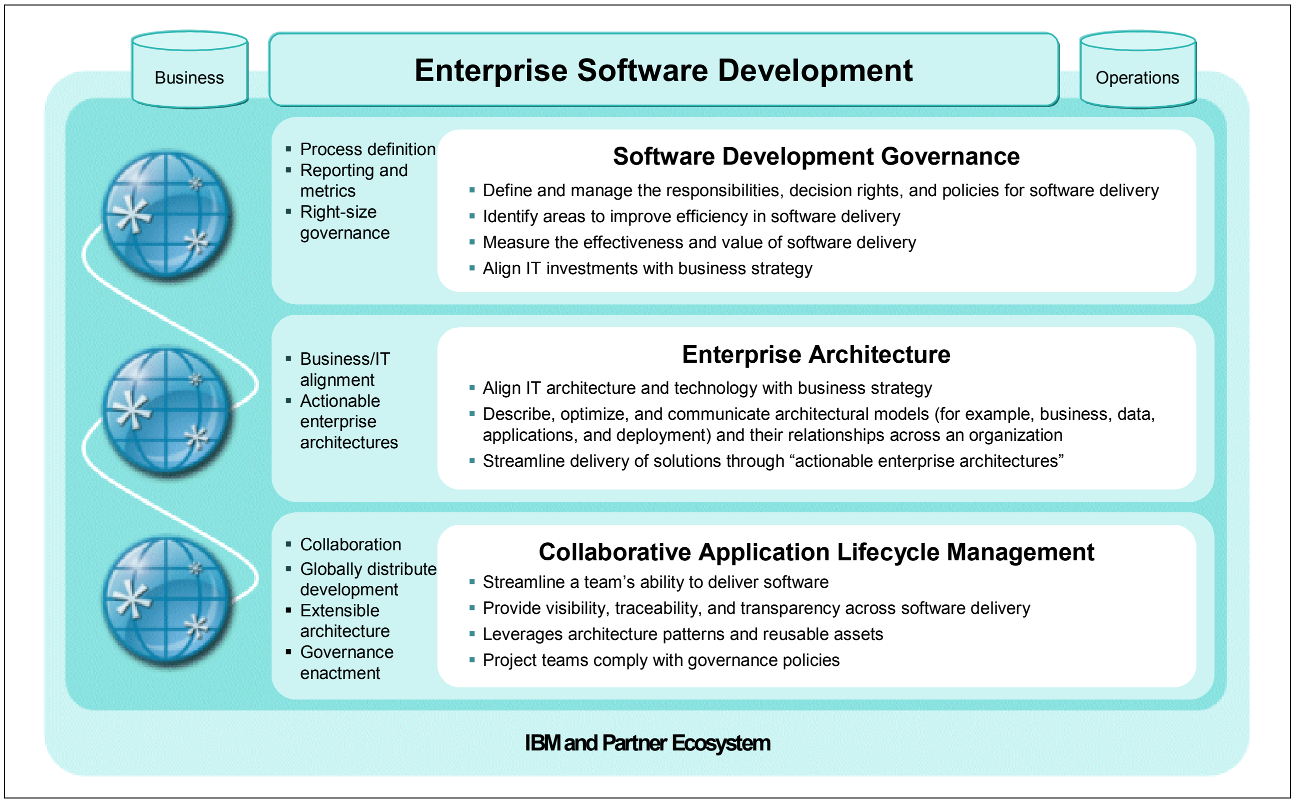

Business
Enterprise Software Development
Operations
Process definition
Reporting and metrics
Right-size governance
Software Development Governance
Define and manage the responsibilities, decision rights, and policies for software delivery
Identify areas to improve efficiency in software delivery
Measure the effectiveness and value of software delivery
Align IT investments with business strategy
Business/IT alignment
Actionable enterprise architectures
Enterprise Architecture
Align IT architecture and technology with business strategy
Describe, optimize, and communicate architectural models (for example, business, data, applications, and deployment) and their relationships across an organization
Streamline delivery of solutions through "actionable enterprise architectures"
Collaboration
Globally distribute development
Extensible architecture
Governance enactment
Collaborative Application Lifecycle Management
Streamline a team's ability to deliver software
Provide visibility, traceability, and transparency across software delivery
Leverages architecture patterns and reusable assets
Project teams comply with governance policies
IBM and Partner Ecosystem

Figure 2-8 Enterprise software development blueprint

2.2 Life-cycle collaboration and management

In this section, we introduce the key themes to consider for CALM and explain the best practices. Details about how each best practice manifests in the life cycle are provided in subsequent chapters and in the context of a CALM scenario.

2.2.1 Success indicators

CALM is analogous in that certain roles produce assets at particular points in the life cycle. To succeed at this orchestration of people and assets, a CALM solution relies on the following pillars, which are shown in Figure 2-9, as underpinnings to effectively develop a release of software:

- Collaboration
- Distribution
- Traceability
- Automation
- Continuous improvement

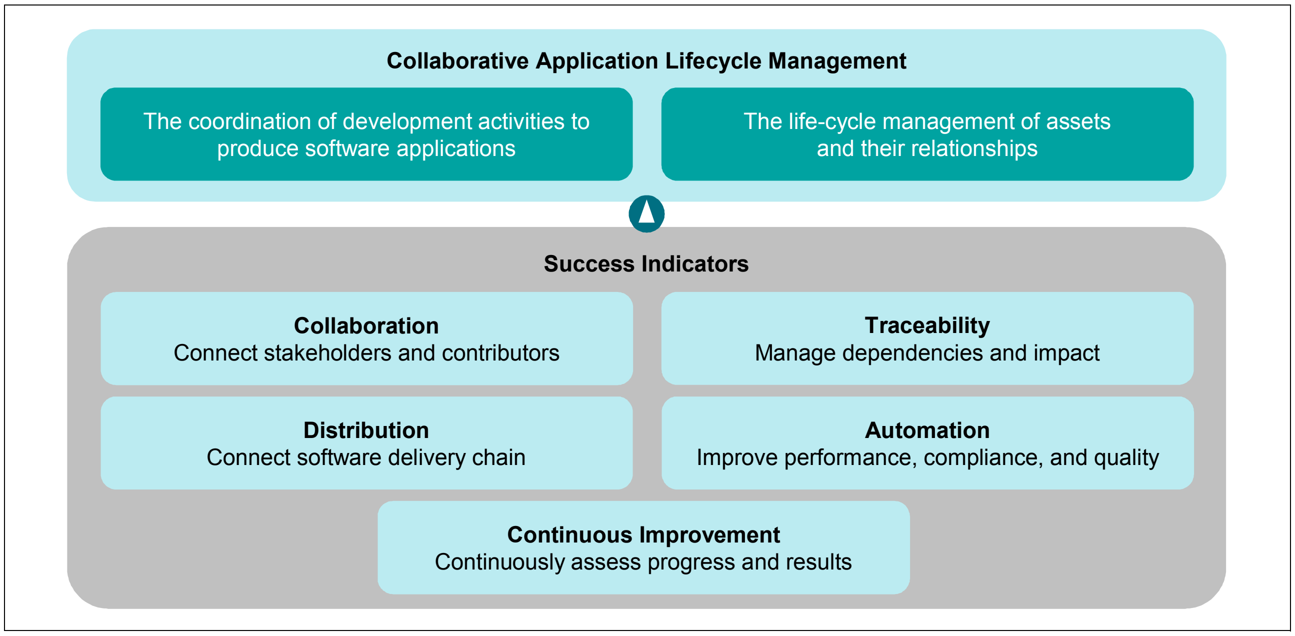

Collaborative Application Lifecycle Management
The coordination of development activities to produce software applications
The life-cycle management of assets and their relationships
Success Indicators
Collaboration
Connect stakeholders and contributors
Traceability
Manage dependencies and impact
Distribution
Connect software delivery chain
Automation
Improve performance, compliance, and quality
Continuous Improvement
Continuously assess progress and results

Figure 2-9 Success indicators for Collaborative Application Lifecycle Management

Collaboration

The "collaborator" is the new role. While each discipline has a set of best practices and common approaches, CALM focuses on the synchronization of the entire team and the handoffs that occur across domains. These integration points facilitate a streamlined development process, or when they are missing, create friction and chaos among the team members. With CALM, it is not satisfactory for a team member to work in a vacuum, solely focused on completing their own tasks. Rather, CALM demands collaboration across team members.

By having all team members aware of what the others are doing, choices can be made that best facilitate the progress of the whole. Members of a soccer team must collaborate to move the ball down the field, or to prevent the opposition from scoring. If each team member acted alone without regard for the other players on the field, their chances for success are reduced dramatically. To improve their chances o f winning, they synchronize their moves to respond to the current situation on the field. All team members must be adept in all of the skills required to play the game.

Similarly, software development is a team sport. Being successful in software development does not mean that you act alone as an analyst, developer, and tester with total disregard for the others. Today, the best team members are the *collaborators*. Collaborators work with each other and respond to the current situation on the project. The agilists were the first to discover this new power of collaboration. Now everyone realizes the need.

Software solutions are the product of many conversations. Team members must be able to interact in many ways as shown in the following examples:

- Team awareness enables teams to know who is online so that they can have a discussion.
- Instant messaging enables team members to instantly communicate regardless of their geographic location. The system tracks the users and whether they are online, thus enabling others to right-click a user name wherever it appears and begin an instant message session. An example is a developer who collaborates with a tester over a defect.
- With RSS Feeds, team members can subscribe to and be notified about events that occur within the repository. Users can subscribe to an artifact and receive notification when someone has made changes, or users can subscribe to the build process or other system level events to receive updated notifications.
- Ratings enable teams to rate the quality or value of a managed resource, thus providing a community view of the perceived value. Poorly rated assets can be improved upon, while highly rated assets are used more often, thus improving the overall quality of the shared assets.
- Tagging enable individuals to tag resources and search on those tags. This frees individuals from relying solely on queries or predefined attributes. Users can tag names that are meaningful to their current work or classification system, thus making it easier to locate assets at the time they are needed.

A CALM solution must support people regardless of who they are and where they are. It must also support their conversations and the assets that they create as a result of these conversations.

Distribution

The support for flexible and agile business sourcing decisions requires a global enterprise CALM platform to integrate distributed roles, teams, workflows, and repositories into a responsive software delivery chain.

Geographically distributed teams are the norm in software development today. Whether team members are in different offices in the same country or different countries, are working from home, or are a trusted third-party provider, development teams are rarely co-located. A CALM solution must take this into consideration by providing high-speed WAN access to all members.

Additionally, the rise of outsourcing work to third-party providers introduces the need for high security standards on the repositories housing the intellectual property. Access to, and visibility of assets within, the repositories must be controlled.

Not only are the teams distributed, but the systems that manage the assets are also distributed throughout the enterprise. There is a natural tension between the need for a consolidated repository versus supporting existing investments in role-specific repositories. Fundamental to both of these needs is a critical demand to harness the chaos and complexity inherent in managing the life cycle for software solutions. Many customers view having a repository to manage all of the assets that are created during the software development process as a means to this end. However, they have already invested in point solutions across multiple vendors.

A CALM solution must connect the people wherever they are located with the assets that they need regardless of where those assets are stored.

Traceability

Traceability involves including links to other resources both internal and external to the repository. The Internet has made the value of linking pages of information obvious. In software development, the ability to link to related resources becomes a fundamental need in completing daily tasks. For example, in writing a test case, a tester must be able to view the requirement that the test will validate. The ability to click a link to the requirement in the context of the task streamlines and simplifies their ability to complete the test case.

Traceability also involves being able to traverse the relationships between resources in a system. For example, having traceable artifacts enables an analyst to ask the system to show how many test cases are written for a specific requirement. Another example is to understand how many defects exist for a specific requirement, such as a use case, feature, or enhancement request.

Traceability also enables a team to respond to a regulatory compliance audit. By having traceable resources, a team can more easily respond to the auditors' questions regarding the number of changes that went into a release, how the changes tested, and the results of the testing.

Traceability streamlines our ability to access resources in context of the work that we are attempting to complete. It also enables us to understand the relationships and dependencies between assets.

Automation

By definition, CALM involves managing a cycle. However, it is more interesting to note that there are cycles within cycles for a single software project. Some of the tasks that are completed by the team are creative in nature and require our attention and skill to resolve. Other tasks, however, are noncreative and repetitive. These noncreative tasks are ripe for automation. Test automation is a prime example and has been around for many years.

Many other forms of automation can be applied to streamline the life cycle. For example, build automation clearly brings enormous value to team members by extending the definition of "conducting the build" to automating the preparation of the build system, applying baselines to source control systems, conducting the build, running build verification tests, and packaging and staging the distribution archive for the test team. Kicking off the build can now include the full process thanks to automation.

Automation can also include "scanning" such as conducting static analysis on the source code. A static analysis scan can be conducted by developers prior to delivering their changes, or it can be added to the build process prior to compilation. This type of analysis improves the quality of the code, ensures consistency across the code base, and makes static analysis a normal operating procedure.

Security and compliance scans can be run against a running application to consistently test the application against a predefined set of criteria. These scans work in a manner similar to virus scanning applications on a home PC. The definition files are created and maintained by the vendor. Organizations that use the scanning software keep their definition files up to date and then run scans against their applications. The scanning software produces the result of the scan, and identified problems can be resolved before releasing the application into production.

Security scans test against a known set of tactics that are used by hackers. By using this type of automation, an enterprise can identify potential threats and close them prior to going to production. Compliance scans work in a similar manner, but test against certain regulations. The organization chooses the regulation with which they must comply and run the scanning solution against their application. The scan identifies areas of noncompliance, thus enabling the team to respond to potential violations.

A CALM solution must facilitate the ability to automate noncreative, repetitive tasks, while capturing the results of the automated process in a meaningful and traceable manner.

Continuous improvement

If CALM seeks to streamline the effectiveness of a software development team, then this team must continually seek areas of improvement. Continuous planning, integration, and testing are three important strategies to take into consideration.

Planning and feedback

Project planning is a continuous effort that involves evaluating the current situation and making course corrections. Project planning includes the following important dimensions:

- Cadence

  Iterations are used to “time box” the software deliverable. Many projects consist of more than one iteration. Teams establish the cadence by choosing the number and length of the iterations. In addition, policies for approaching the iterations are established. Such policies define the frequency of the integration builds, which process to use (Open Unified Process (OpenUP), Eclipse Way, RUP), code review best practices, and the timing for design, development, and solution testing within each iteration.

  Iteration exit criteria can also be set to establish the expectations for each component team. Individual teams can then self-organize within the context of the cadence set by the project leads. For example, one component team employs a continuous build strategy on their component, while another uses a daily build strategy. The example used in this book involves a four-week iteration that is established by the project leaders with a component team working in two-week iterations, where their final milestone build feeds into the larger four-week cycle.

- Transparency

  Iteration plans, work items, and build results are available to all team members and stakeholders, regardless of their geographic location or client technology, for example a Web or rich client. This information is built into the team members’ user interface with seamless integration to the real-time performance of the team. Transparency improves insight into the actual work performed and provides opportunity to review and assess the overall health of the project.

- Health

  Team health involves workload balancing across the individual team members. It also involves tracking the team’s progress toward their goals. Health also has to do with how often the builds are failing or whether the backlog of defects is growing faster than the team is closing them. However, health does not come by merely watching. Instead, health is constantly monitored with action taken to ensure that the team stays on track and the work is distributed across the team members.

- Retrospectives

  At the end of every iteration, teams conduct a retrospective to understand what worked well and what can be improved and to publish the results so that all team members can see them. Improvements are chosen and applied to the next iteration. By conducting retrospectives on a regular basis, teams create a culture of continuous improvement. By acting on ideas and implementing the improvements, teams trend toward healthy interactions and higher quality deliveries. For more information about retrospectives, see the following Web site developed by Norm Kerth:

http://www.retrospectives.com

Integration

Continuous integration can occur at several levels, including the developer, team build, and solution build levels.

As a best practice, developers implement, build, and test their changes before delivering the changes back to the team source code repository. Continuous integration starts with the developers. This best practice can be improved upon. In many cases, the build scripts or build environment that are used by each developer can differ from the build system, which can lead to problems when the developers deliver their changes. For example, what is built in their sandbox environment suddenly breaks when it is built in the team build environment.

To prevent these errors from happening, teams can use a preflight build strategy. In doing so, the developers conduct sandbox builds that use the same environment and build scripts as the team build. Such solutions as Rational Build Forge simplify the preflight build process and ensure that a consistent build environment is used by all members of the team.

Increasing the frequency of the team builds ensures that errors are found early. Some teams employ a continuous build strategy, where every source code delivery triggers a team build. For small teams, continuous build strategies ensure high quality builds and encourage a culture of clean source code delivery. For larger teams, with many developers delivering changes, continuous builds might become too difficult manage. In this case, nightly builds are more appropriate and serve to ensure that all of the deliveries from that day can build without error.

In larger enterprises, where solutions comprise multiple components, each being delivered by a different component team, integration builds become an important part of the continuous improvement strategy. Leaving the integration build for the end of the development cycle can leave the solution team open to last-minute fire drills to fix integration errors. Employing a continuous integration build strategy is in the same spirit of continuous team builds, but at a higher level. Because integration builds bring together multiple components, they occur less frequently than the component build. A weekly integration build helps the team to identify errors early, and the impacted component team can respond while the changes are still fresh in their minds.

Testing

It is no longer acceptable to reserve testing for the final weeks of the project. Rather, responsive teams integrate testing throughout the development cycle as indicated by the following types of testing. Continuous testing naturally contributes to continuous improvements in software quality.

- Developer testing

  Testing is interjected throughout the cycle. Developers are responsible for testing their changes in their sandbox. JUnit tests[4] are the norm for developers. Many agile teams take this one step further with test-driven development (TDD).

- Build verification testing

  Build engineers inject build verification tests into the automated build systems. Conducting the build no longer means “compiling.” Rather, build automation systems now include building the solution, running build verification tests, gathering the results, staging the

[4] http://www.junit.org/

build, and publishing the results of the build. Build health is a key performance indicator for the team. A build that does not pass build verification tests provides an indicator to the developers that they need to implement changes to improve the health of the build, while also warning the testers that the build might not be suitable to deploy to the test lab. A build that passes build verification tests is most likely suitable for testing.

- Test planning

  Test planning becomes a critical part of continuous improvement. By creating and executing against a test plan, the teams can begin to measure their performance. Test plans can be used to define the level of developer, build, and system testing that the team will perform for the project. Test plans can even be defined on an iteration-by-iteration basis.

  A test plan can be used to define which requirements will be tested. For each requirement, a set of test cases is defined and developed. These test cases are then executed against a build of the solution. At each point, the team can measure progress. With a managed test plan, the teams can ask and answer the following questions:

  - How many requirements do not have test cases?
  - How many test scripts need to be written?
  - How many tests are left to execute?

  By asking these questions, the team can continually measure progress and adjust the efforts accordingly.

Teams that develop iteratively include testing as part of the iteration plan. Every iteration involves developer, build verification, functional, integration, and system verification testing. Testers are involved in multiple phases of testing, from functional and integration testing to system, performance, and security testing.

2.2.2 Cycles of activity in the development organization

If you look closer at this relationship between the developed solution and quality management, you notice a tremendous amount of activity. First a best practice for developing software involves the use of *phases* and *iterations* to time box the deliverable and focus the team's efforts. Figure 2-10 illustrates the use of phases and iterations. The first four phases are defined in the Rational Unified Process.[5] The Enterprise Unified Process extends these phases by adding a Production and Retirement phase.[6]

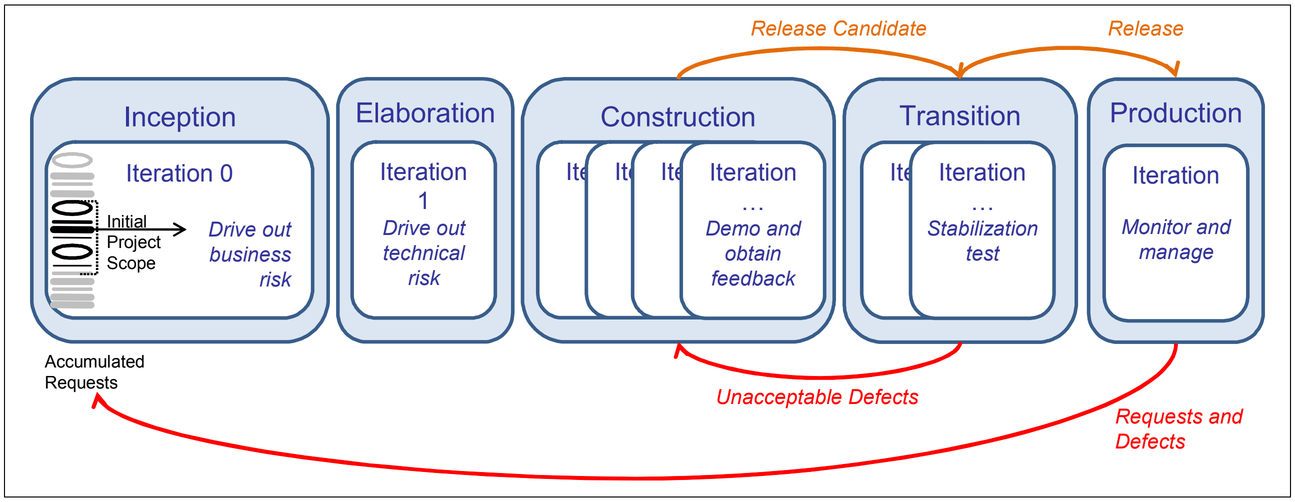


Figure 2-10 A development cycle using phases and iterations

[5] http://www-306.ibm.com/software/awdtools/rup/
[6] http://www.enterpriseunifiedprocess.com/

We summarize of each of the phases as they relate to this book as follows. For more in-depth detail about each of these phases, consult the Rational Unified Process at the following Web address:

http://www-01.ibm.com/software/awdtools/rup/

- In the *Inception phase*, the team leads and product owner come to agreement on a set of requests to define the initial scope of the project. In addition, an initial architectural vision is created, and an assessment of the project team, tools, and environment is completed. For this book, the team in the reference scenario has already completed this phase.
- The *Elaboration phase* focuses on driving out technical risk by producing an end-to-end demonstration of the solution to prove and refine the architecture. Feature details are not addressed, but rather a "skeleton" of the system is completed to prove feasibility and identify areas of technical risk. This phase has also been completed by the team in the reference scenario.
- The *Construction phase* involves one or more iterations that focus on completing the functionality. Each iteration involves planning, design, development, build, and testing. agile teams advocate multiple two- to four-week iterations, while more traditional teams might only have one iteration. This book highlights a single iteration in the Construction phase.
- After the acceptance criteria has been met, the solution moves into the *Transition phase* for stabilization testing. Unacceptable defects found in this phase cause the solution to go back to the Construction phase where the defect is fixed. The release candidate is then sent back to the Transition phase for stabilization testing until all expected tests have passed. This book ends in the Construction phase just as the handoff is about to occur.
- The verified solution is then released into the *Production phase* where it is monitored and managed. Requests and defects found in the Production phase are added to the accumulated request stack for triage. Some requests are addressed immediately with a patch sent to production. Other requests stay in the stack until the next version is funded and the cycle begins again. This book does not address the activities that occur during the Production phase.

A deeper look at the development life cycle reveals cycles within cycles of activity. Each iteration of the Construction phase includes a defined set of activities. We assert that by examining the handoffs that occur in a single iteration of a development project, the value of CALM is understood well enough to scale to a larger life cycle.

Figure 2-11 on page 37 provides an overview of the kinds of activities that are involved in a single construction iteration that are driven by a set of requests.

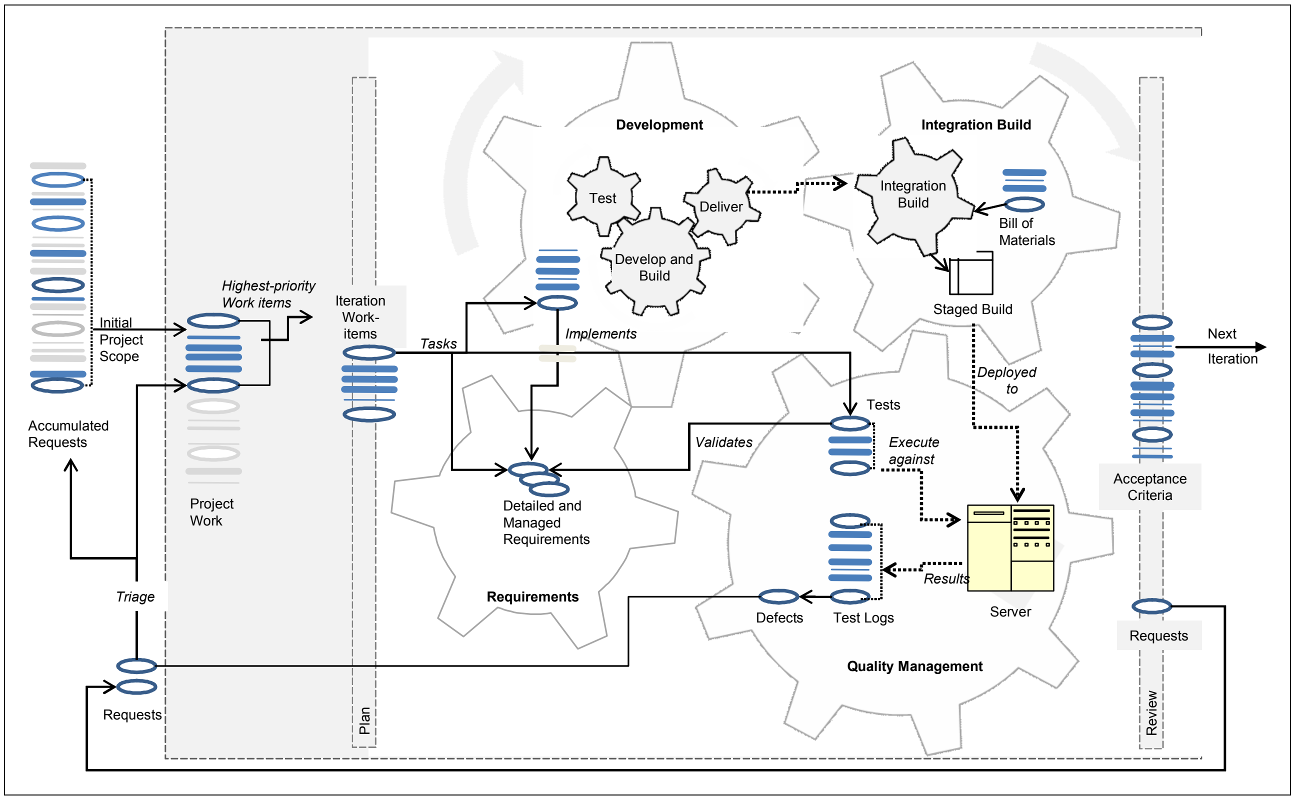


Figure 2-11 A single iteration in a development project

Figure 2-11 shows the following sequence of actions:

1. A project starts by choosing from a backlog of accumulated requests. These requests are added to the project backlog as work items.
2. The highest priority work items are added to an iteration plan, which includes the analysis, development, and testing efforts.
3. Some requests require additional detail. This requirement detail is provided by whomever is best suited for the task, such as the product owner or a developer. These requirements are used by both the development and test teams. For agile teams these requirements are captured as tests. Other teams capture the detail in a set of requirements.
4. The development team uses the requirements to write tests (*test-driven development* in agile terms). Then a developer develops the solution, builds, and tests the code in a sandbox. In agile terms, this concept is called *confirmatory testing*. When the tests pass, the change set is delivered to the team build. After the team build has met a predefined level of quality, the changes are delivered to the integration build.
5. The integration build collects source changes from each of the smaller component teams to produce an integrated build of the full solution. The integration build is staged and then deployed to the test servers.
6. Meanwhile, the testing team has been executing against a test plan by constructing test cases and test scripts to validate the high-level scenarios, the requests, and the requirements of the solution and to investigate potential scenarios that neither the development nor the business stakeholder team's considered. While the development team focuses on low-level, confirmatory tests, the testing team focuses on customer acceptance testing, feature or exploratory testing, performance testing, security testing, policy testing, and regression testing. These tests are executed against a build on a server, and the results of the tests are validated.

7. Defects are added to the request backlog where it is triaged and either added to the work-item stack or postponed and placed on the accumulated request backlog. Defects that are added to the work-item stack are treated like requirements and resolved by the developers by using test-driven development.

When looking at an iteration from a bird's-eye view, the dependencies and relationships across the disciplines become clearer. A single request can drive work that involves analysts, architects, developers, release engineers, testers, and project leaders all working toward the common goal of satisfying a single request.

2.2.3 Scaling agile methods

Agile development is on its way to becoming the dominant development paradigm. Scott Ambler, Practice Leader for Agile Development at IBM, conducts frequent surveys to gauge adoption and success rates. These surveys are showing a growing number of developers and IT managers who believe their teams are using agile methodologies. To learn more about the data and a summary of his surveys, see the following Web site:

http://www.ambysoft.com/surveys

Introducing agile development to the enterprise

When agile techniques were first developed, the teams were smaller and usually co-located. The applications that were developed were relatively small and straightforward. Since agile development has hit the mainstream, the picture has changed significantly, with enterprises seeking to bring agile techniques into larger projects.

This change in scale introduces complexity that needs to be addressed in the following areas as shown in Figure 2-12 on page 39:

- Team size

  In enterprise projects, the teams are likely to be larger, ranging anywhere from fifty to hundreds of people. The concept of a daily stand-up meeting does not make much sense with a team this size, and it becomes more difficult to get everyone together to keep information flowing.

- Team distribution

  We have asserted that globally distributed development is the norm. Effective handoffs between team members become more challenging. Accidental misunderstandings or overlooked activities become more common.

- Application complexity

  Enterprise applications can be extremely complex. Here a team of teams is employed, where each team owns a component of the larger solution, with certain team members overseeing all of the moving parts.

- Audit and compliance

  Again, we have asserted that audit and compliance are now the norm. Regulations can bring additional requirements that must be addressed in software projects.

- Internal governance mandates

  Some organizations might impose specific reporting requirements that impact the development team. By folding these requirements into the development effort, teams can avoid last minute fire drills to collect this information.

- Existing processes

  Many organizations have an existing culture and processes for software development. These processes must be taken into consideration by introducing new agile techniques into the existing approach rather than attempting to overthrow them.

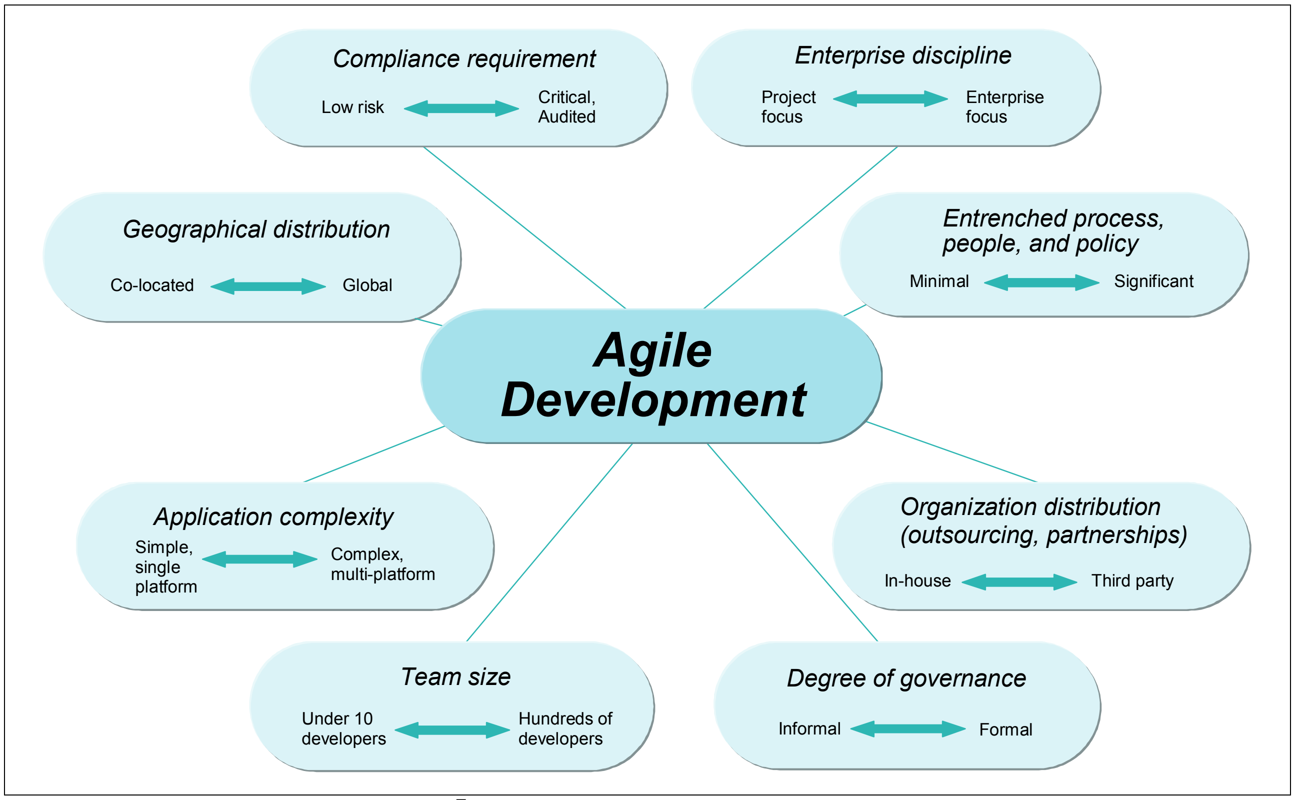


Figure 2-12 Agile mainstream challenges[7]

The good news is that IBM Rational has been helping customers deal with these types of complexity issues for over a decade and can apply their lessons to help you transition to this complex agile environment.

Relativity of agility

The need for processes and enabling technology changes as the development environment increases in complexity. Complexity can occur in a variety of dimensions. For the purposes of understanding how to apply agile development in the enterprise, the challenges are mapped into two broad categories:

- Challenges related to organizational issues
- Challenges related to technical and regulatory factors

[7] Scott Ambler, IBM Software Group, Rational, Practice Leader Agile Development and Per Kroll, Chief Architect - Rational Expertise Development & Innovation (REDI), Project Lead: Eclipse Process Framework

In Figure 2-13, the lower left corner represents the least overall project complexity, while the upper right corner represents the greatest overall picture of project complexity.

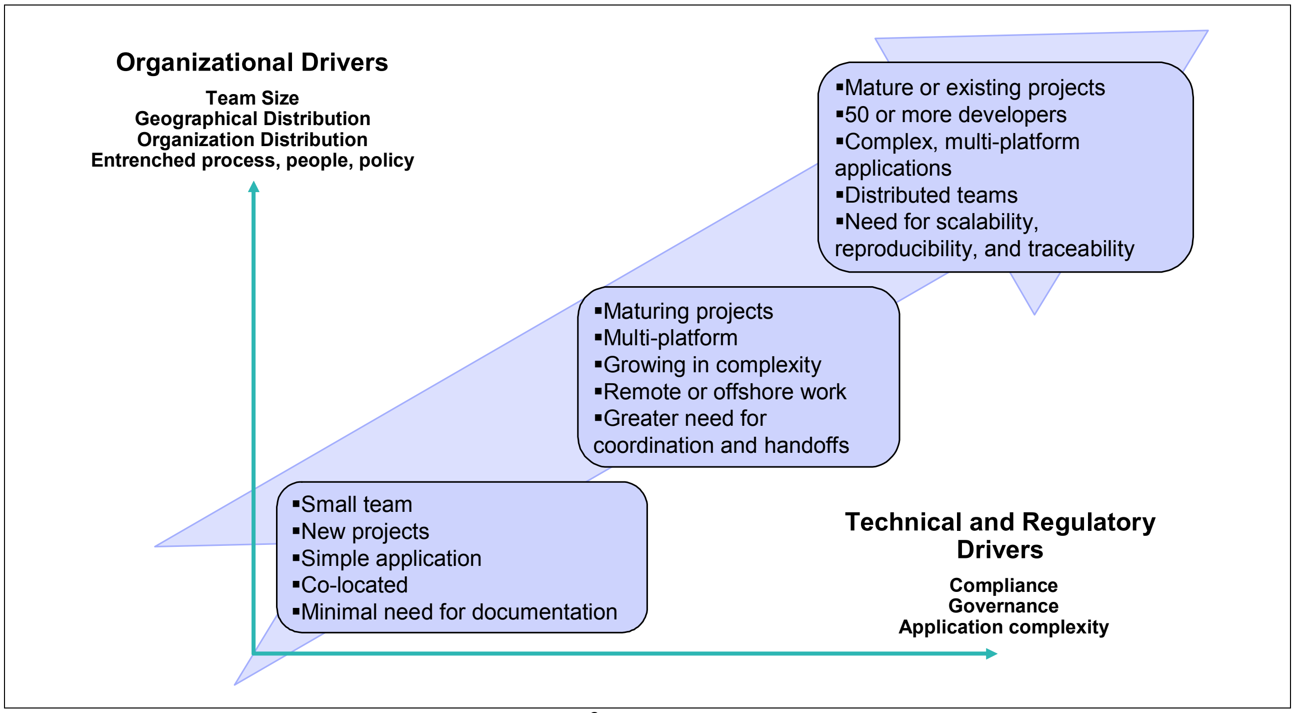

Organizational Drivers
Team Size
Geographical Distribution
Organization Distribution
Entrenched process, people, policy
▪Mature or existing projects
▪50 or more developers
▪Complex, multi-platform applications
▪Distributed teams
▪Need for scalability, reproducibility, and traceability
▪Maturing projects
▪Multi-platform
▪Growing in complexity
▪Remote or offshore work
▪Greater need for coordination and handoffs
▪Small team
▪New projects
▪Simple application
▪Co-located
▪Minimal need for documentation
Technical and Regulatory Drivers
Compliance
Governance
Application complexity

Figure 2-13 Relationship of agility to project dynamics[8]

Agile development started with small, co-located teams of about six developers, working on new projects to produce a simple application. This is represented in the lower left corner of Figure 2-13. In this situation, the organizational, technical, and regulatory drivers are simple. Geographic and organizational distribution is not an issue. The team is free to self-organize. Also compliance and governance drivers are not a consideration.

Mid-size teams are more likely to have some degree of geographic or organizational distribution and entrenched or conflicting processes. Application complexity is growing as indicated by the need to add more team members. Having more team members who are geographically or organizationally distributed creates a need for more coordination and handoffs between team members. Applications with increased complexity, such as multiplatform support, are likely to require additional testing that is performed by a team of testers. These applications also require test servers to be configured and managed in a test environment. Additionally, its more likely that the application is important, and thus more attention must be paid to analysis, architecture, and staging of multiple test environments leading up to the final deployment into production.

The larger the team is, the more complexity it faces in organizational, technical, and regulatory drivers. The situations faced by the mid-size team are amplified on the larger team. To manage the number of trade-offs, disparate approaches, and miscommunications, best practices are put in place and enforced across the team. Cultural awareness becomes an important skill to maintain healthy information exchange and motivation among team members. Compliance with regulatory and governance policies creates a need to document and comply with policies and processes. Yet information produced by the team can be spread across multiple data sources, creating a challenge to effectively managing an audit.

[8] Scott Ambler, IBM Software Group, Rational, Practice Leader Agile Development

The criticality and complexity of the application also increases. The test environments increase in complexity along with the need for a dedicated test team to catch critical defects before the application is deployed. The test environments leading to production become more complex, and the testing strategies for each of the environments increases in sophistication with the addition of performance, security, and policy testing. Careful attention must be paid to analysis and architecture issues such as recovery, fault tolerance, capacity planning, and SLA compliance. Enterprise architecture principles and patterns also come into play.

In mid- to large-sized teams, classic agile strategies must be evaluated against, and some times combined with, traditional approaches to create an approach that best suits the unique needs of the team or project. For example, attempting to hold a stand-up morning meeting with people distributed around the world suddenly seems silly for a large-sized team. However, combining frequent and short time-boxed iterations with dedicated testing efforts can yield tremendous gains. The traditional phased-based approach of the RUP can become more agile by blending fast-paced, “just enough” iterations into each of the phases.

Complexity changes to the development approach

Just as complexity affects a team’s approach to agile development, it also affects their tools and the process.

The effect of complexity on the choice of tools

In Figure 2-14, the software development tools have been mapped along a complexity continuum. Smaller teams with low organizational and technical or compliance complexity appear in the lower left corner, while larger teams with high organizational technical or compliance complexity appear in the upper right corner.

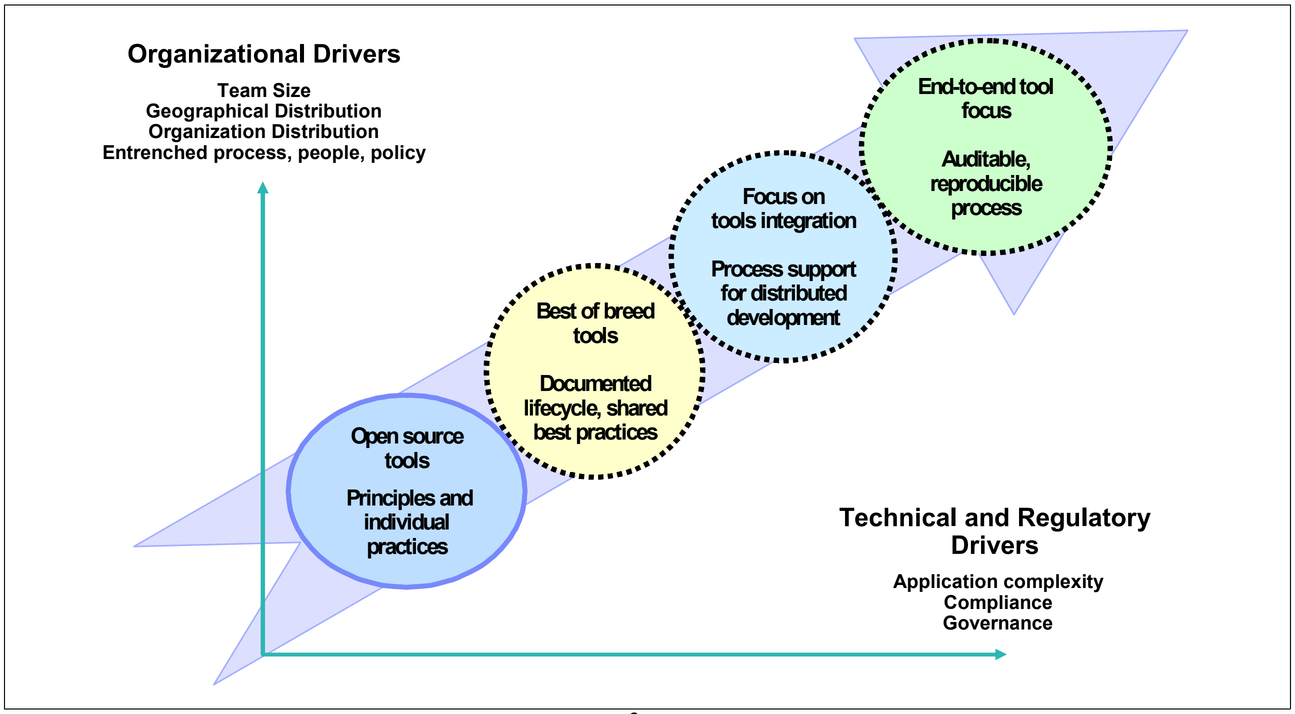

Organizational Drivers
Team Size
Geographical Distribution
Organization Distribution
Entrenched process, people, policy
Open source tools
Principles and individual practices
Best of breed tools
Documented lifecycle, shared best practices
Focus on tools integration
Process support for distributed development
End-to-end tool focus
Auditable, reproducible process
Technical and Regulatory Drivers
Application complexity
Compliance
Governance

Figure 2-14 Complexity changes to the approach for tools[9]

Smaller teams (under 10) with low organizational or technical complexity can self-organize and choose the tools they want to use on their project. A team of this size can agree to a set of principles and practices that describe how to work together and use the tools. Any

[9] Scott Ambler, IBM Software Group, Rational, Practice Leader Agile Development

modifications to the tools, principles, or processes can be discussed among the team members easily and quickly. Open source or introductory level commercial tools might be enough to get the job done and fit into the budget of a small team, provided that there is no serious complexity that the team faces.

Some degree of complexity can be enough to render these tools inadequate. For example, a small team that is developing a product that requires approval by the U.S. Food and Drug Administration (FDA) requires more documentation and traceability across the assets than most open source tools can offer. A small team that needs to work across organizations, geographically distributed teams, or with external vendors might find these tools inadequate in meeting their scalability requirements. Finally, offshore development might tip the scale toward choosing a more robust set of tools.

Best-of-breed tools can make sense when there is a coordinated effort and more than one project. The need for deeper capability increases, and each team adopts a best-of-breed tool for their discipline. The need to communicate more is rising, while the ability to simply modify the process on the fly is reduced. To avoid redundancy and improve awareness across the team, the team might find that they need to document some of the ways they work and that establishing best practices can be beneficial.

A focus on integration occurs as the complexity and team size increases. At this point, the team is large enough that likely multiple managers are in place. As a result, communication becomes more difficult as each team pushes to meet its own priorities, goals, and deadlines. Tools that are immediately integrated and automation strategies begin to look appealing. Understanding and adhering to agreed processes become a necessary part of coordinating their work.

In the most complex cases, multiple complexity variables and a large team size drive a need for an end-to-end tool and an auditable, reproducible process. These teams might be dealing with multiple projects with different requirements, complex systems running on multiple platforms, and advanced quality assurance, security and performance test requirements. Compliance or governance policies mandate that processes exist, are recorded, and are adhered to which can be proven with consistent reporting. At this level, the need to trace requirements to deployment rises to the foreground along with the need to reproduce an application at any point in time.

The effect of complexity on the choice of process

As the tool choice changes and team size and complexity increase, so does the need for process. In Figure 2-15 on page 43, process choices are mapped in similar continuum as used in Figure 2-14 on page 41. For small teams, Extreme Programming (XP), Scrum, OpenUP, and Dynamic Systems Development Method provide enough formality to coordinate the efforts of the team. These processes are designed intentionally to be lightweight and easily modified.

The Eclipse Way and the RUP can be applied to teams that fall in the middle range of size of complexity, while the RUP can scale to accommodate the largest and most complex projects. The Eclipse Way is an agile process that has been used by the Eclipse platform team. By providing support for planning, management, and day-to-day execution of iterative development, test-driven development, and architecture for larger and distributed teams, it suits the needs of the mid-range team. However, teams that are developing safety-critical systems, in need of meeting regulatory compliance requirements, or working in specific technologies, such as SOA, might need to look to the RUP instead.

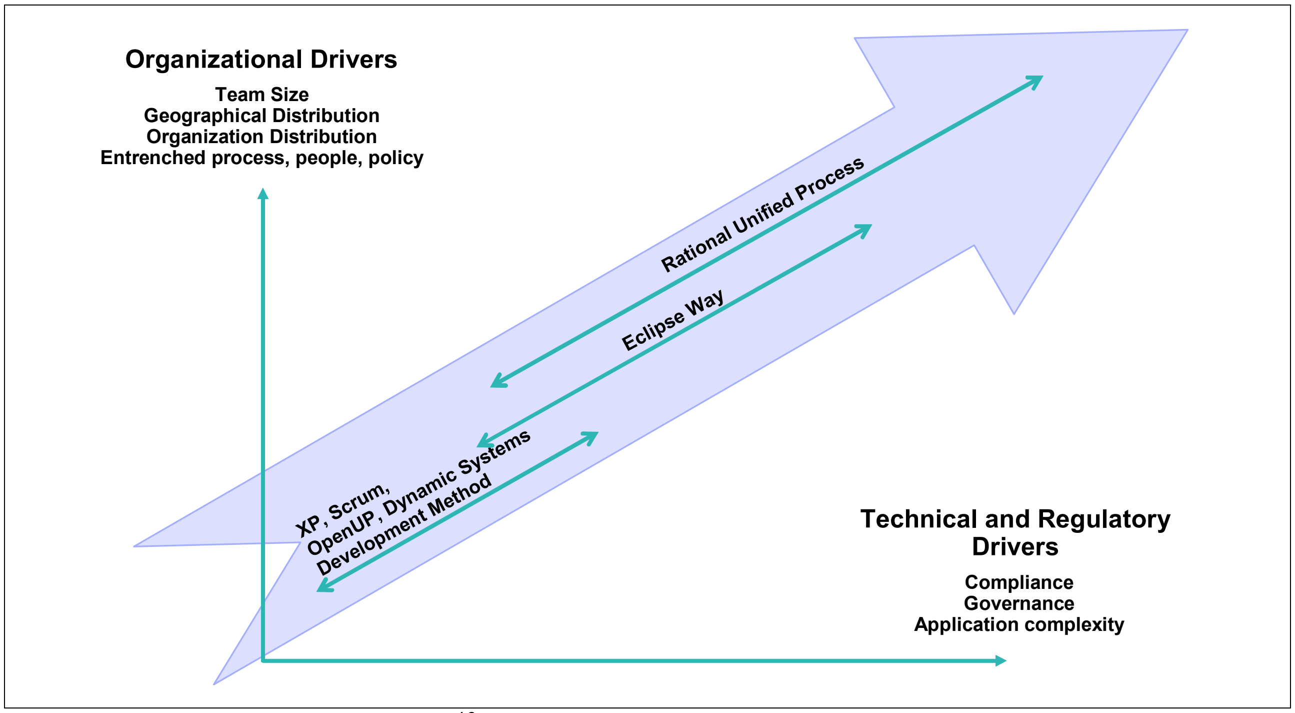

Organizational Drivers
Team Size
Geographical Distribution
Organization Distribution
Entrenched process, people, policy
Rational Unified Process
Eclipse Way
XP, Scrum, OpenUP, Dynamic Systems Development Method
Technical and Regulatory Drivers
Compliance
Governance
Application complexity

Figure 2-15 The agile process continuum[10]

By providing a process framework that consists of many processes and practices, the Rational Unified Process covers the broadest range in the team-size to complexity continuum. Teams can choose a process definition that is close to their needs. All versions of RUP advocate iterative development, continuous integration, stakeholder involvement, and collaboration. The more agile variants include guidance for test-driven development, pair programming, and just enough documentation while the more advanced variants address distributed development, SOA, systems engineering, and compliance. The key for adopting RUP in an agile fashion is to adopt just enough by modifying what is provided or removing what you do not need.

Approaches to Agility at Scale

As more enterprises seek to adopt agile development, they are confronted with the questions of which techniques work with which settings. Every team has a unique set of needs, circumstances, and complexities. While there is no one-size-fits-all answer, we present some ideas in the following sections.

Constant collaboration

The product owner, project manager, and team leads conduct frequent communication with stakeholders to understand their needs, communicate progress, and ask questions. Team members are encouraged to collaborate regardless of their geographic location. Placing tools to facilitate collaboration in the hands of the team members improves their effectiveness and encourages constant collaboration.

Iterative development

Many enterprises have not adopted iterative development, and yet it must be proven as an effective tool for managing risk and achieving user acceptance more easily. Iterations help teams to work out areas of risk while developing a "story" that can be told with the software by

[10] Scott Ambler, IBM Software Group, Rational, Practice Leader Agile Development and Per Kroll, Chief Architect - Rational Expertise Development & Innovation (REDI), Project Lead: Eclipse Process Framework

the end of the iteration. This story is then shared with the stakeholders through demonstration of the working software. The stakeholders can then respond to early versions of the software by making course corrections much easier to accomplish.

With iterative development, the project cadence is set by the leads, but the bottoms-up planning is provided by the individuals. The component teams are allowed to self-organize, provided that they work within the cadence that is set by the project leads. Team leads constantly seek to improve the team's health and efficiency by load balancing and rapid response to change.

Agility of small teams

Component teams can self-organize within the framework set by the leads. Whether they are co-located or distributed, each team can self-organize and choose the techniques that work best for them. For example, implementing daily stand-up meetings and just enough design, conducting continuous builds, and employing test-driven development can all occur at the component team level. Many of these methods do not make sense across the entire solution team. For example, a globally distributed solution team does not hold a 'stand-up' meeting. Instead, it is best left to the teams to decide their culture and techniques, provided that they work within the framework that is established by the solution team leads.

Frequent integration builds

At the component team level, the builds should happen frequently. While some development teams build continuously, others have scheduled daily builds. This ensures the health and quality of each component in the solution.

Each of these component teams feeds into an integration build. Instead of waiting until the end of the cycle to conduct an integration build, an Agility-at-Scale team conducts the integration builds as frequently as possible for the given project. All component teams deliver their code to the integration stream, where the integration build occurs. The integration build includes the running of JUnit tests against the solution build to assess the build quality. Results of the build and the build verification tests are passed to the test team. This helps the team to identify integration problems in the same iteration that they are introduced, thus making it easier to diagnose and fix.

Frequent integration testing

The testing team conducts frequent integration testing based on periodic integration builds. All too often, the test team receives the integration build at the end of the project. By bringing the solution test team into each integration, defects can be identified in the same iteration in which they are introduced. By bringing the test team into the iteration plan, they can wisely create a test plan, determine which test cases need to be written and run, and have a much clearer view of the quality.

In this book, the solution test team, which is led by Tammy, has the opportunity to work with a new solution build every week. This team assesses the integration build results to determine whether to deploy the latest build to the test lab. By conducting solution-level testing as part of the iteration, the feedback loop by the development teams allows the teams to identify and fix defects early and often.

2.2.4 Aligning work assignments

It is no surprise that work is assigned to team members that is expected to be completed in an iteration, and that these work assignments occur for all members of the team. Aligning work assignments is critical to ensuring that every team member knows what they need to do and when they should complete the work.

In the context of CALM, this best practice includes the following actions:

- Understanding the impact of a change to the current plan, knowing that the change can impact analysis, design, development, release engineering, and testing
- Ensuring that all teams can absorb the change and deliver the release on time
- Rolling individual team plans into a cohesive project plan
- Assessing the project's health throughout the cycle and making adjustments to work assignments as needed
- Ensuring that all requirements have been implemented with expected quality before releasing the software to stakeholders

The alignment of work occurs throughout this book. However, it is most apparent in Act 1 of the story, which is covered in Part B, "Act 1: Responding to a change request" on page 77.

2.2.5 Being requirements driven

Being requirements driven is not just a matter of managing requirements. In the context of CALM, requirements drive the scope, focus, and plans for the project. Requirements driven includes the following concepts:

- Stakeholder requests are reviewed and prioritized before detailed requirements are created.
- Approved requests are then elaborated as requirements.
- Requirements are associated with user stories (agile development) or use cases (OpenUP or RUP).
- Iteration plans are driven from requirements for planning purposes. The intent is to know when the requirements will be implemented.
- Test plans reference the requirements that will be tested. The plans align with the development iteration plans. The intent is to know what test cases to write and when the solution will be available in a build for test execution.
- Test cases align with requirements for validation. The intent is to ensure that the test case is written specifically to validate the requirement or requirements.
- Iterations are complete when all planned requirements are implemented and tested, with sufficient quality. The intent is to answer the question: Are we done?
- Defects are probably the most commonly understood artifacts that span multiple roles. Defects found during testing must be reported back to the project leads and developers, so that they can act upon them. Analysts are interested in understanding the quality of the implemented requirement, such as the number of defects that have been reported against this requirement. Therefore, treat defects like requirements that must be addressed, and if possible, manage them in the same repository.

The theme of being requirements driven is explored in the following acts of the CALM scenario:

- Act 1 (planning)
- Impacts Act 2 (what to develop)
- Impacts Act 3 (what is in the build)
- Act 4 (test execution validating requirements)
- Act 5 (are we done)

2.2.6 Striving for build clarity and quality

It is not enough to know if the build passed or failed. In the context of CALM, the more relevant question to the team is: Is this build worth my time? This question might sound arrogant. However, when we consider the complexities that are inherent in deploying a new build into the test lab and running a full suite of tests against it, the reasoning is quite sound. Looking deeper, we can determine the subsequent questions:

- What is implemented in this build?
- What is its quality?

Build clarity includes identifying which requirements are implemented and which defects are fixed. Build quality provides an indicator of which tests were run and with what result (at the build verification test level). This best practice is intertwined with being "requirements driven." Knowing what is implemented in the build implies that the developers are delivering change sets that implement requirements.

Later in the cycle, a project manager or auditor should be able to ask what tests were run for this build in order to determine whether all tests, such as build, regression, performance, system, security, and so on, were run against a build with acceptable results.

This theme spans several of the acts of the CALM scenario:

- In Act 2, changes are delivered by the development team.
- In Act 3, the solution build is produced, which is an aggregate of component builds.
- In Act 4, the theme comes to the surface when the test team needs to decide which daily build to deploy.

3

A scenario for CALM

In this chapter, we describe the scenario-based approach that we used for this book. We provide the background to the story. The rest of the book details the series of events that take place to realize the story and deliver the release of software.

We include the following sections in this chapter:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

3.1 A story telling a tale of integrations

Collaborative Application Lifecycle Management (CALM) involves deep detail in every software development discipline across the software development life cycle. To date, much has been written about the individual disciplines in the life cycle. We could have organized this book around each of the disciplines, the roles, and tools that are used. To repeat such an approach might certainly express the breadth of Application Lifecycle Management (ALM), but might likely fail at highlighting the value of collaboration across the disciplines. As such, it is difficult to choose the topics to discuss in meaningful detail in such a wide domain.

Therefore, rather than focus on individual roles and the tools they use, this book focuses on the interactions that occur across the team members as they deliver a release of software. This is where the power of storytelling comes into play.

The IBM Rational development team has created what we call the *green-thread technique*. A green thread tells a story and, in doing so, explains and highlights the key goals, roles, tools, and integration points along the way. To tell this story, we create a reference company, team members, project, and software application. We create a setting for the story to unfold and present that story in a series of windows or click-throughs of the software applications they used. In doing so, we provide a practical example of how to use the Rational products from the user's point of view. The characters in the story have goals, constraints, interactions with other characters, and ultimately a discrete task that they must complete.

If you are familiar with scenario-based design, you will recognize the similarities. For those of you who know and understand use cases, a green thread is, in effect, a single scenario in a use case that describes a path through a system. At IBM Rational, we realize that, just as use cases help software designers understand the flow of events and alternate paths, a green-thread scenario can help *all team members* to understand the basic principles and goals of a system. Unlike a use-case scenario, the green thread is more personal and animated with real-life characters acting in the context of their organization. The purpose of the green thread is to help us understand the "bigger picture."

The green thread used in this book is called "Geographically Distributed Application Lifecycle Management." The scenario involves a *team* in a fictitious enterprise that must produce a release of software. The focus of the story is on how that team achieves the end result, a release of their software project. The agile technique of "just enough" is applied to each role to describe their contribution to the overall team effort. This allows us to remain focused on how their work influences the other team members. The intent is to show the hand-off points and relationships between the roles, rather than diving into domain-specific detail. For example, the product owner creates a single requirement that is later referenced by the development lead, developer, test lead, tester, and project lead. We provide "just enough" detail in change management to illustrate the discipline and focus on how that change request impacts the other team members.

While this is a single scenario in the deep and wide domain of ALM, we have confidence in its power and applicability. This storyboard has been reviewed with a wide variety of people both external and internal to IBM and is accepted as being representative of many enterprises around the world. The key to reading this book is to focus on the interactions and hand-off points between the characters and their representative teams. Notice how the completion of work by one character is referenced by another character later in the story. This book makes every attempt at highlighting these hand-off points to help you understand the relationships between the roles and assets they produce.

3.1.1 Story objectives

A good story has a set of objectives that it must meet. This storyboard targets enterprises with teams of people distributed around the world. This enterprise-scale story has the following objectives:

- Illustrate the value of ALM.
- Illustrate team collaboration to produce a release of software.
- Illustrate the primary disciplines in the application lifecycle.
- Set the story in a context in an enterprise environment with the following characteristics:
  - Scalable products are delivered as multiple projects.
  - Teams are in distributed locations.
  - Agile development is introduced to existing processes.
  - Governance and compliance are contributing factors to organizational pressures.
  - There is a delivery chain from smaller component teams into an integrated solution.
  - The Rational team-based products are already deployed and in use.
  - The Rational Jazz-based products have been adopted by some of the teams.

3.1.2 The context for the story

Every story needs a context for understanding the plot, such as a time, setting, and any background drama that leads into the opening scene. When reading a screen play, the opening sentences set the stage for the story as in the following example:

Boston, MA 1775, the colonists are becoming weary of the tariffs levied upon them by the King of England. Tensions are mounting as a boat full of tea enters the Boston Harbor.

The following sections serve to provide the setting for this storyboard by introducing the team and how they are organized, the project that they are working on, and an overview for the story that is used for the remainder of the book.

Synopsis: A globally distributed team has received funding to implement the second release of a critical IT application. The team has been working for several weeks and has just begun working on a new iteration. We meet the team on the second day of the second iteration.

3.2 The project

As with any enterprise, multiple business optimization programs are under development at any time. In this story, we assume that the project management office is governing several programs at once. The proposals for the new programs have been validated and approved. Each program is tracked, managed, and released as one with coordination cross individual projects and components.

The story takes place in the context of a team developing a new version of the *Account Opening* project. The Account Opening team is responsible for both a new development project (Release 2) and a maintenance project on the release (Release 1) that is currently in production. This storyboard observes the team as they deliver Release 2 of the project. The solution, as shown in Figure 3-1 on page 50, comprises many components.This story shows how the team leads of the solution collaborate with one of the component teams.

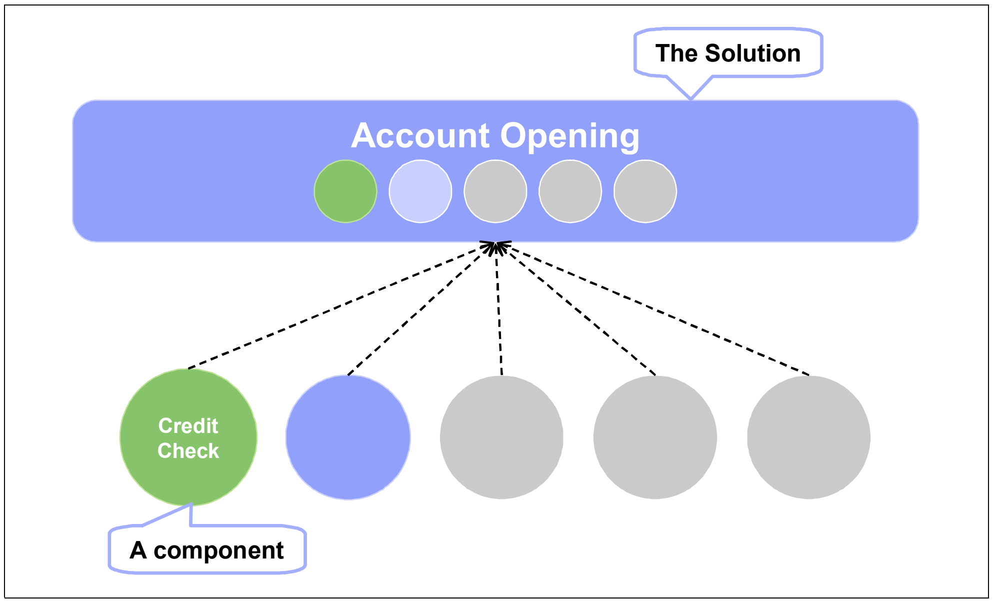


Figure 3-1 The many components of tehe Account Opening solution

The team has grown by acquisitions and outsourcing. The enterprise has accelerated time-to-market for a launch in the European market by acquiring the skills and technologies of a smaller European company. The enterprise has also increased team capacity through a third-party provider based in another country.

The team is globally distributed:

- New development of the second release (Rel2) is delivered from a team that consists of the corporate headquarter team, the acquired team, and the third-party provider.
- Maintenance of the first release (Rel1Maint) is delivered by the third-party provider.

The team has traditionally used the IBM Rational Unified Process (RUP), but is intrigued by the notion of process enactment that comes with IBM Rational Team Concert and the Rational ClearQuest ALM schema. Because this is new, they decided to start small and slowly add a process as the teams became accustomed to using it. As such, they have adopted the Open Unified Process (OpenUP).

This enterprise uses a global delivery platform from IBM. They have already invested in and are using the Rational team-based products such as IBM Rational ClearCase, ClearQuest, and RequisitePro. In addition, they have begun to introduce and adopt the newer products from Rational such as IBM Rational Asset Manager, Rational Build Forge, RequisitePro Composer, Rational Team Concert, Rational Quality Manager, Rational AppScan, and Rational Software Analyzer. Some teams stay on existing platforms and deliver into the global delivery platform that governs the solution delivery, while other teams adopt newer products. This heterogeneous mix of products will remain in the enterprise for years to come, but with a shift toward the newer products where appropriate.

3.3 The software delivery team

The reference team is globally distributed and comprised of several smaller teams. For the purpose of the story, only a few of the teams are brought into the foreground. In this section, we describe the team organization and each of the characters.

3.3.1 A team of teams

The Geographically Distributed ALM reference scenario uses a set of characters and includes a “team of teams” with members who are distributed around the world. Each team member has one or more roles to play and is critical to the success of the project.

Our story revolves around an enterprise solution team, which is made up of several teams. The teams are distributed around the world and consist of both internal teams and a trusted third-party provider. For the purposes of this scenario, we focus on three of the teams, but assume that the other teams will function in a similar manner. The smaller development teams are often referred to as the *component team*. This term is not a technical use of the word, but rather a figurative use to describe a smaller team that owns a discrete portion of the overall solution.

The following teams are involved in this scenario:

- Solution team leads
- An agile development team that was recently acquired by the larger enterprise (the component team)
- A third-party solution testing team

This storyboard has the following characters:

- *Bob* is the product owner. He cares about managing his business and bringing value to his users and stakeholders.
- *Patricia* is the project leader (or project coach). She is responsible for coordinating the efforts of this team of teams.
- *Al* is the solution architect. He works with the development and test teams to ensure that a consistent architectural approach is used and understood.
- *Tammy* is the test lead. She has a globally distributed quality team. Her team is responsible for conducting solution testing, or investigative testing in agile terms.
- *Rebecca* is the release engineer who oversees the solution integration builds, providing global build support to the team.
- *Marco* manages an agile development team. He understands the need to fit into the solution project plan, but seeks to maintain the agile approaches that have made his team successful.
- *Diedrie* is the developer who implements a change request in the scenario.
- *Tanuj* is the tester who is responsible for ensuring the quality of the solution.

Figure 3-2 summarizes the characters and teams in this story.

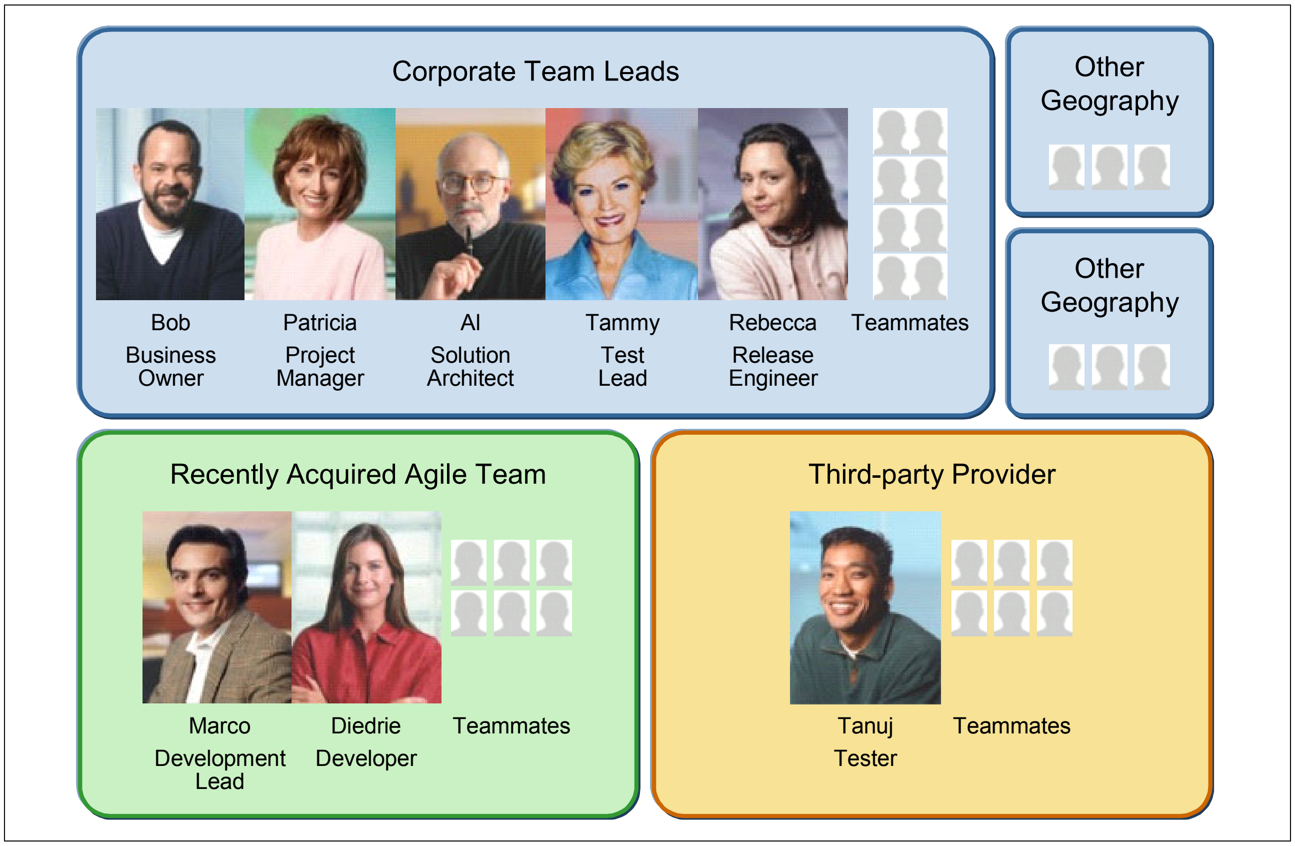
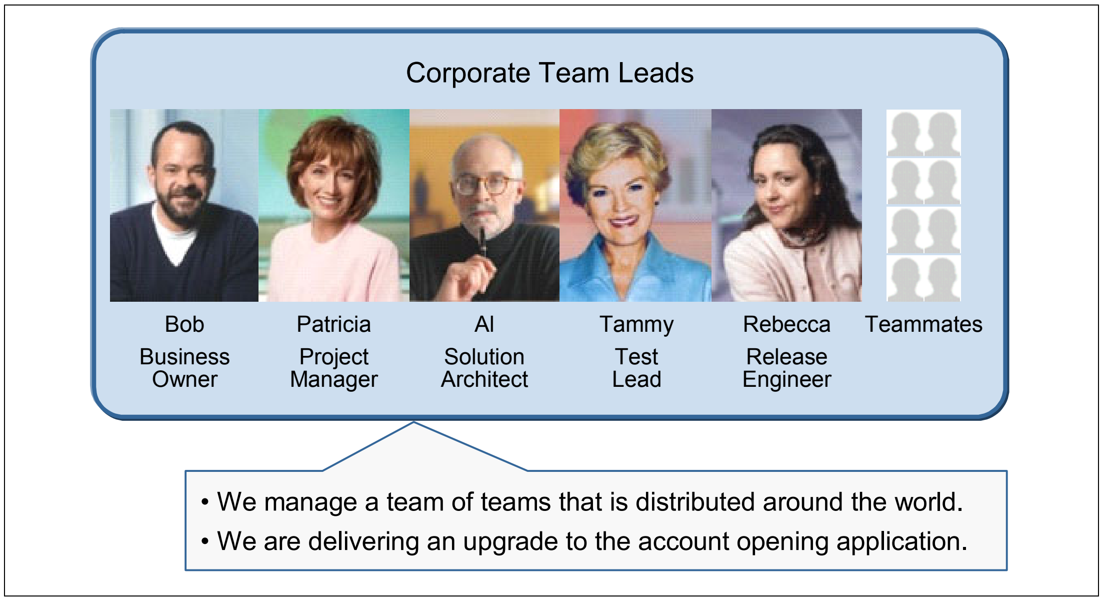


Figure 3-2 A global team made of many teams

The enterprise is adopting agile techniques where appropriate, while also acknowledging that some agile techniques do not apply to their situation. Because this is a large team working with agile techniques within the enterprise, we call this *Agility at Scale*. Additional detail is provided in 3.6.2, "Agility at Scale" on page 67.

3.3.2 The solution team leads

The team of teams is run by a team based at the corporate headquarters. This team works with each of the component teams to coordinate the final solution. Figure 3-3 shows the solutions team leads.



Corporate Team Leads
Bob
Business Owner
Patricia
Project Manager
Al
Solution Architect
Tammy
Test Lead
Rebecca
Release Engineer
Teammates
• We manage a team of teams that is distributed around the world.
• We are delivering an upgrade to the account opening application.

Figure 3-3 Corporate-based team leads

The team leads appear in the following order:

- *Bob* is the product owner. He cares about managing his business and bringing value to his users and stakeholders. As the product owner, working in an agile environment, he works closely with the development team, has ownership over the requirement priorities, and facilitates discussions between the development team and their stakeholders.
- *Patricia* is the project leader (or project coach). She is responsible for coordinating the efforts of this team of teams. As each team produces its iteration plan, Patricia incorporates the plans into the overall project iteration plan. She collaborates with the teams to establish the pace of the iterations (how many iterations for the project and how long each will last). It is her job to ensure that all teams (development and solution test) are working toward the same iteration goals and that the work is aligned across the teams.
- *Tammy* is the test lead. She has a globally distributed quality team. Her team is responsible for conducting solution testing or investigative testing. This includes functional, performance, and security testing at the solution level. This does not include confirmatory testing, such as JUnit, or component-level testing, which is the responsibility of each of the development teams. Tammy's team conducts solution testing as part of each iteration. Her team does a new solution build each week, thus providing an early feedback loop to the development team. She also provides a globally distributed test environment for the project and has ownership of a group of servers in the test lab.
- *Al* is the solution architect. Al works with the development and test leads to ensure the team works from a consistent architectural approach. He provides insight into the iteration plans by identifying the architecturally significant tasks for each iteration. He collaborates with the developers as they design their implementations. He also seeks to reuse existing assets wherever possible to ensure consistency in approach and implementation.
- *Rebecca* is the release engineer who oversees the solution integration builds, providing global build support to the team. She sets the delivery, build, and release policies within

the project. While each component team is responsible for successfully building their component, Rebecca must bring all of the components together into a solution build. To bring agility into this enterprise solution, Rebecca produces a weekly solution build. This build enables the team to diagnose solution build problems as soon as they occur and enables Tammy's team to test the solution build more often.

- *Teammates* are those additional team members, such as the project management office, support personnel, additional testers, and development teams based in the corporate office who also contribute to the project. Their work is not highlighted in this scenario.

3.3.3 The agile development team

A smaller company was acquired by the enterprise and is located in another country. The team from the acquired company seeks to maintain their agile development style while working in the context of the larger solution team. For the most part, the team, which is shown in Figure 3-4, is co-located but they interact with many other teams in the enterprise.

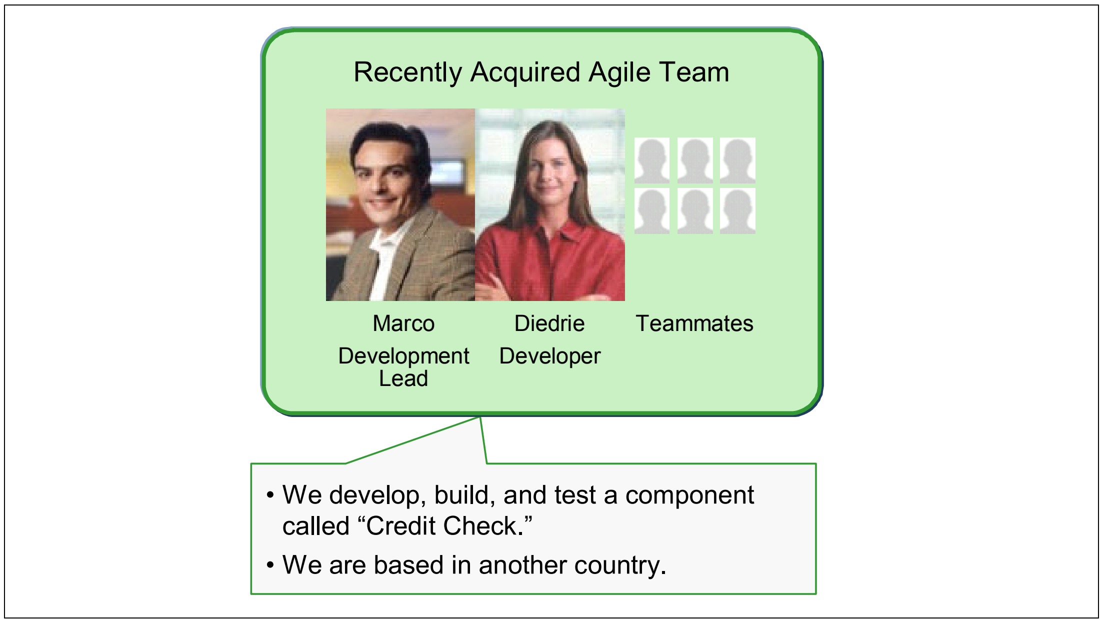

Recently Acquired Agile Team
Marco
Development Lead
Diedrie
Developer
Teammates
• We develop, build, and test a component called "Credit Check."
• We are based in another country.

Figure 3-4 The smaller agile team who owns the 'Credit Check' component

The characters appear in the following order:

- *Marco* manages an agile development team. He understands the need to fit into the solution project plan, but seeks to maintain the agile approaches that have made his team successful. His team is self-organized and uses more frequent iterations than the rest of the project team. Marco still conducts daily stand-up meetings with his team and employs test-driven development techniques on his component. He is also a developer.
- *Diedrie* is the developer who implements a change request in the scenario. She is a generalizing specialist on Marco's team who designs and implements changes and oversees the component builds.[1]
- *Teammates* are those developers on the team whose activities are similar to Diedrie's.

[1] See "Generalizing Specialists: Improving Your IT Career Skills" by Scott Ambler at the following Web address: `http://www.agilemodeling.com/essays/generalizingSpecialists.htm`

3.3.4 The solution testing team

The solution testing team works under the direction of Tammy, who is the test lead. Some of the solution testing has been out sourced to a trusted third-party provider that is located in another country in a different time zone.

Figure 3-5 shows the solution test team.

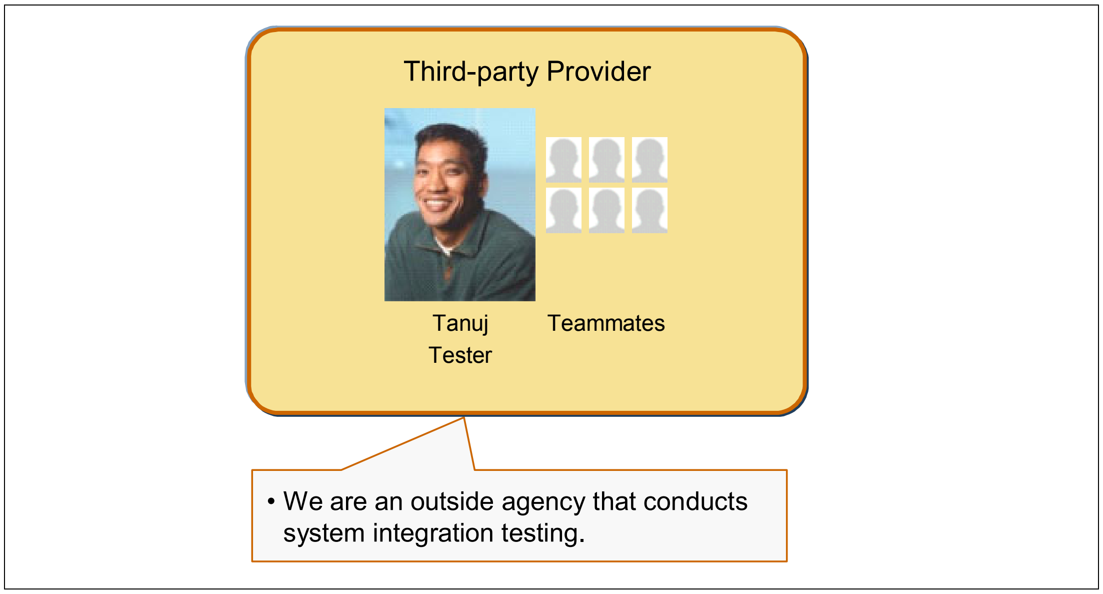

Third-party Provider
Tanuj
Tester
Teammates
• We are an outside agency that conducts system integration testing.

Figure 3-5 The third-party provider team that conducts system integration testing

The following characters conduct the testing:

- *Tanuj* is the tester. He is responsible for creating the test cases and test scripts for ensuring the quality of the solution. He also executes the tests and analyzes the results. He logs defects when needed. Tanuj and his teammates employ a full battery of tests including manual, functional, and performance tests.
- *Teammates* are those many other testers who conduct activities similar to Tanuj. Some of these teammates are based in the third-party company with Tanuj. Others are based at the headquarters with Tammy.

For the purposes of this scenario, we highlight the interactions between Tammy and Tanuj.

3.4 The approach

In this section, we describe the approach that is used by the team in the scenario. They have heard a lot about agile development techniques and have begun to adopt those that apply to their project and team size. Some of the agile techniques are discussed in this section. See 3.6.2, “Agility at Scale” on page 67, which describes these approaches in more detail.

3.4.1 Phases and iterations for establishing cadence

The team is managing against phases and iterations. It is the first time that they have used an iterative process and are starting small by adopting OpenUP. They have adopted the four phases of Inception, Elaboration, Construction, and Transition and define iterations within each of these phases as illustrated in Figure 3-6.

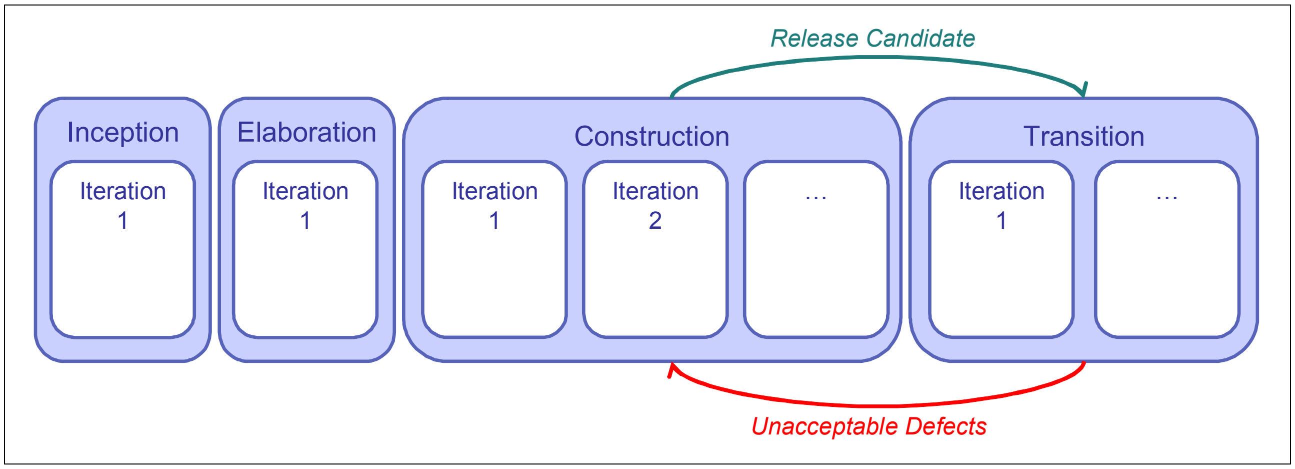


Figure 3-6 OpenUP phases with time-boxed iterations

Prior to the Inception phase, Bob, the product owner, developed a proposal for an upgrade to the Account Opening project, which was reviewed and funded. He collaborated with Al, the solution architect, and Patricia, the project manager, to solidify his proposal and build his business case. When his proposal was approved, an initial project team was created, and they began the Inception phase for the project.

During the Inception phase, Bob, Patricia, and Al collaborated to initiate the project. Bob prioritized and refined the requirements. Al developed a technical approach and outlined the architecture. He and Patricia collaborated to develop an estimated project plan that included team sizes, skill sets, and software and hardware needs. They collaborated with other development leads and test leads to contribute to and confirm the plans. They took the approach of doing just enough to put their plans in place and completed the phase in less than one week. The goal of this phase was to drive out business risk.

During the Elaboration phase, more team members were added to the project, and the team quickly refined the requirements and architecture. A prototype of the system was developed and tested by a small team to prove the architecture. The prototype was demonstrated to Bob and additional stakeholders to verify the approach and obtain additional feedback and direction. This phase lasted three weeks. The goal of this phase was to drive out technical risk.

During the Construction phase, the full development team was brought into the project to develop the solution. Feedback from the Elaboration phase was shared with the team, and requirements were reviewed, refined, and prioritized. The team agreed to run four-week iterations. This point is where the scenario begins. When the Construction phase is complete, the team will hand the solution to the operations team for the roll out into production.

During the Transition phase, the operations team will conduct additional tests such as performance, security, and user-acceptance testing. If unacceptable defects are found, the solution is sent back to the Construction phase to be fixed. This loop continues until the solution is rolled into production.

The Production phase (not pictured in Figure 3-6 on page 56) continues until a new version is released. Until then, the solution is monitored, managed, and maintained with hot fixes and patches.

3.4.2 Frequent builds to drive clarity and quality

To drive quality deep into their development style, each team builds their component on a continuous or daily basis, while the integration build is scheduled to occur weekly as illustrated in Figure 3-7.

Diedrie is a developer who also maintains her team's build system. She ensures that the continuous builds flow smoothly. Other teams can choose a different approach to their builds, provided that they occur often. The example shows two other teams that have daily scheduled builds.

Rebecca resides over the integration build that is scheduled to occur every week. As each team achieves a successful and stable build, they deliver their source to the solution integration stream. By conducting weekly integration builds, the team can quickly respond to problems in the build or that are identified by the test team.

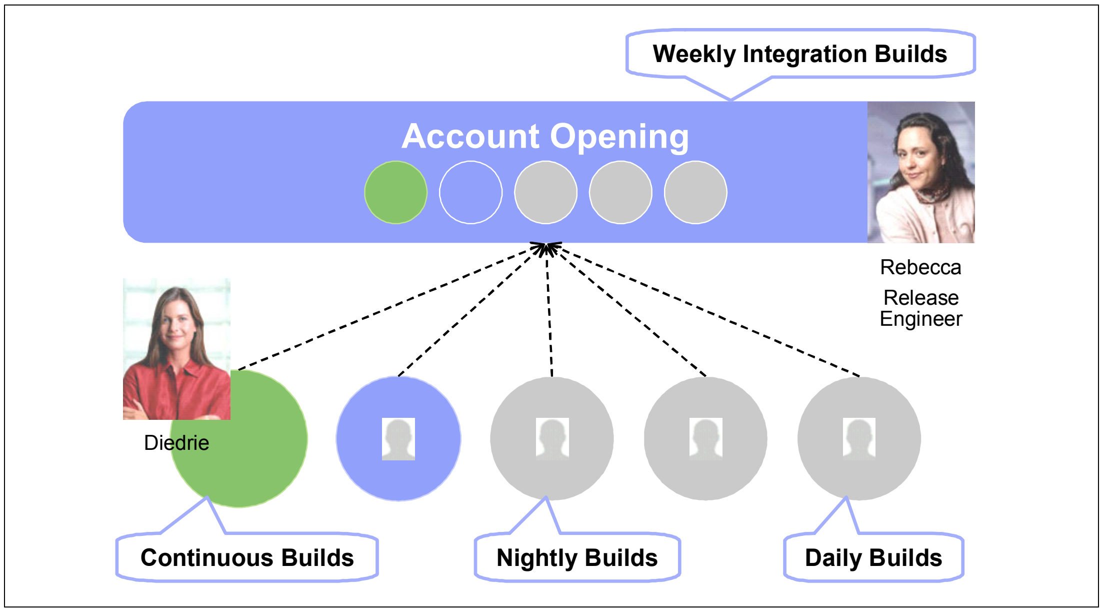


Figure 3-7 Team builds occurring daily, with integration builds occurring weekly

3.4.3 Testing to drive quality

Each of the development teams conducts confirmatory testing on their own component. They use a combination of unit tests and functional tests to drive quality into their component. In addition, frequent customer testing and feedback brings insight and clarity into the quality of the component.

The weekly integration build is used by the solution test team throughout the iteration. In agile terms, this is called *investigative testing*, where many forms of testing take place, such as exploratory, scenario, system, and user testing.

Tammy's team has access to weekly integration builds. Her team deploys these builds into the test lab and conducts tests. The test effort is aligned with the requirements and development plan, which enables the test team to be focused on which tests to run in each integration build. The last week of every iteration is dedicated to fixing defects.

Figure 3-8 illustrates the system testing that is done.

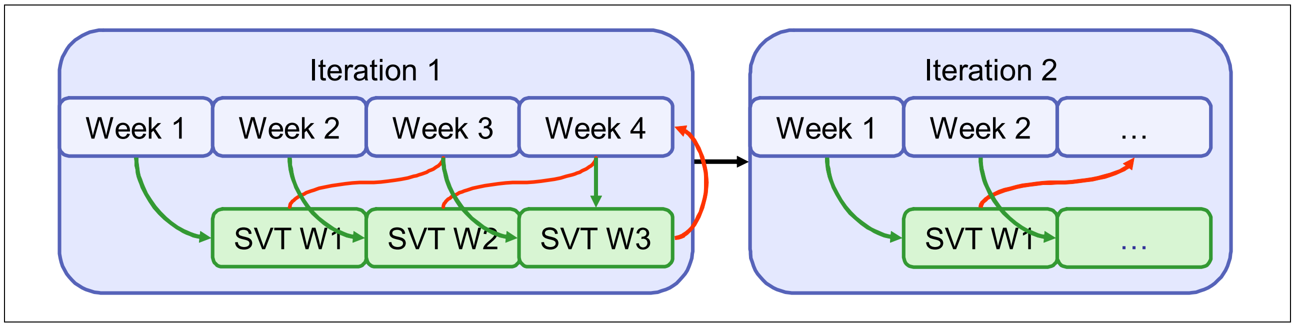


Figure 3-8 System testing occurring during the iteration

3.4.4 Lean governance

Gantt charts, which are large enough to form wall paper when printed, are not used by this team. Instead the team leads established a plan for four-week iterations to which all teams are held accountable. The team has agreed to use OpenUP as the development process and all teams are adhering to it. Because both Rational Team Concert and ClearQuest provide support for using OpenUp, adhering to the process is a matter of receiving work items and completing them. This is considered a *top-down approach* for managing the team.

At the same time, each team member owns and manages their own work items. Team members provide estimates for their work that rolls up into an overall plan that is managed by their CALM solution. The team leads can view the iteration plan, the current workload and estimates, and respond appropriately to maintain a healthy workload for the iteration. This is considered a *bottom-up approach* to managing the team.

Both the bottom-up and top-down approaches are blended to create the healthiest and most dynamic environment possible given the size of the team.

3.5 A story told act by act: Completing an iteration

Synopsis: A globally distributed team has received funding to implement the second release of a critical IT application. The team has already completed the Inception and Elaboration phases and has just begun the final iteration of the Construction phase. We meet the team on the first day of the iteration.

The storyboard is divided into five major acts, each with one or more scenes. This division is similar to that of a movie script or play. Imagine the curtain and lights going up in the theater at the end of each act. Each of the major acts stands as a milestone in the overall life cycle, while each scene within the act demonstrates how one or more of the characters completes a specific task. By dividing the story into separate acts, the following objectives are achieved:

- The story is more consumable and reads like a play rather than an technical journal.
- The goals, resources, and character interactions can be clearly defined in the context of the act, thus making their interdependency clear and highlighting the value of having a CALM solution.

- Each act builds on the previous act, but can be read independently of the others. Therefore, you can jump directly to the part that is of interest to you.
- The life-cycle assets that are produced and consumed by each act can be highlighted in context of how they are used by the characters.

Figure 3-9 illustrates the five major acts in the scenario. Each act has several scenes as indicated by the numbered boxes. The story starts with Act 1, where Bob submits a request for the iteration, followed by the development (Act 2), integration build (Act 3), testing (Act 4), and the delivery of the solution at the end of a four-week iteration (Act 5). Each part of this book corresponds to one of the acts in the storyboard.

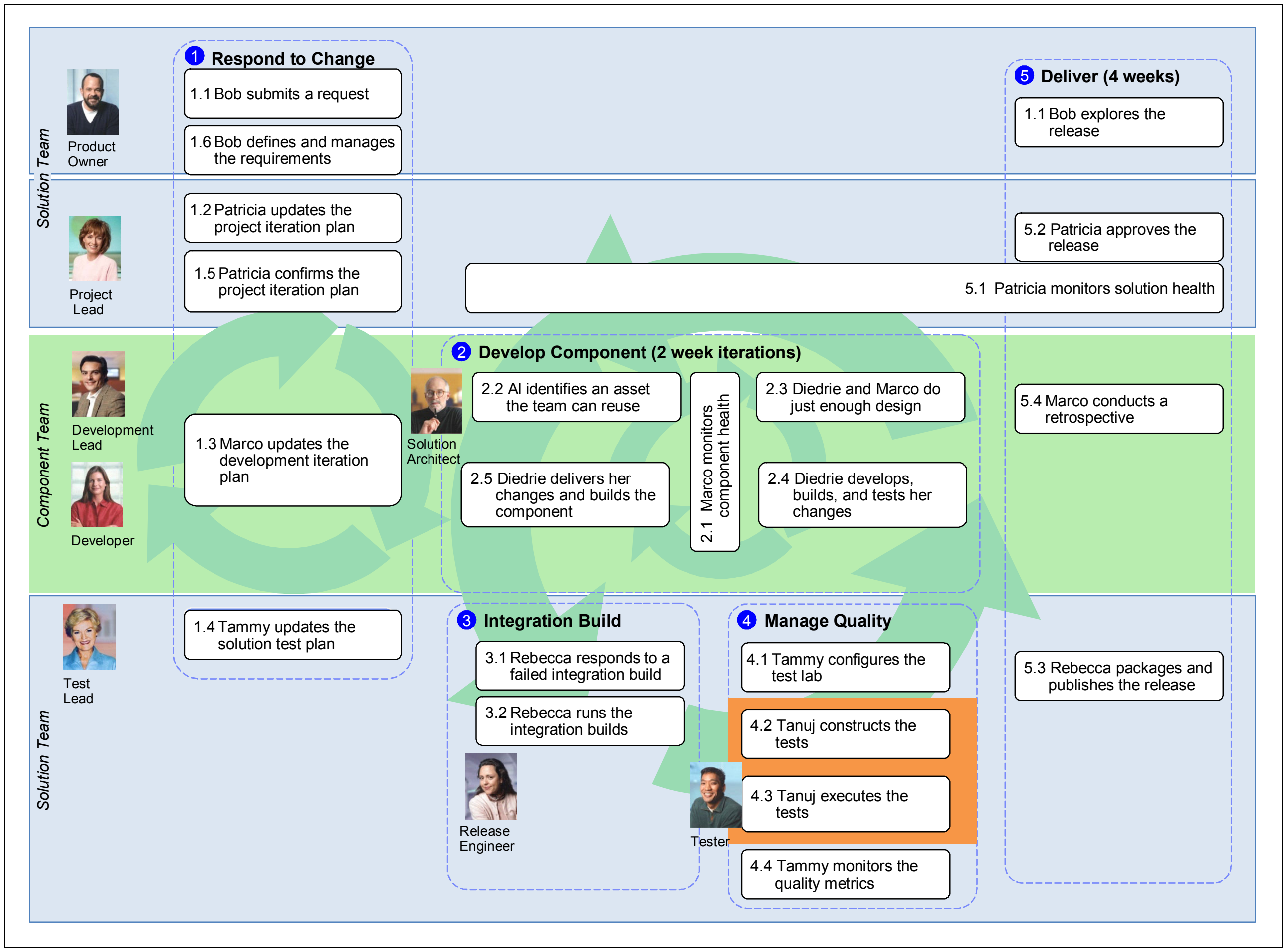

Solution Team
Product Owner
Project Lead
Component Team
Development Lead
Developer
Solution Team
Test Lead
1 Respond to Change
1.1 Bob submits a request
1.6 Bob defines and manages the requirements
1.2 Patricia updates the project iteration plan
1.5 Patricia confirms the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
5 Deliver (4 weeks)
1.1 Bob explores the release
5.2 Patricia approves the release
5.1 Patricia monitors solution health
5.4 Marco conducts a retrospective
5.3 Rebecca packages and publishes the release
2 Develop Component (2 week iterations)
Solution Architect
2.2 Al identifies an asset the team can reuse
2.1 Marco monitors component health
2.3 Diedrie and Marco do just enough design
2.5 Diedrie delivers her changes and builds the component
2.4 Diedrie develops, builds, and tests her changes
3 Integration Build
3.1 Rebecca responds to a failed integration build
3.2 Rebecca runs the integration builds
Release Engineer
4 Manage Quality
4.1 Tammy configures the test lab
4.2 Tanuj constructs the tests
4.3 Tanuj executes the tests
Tester
4.4 Tammy monitors the quality metrics

Figure 3-9 Geographically Distributed Application Lifecycle Management scenario used by this book

The following subsections provide a synopsis of each of the acts shown in Figure 3-9 and provides a pointer to the part of the book that details that act:

- Part B, “Act 1: Responding to a change request” on page 77
- Part C, “Act 2: Collaborative development” on page 211
- Part D, “Act 3: Enterprise integration builds” on page 313
- Part E, “Act 4: Managing quality” on page 387
- Part F, “Act 5: Delivering the solution” on page 479

The team has already completed the Inception and Elaboration phases and is now in the final iteration of the Construction phase.

3.5.1 Act 1: Responding to a change request

Synopsis: The product owner submits a high priority request for the current iteration. The team reviews the request and determines that it can be contained in the current iteration plan. The impacted team leads update their plans, and Patricia updates the solution iteration plan. Bob provides additional detail and definition to the requirement.

The purpose of this act is to illustrate how the team collaborates to quickly triage, and plans to implement, a new request. The following primary goals are illustrated in this act as shown in Figure 3-10:

- The team leads update their plans to reflect the decision to implement the request.
- The product owner details and defines his request by elaborating on the requirements.

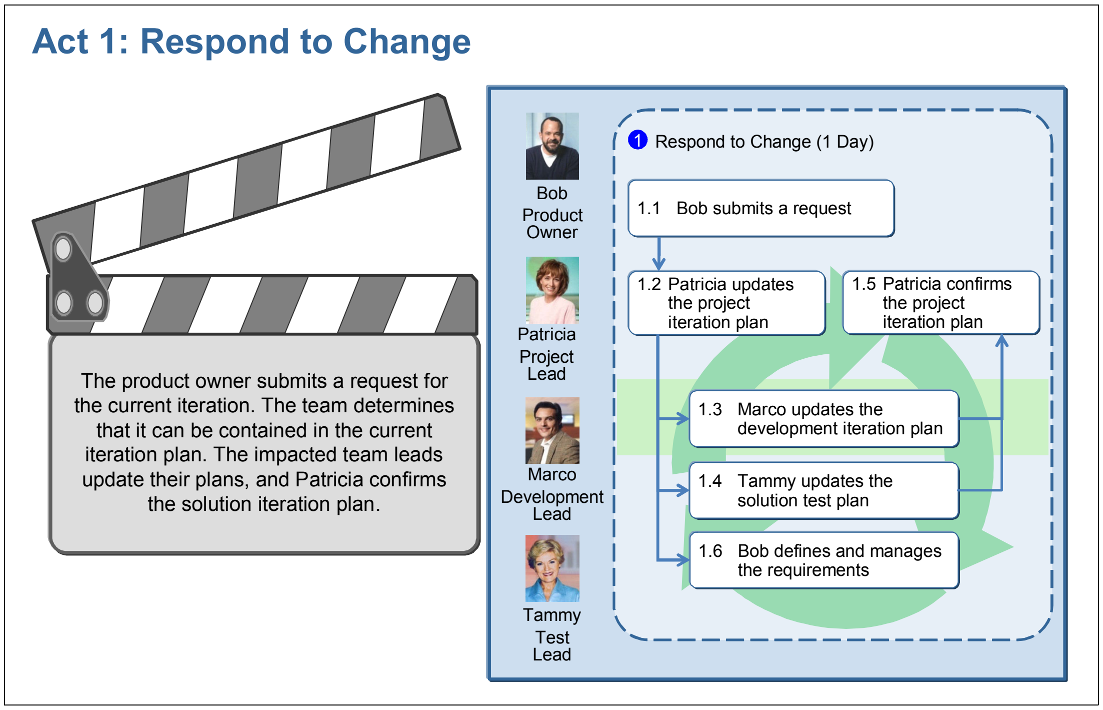

Act 1: Respond to Change
The product owner submits a request for the current iteration. The team determines that it can be contained in the current iteration plan. The impacted team leads update their plans, and Patricia confirms the solution iteration plan.
Bob Product Owner
Patricia Project Lead
Marco Development Lead
Tammy Test Lead
1 Respond to Change (1 Day)
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.5 Patricia confirms the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements

Figure 3-10 The scenes in Act 1: Responding to change request

As shown in Figure 3-10, these goals are expressed in the following scenes:

1. Bob submits a request.

   Bob, the product owner, submits a request, setting it to the highest priority and assigning it to the current iteration. In the request, he references the user interface (UI) standards that the enterprise uses for all of its customer-facing applications. He creates a task for Patricia and her team to size his request.

2. Patricia plans the project iteration.

   Patricia assigns work to Bob to detail the requirement, Marco to implement it, and Tammy to ensure that the request is tested per the requirements that Bob provides. This work is linked back to the original request for traceability.

3. Marco updates the iteration plan.

   Marco leads the agile development team. He conducts an iteration planning session with his team where the work item to implement Bob's new request comes up for discussion. They contact Al, the architect who identifies a reusable asset for UI branding. They agree to explore its use for the implementation. Marco assigns the branding work to Diedrie for implementation in this iteration.

4. Tammy updates the solution test plan.

   Tammy receives a work item from Patricia and updates her test plan. She creates a test case and adds it to her test plan. Tammy notices that Al is suggesting a reusable asset. She looks at the asset and discovers a set of tests that are to be used when implementing this asset. Tammy updates the test plan and assigns the testing to Tanuj. Finally, she confirms the availability of the required test lab servers.

5. Bob defines and manages the requirements.

   Bob receives a work assignment from Patricia to provide additional detail to his request. He creates a sketch and a requirement to illustrate what he wants. He claims his work complete and links his requirements to his work item.

6. Patricia confirms the project iteration plan.

   Patricia reviews the task assignment to see how Marco, Tammy, and Bob are progressing on their work items.

We explore the following concepts in this act:

- The bridge between the business and development team via a request
- How a request drives changes to development and test plans that are managed in separate repositories
- How a request is elaborated with additional detail
- How Bob can clearly communicate his requirements by providing a sketch

See Part B, “Act 1: Responding to a change request” on page 77, for a full elaboration of this act.

3.5.2 Act 2: Collaborative development

Synopsis: The team develops, validates, and builds the required changes to their component. The source code is delivered to the solution integration stream.

The purpose of this act is to illustrate how the team collaborates to quickly design, implement, build, validate, and implement the request. The primary goals illustrated in this act are as follows and as shown in Figure 3-11 on page 62:

- Marco works with the team to identify the appropriate owner to implement the request.
- The team identifies and incorporates an asset for reuse.
- A developer collaborates with other team members to develop, test, and build the component with the changes to satisfy the request.
- Changes are delivered for the weekly integration build.

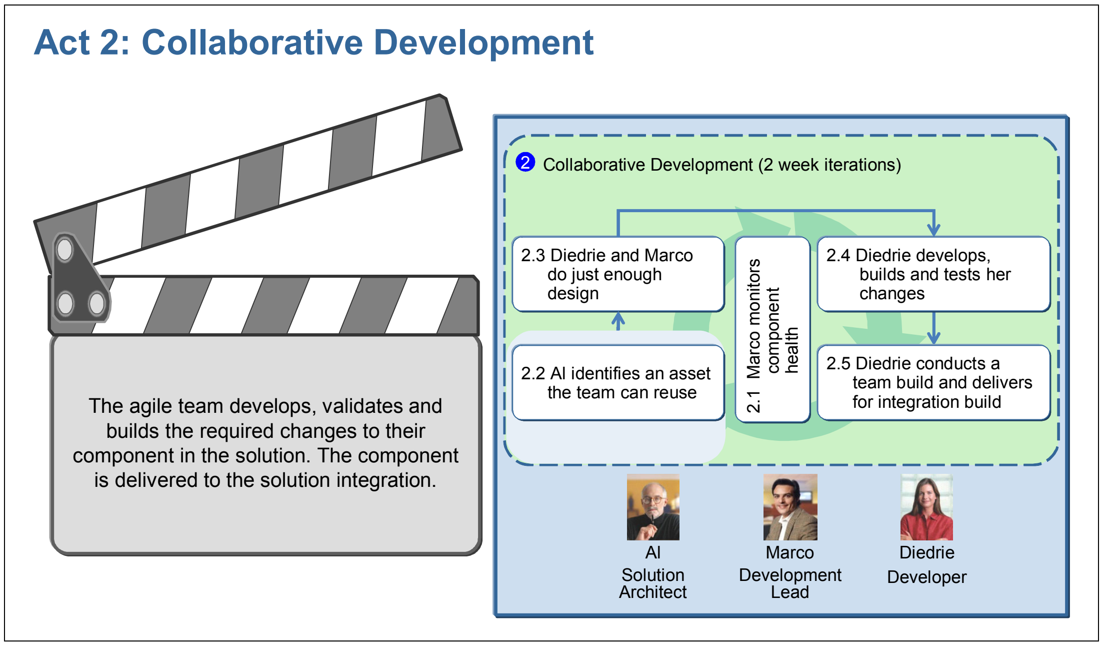

Act 2: Collaborative Development
The agile team develops, validates and builds the required changes to their component in the solution. The component is delivered to the solution integration.
2 Collaborative Development (2 week iterations)
2.3 Diedrie and Marco do just enough design
2.1 Marco monitors component health
2.4 Diedrie develops, builds and tests her changes
2.2 Al identifies an asset the team can reuse
2.5 Diedrie conducts a team build and delivers for integration build
Al Solution Architect
Marco Development Lead
Diedrie Developer

Figure 3-11 The scenes in Act 2: Collaborative development

As shown in Figure 3-11, these goals are expressed in the following scenes:

1. Monitor component health. Marco conducts daily stand-up meetings to adjust the work for the day. Diedrie and Marco collaborate on the design of Bob's request and ask Al to guide them on reusable assets for application branding. Diedrie and Marco examine the asset and perform just enough analysis and design in an experimental development environment.
2. Al identifies an asset the team can reuse and shares it with the team.
3. Diedrie and Marco do just enough design. The team practices test-driven development, and Diedrie develops her test cases before writing the code.
4. Diedrie develops, builds, and tests her changes on her local machine. Then she makes the required updates to the build and build verification test scripts and runs a private build. She delivers her changes and monitors the team's continuous build.
5. Diedrie conducts a team build and delivers to the integration build. With a successful build, she delivers her changes to the integration stream.

We explore the following concepts in this act:

- Asset reuse
- Test-driven development
- Development team collaboration

See Part C, "Act 2: Collaborative development" on page 211, for a full elaboration of this act.

3.5.3 Act 3: Enterprise integration builds

Synopsis: The delivered changes are integrated, built, and verified on a weekly basis. The release engineer is monitoring the build process.

The purpose of this act is to demonstrate the power of a centralized and automated integration build. The primary goals illustrated in this act are as follows and as shown in Figure 3-12:

- Rebecca automates the build process to gain efficiency and predictability in the builds.
- Rebecca rapidly responds to the build failures and restarts the build.
- Rebecca integrates third-party applications to enable the automation of the full build process from gathering the latest source, to running static analysis, capturing the delivered activities in Rational ClearQuest records, and staging the resulting build.
- Rebecca uses environments, selectors, and filters to separate the build process from the hardware for which its compiled.

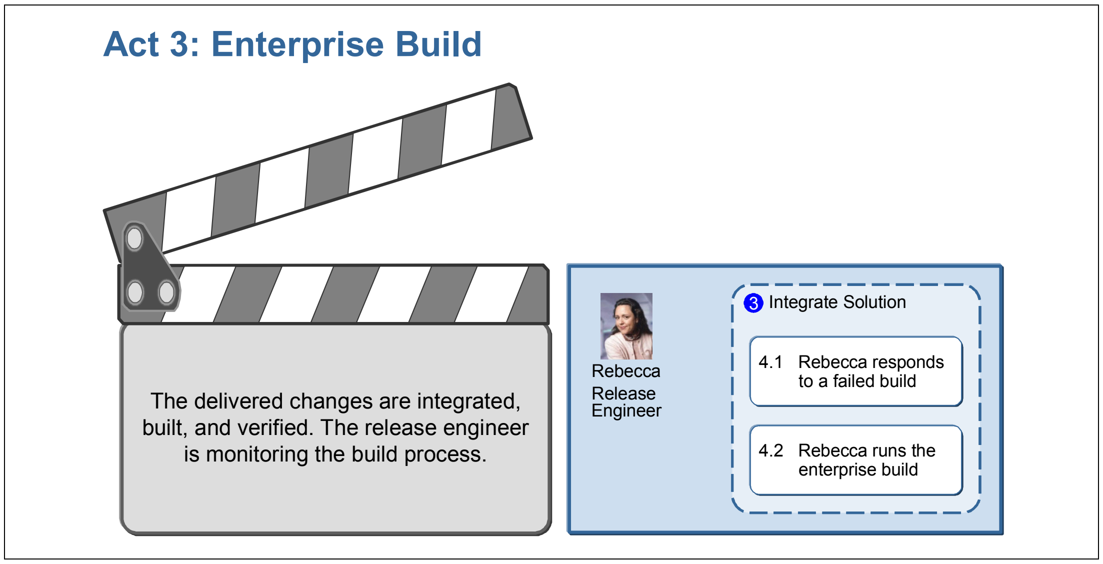

Act 3: Enterprise Build
The delivered changes are integrated, built, and verified. The release engineer is monitoring the build process.
Rebecca
Release Engineer
3 Integrate Solution
4.1 Rebecca responds to a failed build
4.2 Rebecca runs the enterprise build

Figure 3-12 The scenes in Act 3: Enterprise integration builds

As shown in Figure 3-12, these goals are expressed in the following scenes:

1. Rebecca responds to a failed build.

   Rebecca monitors the range of build projects that she is responsible for. She is notified that the weekly Account Opening integration build failed. She inspects the build and resolves the problem.

2. Rebecca runs the enterprise build.

   Rebecca runs a new Account Opening integration build, including the latest changes. She has automated the complete build process, which includes the creation of baselines, an application build, build verification tests, build staging, and build announcements. The build processes are scheduled to run weekly. She also has the option to manually request builds on demand. The output builds are automatically posted and announced for team consumption, the build verification test results are added, and a report of “what’s in” is generated. The availability and staging location is included in release notifications. The build is successful and is announced to the team.

We explore the following concepts in this act:

- Resolving a build failure
- Integration with Rational ClearCase for source code control, baselining, and staging the completed build
- Integration with Rational ClearQuest to establish baseline and build records that provide information about the build to the Rational ClearQuest users
- Integration with Rational Software Analyzer to perform static code analysis
- Automated notifications of build success or failure

See Part D, “Act 3: Enterprise integration builds” on page 313, for a full elaboration of this act.

3.5.4 Act 4: Managing quality

Synopsis: The stability and quality of the integration builds are tested by the globally distributed quality team.

The purpose of this act is to demonstrate how the test plan organizes the team’s test effort. The primary goals as illustrated in Figure 3-13 for this act are as follows:

- Obtain notification of a new build.
- Prepare the test servers with the build.
- Execute tests, evaluate the results, and file defects.
- Monitor quality and test team progress.

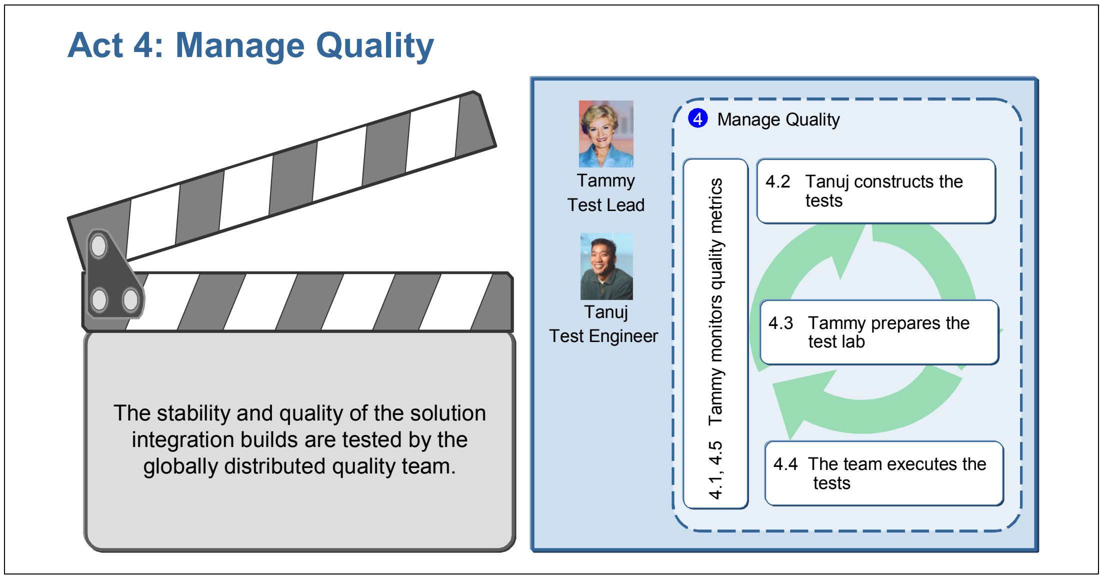

Act 4: Manage Quality
The stability and quality of the solution integration builds are tested by the globally distributed quality team.
Tammy
Test Lead
Tanuj
Test Engineer
4 Manage Quality
4.1, 4.5 Tammy monitors quality metrics
4.2 Tanuj constructs the tests
4.3 Tammy prepares the test lab
4.4 The team executes the tests

Figure 3-13 The scenes in Act 4: Managing quality

As shown in Figure 3-13, these goals are expressed in the following scenes:

- Tammy monitors quality by ensuring that her test plan is up to date with the latest requirements from Bob.
- Tanuj constructs the test cases, test scripts, and test execution records that are needed to sufficiently test the change.
- Tammy deploys the build to the test lab and notifies the team that the servers are ready for testing.

- The team executes the tests. Tanuj executes the tests to validate that Bob's request has been developed as requested and finds a defect. He collaborates with Diedrie to close the defect. Tammy executes a security scan and evaluates the results.
- Tammy confirms that all tests have been executed and reviews the quality metrics.

We explore the concept of quality management in this act.

See Part E, "Act 4: Managing quality" on page 387, for a full elaboration of this act.

3.5.5 Act 5: Delivering the solution

Synopsis: The readiness of the release is assessed and the solution is delivered.

The purpose of this act is to demonstrate how the releases the iteration. The primary goals as illustrated in Figure 3-14 for this act are as follows:

- Govern the delivery process for changes to close the release.
- Confirm that they have met the exit criteria and quality metrics.
- Package and publish the release to their stakeholders.
- Learn from the iteration.

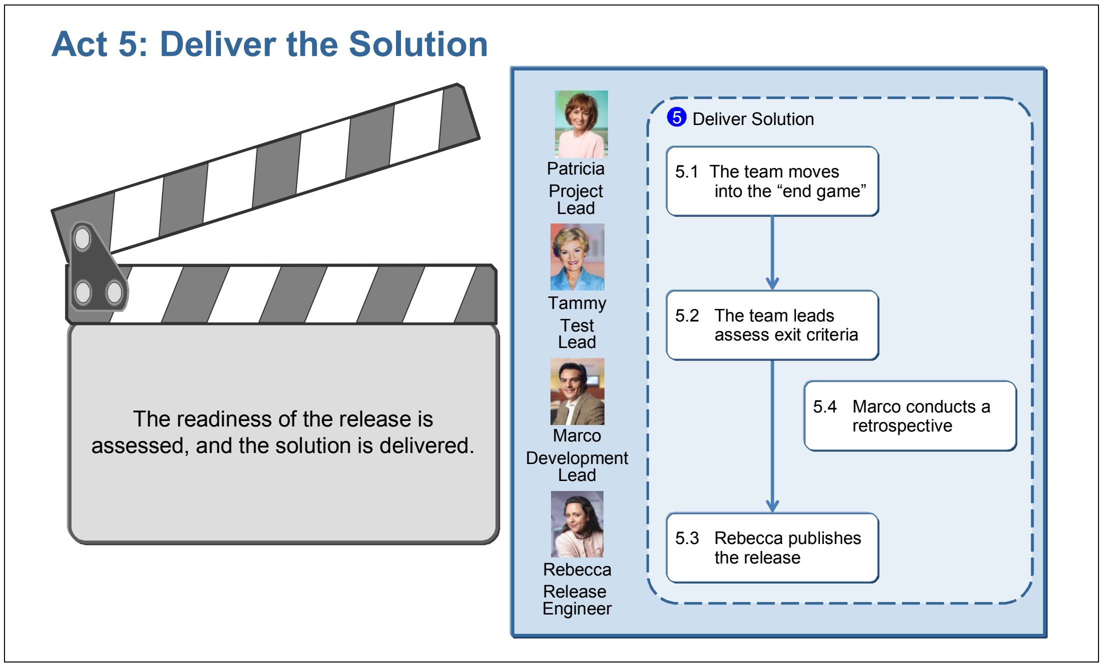

Act 5: Deliver the Solution
The readiness of the release is assessed, and the solution is delivered.
Patricia Project Lead
Tammy Test Lead
Marco Development Lead
Rebecca Release Engineer
5 Deliver Solution
5.1 The team moves into the "end game"
5.2 The team leads assess exit criteria
5.4 Marco conducts a retrospective
5.3 Rebecca publishes the release

Figure 3-14 The scenes in Act 5: Delivering the solution

As shown in Figure 3-14, these goals are expressed in the following scenes:

- The team moves into the "end game."
- The team leads assess the exit criteria.
- Rebecca publishes the release.
- Marco conducts a retrospective.

The following concepts are explored in this act:

- Governing the release to restrict source code changes
- Assessing and approving a release
- Packaging and publishing a release for re-use by external stakeholders.
- Continual improvement

See Part F, “Act 5: Delivering the solution” on page 479, for a full elaboration of this act.

3.6 Life-cycle collaboration

In this section, we highlight how collaboration occurs across the various disciplines that are involved in producing a release of software.

3.6.1 Life-cycle assets in this CALM scenario

Each of the acts in the storyboard create or reference a set of assets. Assets that are created in one act are often referred to in a subsequent act. Each act contains a section that identifies the assets that are used or referenced. Figure 3-15 shows an overview of the assets.

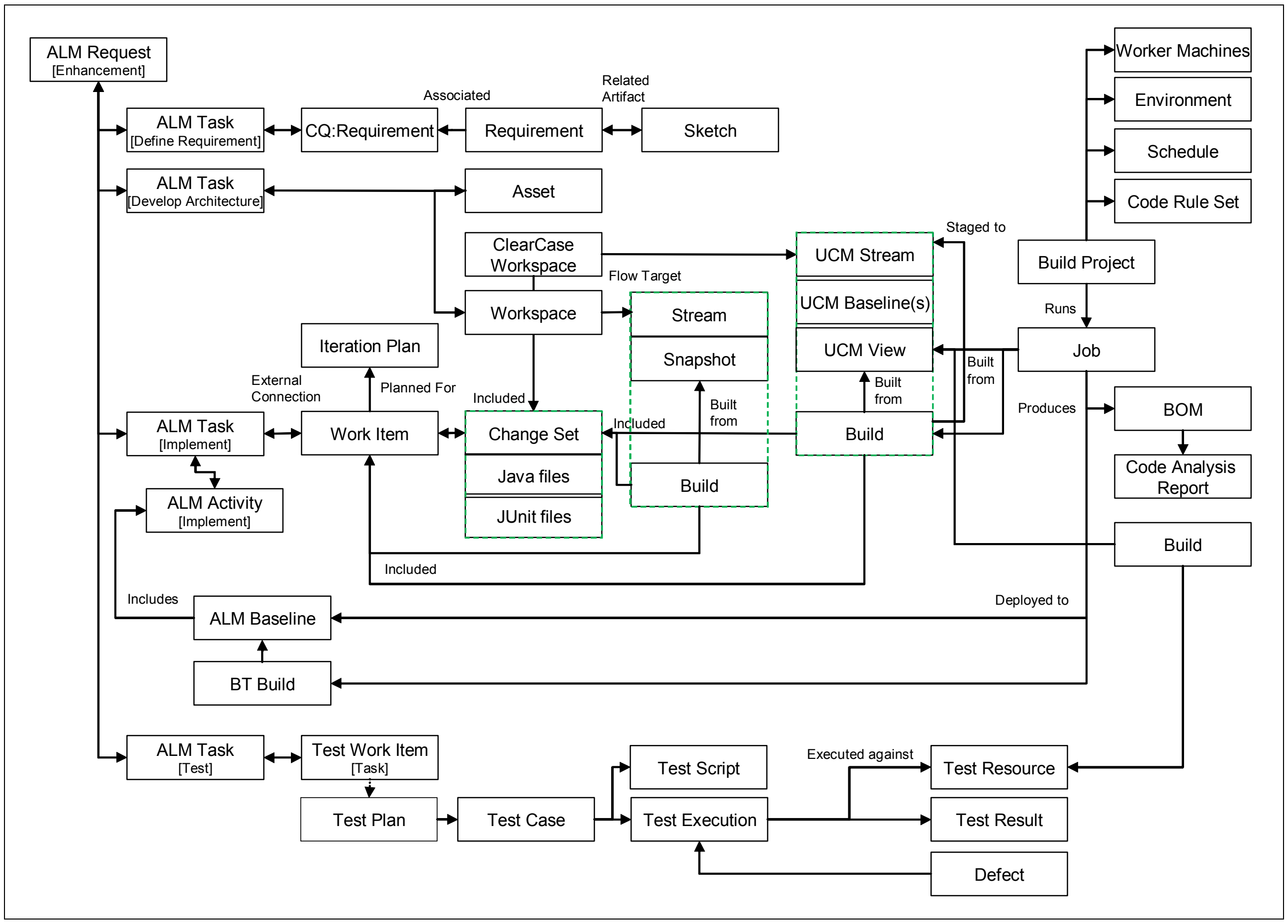


Figure 3-15 Life-cycle assets that are created or referenced in the storyboard

In Act 1, the ALMRequest [Enhancement] asset is created, which drives four ALMTasks:

- [Develop the architecture] for the solution architect that appears as a task in Rational ClearQuest
- [Implement] for the development team that appears as a work item in Rational Team Concert. The work item is added to an iteration plan.
- [Test] for the test team that is added to the test plan.
- [Define the requirements] for the product owner who creates a sketch that is linked to a requirement, which in turn, is linked to the ALMActivity.

In Act 2, Marco and his team work with the following items:

- A reusable asset is located and added to the developers' workspace.
- The work item is implemented by a change set that contains a set of files.
- The change set is delivered to the team build.
- Upon verification of a healthy build, the change set is delivered to the Unified Change Management (UCM) integration stream.

In Act 3, the Rational Build Forge project automates the following tasks:

- Creates a UCM view for the source code
- Runs code analysis against the source and produces a report
- Produces the build and checks the distribution archive into Rational ClearCase
- Creates a BTBuild and ALMBaseline record in Rational ClearQuest
- Creates a bill of materials

The project uses worker machines, which provide a solution environment with abstracted variables, a schedule, and a code rule set (used by Code Analysis).

In Act 4, Tammy and her team work with the following assets:

- The test plan is linked to requirements in RequisitePro.
- A build is deployed to a managed server in the lab.
- Test cases are updated with detail including an association with the imported requirements from RequisitePro.
- A test script is written and associated with the test case.
- Test execution work items are generated for multiple browser configurations.
- Test results exist for each test execution record.
- A defect is associated with a line in the test execution.

In Act 5, the release delivery is supported by the following assets:

- Approvals are given by the project leadership team members.
- Notes from retrospectives are created by the project teams.
- Release packages are created and submitted as request assets.

3.6.2 Agility at Scale

The enterprise in this storyboard is undergoing a transformation. The teams have heard about agile development and have begun to incorporate some of the techniques into the development practice.

Because this is a large team, which includes a team of teams, not all agile practices make sense. In this section, we discuss the agile practices that have been adopted by the enterprise team, which are displayed in the callouts in Figure 3-16.

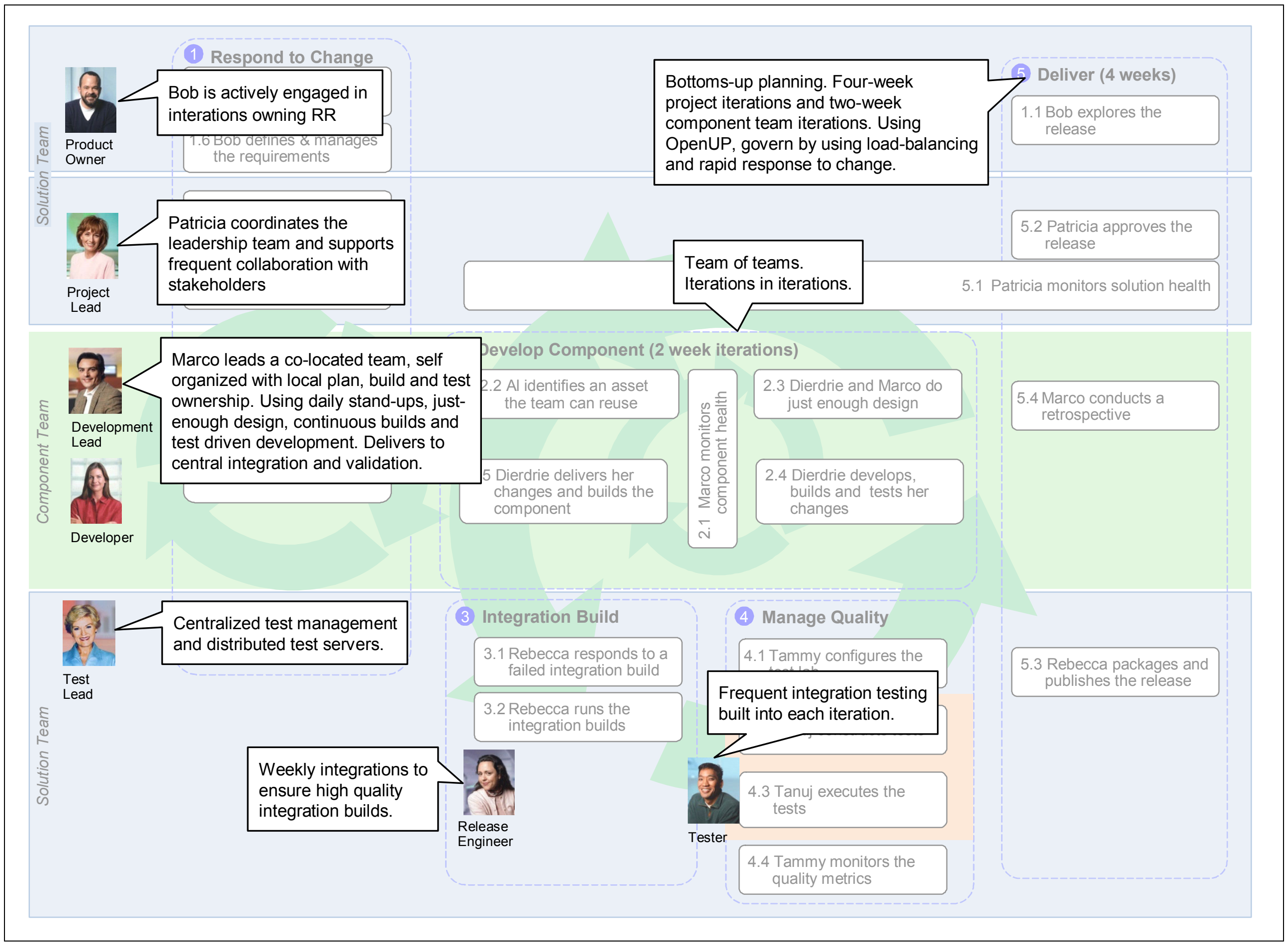

1 Respond to Change
Bob is actively engaged in interations owning RR
1.6 Bob defines & manages the requirements
Product Owner
Solution Team
Bottoms-up planning. Four-week project iterations and two-week component team iterations. Using OpenUP, govern by using load-balancing and rapid response to change.
5 Deliver (4 weeks)
1.1 Bob explores the release
Patricia coordinates the leadership team and supports frequent collaboration with stakeholders
Project Lead
5.2 Patricia approves the release
Team of teams. Iterations in iterations.
5.1 Patricia monitors solution health
Develop Component (2 week iterations)
Marco leads a co-located team, self organized with local plan, build and test ownership. Using daily stand-ups, just-enough design, continuous builds and test driven development. Delivers to central integration and validation.
Development Lead
Developer
Component Team
2.2 Al identifies an asset the team can reuse
2.1 Marco monitors component health
2.3 Dierdrie and Marco do just enough design
5.4 Marco conducts a retrospective
5 Dierdrie delivers her changes and builds the component
2.4 Dierdrie develops, builds and tests her changes
Centralized test management and distributed test servers.
Test Lead
3 Integration Build
4 Manage Quality
3.1 Rebecca responds to a failed integration build
4.1 Tammy configures the test lab
5.3 Rebecca packages and publishes the release
3.2 Rebecca runs the integration builds
Frequent integration testing built into each iteration.
Weekly integrations to ensure high quality integration builds.
Release Engineer
Tester
4.3 Tanuj executes the tests
4.4 Tammy monitors the quality metrics

Figure 3-16 An Agility-at-Scale CALM scenario

The actively engaged product owner

Bob is the product owner for the application. He prioritizes the requirements and is actively engaged in the iterations. He has access to the iteration builds and uses milestones to prioritize the next iteration. He works closely with his stakeholders to receive early feedback on the software and to provide guidance back to his team.

Iterative development

Patricia has set the cadence for the project at four-week iterations. Within each of these iterations, the team prioritizes requirements, does just enough design, implements and tests the team builds, builds the entire solution, and tests the solution. Patricia works with all of the teams on a frequent basis to assess the team's health and ability to deliver the iteration. Modifications to the plan occur rapidly with little ceremony to maintain a realistic view of the iteration. Patricia has frequent contact with the stakeholders to review progress and receive feedback on their direction.

The component teams are allowed to self-organize provided that they work within the project framework. In this storyboard, the acquired team, which is led by Marco, has decided to work in two-week iterations.

Agile small teams

The component team is relatively small but also geographically distributed, and some of the members work from their home office. Marco continues to conduct daily stand-up meetings with his team to set the priorities for each day. They are a team of generalists who own every aspect of their part of the component, from analysis to testing. They use a continuous build strategy to catch build problems early. They also use the agile test-driven development technique. Each week they promote a good build to the integration build.

Frequent integration builds

Integration builds occur on a weekly basis. All component teams deliver their code to the integration stream, where the weekly integration build occurs. The integration build includes running of tests against the solution build to assess the build quality. Results of the build and the build verification tests are passed to the test team.

Frequent integration testing

The solution test team, which is led by Tammy, has the opportunity to work with a new solution build every week. The team assesses the integration build results to determine whether they must deploy the latest build into the test lab. By conducting solution-level testing as part of the iteration, the feedback loop by the development teams allows the teams to identify and fix defects early and often.

3.7 Reference architecture and configuration

This scenario in this book was written to show the full power of the Rational team-based products. Many combinations of products can be used to define a CALM solution. This is a reference implementation, but is by no means, the only implementation. In this section, we describe the reference implementation that is used to build this story.

3.7.1 An enterprise CALM solution

The configuration of software products and servers is distributed across several geographic locations including the corporate headquarters, the acquired company's offices, and the trusted third-party provider. Figure 3-17 illustrates the software and servers at each location.

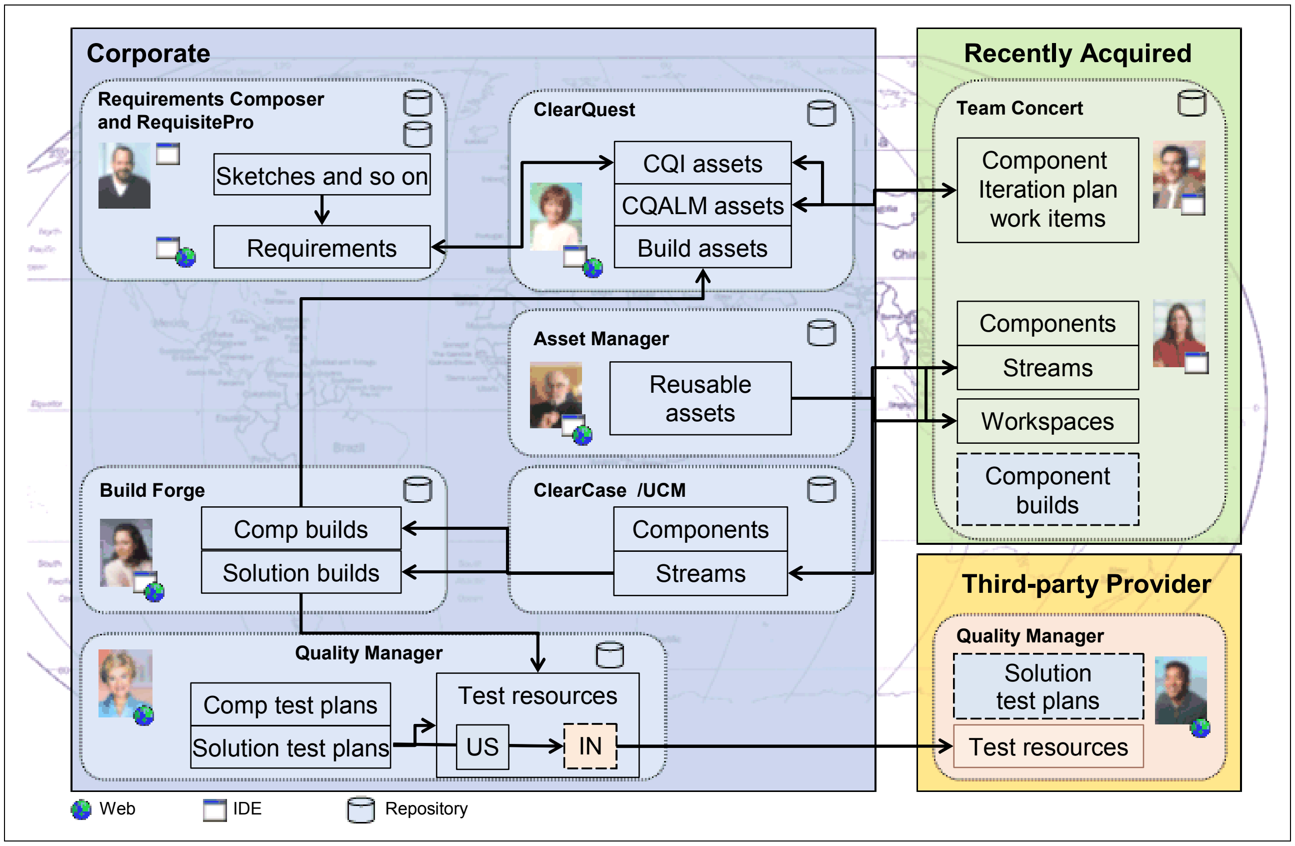


Figure 3-17 The software configuration used in this scenario

The product owner

The product owner uses the following applications:

- Rational RequisitePro to manage and trace requirements
- Rational Requirements Composer to define, illustrate, and storyboard requirements
- Rational ClearQuest to manage change requests

The project manager

The project manager uses Rational ClearQuest with integrations to Rational Team Concert and Rational Quality Manager.

The development lead

The development lead uses Rational Team Concert with an integration to Rational ClearQuest and Rational ClearCase.

The developer

The developer uses Rational Team Concert to develop, unit test, and build the software component.

The architect

The architect uses the following applications:

- Rational Asset Manager to identify a reusable asset
- Rational ClearQuest to size a request and help the team determine whether they can contain the request in the iteration

The release engineer

The release engineer uses the following applications:

- Rational Build Forge Enterprise with adapters to Rational ClearCase, Rational ClearQuest, and Rational Software Analyzer to automate the entire build process.
- Rational ClearCase uses the ClearCase Connector to Rational Team Concert and manages the source for the entire solution. Rational ClearCase manages the solution source code. Source code from Rational Team Concert is copied to Rational ClearCase by using the Rational ClearCase Connector.

The test lead

The test lead uses Rational Quality Manager with integrations to Rational ClearQuest, Rational RequisitePro, and Rational AppScan Tester Edition to plan, manage, construct, execute, and analyze the testing effort.

The tester

The tester uses Rational Quality Manager to construct and execute tests. In addition, the following products are referenced, but no detailed information is provided:

- Rational Functional Tester
- Rational Performance Tester
- Rational Services Tester

The full product list

Detailed information in the context of the storyboard is provided on how to use the following products (in order of appearance):

- Rational ClearQuest 7.1.0.0
- Rational RequisitePro 7.1.0.0
- Rational RequisitePro Composer 1.0.0.0
- Rational Team Concert 1.0.0.0
- Rational Asset Manager 7.1.0.1
- Rational Build Forge 7.1.0.0
- Rational Software Analyzer 7.0.0.0
- Rational Quality Manager 1.0.0.0
- Rational AppScan 5.6.0.0

Additional products referenced in the scenario

The following products are referenced in the scenario as potential integrations. Information about how to integrate and use these products is not provided in this book.

- Rational ClearCase 7.1.0.0
- Rational Functional Tester 8.0.0.0
- Rational Performance Tester 8.0.0.0
- Rational Services Tester 8.0.0.0
- Rational ClearQuest Test Manager 7.0.1.0

3.7.2 Product integrations for this scenario

In this section, we provide a brief overview of the product integrations that are used to support this storyboard.

Rational RequisitePro and Rational Requirements Composer

Rational Requirements Composer is a new product to define requirements. It provides a rich user interface for sketching and storyboarding ideas and business processes. The product has a database for managing the sketches. An Eclipse user interface is used to create the sketches, storyboards, and business processes.

Rational RequisitePro manages requirements in a database. The sketches that are created in Rational RequisitePro Composer are linked to the requirements that are managed by Rational RequisitePro. When viewing a requirement in Rational RequisitePro, the sketch can viewed by clicking its link when the Rational RequisitePro Composer user interface is installed on the local desktop.

In this reference scenario, the Rational RequistePro and Requirements Composer databases were installed on the same server. The Rational Requirements Composer user interface was installed on a desktop. The assets created by Rational Requirements Composer are linked to the requirements that are managed by Rational RequisitePro by using the integration that is provided by these products.

Rational RequisitePro and Rational ClearQuest

Rational ClearQuest 7.1 was installed on a server by using an IBM DB2 database. The Rational ClearQuest ALM schema and sample database were used.

Rational ClearQuest and RequisitePro have an existing integration that was used to support the storyboard. The Rational RequisitePro package was applied to the new Rational ClearQuest ALM schema by using the ALMTask and ALMActivity records. This creates a Requirements tab on each record that contains links to requirements that are managed by Rational RequisitePro.

Rational ClearQuest and the ClearQuest Connectors to Jazz product-based repositories

The Rational ClearQuest Connector was used to connect the Jazz product-based repositories, which are Rational Team Concert and Rational Quality Manager.

The Rational ClearQuest Task and Activity records are candidates for interoperating by mapping these records to the Jazz work item. The ALMTask is mapped to the work item of the type “Task.” The ALMActivity is also mapped to the Jazz work item “Task,” but with a parent relationship back to its corresponding task.

Rational Team Concert and the ClearCase Connector

The ClearCase Connector is used to interoperate the source changes in Rational Team Concert over to the integration stream in Rational ClearCase.

Rational Build Forge, Rational ClearQuest, Rational ClearCase, and Rational Software Analyzer

Rational Build Forge is used to conduct the integration builds. It uses the Rational ClearCase adapter to collect the latest source, baseline the source, and check the packaged build back into source control for staging.

Rational Build Forge also integrates with Rational ClearQuest to create build and baseline records that are used to identify the build status, the baseline name, and delivered activities between baselines.

Rational Software Analyzer is integrated to run static analysis on the source code prior to compilation.

Rational Quality Manager and Rational AppScan

Rational Quality Manager provides integrations with test execution products. To illustrate this capability, an integration with Rational AppScan is demonstrated.

3.7.3 Supporting distributed teams

The scenario described in 3.3, "The software delivery team" on page 50, focuses the story on the Account Opening project leadership team, which is led by Patricia, and the CreditCheck component development team, which is led by Marco. The description in this book on the deployment and configuration of a collaborative development environment has been focused on in this core scenario.

However, the Account Opening project is run by a "team of teams." While the CreditCheck team is one subteam, there are may other teams. In this section, we describe some of the considerations in deploying an enterprise-sized collaborative development platform for the Account Opening project. For larger Geographically Distributed Development teams, the collaborative development platform must support local and remote access. Repository replication has to be taken into consideration.

Some general site topology considerations apply:

- Identify core sites

  A *core site* is often the owner of a program, an application, or a project. Core sites have IT administration and can manage a global collaborative development platform.

- Identify remote sites

  A *remote site* often contributes to collaborative development by having team members join projects. A remote site is often lacking the administration staff to locally manage a development infrastructure. Team members rather connect remotely to the services of a core site.

- Identify mobility needs

  Some team members are needed for mobility, which might entail working from home or from another temporary remote location. It might also require frequent switching between sites.

In the context of this book scenario, it is reasonable to make the following assumptions regarding the deployment of an enterprise-sized collaborative development platform for the Account Opening project:

- The Account Opening project depends on the teams and team members in multiple geographies. The team of teams is distributed over multiple development sites and over multiple continents.
- Larger core sites in the enterprise form hubs that serve a region of development sites with collaborative development platform services. The Account Opening project teams use two or more hubs that serve its geographic regions respectively.
- Team members are mobile, from time to time, and require remote access from home or from other temporary non-office locations.
- The project depends on external service providers, for example in the test team, that from their external site connect remotely and securely to the Account Opening project repositories.

When deploying a collaborative development platform (Figure 3-18 on page 75) for the geographically distributed Account Opening project, the following considerations apply:

- The Requirements repository is deployed as a central service to one of the core sites. It is advised that the site is chosen, so that the repository is co-located with business and application stakeholders. It is also advised that application stakeholders use a requisite Web client to access requirements artifacts. If the integration with Rational ClearQuest is used, mastership of the requirements records must be kept at the core site that hosts the requirements repository. The integration of Rational RequisitePro and Rational ClearQuest does not update records that are mastered remotely.
- The Rational ClearQuest and Rational ClearCase repositories are deployed to all core sites with a component development team. Rational ClearQuest or Rational ClearCase Multisite are used to replicate information across the sites. Mastership of Rational ClearQuest ALM artifacts is automatically set and managed by the Owner field on the ALM records. This implies that ALM Request records stay with Bob because he is the owner. The ALM Task records stays with Patricia, and the ALM Activity records are remastered to the replica for the owner of the activity. Other ALM records that are related to project ALM Work Configurations stay at the site of the project administrator.
- Some project component teams, for example the CreditCheck team, use Rational Team Concert. For those teams, a Jazz server is deployed. To establish interoperability between the Rational ClearQuest clan, a ClearQuest Gateway server is deployed at one of the core sites and co-located with one of the Rational ClearQuest MultiSite replicas. In Rational Team Concert 1.0, the Rational ClearQuest Connector supports one gateway per clan. The deployment location of the Jazz server is not required to be co-located with the Rational ClearQuest clan or the ClearQuest Gateway.

  In our scenario, the Jazz server is managed by the CreditCheck team. It is advised to consider mastership and set up Rational Team Concert users in Rational ClearQuest with the appropriate mastership properties. The Rational ClearQuest query that is used to select the records to be synchronized with Rational Team Concert should also be configured with a filter that prevents records with remote mastership to be synchronized.
- A project area and team areas for each component team are deployed to the Jazz server. Multiple projects can share a Jazz server that then serve multiple project areas. Each component team manages their team area for their component.

- Some component teams, such as the CreditCheck team, might have local build services. Other teams rely on the central build services in the Account Opening project that is run by the release engineer, Rebecca. Component teams can also decide to integrate with the central build service, but take ownership of all aspects on managing and monitoring the component builds.

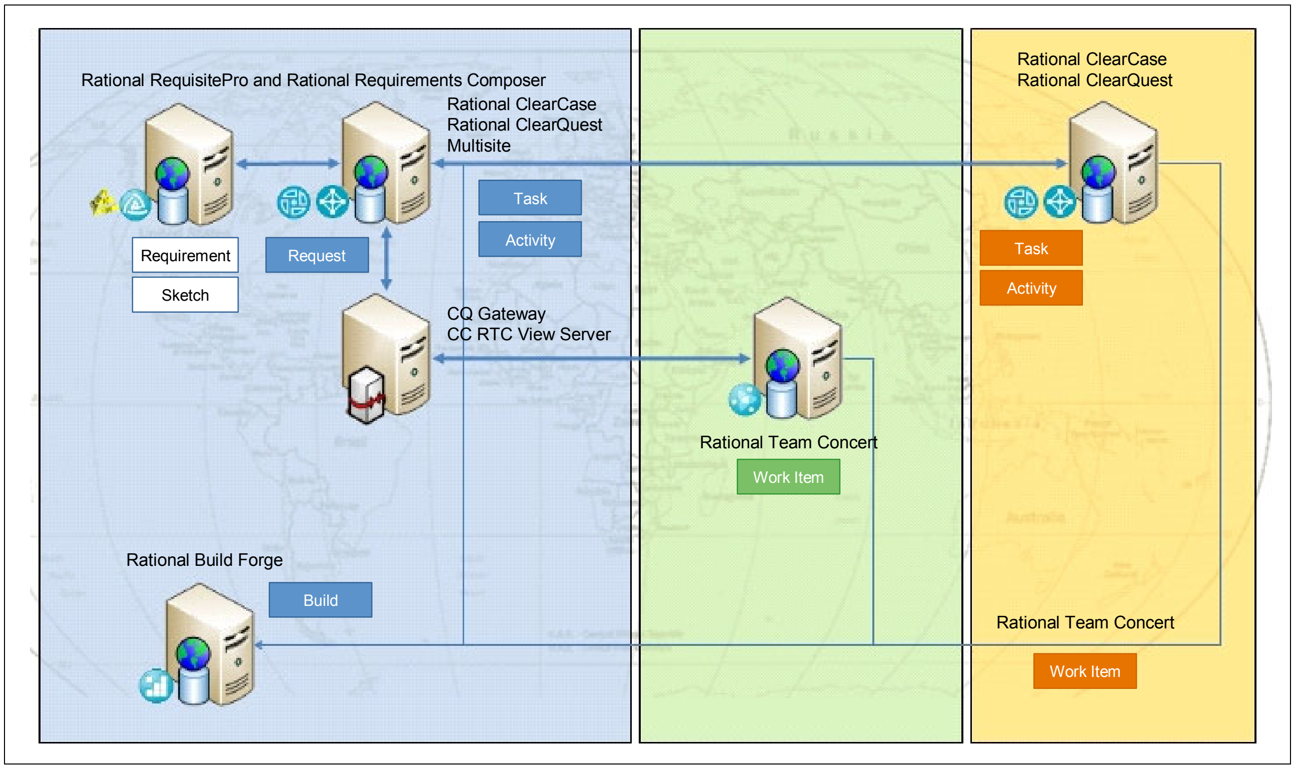


Figure 3-18 View of the collaborative development platform deployed to support the Account Opening project

Part B

Act 1: Responding to a change request

In this part, we introduce and describe the first act in the storyboard. The first act involves the submission of a new request against a project iteration and the teams' response of planning the work in the current iteration. Two important domains are discussed in this part:

- Change management
- Requirements definition and management

Chapter 4, "The team responds to a requirement change" on page 79, provides information about change management and requirements definition and management as they relate to the scenario. Chapter 5, "Rational ClearQuest, Requirements Composer, and RequisitePro to manage stakeholder requests" on page 115, provides details about the IBM Rational products that are used to support this act of the storyboard.

Role-based guide: To understand how the content in this part applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

4

The team responds to a requirement change

In this chapter, we explain how the team responds to a requirement change. As discussed in Part A, “Collaborative Application Lifecycle Management defined” on page 1, in the scenario for Collaborative Application Lifecycle Management (CALM), the storyboard is divided into five major acts. This chapter and Chapter 5, “Rational ClearQuest, Requirements Composer, and RequisitePro to manage stakeholder requests” on page 115, constitute Part B, “Act 1: Responding to a change request” on page 77, of the storyboard. Act 1 entails responding to a change request. Most acts in the storyboard cover one primary discipline. Act 1 is unique in that is covers both change management and requirements definition and management.

In this chapter, we introduce the following concepts:

- Change management and requirements definition and management
- A reference scenario for managing a requirement change
- How this scenario impacts subsequent scenarios in the life cycle
- Considerations for change management and requirements definition and management
- Considerations and variations on the scenario

Specifically this chapter includes the following sections:

- 4.1, “Introduction to change management” on page 80
- 4.2, “A reference scenario for responding to a change request” on page 105
- 4.3, “Considerations in change management” on page 112

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

4.1 Introduction to change management

New market realities are driving changes in change management and requirements definition and management. In this section, we discuss these market shifts and the impact on the software development disciplines.

4.1.1 The changing change management market

The change management discipline that traditionally placed an emphasis on the management of defects is undergoing a significant transformation. Several factors are driving this change as discussed in 2.1.1, “Changes in the ALM market” on page 16. The most important factor is the shift to focus on business outcomes. The business, development, and IT operations teams must collaboratively deliver software that delivers value to the customer and the bottom line. As such, there is increased demand on the development team to respond to requests and deliver software releases. Introducing a high level of collaboration is critical because development processes, such as change management, remain fundamentally difficult due to shifting requirements, disconnected teams, and other real-time changes.

To respond to the business needs of cost containment and increased capacity, development organizations continually seek and refine geographic distribution. These distributed teams deliver new applications, maintain existing application, or refactor existing applications. The challenge in change management is to have organizationally disparate team members from the business, development, and operations, who are also geographically distributed, act as a team that can quickly respond to new requests. Team members who are often distributed over multiple continents and time zones require a new set of collaborative development principles in change management to effectively form the software supply chain that connects stakeholder requests to integrated components and applications.

To respond to customer needs, agile methods have emerged that emphasize the importance of continuous stakeholder involvement and guidance. This interaction between the users and the development team drives transparency into the change management workflow. This interaction benefits both parties. The development team receives early and continuous feedback, which contributes to the success and viability of the software they are producing. The product owners, analysts, program office, and other cross-organizational stakeholders know that their requests are taken seriously by the development team and have easy access to project information to provide timely input.

All parties that are involved have a stake in the success of the project, and through constant feedback loops, they obtain insight into the vocabulary, culture, and unique challenges that the other faces. Managing requests and defects that impact each iteration becomes a critical process in establishing this feedback loop. The team’s ability to deliver software that meets the needs of the user improves in a timely and efficient manner improves with each new release.

As teams encourage feedback from their stakeholders, the number of requests increases. Therefore, change management involves actively triaging and sizing change requests to assess their impact on iteration plans. In planning iterations, distributed teams must orchestrate the collaboration that is needed to analyze and prioritize changes. Most change requests require work to be completed by more than one team. Most requests require work from both development and test teams. Other requests can involve additional work from an analyst to detail requirements or can significantly impact the architecture.

To deliver complete functionality with sufficient quality, all of this work must be added to the iteration plan, assigned and monitored. Additionally, regulatory compliance demands that a team demonstrates what work went into a change, how it was implemented, built, and tested. Coordinating the work that is related to a single change request not only improves the team's ability to deliver high quality iterations, but can also help in responding to regulatory audits.

It is clear that today's software development teams must take a shared responsibility in the project success. Additionally, agile methods underscore the importance that team members participate in many roles crossing the traditional discipline silos. Disparate tools, often separated by disciplines, are no longer enough. People and work cross the traditional boundaries of software development disciplines. A single team member might need access to the requirements definition, the models, source code, build, and test cases. As such, the team member might need to collaborate with the team members who produced these assets.

The role of change management along with CALM is to link the people, information, and tools by using a streamlined and iteration-specific process. The change is understood and tracked across team members, disciplines, repositories, and geographies. To do so, teams need insight into the project status to enable them to respond to change and track project health.

All of these factors require development teams to communicate and monitor progress throughout the development life cycle and to work together to find solutions that deliver effective software shipped on time and on budget. To succeed at this orchestration of workflows, teams, and assets, a CALM solution relies upon the following success indicators as underpinnings to effectively develop a release of software, as discussed in 2.2.1, "Success indicators" on page 30. Collaborative change management takes a central role in contributing to successful CALM.

- Collaboration

  Team members must collaborate when submitting, analyzing, elaborating, triage sizing, planning, and delivering change requests. The emphasis here is on the *team*. A change typically impacts more than one discipline, and as such, all disciplines are taken into account when addressing a change request. By doing so, a team is better equipped to orchestrate the work in delivering committed changes with sufficient functionality and quality.

- Traceability

  Team members must ensure that the results of their changes are traceable to the originating request and ensure that the build delivers the changes. An understanding of the delivered changes helps the test team to efficiently and thoroughly target their testing effort. These activities connect the requirements definition, enterprise build, and change management workflows.

- Distribution

  Teams that are distributed must ensure a close connection to the software delivery chain that integrates their changes into versions of the solution. These teams are likely to use separate repositories to organize assets. It is critical that these assets are linked and the team's distribution does not break traceability.

- Automation

  Teams can improve their performance and the quality of delivered applications by automating parts of the change management workflow and by automating the management of application life-cycle assets. Traceability adds significant value to the governance aspects of change management, but requires consistency and discipline to maintain. By automating the creation and maintenance of asset traceability, the team can more effectively apply change management and leverage its value. Many of the change processes can also be seamlessly integrated into team collaboration to alleviate the burden of maintaining traceability across change assets.

- Continuous improvement

  Teams seek strategies to continuously improve the change process. Conducting retrospectives at the end of each iteration and adopting lessons learned will aid in process improvements and reduce friction.

Change management workflows in collaborative development

The change management workflows in collaborative development support the teams' need to engage stakeholders in providing application requests, manage and predict the impact of these requests, and ensure the delivery of the committed change in a timely manner, with expected functionality and with sufficient quality. Change management has transformed into workflows that consist of the following actions:

- Submitting change requests (often as stakeholder requests or release requirements)
- Approving (or rejecting) requests
- Planning and estimating the work and delivery of approved requests
- Monitoring the work and delivery of planned requests
- Signing off on the completed delivery of requests
- Guarding the delivery of unapproved changes

As exemplified in this scenario, a single software request impacts the design, development, build, and testing of an application. Each role during the software development process produces content that contributes to the design, implementation, and testing of that request. Understanding and managing the amount of effort involved to satisfy each request is critical for a team to deliver on time or under budget. The project manager must have confidence that all of the requests have been analyzed, implemented, and tested with sufficient quality before agreeing to deliver the solution. The challenge for software development teams is not in creating a single asset (source code, requirement, or test case), but rather in understanding the relationships between those assets.

The change management discipline spans the full scope of software development and delivery, as a single change is tracked from the initial triage and planning, through the final delivery of the release. In this book, we touch on change management as follows:

- In Part B, "Act 1: Responding to a change request" on page 77, we discuss the leading part of change management by discussing requirements definition and management. That is, we discuss the initial steps of change management where requests are submitted, triaged, approved, estimated, and planned.
- In Part C, "Act 2: Collaborative development" on page 211, the change is implemented.
- In Part D, "Act 3: Enterprise integration builds" on page 313, the change is incorporated into the solution build as part of enterprise build management.
- In Part E, "Act 4: Managing quality" on page 387, the change is tested as part of the team's quality management strategy.
- Finally, in Part F, "Act 5: Delivering the solution" on page 479, the change is delivered as part of the solution delivery.

The chapters in which we discuss change management also exemplify the benefits of ready-to-use Rational Application Lifecycle Management (ALM) solutions that provide support for managing many of the challenges presented by Geographically Distributed Development (GDD) and CALM. The Rational CALM solutions provide support for a streamlined, Agility-at-Scale application development process that is both role-based and process-driven.

4.1.2 The changing requirements definition and management market

Teams in the software development market of today must be adaptive. Months of long requirements analysis phases are becoming a thing of the past. Yet, requirements help provide a foundation for change management. Requirements are the items generally in the analysis discipline, providing information to a development team in order to create a successful software application. Requirements are leveraged throughout the entire life cycle from defining the solution use cases to the solution validation test cases. Requirements help to answer such key questions as: "Do we have sufficient functionality with sufficient quality?"

Project failure is often indicative of poor requirements definition or management. Failures tied to requirement problems include incomplete requirements or incorrect requirements. In some cases, nearly half of the software development budget can be consumed by poor requirements. Requirement problems that are found later in the cycle prove to be more costly. A requirement problem found in the maintenance phase of a project might be up to 200% more expensive than requirement errors discovered and addressed during an early phase of the project. For example, requirements errors that are discovered during the Inception phase are much less expensive to correct than those found during the Construction phase. Figure 4-1 shows details of the phases of the Open Unified Process (OpenUP) and Rational Unified Process (RUP).

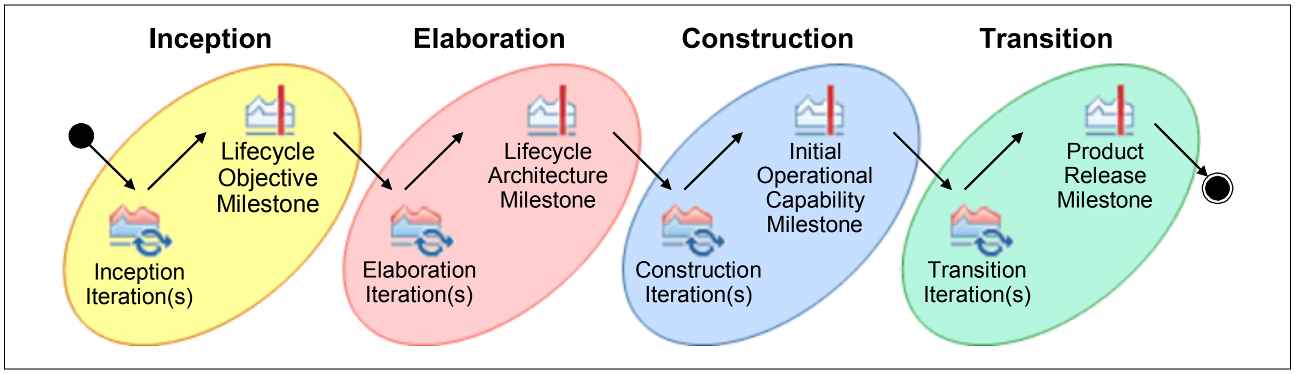

Inception
Elaboration
Construction
Transition
Lifecycle Objective Milestone
Inception Iteration(s)
Lifecycle Architecture Milestone
Elaboration Iteration(s)
Initial Operational Capability Milestone
Construction Iteration(s)
Product Release Milestone
Transition Iteration(s)

Figure 4-1 OpenUP and RUP phases

Misunderstood or badly communicated requirements are a key risk factor in GDD projects. In a 24x7 globally competitive market, the stakes involved in getting it right are higher than they have ever been. However, teams can no longer risk "analysis paralysis." The go-to-market time is faster than in the past and requires organizations to collaborate with their stakeholders and obtain constant feedback and iterative refinement of assets.

Therefore, organizations are looking for ways to improve the requirements process by finding a balance for what is enough to get started on prototyping and then using feedback loops to gain greater clarity as the project progresses. This desire to expedite the requirements process often calls for leveraging different types of techniques to improve the communication and definition of requirements.

The article "Examining the 'Big Requirements Up Front (BRUF) Approach'" describes issues that occur during requirements management, the importance of writing just enough detail early in the requirements process, and collaborating with stakeholders to obtain feedback. You can find the article on the Web at the following address:

http://www.agilemodeling.com/essays/examiningBRUF.htm

Inputs to requirements come in many forms such as business process sketches with predefined key performance indicators, application sketches, storyboards, process flows, and diagrams. These assets are important because they aid in the translation of business goals and objectives to requirements. Rich text and images live side by side to help analysts convey information by using a rich and detailed language.

Requirements definition and management tooling must be adaptive to these changing techniques. Solution teams no longer center their requirements processes solely around the management of requirement information. It is no longer sufficient to capture and track requirements across the life cycle. Organizations realize the value in capturing the “right” requirements at the appropriate level of detail that is useful for the current phase of the software development life cycle. Involving a larger set of business oriented stakeholders to be involved in the requirements definition process is crucial. The need to align business and IT is bringing definition techniques “front and center” as a key element to the requirements process.

Requirements definition, also known as *requirements development*, is the identification of needs that a project must satisfy from its many stakeholders. It is an iterative process where we define and refine problem statements. We also conceptualize possible solutions and their impacts, involving the understanding of business goals and then eliciting, defining, and elaborating on the user and software requirements that align with those goals.

Requirements definition focuses on the elicitation techniques that are performed by a solution team to better understand stakeholder needs and desires. With requirements definition activities, teams can obtain constant feedback from stakeholders and refine requirement assets. Sketches, storyboards, and process flows are quickly created, reviewed, and refined. The meaningful designs are retained, and the others discarded. Effective organizations allow their requirements definition process to support an environment where the team works with stakeholders to learn, communicate, and share a common vision.

Creativity is the key to requirements definition. The process can be fast and loose until a shared vision begins to stabilize. Requirement assets that are derived during the requirements definition process are refined to describe capability statements to which the system will conform. How the requirements are expressed depends upon the requirement process that is followed by the software development organization.

In contrast, *requirements management* refers to activities that are undertaken by a product or software teams in order to gather, store, track, prioritize, and implement those requirements. It is the process of communicating and controlling a project scope while incorporating changes at the same time. It describes the process where a common understanding of requirements is agreed upon by the stakeholder and the development organization and the requirements are tracked across the software development life cycle. Requirements management provides a systematic approach to gathering, organizing, documenting, and managing the changing requirements of an application. Where requirements definition is free form and creative in order to capture the ideas, requirements management brings order and organization to the final set that will be implemented.

The following key market trends are impacting requirements definition and management:

- The spreadsheet and document-centric requirement elicitation approach is no longer sufficient to capture the breadth of information for requirements. These methods make it difficult to identify and address requirements change, categorize requirements, and extract meaningful information about the requirements. All too often, the big picture or story is lost in a long list of minute details, and analysis paralysis sets in as teams struggle to organize and prioritize long lists of line items.
- Better requirement elicitation and definition techniques are required to align business and IT. Solution teams that do not understand the “business” or their stakeholders spend too many cycles refining requirement content later in the development life cycle because the

requirements do not address business problems. Or worse, the software might never meet the needs of the user due to a lack of understanding. To meet the needs of users, the business analysts must be empowered to capture and communicate their ideas to the development team.

- Improved collaboration is crucial across the software development life cycle regardless if it is between the customer and IT or within the solution team. All stakeholders in the project must be on the same page and have a common vision. The team must design a system that satisfies the request. The developers must implement the same vision that the architect defined. Certainly the test team must test from the same set of requirements to confirm that the implementation meets the original need. Without this common vision, the solution team cannot build something that addresses the business problem.
- Process rigor depends upon the type of project. Software development projects and processes come in all sizes. There is not a one-size-fits-all approach. Some projects are amenable to agile techniques, but more rigorous requirement processes can be adapted where required for other projects. The agile method places rigor through continuous feedback between stakeholders and the team, collaboration, and requirements iteration, where traditional attempts to obtain rigor are through process-driven approaches. CALM tools must be flexible and allow for various organization types to be effective regardless of whether the organization is using agile, iterative, or waterfall development methods.
- There is the desire to improve effectiveness of distributed teams. Teams are not always co-located. If teammates are not in the same building, it becomes more crucial to provide tools that support collaboration and clear articulation of the requirements.
- Organizations are moving away from a “silo” role-based approach in the software development processes and moving toward activity-driven functions. Traditional role lines are blurring. Team members are wearing more than one hat and performing many activities across the software development life cycle such as requirements elicitation, definition, and design techniques. In a software development market with two- to four-month project schedules, a silo approach is ineffective due to wasted time in transitioning information between roles. Expertise in a single discipline is no longer the norm.

  Effective teams collaborate in the requirements process and share information of the requirement assets. The “collaborator” is the new role. In agile development, these persons are referred to as *generalizing specialists* as referenced in the article “Generalizing Specialists: Improving Your IT Career Skills” at the following Web address:

  http://www.agilemodeling.com/essays/generalizingSpecialists.htm

Requirements and Collaborative Application Lifecycle Management

ALM focuses on the synchronization of the entire team and the handoffs that occur across disciplines. Requirements are assets in CALM that are used across disciplines by many roles. Although these assets can be used across disciplines, it is not good enough to hand off requirements. Information that is provided must be articulated in a clear and concise language.

The evolving IT development processes necessitate greater discipline across the software development life cycle than traditional methods. This point is also true in the area of requirements definition. During milestone reviews, quality and consumability of requirements are real. The information that is provided by stakeholders (as requirements) is critical to the development organization. The stakeholders must be able to express and prioritize their needs. Conventional elicitation techniques often are inadequate to provide the depth of information that is needed to drive the requirements process. Collaboration between stakeholders and the development team must be supported through the life cycle as illustrated in Figure 4-2 on page 86.

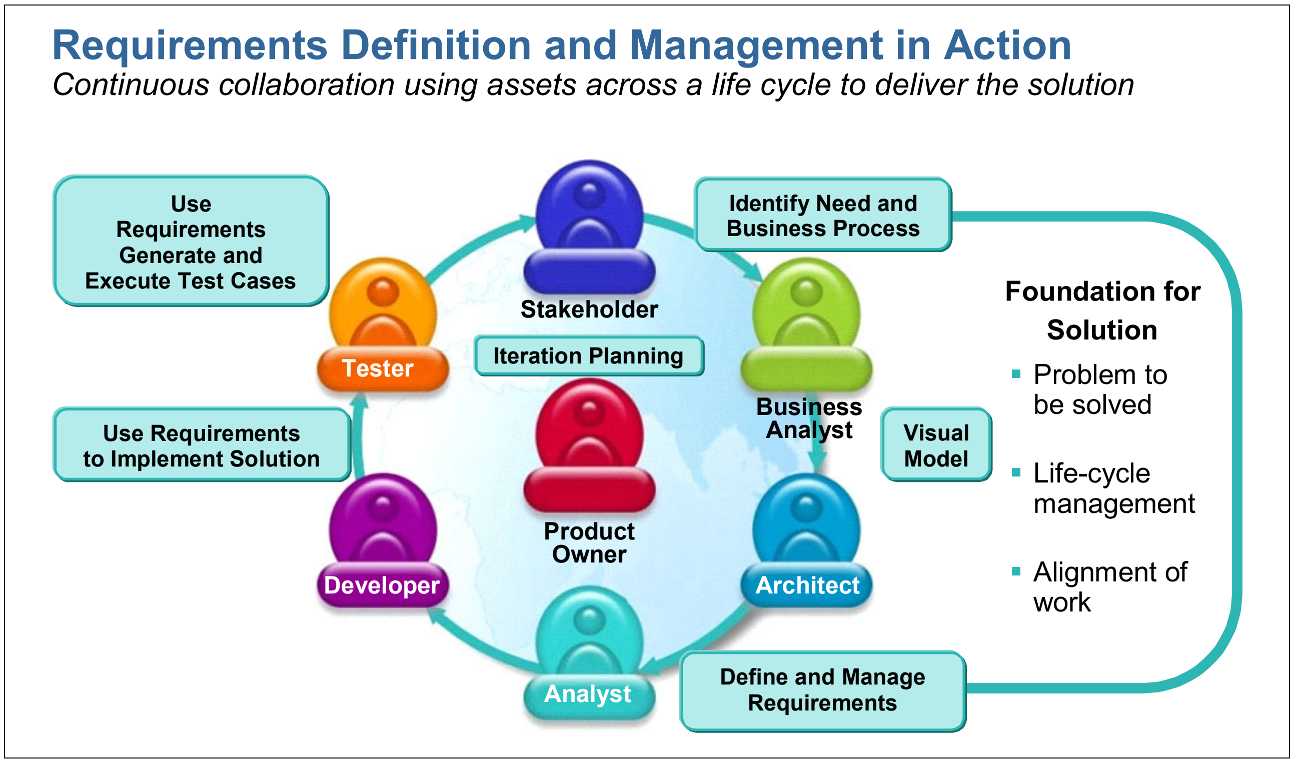

Requirements Definition and Management in Action
Continuous collaboration using assets across a life cycle to deliver the solution
Use Requirements Generate and Execute Test Cases
Stakeholder
Identify Need and Business Process
Tester
Iteration Planning
Business Analyst
Foundation for Solution
Problem to be solved
Life-cycle management
Alignment of work
Visual Model
Use Requirements to Implement Solution
Product Owner
Developer
Architect
Analyst
Define and Manage Requirements

Figure 4-2 Collaborative requirements definition and management

The requirements process begins with the definition of a need by a stakeholder. The stakeholder has the opportunity to convey their business problem by using the language and vocabulary of the business and its customers. Requirements definition and management activities ensure that the business can capture and communicate their ideas to the solution team, and at the same time, the solution team can learn the nuances and expectations of their customer. This is the point where the business meets development. It is in the stakeholders' best interest to clearly articulate the need, and having the tools that capture the rich detail and nature of the request facilitates that goal.

The stakeholders' request is reviewed to assess if it aligns with the business goals and strategy for the project team. If the request aligns with the strategy and can be contained within a given iteration, the request is added to the iteration plan. See "Work management and iteration planning" on page 89. The iteration plan contains a list of items that must be resolved. Members of the solution team collaborate with their peers (project manager, developers, analysts, and so on) as part of the planning process. As part of the planning process, activities are created and executed upon for a given iteration.

Each person on the solution team plays a critical role or roles in implementing the solution. Information defined by the stakeholder is used to complete activities across multiple software disciplines. These activities can include additional elicitation techniques to provide a context for the request when the request does not provide the "full picture" that is required to implement a solution. In this case, the business analyst contributes to the effort by providing context around the statement that is expressed by the stakeholder. Context is provided in the form of a business process or application sketches, storyboards, models, and other rich text content.

These kinds of activities allow an organization to capture the "details" of the business that might not be included in the request. Requirements content evolves through these efforts. It is a critical point as the business and solution team come together to create a common vision for the solution to be developed. Need statements are shared and discussed to ensure a common understanding, and as such, they provide the foundation for the requirements process. The team then creates a solution to address these need statements.

The analyst then organizes and manages the requirements. Management of the requirements includes capturing important requirement descriptors such as priority, release, origin, and others. Additionally, pertinent traceability information for related requirements can also be captured.

Other team members who work with the analyst use the requirements to complete their activities. An architect reviews the requirements and defines how an application will be implemented to solve the stakeholder request. This information can be captured in the form of design.

Developers use the requirement and design information from previous activities to implement the solution. The developer now has an understanding of the following concerns:

- The business need and problem
- What is expected to address that need
- Potential designs or constraints that impact the implementation of the requirement

The design information can be newly defined or used from an existing asset. The solution should address the original need, no more or no less, and be as simple as possible to resolve the need.

The tester also uses the requirements. Requirements identify what should be tested. Their testing confirms that the implementation meets the original stakeholder request. Functional verification testing is an important form of testing. Test cases link to requirements. With this traceability, a test team can determine test coverage (how many requirements without test cases). Additionally testers can capture how many tests have been executed for a given requirement. Some industries, such as the pharmaceutical industry, require that each requirement has one or more tests.

Test organizations might conduct different types of testing. Some organizations perform functional verification test (FVT) and system verification test (SVT). In other cases, a combination of FVT or SVT and JUnit tests are performed. Regardless of the testing process performed, the tests that are conducted will use requirements assets. The test plans that are created align the functions in the system that must be tested. If a team is adopting a user story or use-case driven approach, the scenario information that is captured helps identify test-case information. Test coverage is verified by reviewing the requirements. A test team can verify all function points, and scenarios are tested in a system.

A football analogy

Software development is a team sport. The business goal or objective in this case is to “win” or to “score more goals” than the other team. The acceptable requirements, then, should always support those goals. If not, the project manager throws them out.

Requests are the assets that are triaged, sized, and planned. The requirements add detail to a request. Requests are the “plays” that the solution team executes upon. Requirements tell what each player must do in that play (for example, run long, sweep right, or sweep left). The project manager is the “captain” who assesses the situation on the field. This person must evaluate the development situation. The project manager must coordinate and work with the team so that they are adaptive and able to address change as they encounter it. The project manager depends on the team to identify what items they can contain within a given iteration.

The development iteration is like the “football game” (referred to as “soccer” in the U.S.). The cadence of the iteration is like managing the “clock” on the field. The project manager and team must identify the length of the iteration and which requirements can be contained within a given iteration. The project manager and team work together to execute delivery of the appropriate requirements across several iterations. The successful delivery of a function within a given iteration allows an organization to meet the “end game” goals that are defined in the iteration plans. The end result is providing a solution that meets the stakeholders’ needs on time and on budget.

Continuing with the sports analogy, in team sports, everyone must work together and play their part to win. Development organizations are no different than a sports team. The development organization must understand the “vision” of the project and work together to execute that vision. The team must work together and understand actions that must be done to succeed. The team must use information that is available to them and effectively adapt to change, to avoid chaos and friction within the team. If everyone works on different requests, there is chaos. The team must work from the same set of stakeholder requests. The requirements provide detail, while the original request provides the context. The request is implemented and tested by using the requirements as validation and verification points.

The software development life cycle is like a football game. A football team is not steadfast in all the plays it will execute up front. A set of potential plays are identified before the game. However as the game evolves different plays are executed depending upon the situation. Plays are adjusted as the game unfolds.

Development organizations are similar to a “football team.” They must be effective to adapt to changing needs of their stakeholders in the form of requirements by using a change request. Additionally, development organizations must be adept to the changing industry and changing project. Development teams that practice iterative development, agile techniques, or both are effective at adapting to changing needs. They break the development life cycle into measurable chunks or iterations, much like a football game is broken into halves. The development team is adaptive to change in that they manage requests into various iterations. Additionally requirements are adapted as more information is gleaned.

Organizations that are more waterfall in nature are often bound by decisions made early in the project. These organizations are not as effective to easily address change as the project unfolds. When viewed from the perspective of this sports analogy, you can make the association that waterfall requirements analysis is similar to a coach selecting a set of plays before the game and sticking to them. Changing the game plan is more measured in that the requirement change is controlled through a change request process.

4.1.3 Collaborative development blueprint and change management

Collaborative development is a core component of the total ALM blueprint for software development and delivery. See 2.1.2, “A CALM blueprint to streamline software delivery” on page 24. A collaborative development blueprint, as seen in Figure 4-3 on page 89, is provided to define the key areas of this space. Support for change management is a key component of collaborative development as it impacts iteration planning, work management, and team health and transparency. In this section, we discuss the following parts of the collaborative development blueprint:

- Iteration planning and work item management
- Team health, transparency, and collaboration

The rest of the collaborative development blueprint is covered later in this book. In this section, we discuss only those aspects that are related to change management. The source code management part, and further details about work management and team health, of the collaborative development blueprint are discussed in 6.1.2, “Collaborative development blueprint” on page 217.

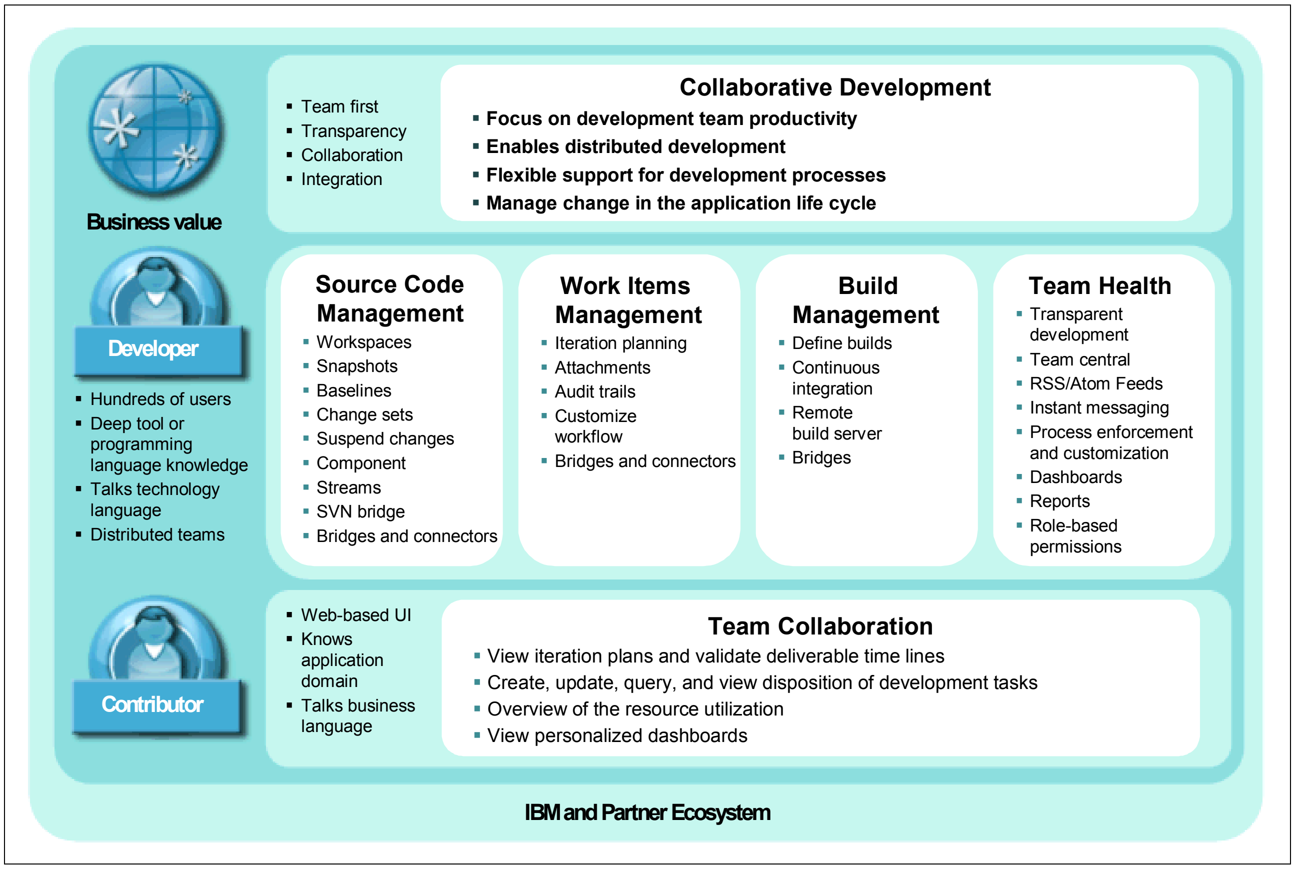


Figure 4-3 Change management blueprint

Work management and iteration planning

Development projects that are adopting an iterative development approach organize the project into a sequence of phases or iterations. For example, the open source OpenUP defines four phases: Inception, Elaboration, Construction, and Transition. Each of these phases has one or more iterations.

OpenUP: To learn more about OpenUP, see the following Web address:

`http://epf.eclipse.org/wikis/openup/`

Iterations focus the software development team on delivering and demonstrating working code at frequent iteration milestones. The number of weeks that are assigned to each time-boxed iteration can vary depending on the project size. Typically iterations are four weeks, although some teams find success by using two-week iterations, while other teams stretch to six-week iterations. The goal of having short iterations is to create an environment where the team is focused on high-value work with no room for non-essential work or bureaucracy. Using short iterations creates a healthy team focus, ensures the work is of value to the solution stakeholders, and reduces risk by demonstrating working code at the end of every iteration.

While the length of each iteration is fixed, the scope is not. This means that work will be pushed off the iteration stack and replanned to the next iteration. This fluidity of scope is a core part of the change management workflow as supported by the blueprint.

Iteration planning, estimation, and progress tracking are centered on items of work. An iteration plan is created by fitting top-priority work items within the time-boxed iteration. Work items can be organized by themes, plan items, and stories. Change requests are handled as work items. Each request represents a work item that must be triaged by the team. Estimation techniques are used to plan the stack of work items, but the estimates have an acceptable uncertainty due to the lack of detail at the project and iteration inception. This iteration planning is repeated for each iteration within a release.

Practices in work management

In a fast paced iteration, there is little time for top-down assignment of work by using Gantt charts. Instead, when a team decides to take on a work item, the work is assigned to one or several team members. Each individual team member is responsible for managing their workload. They commit to a reasonable amount of work for the iteration and work with the team to resolve work overload. Hence, software development teams self-organize and determine the work that must be done. This work is reflected in an iteration plan that is directly tied to the teams work assignments.

The project manager uses the team performance from previous iterations to ensure that the team commits to a reasonable amount of work for the iteration. The transparency of real-time information about the iteration plan, project health, and team member load is a key aspect to the success of self-organizing teams and is supported by the collaborative development blueprint.

Just as a project goes through a life cycle, iterations also go through a life cycle but with a different focus for work management depending on whether the iteration is in the first versus the last week of the iteration. Consider the following examples:

- Iteration planning

  The team starts each iteration by collectively selecting and organizing the work for the iteration.

- Weekly builds

  The team strives to stabilize the changes, so that weekly integrations can be delivered to the software supply chain, that is, the solution integration.

- Stabilizing the iteration release

  The team applies a stricter change control as the iteration release is stabilized.

- Iteration retrospective

  The team learns from the iteration, and the plan process changes for the next iteration.

The collaborative development blueprint highlights the need for the workflow customization and process enactments to be an integrated capability in work management and iteration planning.

Managing change requests

All work begins with some form of request. The request describes a need identified by any stakeholder on the project. A request is designed to allow anyone to submit petitions to the team, such as a request for a new feature (enhancement) or a defect report. However, requests are not limited to just features or defects. A request can come in any type defined by the project team by using process customization. For example, a project lead might receive a request to find additional members for the development team.

Stakeholders periodically check the status of the request and respond to any questions or comments from the team as the request is triaged, prioritized, planned, estimated, and delivered. The tight collaboration between stakeholders and development team is key in collaborative development.

The categorization of requests provides an indication of which team is impacted. That team reviews the request and determines if and when it can be addressed. If the request will be addressed in a particular project, a work item is created for that project and associated with the request, as discussed in the next section. Anyone who is looking at a request can see which work items are needed to fulfill it. Anyone who is looking at a work item can see which request it is intended to complete.

A request has a defined life cycle that governs its workflow. For example, a request can be approved, withdrawn, or rejected. When in the rejected or withdrawn state, stakeholders can decide to re-open the request. The Rational ALM solution, which implements the collaborative development blueprint, uses various repository objects to realize change management. In the ClearQuest ALM solution, a new record type, called *ALMRequest*, maps to a request. Also, a family of record types, called *ALMTasks* and *ALMActivity*, map to work items. The Rational Team Concert environment uses “work item.”

Assigning and aligning work

Any one can submit a request. However, not all requests are completed. Each request can drive work items for one or more team members. Understanding the amount of work that is involved is essential for successful iteration planning. For example, a defect impacts a developer and tester, while a new feature can impact an analyst, architect, one or more developers, the release engineer, and one or more testers. The ability to trace work items back to the originating requests helps the development team understand when all of the work related to a single request is complete. This is particularly important when the impacted team members are using different tools to manage their work.

The request is purposely defined outside of the context of a project. After all, requests come in many forms. Some requests apply to the current solution under development, while others apply to the version in production, and others seek enhancement to a future release.

Work items, however, are assigned to each of the impacted team members in the context of a project. A request that is found in a previous version of the software might be addressed in the project that is currently under development. In some cases, many tasks might be required to complete a single request. Alternatively, a single request might be addressed in more than one project. Resolving a request in most cases involves multiple disciplines. For example, the work needed to satisfy the request might impact the development and test teams. Therefore, work items that are created for both development and test are added to the iteration plan for the current project.

As noted previously, work items are represented in the ClearQuest ALM solution as task and activity records. When using the ClearQuest ALM solution, a *task* is an artifact that commits a team to act on a request within the context of a specific project. A task commonly requires more than one person to complete all of the work involved.

Therefore, *activities* collectively complete a task. Multiple activities can be associated with a task. A testing task provides a good example. A single test task can involve updating the test plan, creating test cases, creating test scripts, and running the tests. Alternatively, the task to implement an enhancement highlights the need to assign activities to across disciplines, such as assigning work for design, implementation, and testing.

In Rational Team Concert, a work item can have parent/child relationships to other work items to link work items. For example, a work item with child work items is similar to a ClearQuest ALM task with child activities.

The ability to track what work is assigned and completed for each request helps the project teams assess their progress and determine if an additional request can be absorbed into an iteration.

Assessing results

While the request describes the identified need and is owned by any stakeholder on the project, work items track the work that is assigned and must be completed in order to resolve the request. This allows you to compare the number of requests to the number of committed work. This also allows project managers to see how much work is involved in each request and how much of the work is completed or outstanding.

Additionally, there are times when a single request may impact more than one project, or iteration. This separation of requests for work and the commitment to do work allows each impacted project to manage the work as it relates to satisfying the request in that specific project context.

The capability to align work, as part of iteration planning and tracking, is essential as the completion of a request. Hence a request is not completed until all work, in the form of tasks and activities, or work items, are complete. The team is done when all work related to the committed requests is complete.

Work can be assigned to iterations of the project. This allows the project team to balance their workload across iterations. Health metrics, in the form of dashboards, team centrals, reports, or queries, can be created to see how the work is distributed across the team. This insight helps the project manager to spread the workload evenly across the team members and avoid critical path situations. Charts also help the project manager ensure that all work is assigned.

Team health, transparency, and collaboration

Transparency is a fundamental quality for team adopting Agility at Scale. Team health and team transparency empower the organization with self-direction in order to respond effectively and rapidly to project challenges and changing project needs. A self-organizing team has the authority to configure the work that it will perform and the responsibility to do that work in the way that it chooses.

The capabilities of team health and team transparency enable teams to assess iteration status, workload balance, and over commitments. Team health and team transparency enable team members to act on work items as needed by the team, rather than only have work assigned to them by their manager. The work management, as discussed in the previous section, that provides triage and direction still remains essential to ensure that the project remains focused on delivery commitments, but with transparency and self-direction.

The Rational CALM solution for implementing the collaborative development blueprint uses various platform capabilities to realize team health and team transparency.

Rational Team Concert provides team health and team transparency in a completely transparent team environment based on the collaborative capabilities in the IBM Jazz products. Team members can use Iteration Plans, the Team Central view, MyWork view, RSS Feeds, Chats, Notifications, Dashboards, and extensive querying and reporting capabilities. In the Rational ClearQuest CALM solution, team health and team transparency are provided by using in-context collaboration, notifications, reporting, and queries.

4.1.4 Requirements definition and management blueprint

To address the needs of the market, Figure 4-4 illustrates the IBM Rational requirements definition and management blueprint. In this blueprint, the requirements effort revolves around the need. A need is defined in the form of a request. By understanding the need expressed by a stakeholder and the business problem, a solution team can implement a solution to address the need.

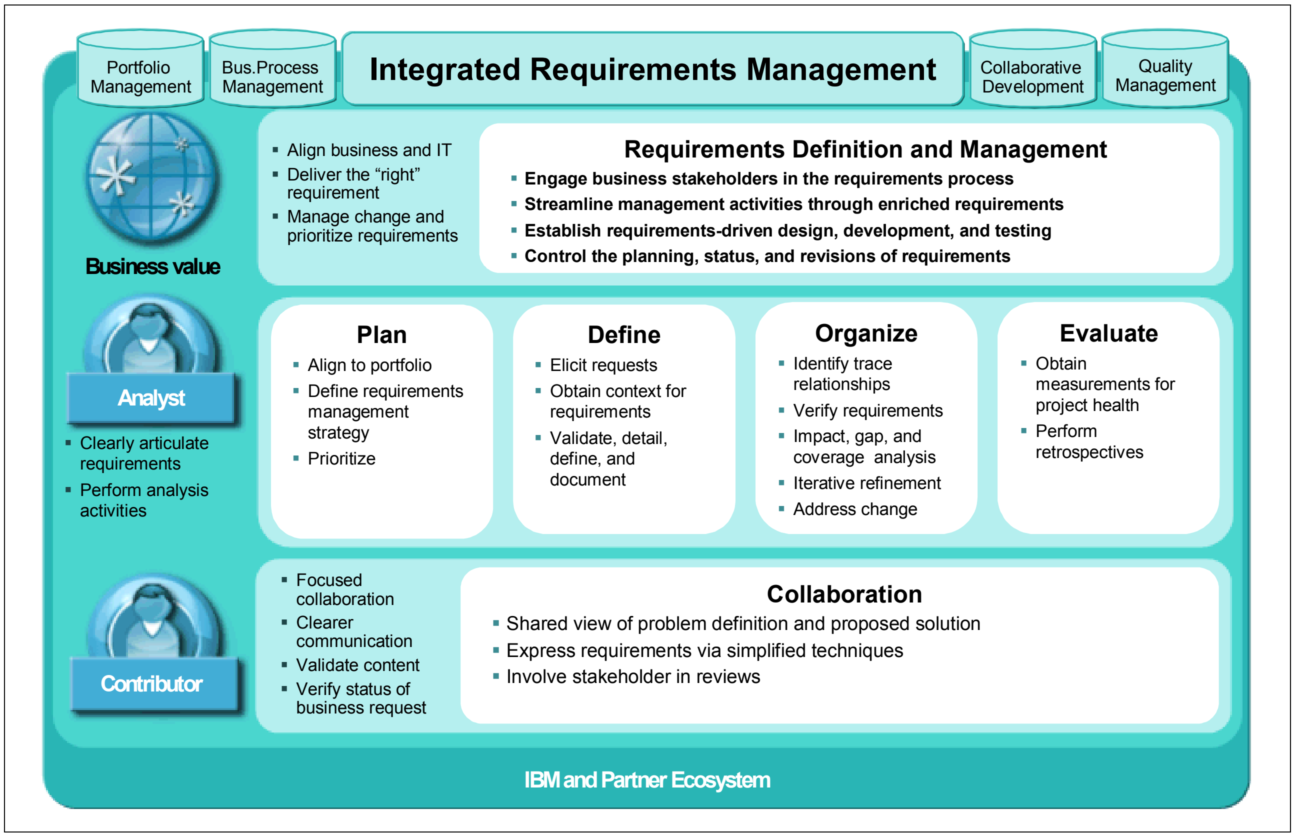

Portfolio Management
Bus.Process Management
Integrated Requirements Management
Collaborative Development
Quality Management
Business value
Align business and IT
Deliver the "right" requirement
Manage change and prioritize requirements
Requirements Definition and Management
Engage business stakeholders in the requirements process
Streamline management activities through enriched requirements
Establish requirements-driven design, development, and testing
Control the planning, status, and revisions of requirements
Analyst
Clearly articulate requirements
Perform analysis activities
Plan
Align to portfolio
Define requirements management strategy
Prioritize
Define
Elicit requests
Obtain context for requirements
Validate, detail, define, and document
Organize
Identify trace relationships
Verify requirements
Impact, gap, and coverage analysis
Iterative refinement
Address change
Evaluate
Obtain measurements for project health
Perform retrospectives
Contributor
Focused collaboration
Clearer communication
Validate content
Verify status of business request
Collaboration
Shared view of problem definition and proposed solution
Express requirements via simplified techniques
Involve stakeholder in reviews
IBM and Partner Ecosystem

Figure 4-4 Requirements definition and management blueprint

The requirements definition focuses on the elicitation techniques that are used for capturing high level requirements and how more detailed requirements are derived from those statements. Requirements management focuses on how requirements content is used throughout the software development process. Requirements assets are organized in a central repository and leveraged by the solution team throughout the CALM life cycle.

Requirements definition and management includes the following components: manage, plan, define, organize, and evaluate. A CALM solution should allow for a repository to define, manage, and organize the requirements effort. See Figure 4-4 for detail.

- The product owner and team can review and ensure that the stakeholders' need is resolved by an implemented requirement.
- Analysts and stakeholders have the ability to obtain a better understanding of the business through the capture of storyboards and process models. This information supplements the need and provides context.
- The requirements become drivers for project planning activities.
- By having all of the requirement assets at their fingertips, the solution team can leverage the requirements.

- The development, test, and documentation organizations leverage requirements content as a foundation to support architecture and development, test, and documentation activities.

Requirements definition and management

Integrated requirements management starts with requirements definition and management, which coordinates the disciplines of requirements planning, definition, organization, and evaluation. An integrated requirements management system provides a repository for managing and organizing the requirements effort. By placing all assets in the requirements management system, the development team has a clearer indicator of their progress.

Requirements impact the rest of the disciplines in software development. Requirements are linked to requirements plans, sketches, and storyboards, and business process sketches help to articulate the details of the requirement included in the plan. They are organized and traced in the context of the plan. Plans are baselined to capture their content at a particular point in time. Requirements are implemented and tested. Defects are reported and linked back to the requirements. Reports become much easier to create and manage.

Plan

As mentioned earlier, requirements are an important part of the project planning activities. Planning activities can be separated into two areas:

- Alignment to a portfolio or business goals
- Defining a requirements management strategy

Both of these planning activities use requirements as inputs to planning decisions. However, the activities performed within them are different. All aspects of project planning is not discussed here. For more information, see “Assigning and aligning work” on page 91.

Aligning to the portfolio

Alignment to a portfolio simply identifies the agreed upon list of needs that will be fulfilled within a given release. Planning activities can vary for organizations depending on the software development process that is followed. Just as shoes come in many sizes, so do organizations, and their planning processes can also vary. In this section, we present some examples for organizations to consider for planning.

Large enterprise organizations can use a program office to help review and filter requests and identify a list of needs that align with a yearly strategy. Only these requests are directed to development organizations to implement. This organization uses a top-down approach.

An agile organization is different. The team works with the project manager to define the planning activities as mentioned in “Work management and iteration planning” on page 89. Such organizations use a bottom-up approach. In these organizations, leads or stand-up meeting masters serve as a team liaison. Prioritized requirements are reviewed and assigned to iterations.

Other organizations might combine agile and larger scale planning approaches. In this approach, requirements are aligned to an overall project plan for the solution. The project plan might identify four- to six-week iterations for the larger solution as a whole. In this example, smaller component teams might adopt agile approaches, follow two-week iterations, and deliver functions into the larger solution iteration.

Regardless of the organizational type, ALM tooling and processes should support portfolio planning activities. Here are some questions to consider for requirements planning:

- As a *stakeholder or customer*, you are concerned with the following information and anticipate that planning activities should provide this information:
  - What are the requirements or set of requirements that implement your request? That is, has it been considered in scope for an iteration?
  - What is the business priority of your request?
  - In which iteration or major release will your request implemented?
  - If your request has been considered in scope for an iteration, what is the requirement that will implement your request?
- As a *project lead*, you are concerned with the following information and anticipate that planning activities should provide this information:
  - What is the given status of the backlog? That is, is the request in the backlog considered in scope for an iteration?
  - Do any of the requests require reprioritization and validation with the customer or stakeholder for the iteration?
  - Which requirements must be removed from the current release to accommodate these higher priority requirements?
  - Do you need to obtain additional information from anyone to set context on the requirements?
  - Is the request mapping to themes for the iteration? Additionally, are requirements mapping to themes as well?
  - Do you need to work with anyone to address risks and define mitigation strategies?
  - Are any items blocking progress?
  - Who will collaborate from a pair-programming perspective to achieve continued progress?
  - What is the quality of the iteration thus far? That is, are the requirements that are implemented and coded passing?
  - Is there more work that needs to be completed?
- As a *team member*, you are concerned with the following information and anticipate that planning activities should provide this information:
  - What is the backlog of requests that must be reviewed so that your team can help provide guidance of what might be contained in an iteration?
  - What are the themes for the release so that you can help identify requests in the backlog that map to a theme?
  - What is the list of work items that you must complete and by when?
  - What additional information is available regarding this work item?
    - What is the request that initiated this work item?
    - What requirements information provides context for the request?
  - Do any items require your input?
  - Are there blocking issues that require mitigation in order to address a requirement for an iteration?

Defining the requirements management strategy

The requirements management strategy defines how requirements are managed for a given release, iteration, or project. During project startup, you must answer these questions: What are the factors that affect how requirements are managed, and what is the requirements process followed by an organization?

Organizations must identify basic process requirements for a project. These activities might entail agile development by nature or more rigorous methods if the organization aligns to Capability Maturity Model Index (CMMI) or International Organization for Standard (ISO) standards. The intention for process definition is to identify just enough process so that team members and stakeholders understand and agree upon the information that is conveyed for requirements in a project. If a team is following the RUP, this information is typically captured in a Requirements Management Plan.

Tip: You can find examples of the usage of Requirements Management Plans in Rational Method Composer.

The following questions are a subset of probable questions to ask at the beginning of a project:

- What is the requirement process that will be followed in the project, because this decision will impact other decisions?
  - Will the team follow iterative, agile, or waterfall development practices?
  - What are the implications of the process choice?

    Depending on the type of process that is followed, the kind of requirements captured, project planning, reviews are all impacted.
- What are the pertinent types of requirements that must be captured in the projects?
- What kind of information is expected from a stakeholder and in what form? For example, should requirements be defined in the form of a need statement, user story, and so on?
- What are pertinent categories for further refining requirements?
- Are any traceability relationships required between requirements or other assets across the software development life cycle?
- How will progress, that is the implementation of the requirements, be measured?
- What review mechanisms will be used for validation of requirements, and what verification or acceptance criteria might occur?
- Are deliverables required for the project?

Prioritizing

Identifying the importance of requirements is key to the requirements process. Prioritization of the requirements aids in project planning activities to be completed in later portions of the software development life cycle. Prioritization helps a solution team assess which functional elements must be in a release, where other functions might be nice to have.

Define

As mentioned earlier, understanding the needs of stakeholders and their business process is critical in the requirements process. Different types of activities can be performed for elicitation and definition. A CALM solution should support these activities.

Eliciting requests

Requests allow a development organization to understand stakeholder needs. They are a fundamental piece of the requirements process because they provide an understanding of the business problem that must be solved.

Obtaining context for requirements

The requirements definition process has the following fundamental components as illustrated in Figure 4-5:

- Requests
- Rich text documents
- Business processes
- Business objectives
- Glossary
- Prototypes
- Storyboards and application sketches
- Text-to-visual transformation
- Use cases

Although not all of these components come into play on every project, a combination of them makes sense for all projects.

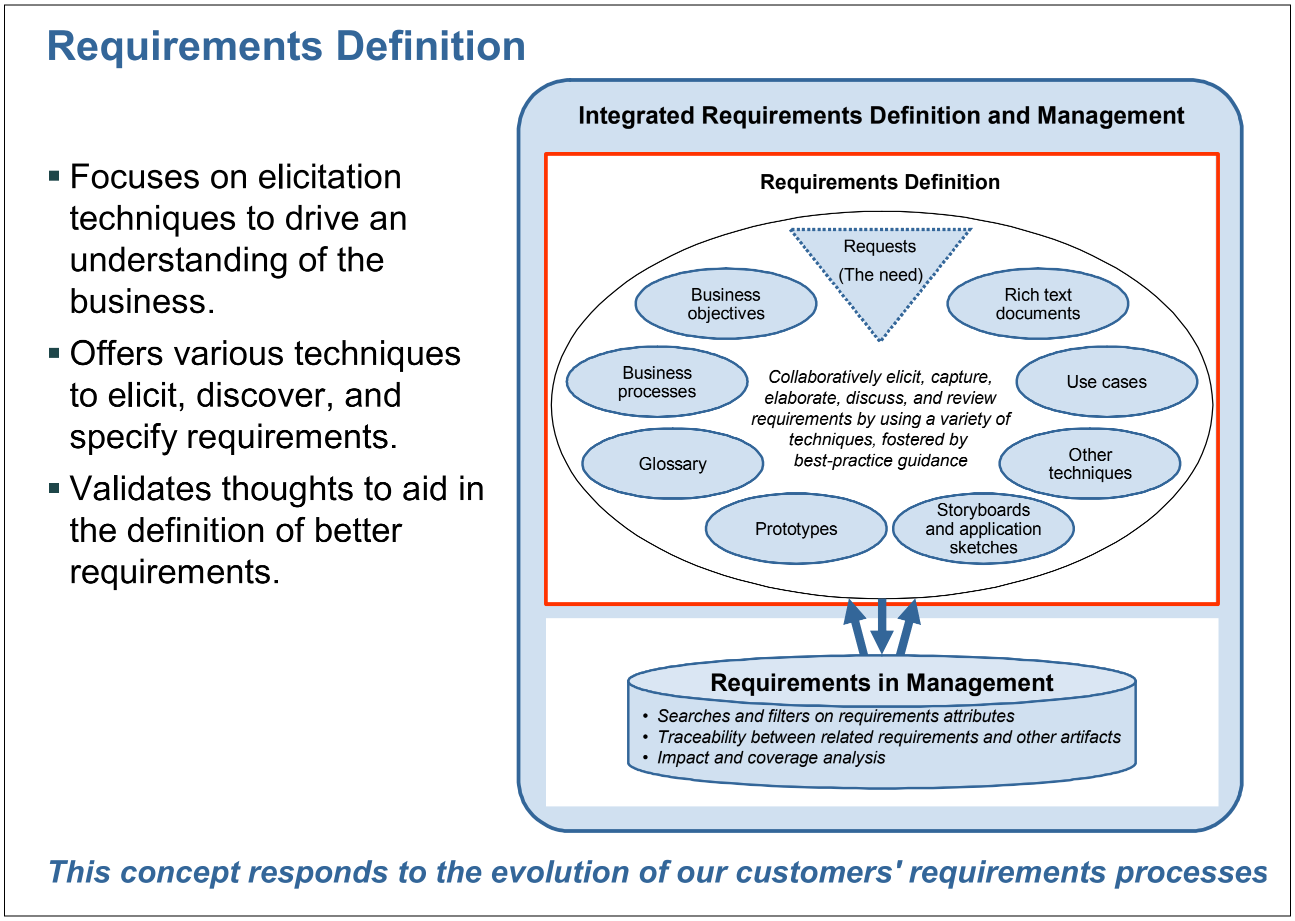

Requirements Definition
Focuses on elicitation techniques to drive an understanding of the business.
Offers various techniques to elicit, discover, and specify requirements.
Validates thoughts to aid in the definition of better requirements.
Integrated Requirements Definition and Management
Requirements Definition
Requests (The need)
Business objectives
Rich text documents
Business processes
Use cases
Collaboratively elicit, capture, elaborate, discuss, and review requirements by using a variety of techniques, fostered by best-practice guidance
Glossary
Other techniques
Prototypes
Storyboards and application sketches
Requirements in Management
Searches and filters on requirements attributes
Traceability between related requirements and other artifacts
Impact and coverage analysis
This concept responds to the evolution of our customers' requirements processes

Figure 4-5 Integrated requirements definition and management - Requirements definition

The fundamental components to the requirements definition process are explained as follows:

- Rich text documents

  Rich text documents can be used throughout the requirements elicitation process. For example, an analyst might conduct interviews to obtain a better understanding of the customer's business. Information obtained from the interviews is key to providing context for defining requirements.

- Business processes

  Understanding the business is a key element to the requirements process. A good understanding of *business processes* is important for building the right systems. Capturing a current or "as-is" business process and later evaluating it to define a future or "to-be" business process allows teams to define requirements that support the "to-be" process. The "to-be" process, in turn, supports the business goals and objectives. More value is added to the solution if information is captured about people's roles and responsibilities, as well as definitions of what items are handled by the business as a basis for building the system.

- Business objectives

  A *business objective* is a need that must be satisfied by the business. Business objectives describe the desired value and are used to plan and manage the activities of the business. All requirements should support at least one business objective.

- Glossary

  Creating a *glossary* and capturing terms helps provide an understanding of industry specific terms and so on. These terms are important for the requirements process, so that all stakeholders involved in the development process can understand the relevant terms.

- Prototypes

  Prototypes provide a working example to illustrate areas of design, features, and so on to gather early feedback from stakeholders.

- Storyboards and application sketches

  Storyboards are a sequence of sketches or wire frames that define scenarios based on a series of tasks. They allow for multiple threads and depend on the path that the user takes through the system. Application sketches provide a high level view of a specific scenario. Both provide a visual context of a GUI.

- Other techniques

  The capture of written process descriptions and transference into business flow diagrams is helpful in the requirements process. Textual transformation to a process flow diagram provides a visual representation of the information.

- Use case

  A *use case* is a sequence of events that provides an observable result to an actor. Use cases are helpful because they provide a "story" or scenario. They capture the functional requirements of a system.

- Further detail requirements

  This information is derived from a stakeholder's needs. Often times more detailed requirements, such as business rules and nonfunctional requirements, provide pertinent supplemental detail that supports higher-level requirements defined earlier in the requirements definition and management process.

Validating, detailing, defining, and documenting

Request information and other business details are validated by the solution team to ensure that they understand the stakeholder's business. After this understanding is validated, analysts might detail and further refine requirements to be used by others in the application lifecycle. As part of the validation process, an analyst may ascertain if additional information must be captured in order to accommodate the request.

Considerations

The following questions and comments are a subset of probable questions and comments to consider as a result of the definition process:

- Are the need statement and business problem understood? If they are not understood, what kind of elicitation and definition activities might occur to drive an understanding of the business (sketches, storyboards, and so on).
- Has the request been validated? Is there an agreement on the priority of the request?
- Have supplemental detailed requirements been captured to augment higher level requirements as needed?

Organize

Often the focus in requirements processes in the past has been on the management aspects of requirements. Requirements are captured, organized, and traced to requirements and other assets in the software development life cycle. Traceability relationships often have become behavioral where teams define relationships just to have them. This type of activity, if not measured, can become burdensome to some organizations and not necessarily provide value. The true intention of setting traceability is to provide an understanding of requirement relationships and potential impacts for change.

Bob, the analyst manager, defines the following feature-level requirements that have other requirements that are traced to them:

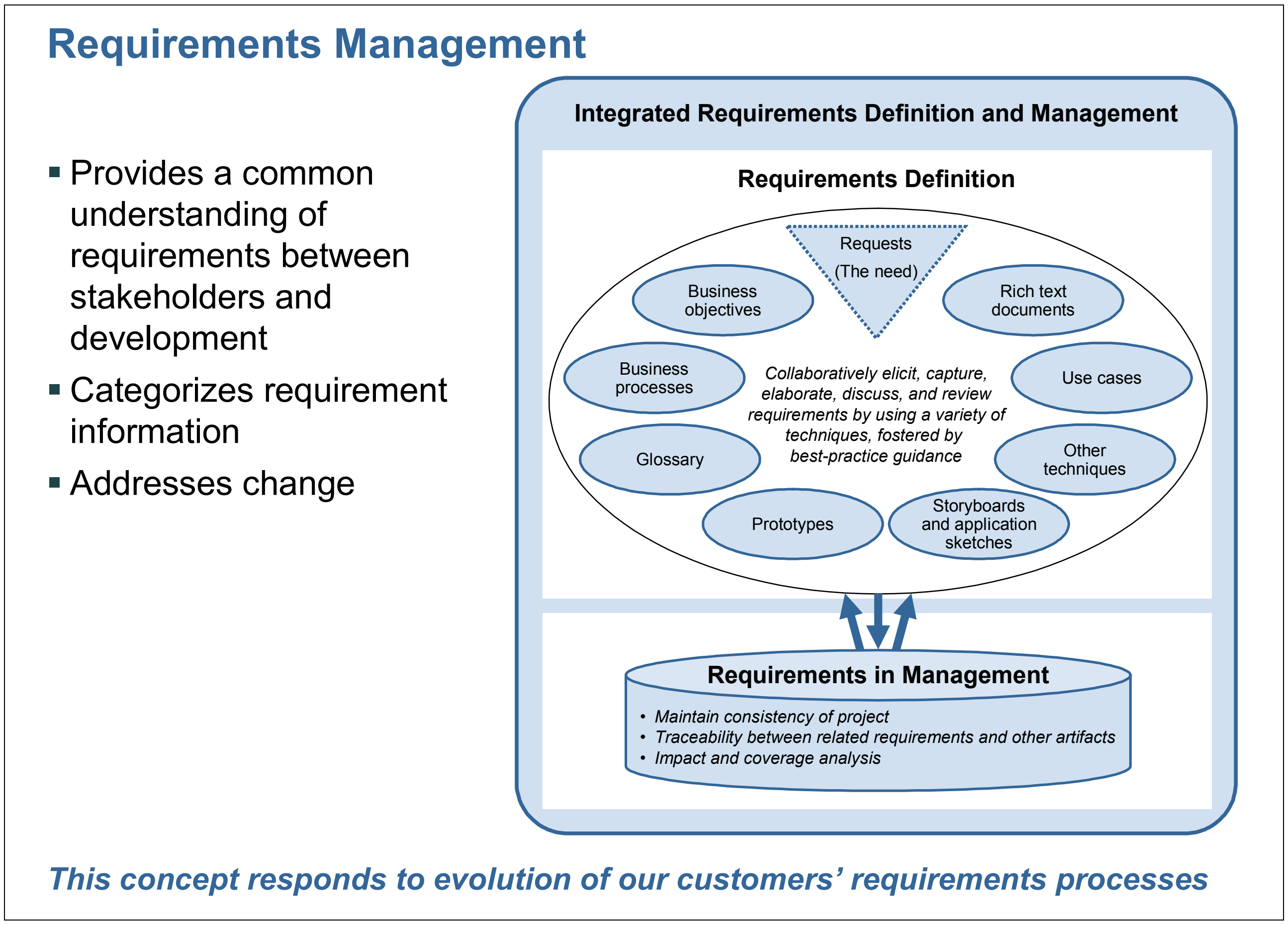
- 1...n use cases that development will implement, which trace back to the feature
- 1...n test cases that test teams will validate

Any change at the feature level creates a suspect link down the chain, so that development and test can react to the change and update assets. *Suspect links* identify requirement trace relationships that are impacted from a requirement change. If a requirement changes on either side of the trace, the relationship is marked suspect.

It is not enough just to have traceability without a defined process to monitor and address the change. Otherwise, the trace relationships and change impacts are not worth much. Teams must build a process around trace relationships that are identified. Stakeholders downstream must monitor for change and react.

Other types of organizational approaches entail capturing many types or kinds of requirements. Details can be depicted through technical decomposition of requirements. As mentioned in 4.1.4, "Requirements definition and management blueprint" on page 93, organizations should identify enough detail for the kinds of requirements that are captured, the trace relationships that are set to provide value to the stakeholder, and other persons who are involved in the requirements process. Organizations that adopt agile and lean approaches focus only on those activities that provide value to the customer. If traceability relationships do not provide value to the customer, these teams might choose to implement minimal traceability. A CALM solution should be flexible enough to support detailed process-centric approaches to organizing requirements to more Agility-at-Scale-oriented processes.

In the sections that follow, we describe different activities that occur during the management of requirements. These activities help in the refinement of requirements as a solution team has a better understanding of requirements. The management activities help provide a clearer perspective of the objective. The intention to perform these activities is to make the requirements more consumable across the software development life cycle. These components, as illustrated in Figure 4-6, describe management activities and include attributes, traceability, and impact and coverage analysis.


Requirements Management

- Provides a common understanding of requirements between stakeholders and development
- Categorizes requirement information
- Addresses change

This concept responds to evolution of our customers' requirements processes

Integrated Requirements Definition and Management
Requirements Definition
Requests
(The need)
Business objectives
Rich text documents
Business processes
Use cases
Collaboratively elicit, capture, elaborate, discuss, and review requirements by using a variety of techniques, fostered by best-practice guidance
Glossary
Other techniques
Prototypes
Storyboards and application sketches
Requirements in Management
• Maintain consistency of project
• Traceability between related requirements and other artifacts
• Impact and coverage analysis

Figure 4-6 Integrated requirements definition and management - Requirements management

Identifying trace relationships

As the software development life cycle proceeds, more information is obtained about requirements. As a better understanding of the problem to be solved is gained, how the problem will be solved and additional information for the requirement can be captured. Some of this information might include complexity, iteration or release, and other elements. Attributes provide a mechanism to categorize requirements. Categorization of requirements aid in project planning activities and help provide pertinent details of requirements to the solution team.

Traceability is also helpful in organizing requirements. It identifies pertinent dependency relationships between requirements across the software development life cycle. Traceability also aids in describing how requirements are related to other assets across the software development life cycle including test, design, and so on. The main goal for setting traceability relationships is identify which requirements are related to one another and to perform accurate change impact analysis. Trace relationships between requirements help identify the origin of any requirements; from a design perspective, the elements that are realized from higher level requirements; other assets that leverage requirements (that is, test assets); and the changes or defects that impact requirements.

Verifying requirements and impact, gap, or coverage analysis

Verification activities are crucial in requirements management. These activities help teams ensure quality. Additionally, the team can verify that the software fulfills requirements, and only those requirements expressed.

Impact, gap, and coverage analysis

Traceability is also helpful with requirements change management as it identifies how requirement changes can impact a related requirement or requirements. These changes can impact the software development life cycle, such as the resources, schedule, and so on. Impact analysis is helpful when reviewing relationships between requirements. A CALM solution should allow a team to easily identify what has changed and anticipate the impact of that change. For example, in Figure 4-7, if the stakeholder request changes, it might have an impact on the feature that is traced to it.

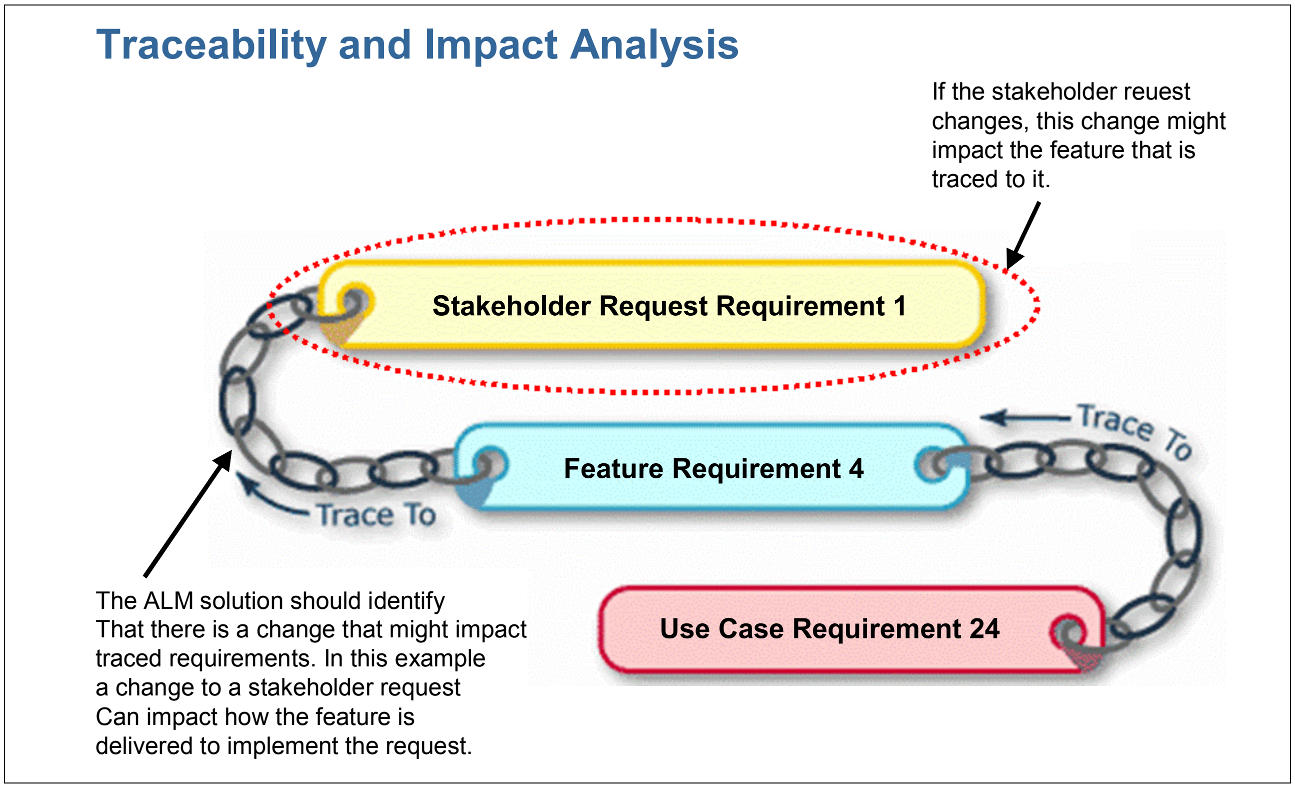

Traceability and Impact Analysis
If the stakeholder reuest changes, this change might impact the feature that is traced to it.
Stakeholder Request Requirement 1
Trace To
Feature Requirement 4
Trace To
Use Case Requirement 24
The ALM solution should identify That there is a change that might impact traced requirements. In this example a change to a stakeholder request Can impact how the feature is delivered to implement the request.

Figure 4-7 Traceability and impact analysis

Gap analysis information is also helpful as team members are able to discover missing relationships where anticipated. If gaps exist, there is potential scope creep. In the example in Figure 4-8, there is a feature requirement that is not traced to a stakeholder request. It is more important than ever to control scope creep with lean development and limited resources to deliver solutions.

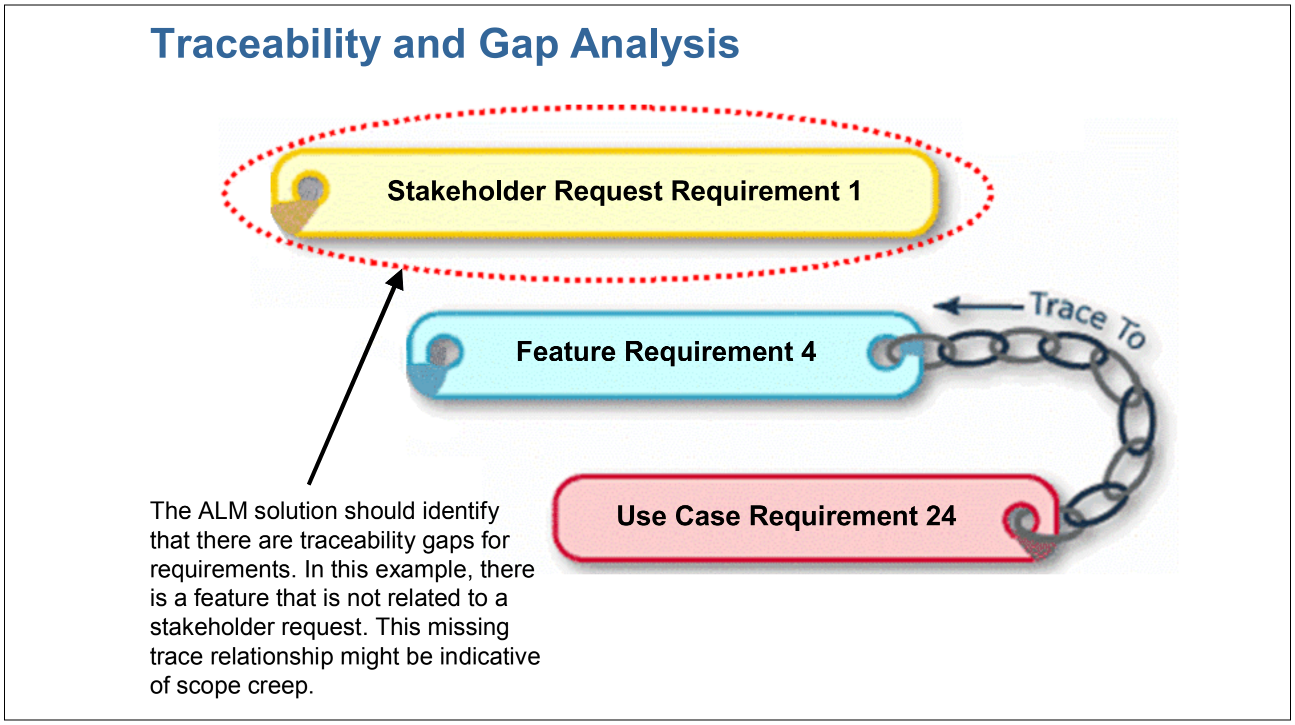

Traceability and Gap Analysis
Stakeholder Request Requirement 1
Trace To
Feature Requirement 4
Use Case Requirement 24
The ALM solution should identify that there are traceability gaps for requirements. In this example, there is a feature that is not related to a stakeholder request. This missing trace relationship might be indicative of scope creep.

Figure 4-8 Traceability and gap analysis

Coverage analysis is also helpful for teams. Information should be usable across the software development life cycle. Coverage analysis identifies how assets are related to one another across disciplines. For example, in Figure 4-9, the feature requirements are leveraged in a test plan, identifying the function that will be tested.

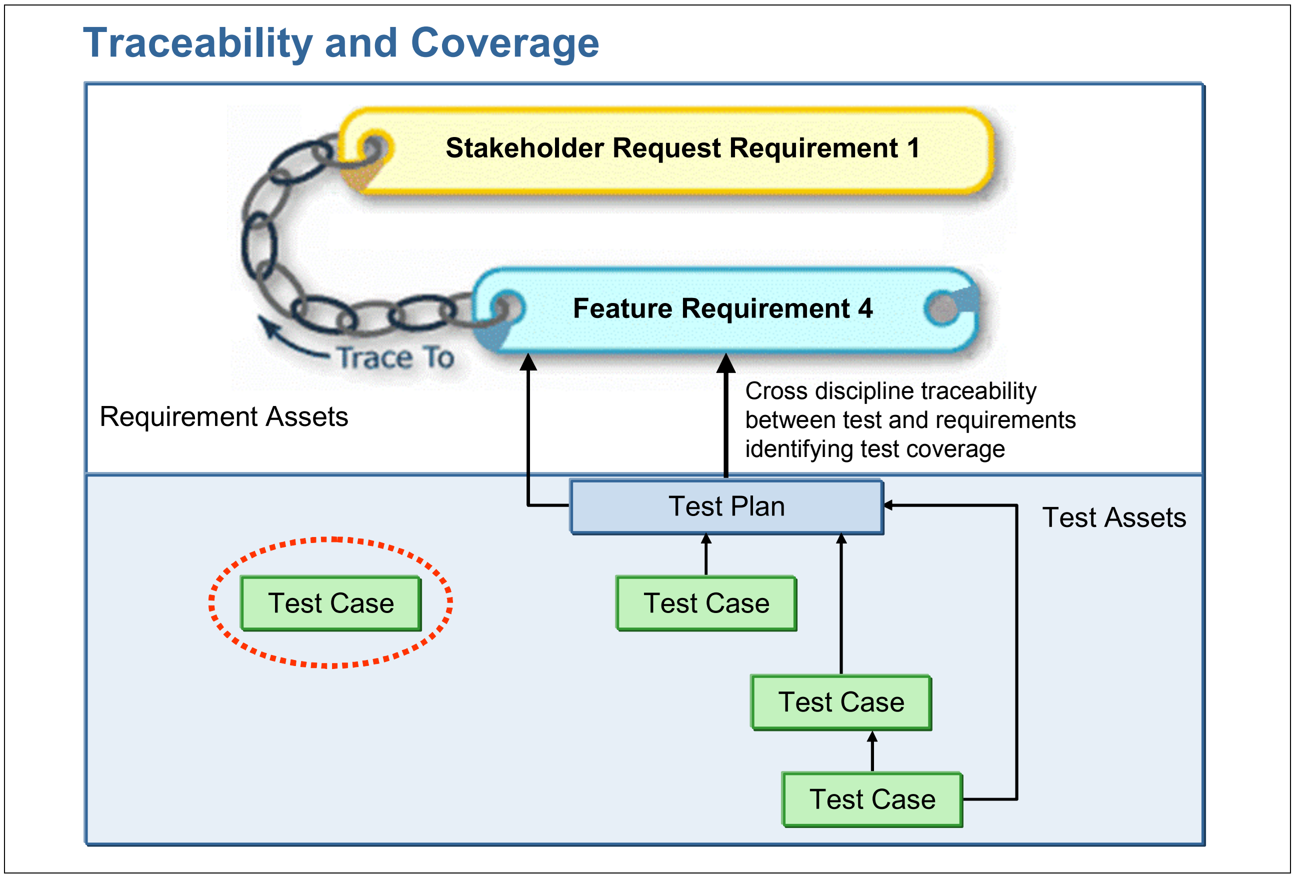

Traceability and Coverage
Stakeholder Request Requirement 1
Feature Requirement 4
Trace To
Requirement Assets
Cross discipline traceability between test and requirements identifying test coverage
Test Plan
Test Assets
Test Case
Test Case
Test Case
Test Case

Figure 4-9 Traceability and coverage analysis

Iterative refinement

As the solution team gains further understanding of how to build a solution, the requirements will become more detailed. Abstraction of the requirements naturally occurs as the context for the requirements is obtained.

Accommodating change

Requirements definition and management is impacted by change, just like other disciplines. Change is inevitable in the requirements life cycle. It is important to note how teams address change and incorporate it into their requirements process. This section highlights information about change management and its relationship with requirements. Organizations must identify a change management strategy for requirements and ensure that the tooling that they use supports those changes.

Change requests should be reviewed just like any other type of request. A *change request* is something that changes the scope for requirements after the scope has been agreed upon. The change request must be reviewed to assess if it makes sense to incorporate the change. When reviewing the change request, it is important to determine the type of request. The kind of change request might have a different impact upon the requirements process. If the change is simply a "document" change for a type of formal deliverable, the process to implement the change is likely less stringent.

Another type of change can be a content type of change. A *content change* is something that can simply change the language that is used to express a requirement. A potential example is the requirement text must be rephrased to be better understood. For example, the original requirement indicates “provide a fast search mechanism.” The rephrased requirement states “provide the ability to search on customer information and provide a result set within 5 seconds.”

The last type of change is a functional change. A *functional change* is something that impacts the scope of requirement content to be delivered in a solution. The functional change might entail the addition of a new requirement or removal of a requirement to an iteration. Often times when changes are incorporated, concessions must be made in order to include higher priority requirements and descope lower priority requirements.

Considerations

The following questions and comments are a subset of probable questions and comments to consider as a result of the organization process:

- Categorize requirements to help in project planning activities and to help provide supplemental information about the requirement. Consider the type of metrics that must be captured about requirements for planning activities.
- Depending on the requirements process followed, identify key trace relationships for different types of requirements.
- Use traceability as needed as part of the verification process to ensure implemented requirement resolve original need statement.
- Use impact analysis to help define how requirement change may impact other requirements, or potentially project planning activities.
- Identify requirement gaps to ensure that neither of the following conditions exist:
  - Missing requirements where requirements should be defined
  - Scope creep, new requirements added late in life cycle without identifying the impacts

  Only requirements that address original need statements should be included.
- Perform coverage analysis for use requirements in other portions such as the test life cycle to ensure that requirements are tested.
- Identify a change request process and ensure that tools support the process. Obtain an understanding of which requirements will be impacted before implementing the change.

Evaluate

Identifying the results of a project is important to the requirements process. A solution team is considered effective if the correct solution is delivered on time and on budget. Performing this task is always a challenge, and there is always room for improvement. Lessons learned might be applied to project planning activities for future projects.

Obtaining measurements for project health

As part of the assessment process, *metrics capture* is important. Metrics capture is described in more detail in Chapter 14, “Delivering and measuring success in Application Lifecycle Management” on page 533. As part of the project startup or planning activities, organizations must identify the kind of measurements that are important in regard to the software development process.

Performing retrospectives

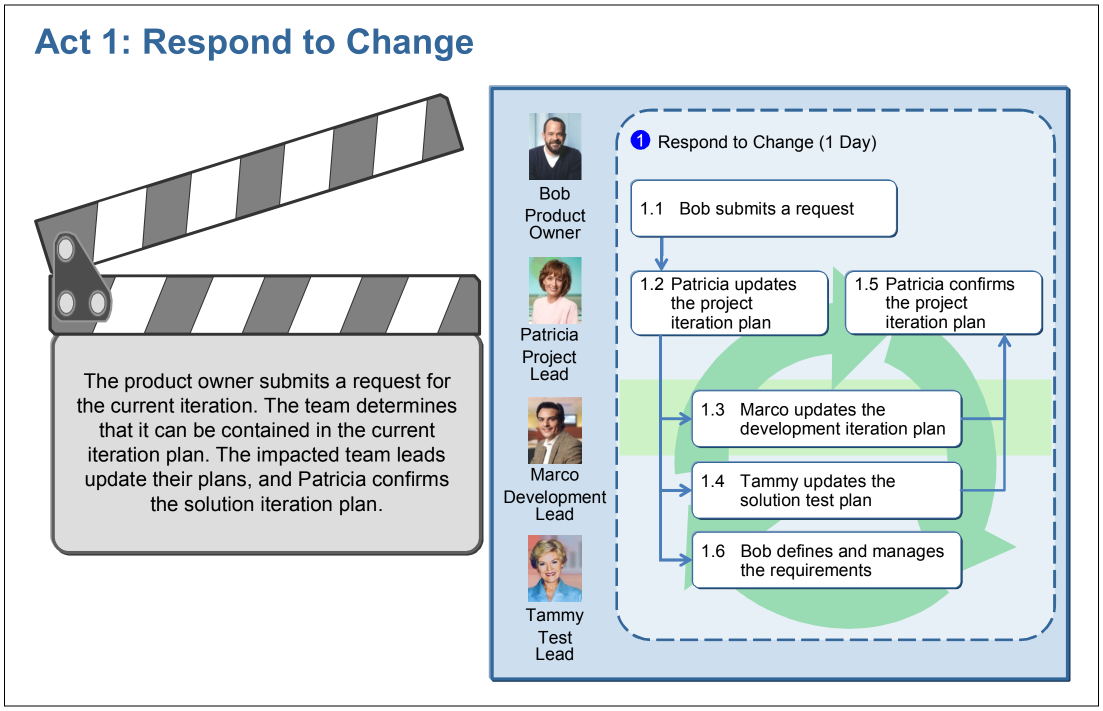
The same holds true for capturing information about project health. This type of information can also be used for retrospectives. Metric information and project health details can help project teams improve upon their requirements process, so that they are more effective.

4.2 A reference scenario for responding to a change request

In this section, we provide an overview of the steps taken by the project team to address change request. In Chapter 5, “Rational ClearQuest, Requirements Composer, and RequisitePro to manage stakeholder requests” on page 115, this workflow is demonstrated in the Rational products.

This scenario provides an introduction for the scenarios to come. The requirements that are captured here provide the building blocks for other acts in the book. Although there are many paths that the “Respond to Change” story may cover, the story described in the following pages is a simple one. A new request is entered in the system and is reviewed, prioritized, sized, and scoped to implement within a given iteration. The team executes against the iteration plans to address the need identified in the request.

Additional scenarios and best practices are covered in the 4.3, “Considerations in change management” on page 112. The scenario begins when Bob is notified of the release and reviews it for accuracy. Figure 4-10 shows a glimpse of Act 1: Responding to a change request.



Act 1: Respond to Change
The product owner submits a request for the current iteration. The team determines that it can be contained in the current iteration plan. The impacted team leads update their plans, and Patricia confirms the solution iteration plan.
Bob Product Owner
Patricia Project Lead
Marco Development Lead
Tammy Test Lead
1 Respond to Change (1 Day)
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.5 Patricia confirms the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements

Figure 4-10 Act 1 - The team responding to a new change request

The requirements definition and management story begins with the product owner. This scenario begins when Bob reviews the iteration plan and realizes that the user interface branding is not included in the iteration. A new request is entered for the UI branding.

This act includes the following scenes:

- Bob submits a request.
- Patricia updates the project iteration plan.
- Marco updates the development iteration plan.
- Tammy updates the solution test plan.
- Patricia confirms the project iteration plan.
- Bob defines and manages the requirements.

4.2.1 The actors

This scenario includes the several key actors as described in this section.

Bob is the name of the product owner. He cares about managing his business and bringing value to his users and stakeholders. As the business application owner, working in an agile environment, he works closely with the development team and has ownership over the requirements priorities. Additionally, he ensures that the request information has appropriate detail so that it can be implemented.

He wears more than one hat, however, in this story. He also performs analysis activities in the requirements process. As an analyst, he performs elicitation techniques such as application sketches business process definition, and storyboards, to provide context for requirements. He also refines and manages requirements as needed.

Patricia is the name of the project leader. She is responsible for coordinating the efforts of this team of teams. As each team produces its iteration plan, Patricia incorporates the plans into the overall project iteration plan. She works with Al to establish the pace of the iterations (how many iterations for the project and how long each will last). Her job ensures that all teams (development and solution test) are working toward the same iteration goals and that the work is aligned across the teams.

Al is the name of the solution architect. He works with the development and test leads to ensure that the team works from a consistent architectural approach. He provides insight into the iteration plans by identifying the architectural significant tasks for each iteration. He also seeks to reuse existing assets wherever possible to ensure consistency in approach and implementation.

Marco leads an agile development team. He understands the need to fit into Patricia's project plan, but seeks to maintain the agile approaches that have made his team successful. His team is self-organized and uses more frequent iterations than the rest of the project team. Marco still conducts daily stand-up meetings with his team and employs test-driven development techniques on his component.

Tammy is the name of the test lead. Her team is responsible for conducting solution testing, which includes functional, performance, and security testing at the solution level. The testing does not include JUnit or component level testing, which is the responsibility of each team. Tammy's team conducts solution testing as part of the iteration. They take a new solution build each week, thus providing an early feedback loop to the development team, when there are defects at the solution level.

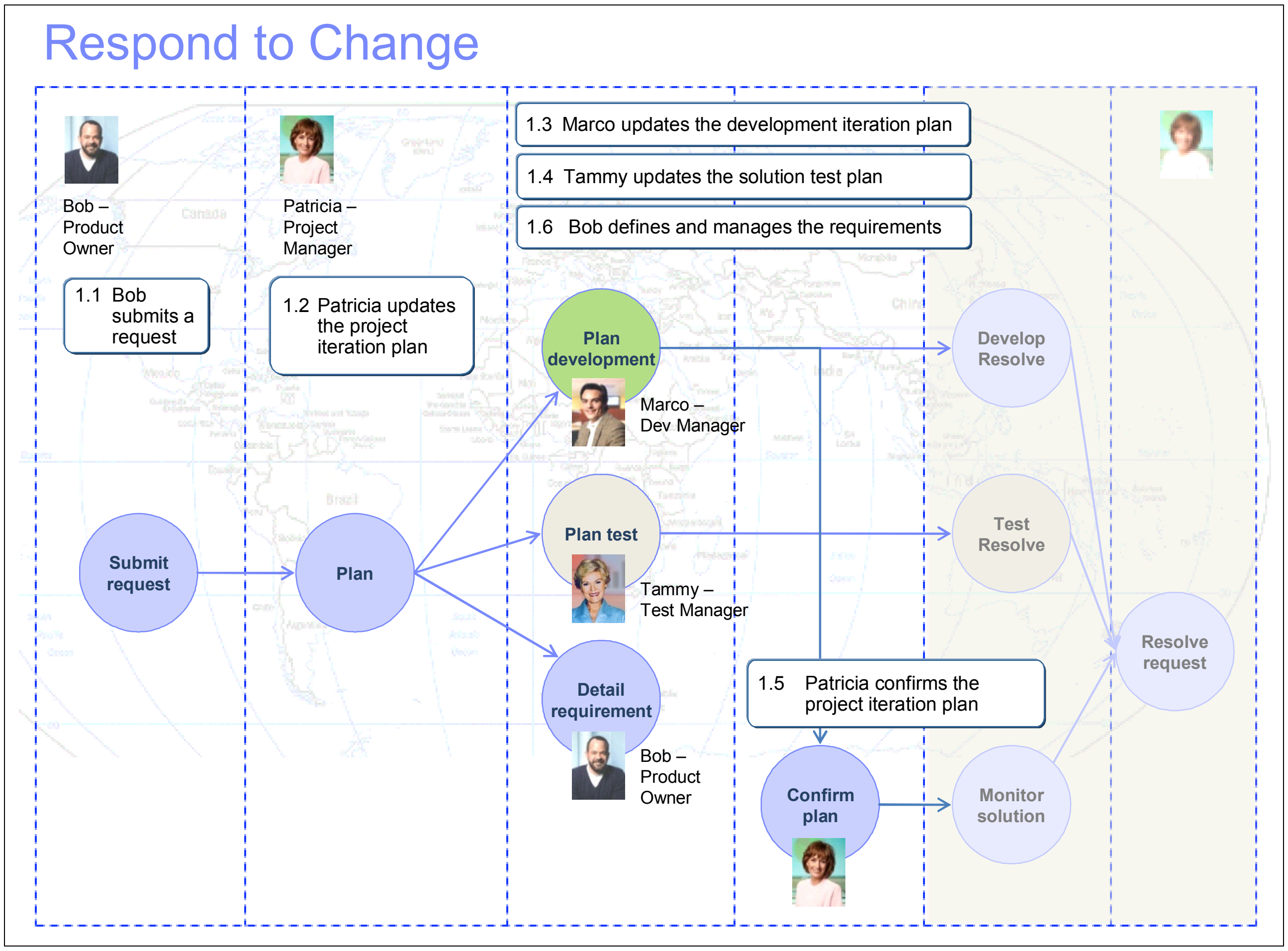
4.2.2 The workflow

Act 1 has the following workflow. The steps are described in more detail in the subsequent headings.

1. Bob, the product owner submits a new request.
2. Patricia, the project leader, is informed of the request and updates the project iteration plan to include the request.
3. The request is specific to GUI re-brand changes for the Credit Check component. Marco updates the development iteration plan to include the request in the next iteration plan for the component team.
4. The request must be included in the test assets, and Tammy updates the solution test plan to include it.

5. Patricia confirms the iteration plan to indicate the updates from the component team.
6. Bob defines and manages the requirements that are related to the request.

Figure 4-11 displays the activities that occur in Act 1: Respond to change.



Respond to Change
Bob – Product Owner
Patricia – Project Manager
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements
Plan development
Marco – Dev Manager
Plan test
Tammy – Test Manager
Detail requirement
Bob – Product Owner
Submit request
Plan
Develop Resolve
Test Resolve
Resolve request
1.5 Patricia confirms the project iteration plan
Confirm plan
Monitor solution

Figure 4-11 The flow of steps for Act 1: Responding to a change request

4.2.3 Bob submits a request

Synopsis: Bob, the product owner for the Account Opening processes, has been awaiting the announcement that the next milestone release is available. He wants to review the milestone and confirm its capabilities before submitting his finalized requirements for the next project iteration milestone. Bob receives the release notification and logs into the demonstration installation to run through the key processes for which he is responsible.

During his exploration of the milestone release, Bob finds that the release does not include the UI branding experience that he expected. His UI requirements were partially delivered, partially descoped and somewhat misunderstood. The inconsistencies are evident in the CreditCheck process. Bob believes that it is critical to ensure that the Account Opening project and associated applications have a consistent look to their interface and way in which they are used for the company for branding. All components must meet these characteristics.

Bob decides to submit a new request to have the UI branding completed in the next project iteration. He opens his Web browser and logs into the change management tool. He creates a new request where he references the UI standards that the enterprise uses for all of its customer-facing applications. Before finalizing and submitting the new request, he associates the request with the Account Opening second release project, AO_Rel2, sets a high severity on the request, and sets himself as the owner of the request.

Bob now wants the project team to take responsibility for the new request. He expects the request to show up in Patricia's triage queue and that she will update her iteration plan accordingly.

The workflow in this scene captures how Bob performs the following tasks:

- Creates a new request
- Assigns ownership of planning and delivery of the request to the development team

4.2.4 Patricia updates the project iteration plan

Synopsis: Patricia, the Account Opening project manager, is responsible for the overall project iteration plan and the project team coordination. She and her leadership team are triaging new requests on the Account Opening project. She reviews the UI branding request, which was submitted to by Bob, and agrees that the work has a high priority and must be completed in this iteration. She realizes this request impacts her project plan, the CreditCheck component plan, and the test plan. She must create tasks to plan, assign, and align the work across the team in order to implement the request.

Patricia assesses the properties of the request, such as size, priority, and risk. She ascertains that the depth of this request is not difficult. As a result, no type of formalized sizing activities have to occur by her team leads.

Patricia updates her project plan by creating and assigning new tasks for the impacted team leads to plan. She assigns a task to Marco for him to plan his implementation, a task to Tammy for her to update her test plan, and a task to Bob to define the requirements in further detail.

The workflow in this scene captures how Patricia performs the following tasks:

- Triages a requested change for the next iteration
- Plans work for a request by creating and assigning tasks to her team

The sizing of requests is explained in 4.3.2, “Sizing requests” on page 113.

4.2.5 Marco updates the development iteration plan

Synopsis: Marco is the leader of the agile development team that owns the Credit Check component. His agile team runs two-week iterations, while the project runs four-week iterations. Marco and his team are actively planning the project C2 iteration as their next C2A and C2B iterations. Marco conducts an iteration planning meeting with his team where Bob’s new request is reviewed.

The team looks at the request and discusses priorities and approaches. Diedrie indicates her interest in owning the changes. This request aligns with other UI work that she has scheduled for the next iteration. The team settles on the decision that the work is of medium priority, that it should be planned for the second C2B iteration, and that Diedrie should take ownership of the development task. Marco updates the development plan, updates the priority, and assigns the work to Diedrie.

After the meeting Diedrie completes her planning by organizing her assignments for the iteration. She estimates each of the work items and sets preliminary due dates for each of the changes. She finds that she is overcommitted for the iteration and identifies lower priority work that can be taken off her list. She collaborates with Marco and others on the team to balance work or replan for later iterations.

The workflow in this scene captures how the agile team performs the following tasks:

- Reviews, plans, and assigns work related to Bob’s request
- Forms, organizes, and balances the iteration plan

4.2.6 Tammy updates the solution test plan

Synopsis: Tammy and her team conduct iteration planning for the final milestone. The team already has their test plan strategy set for the final milestone with a focus on stability, load, and performance testing. However, Tammy also must ensure that all requirements for the iteration are covered by functional testing.

A few new requests have been added to the iteration and Bob’s UI branding request is one of them. Tammy finds that UI branding is not covered by any existing test cases. The test plan must be updated.

Tammy starts updating the iteration test plan. She accesses the test repository and opens the test plan. She extends the UI testing section in the plan with a new test case. She assigns the new test case to Tanuj for elaboration, creation, and configuration.

Tammy also confirms the availability of the required test servers. She finds that she must schedule more time and servers at the test lab. She adds a request for another test server and validates that resource demand and availability match in the new test plan.

The workflow in this scene captures how Tammy performs the following tasks:

- Updates the iteration test plan and assigns tests that are related to Bob’s request
- Allocates the required test servers to execute her test plan

4.2.7 Patricia confirms the project iteration plan

Synopsis: Patricia reviews the overall project iteration plan. Her component teams have completed their iteration planning, and their changes are reflected back into her project plan. She ensures that the schedule is still on track and that shared work is aligned for the iteration.

To ensure work alignment, Patricia runs queries to identify any tasks that might have been pushed off the iteration plan and hence might jeopardize the complete delivery of a request. She identifies the implementation related to the UI branding that is owned by Diedrie. This is a high priority change that the team has demoted to a medium priority and pushed to their second C2B iteration. Patricia disagrees with this planning and collaborates with Marco and Diedrie to lower the risk by delivering the changes earlier.

Patricia, Marco, and Diedrie settle on a plan where the delivery is made as early as possible. As requested by Diedrie, Patricia assigns Al to help the team and identifies a reusable component for UI branding. As Diedrie updates her work schedule, Patricia can view the updates to her project plan.

The workflow in this scene captures how Patricia performs the following tasks:

- Confirms the project iteration plan
- Collaborates with her team on iteration plans

4.2.8 Bob defines and manages the requirements

Synopsis: Bob finds that there is a new task from Patricia, the Account Opening project manager, for him to define requirements for the re-branding of Account Opening. This is related to the request he entered previously as the product owner.

Now that his request was accepted by the team, Bob documents it in the Requirements Management tool to manage it through the development cycle and add detail as the request is refined. He knows that there is not enough detail to capture the intent of the UI branding and realizes he can capture more information in the form of a sketch to provide more context to the request.

Bob uses a Requirements Definition tool so that he can create simple application sketches and storyboards. After Bob creates the application sketch, he requests feedback from his peers to ensure that everyone understands the UI branding requirement. His teammates provide feedback about his sketches in the form of comments that he can review. Upon agreement, Bob continues detailing and managing the requirements for this request.

Bob's team is following an Agility at Scale approach. As a result, he captures a feature requirement that provides more detail regarding the UI re-brand request and identifies the elements that are being re-branded. He also identifies a nonfunctional requirement to capture the need to comply with the corporate branding guidelines. He traces these requirements back to the request.

Now that Bob has completed detailing the requirements, he completes the task that was assigned to him by Patricia. To do so, he also associates the feature requirement to the request and sets the task to complete. Patricia receives notification that Bob's task is completed. She makes sure that Marco, Diedrie, and Tammy are also aware of this, so that they can begin their work.

In the define and manage the requirements scene, the following steps are taken as illustrated in Figure 4-12 on page 112.

1. Review the work.
2. Elaborate and define the requirements.
3. Manage the requirements.
4. Complete the work.

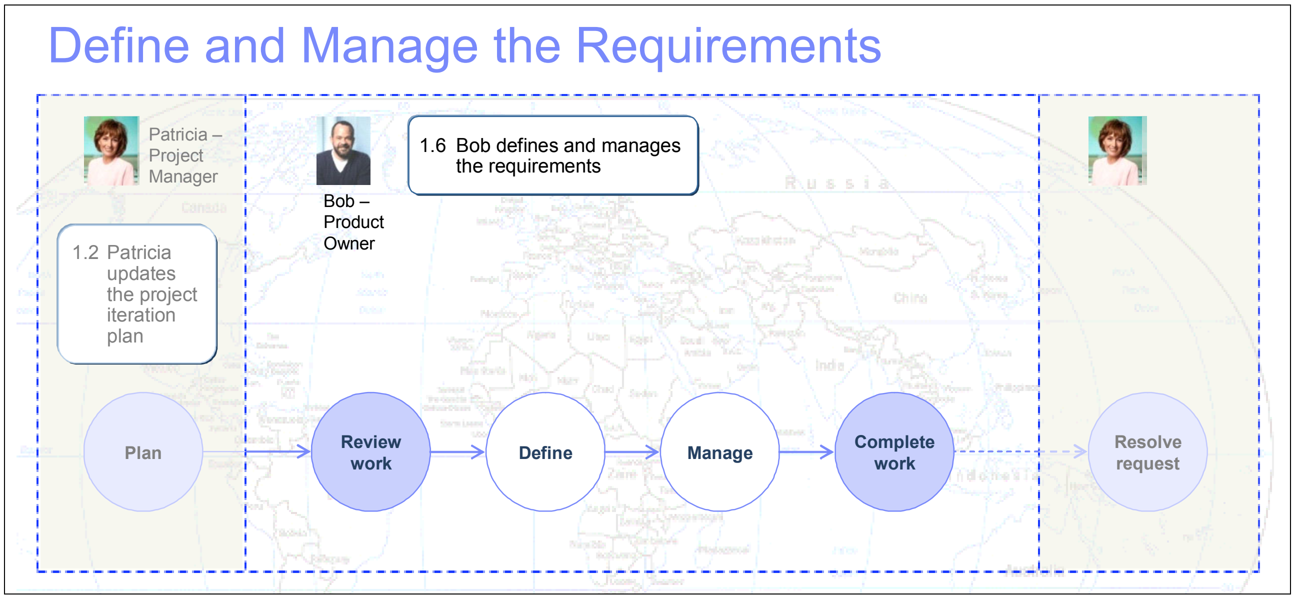

Define and Manage the Requirements
Patricia – Project Manager
1.2 Patricia updates the project iteration plan
Bob – Product Owner
1.6 Bob defines and manages the requirements
Plan
Review work
Define
Manage
Complete work
Resolve request

Figure 4-12 The flow of steps for defining and managing the requirements

4.3 Considerations in change management

In this section, we discuss pertinent activities for change management. These activities include analyzing and prioritizing requests as well as sizing them.

4.3.1 Analyzing and prioritizing requests

The scenario described in 4.2.2, "The workflow" on page 106, focuses the requirements and change management workflows on the alignment of work and iteration planning in the Account Opening project. However, requirements and change management at an enterprise scale often include a requirements and portfolio analysis phase, which involves the business stakeholders, the product owner and the program office, as outlined in 3.2, "The project" on page 49.

Because analyzing and prioritizing requests, and the associated workflows, are out of scope for this book, considerations are briefly discussed in this section. When extending the requirements and change management with the analysis and prioritization workflows, take note of the following considerations.

Requests that are submitted to a project are reviewed by the product owner. In this story, Bob, the product owner, assesses the following information:

- Is the information that is provided by the stakeholder sufficient?
- Is the level of detail that is provided adequate?
- Was the business problem captured and can it be understood?
- Does this request fit into the scope of a business strategy, goal, or theme for the project?
- Does the stakeholder's priority fall in line with the priority for the business?

Prioritization is simply comparing the stakeholder's priority for the request versus the importance and priority of the product. These views on priorities might not always align. Some type of collaboration might need to occur between the stakeholder and the business where concessions are made about the priority.

The end goal is to identify a prioritized list of requests that are taken forward for the iteration. This prioritized list may be identified as the backlog if following agile practices. This list should be pruned for requests to implement for upcoming iterations. Review and reprioritization of these items can occur in later activities as the requirement or development team refines the request into requirements. This list is used, is reviewed, and is “fluid” as the team is required to respond to change.

4.3.2 Sizing requests

This scenario focuses the story on the addition of a single request to the previously committed release requirements stack for the next iteration in the Account Opening project. Bob, the product owner, considers the request to be of high business priority and expects the development team to deliver the changes as soon as possible. Also, the size and scope of the changes are understood by the project.

However, in most cases, the size of work related to delivering more complex requests might not be intuitively understood. Also, the delivery of a complex request, in most cases, spans multiple components and multiple development disciplines such as development and test. The feasibility to deliver a request in an iteration might not be consolidated across the project.

In extending the scenario to sizing, Patricia, the project manager, reviews a request that has been submitted to the Account Opening project by Bob. In order to assess whether the team can implement the request within the given iteration, she consults various team members to identify if the request might be contained within the confines of the iteration. To make this assessment, initial sizing must occur by the following teams:

- Architecture team, which is represented by Al, the solution architect
- Component team, which is represented by Marco, the CreditCheck component lead
- Test team, which is represented by Tammy, the test lead

Each person’s input is critical because the requirement will not be implemented in the given iteration if it cannot be contained. The following elements, among others, might be considered by as part of the sizing:

- The iteration to implement the solution
- The skills and availability of resources to implement the solution
- The architectural significance of implementing the request
- The demand for additional testing or at worst additional testing approaches
- The availability of re-usable components, frameworks, or test assets that can be leveraged

Using the ClearQuest ALM solution can be configured to manage sizing assignments. In such a work configuration, Patricia creates a task, of type size, for tracking sizing. The task, with Patricia as the owner, is now available to associate sizing activities for the task. She creates sizing activities and assigns the activities to the respective team. Team members make their assessments and can use the activity resolution field to indicate if the request can be contained and in what iteration. Using the collaboration capabilities in the CALM solution, teams can collaborate and validate the estimates.

For details about extending the OpenUP configuration for sizing, see the following sections:

- “Configuring ClearQuest ALM for OpenUp” on page 195
- “Adding optional resolution codes” on page 576

4.3.3 Rational Team Concert for stakeholder requests

This book captures end-to-end CALM by using the Rational CALM solution. This solution includes the Rational ClearQuest CALM solution, Rational RequisitePro, and Rational Requirements Composer.

However, some projects or teams might decide to deploy reduced application life-cycle support as appropriate to their practices, maturity, or process needs. The collaborative development blueprint captures the minimal core capabilities in requirements and change management workflows. In this section, we discuss the considerations when deploying application life-cycle support by using Rational Team Concert. See 6.3.1, “Lifecycle solution for small teams” on page 227, for a similar discussion about the considerations for collaborative development. Teams that use Rational Team Concert for application lifecycle gain the following benefits:

- Web-based access for project stakeholders
- Requirements and change workflows, and process enactment for an agile way of working by using the Eclipse Way process
- The creation, update, and query of requests and view disposition of development tasks
- Personalized project dashboards to team health

Teams who are using Rational Team Concert for end-to-end CALM might experience the following limitations:

- Limited requirements management and definition by using work items
- Limited quality management by using build validation with JUnit test cases
- Limited team size or distributed repositories

For more information about using the Eclipse Way process, see the Jazz Web site at the following address:

http://jazz.net

5

Rational ClearQuest, Requirements Composer, and RequisitePro to manage stakeholder requests

In this chapter, we provide a detailed demonstration of Part B, “Act 1: Responding to a change request” on page 77, and Chapter 4, “The team responds to a requirement change” on page 79. The purpose of this chapter is to provide a pragmatic demonstration of how the roles in this scenario use the Rational products to accomplish their tasks.

We discuss the following topics in this chapter:

- An overview of the product features used in the scenario
- A step-by-step demonstration of the products in the scenario
- A summary of the assets that are created and used by the team
- How to measure success for this scenario
- How the products that are used fit into a larger enterprise Application Lifecycle Management (ALM) scenario and how they are configured
- Tips and tricks for resolving known problems

Specifically, this chapter includes the following sections:

- 5.7, “Planning and measuring success in change management” on page 177
- 5.8, “Reference architecture and configuration” on page 186
- 5.9, “Problem determination and known workarounds” on page 203

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

5.1 Act 1: Responding to a change request

In this chapter, we discuss step by step how the characters in the story complete Act 1 of the storyboard, which is illustrated in Figure 5-1.

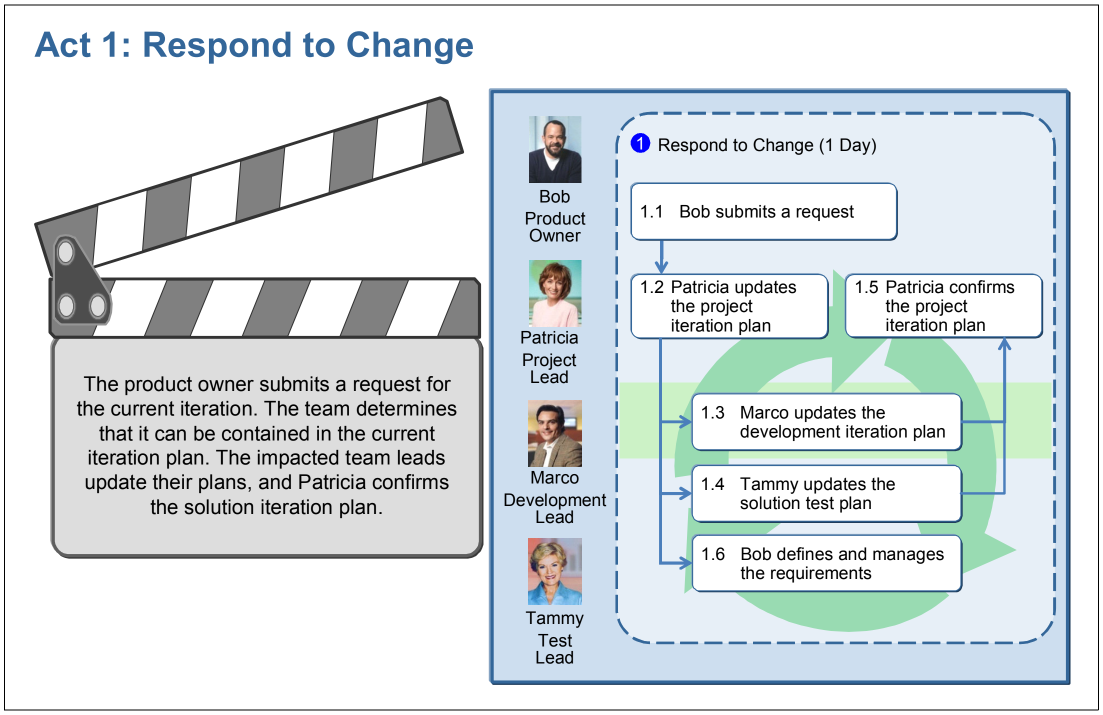

Act 1: Respond to Change
The product owner submits a request for the current iteration. The team determines that it can be contained in the current iteration plan. The impacted team leads update their plans, and Patricia confirms the solution iteration plan.
Bob Product Owner
Patricia Project Lead
Marco Development Lead
Tammy Test Lead
1 Respond to Change (1 Day)
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.5 Patricia confirms the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements

Figure 5-1 The scenes in Act 1: Responding to a change request

This act contains the following scenes:

- Bob submits a request.
- Patricia updates the project iteration plan.
- Marco updates the development iteration plan.
- Tammy updates the solution test plan.
- Bob defines and manages the requirements.
- Patricia confirms the project iteration plan.

The following IBM Rational products are used in this act:

- IBM Rational RequisitePro 7.1.0.0
- IBM Rational Requirements Composer 7.1.0.0
- IBM Rational ClearQuest 7.1.0.0
- IBM Rational Quality Manager 8.0 beta

In addition, the following products were integrated:

- Rational RequisitePro and Rational Requirements Composer for requirements management and storyboarding
- Rational RequisitePro and Rational ClearQuest for traceability between change requests and requirements
- Rational ClearQuest and Rational Team Concert and Rational Quality Manager for team interoperability in a geographically distributed environment

5.2 Rational RequisitePro and Rational Requirements Composer

Rational RequisitePro is a requirements management application. With it, project teams can manage their requirements, write good use cases, improve traceability, strengthen collaboration, reduce project risk, and increase quality. Beginning with release 7.1.0.0, RequisitePro supports an integration with Rational Requirements Composer.

Rational Requirements Composer enables requirements management definition. Rational Requirements Composer fosters focused, natural, real-time, contextual collaboration by using various techniques, vocabularies, and artifact types. This new tool provides improvement in the following key areas:

- Improved requirements definition, validation, and management of requirements change through the software development life cycle
- Increased and clearer communication among business stakeholders and IT delivery teams wherever they are located
- Less project rework, faster project execution, and lower-cost delivery

Rational RequisitePro

Rational RequisitePro is a full featured, flexible, and integrated requirements management tool. Requirements management is key to a project's success. Rational RequisitePro is a requirements management solution that aids teams in organizing their requirements. It provides collaboration capabilities to help ensure accurate communication and management of requirements as they evolve. Additionally, requirements can be categorized by using attribute functions in order to define prioritization of requirements, for example.

The Extensibility Interface API is included and allows access to requirement information outside of the context of the tooling. As a result, Rational RequisitePro can be extended by using this API to extract information from the database or import information from external sources or applications into Rational RequisitePro.

Rational RequisitePro offers the following key advantages among others:

- The ability to capture, track, manage, and analyze different types of requirements
- Dynamic integration between the Microsoft® Word application and a requirements database

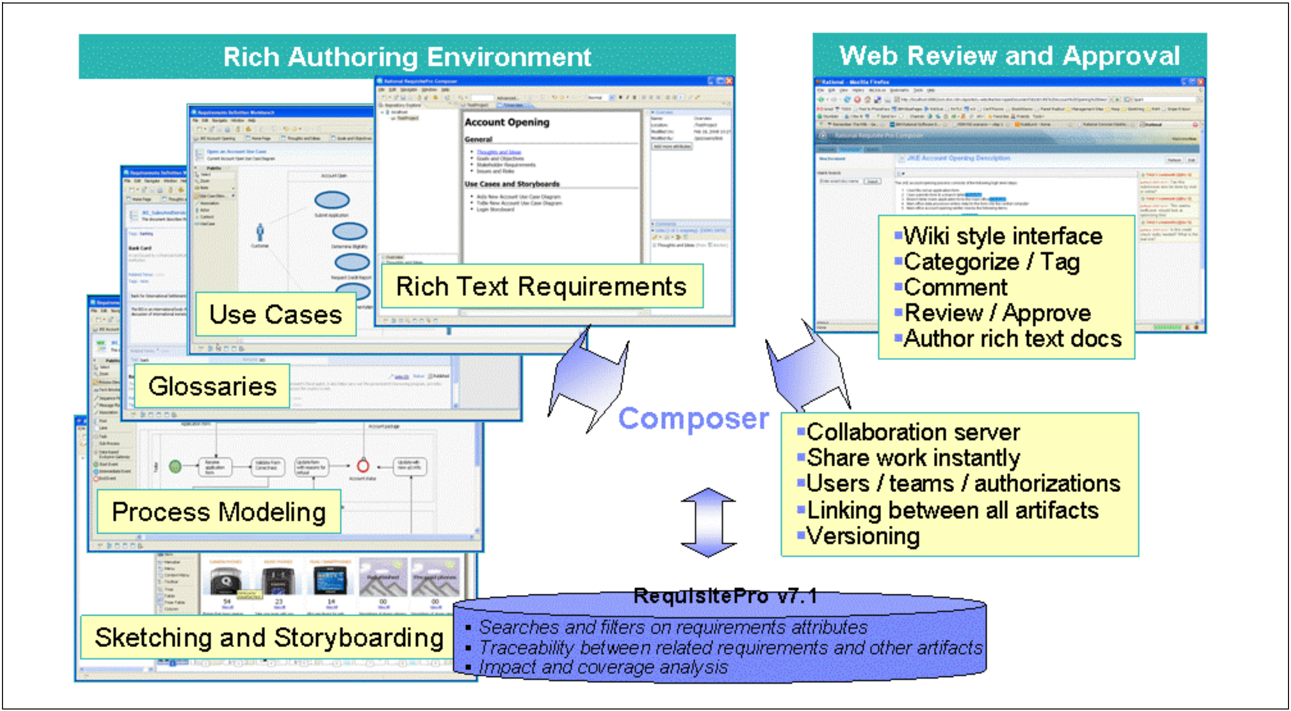
  The requirement content stored in the database aids teams in organizing, prioritizing, and tracking project requirements, which is something that the Word application alone cannot do well.
- The ability to specify who is authorized to modify requirements, attributes, and documents
- In-depth traceability and coverage analysis

- Understanding of the impact of changing requirements
- Support for distributed team members by the Web
- Integratable with marketplace-leading life-cycle tools including solutions for business modeling, defect and change tracking, visual modeling, automated testing, configuration management, and process guidelines

  These integration points help support the entire software development team and the software development process.

Rational Requirements Composer

Team members require a number of techniques in order to elicit requirements information. Rational Requirements Composer helps in this area because it provides functions to capture sketches and storyboards, use cases, glossaries, and process models. With these functions, analysts can provide context information for requirements as shown in Figure 5-2. After these requirements are defined, they can be managed in Rational RequisitePro. Rational Requirements Composer is a complement to Rational RequisitePro.



Rich Authoring Environment
Web Review and Approval
Rich Text Requirements
Use Cases
Glossaries
Process Modeling
Sketching and Storyboarding
Wiki style interface
Categorize / Tag
Comment
Review / Approve
Author rich text docs
Composer
Collaboration server
Share work instantly
Users / teams / authorizations
Linking between all artifacts
Versioning
RequisitePro v7.1
Searches and filters on requirements attributes
Traceability between related requirements and other artifacts
Impact and coverage analysis

Figure 5-2 Rational Requirements Composer

5.3 Rational ClearQuest

Rational ClearQuest is flexible and powerful change management solution that provides defect tracking, process automation, reporting, and life-cycle traceability for better visibility and control of the software development life cycle. Rational ClearQuest provides a client based on Eclipse technologies. It provides a Web interface to allow Geographically Distributed Development (GDD) teams the ability to use a centralized data repository for change management.

In addition, databases can be multi-sited to share data across a wide area network (WAN). An API is included to all scripted access to database information. Rational ClearQuest offers the following key advantages among others:

- ALM package for managing a team's work in the context of secure and role-based projects
- Real-time reporting and process enforcement to improve project visibility and control
- Automated workflows and e-mail notifications to enhance team communication and coordination
- Access control, electronic signatures, repeatable processes, and audit trails to simplify compliance management
- Web interface for easy access from virtually anywhere
- Integration with requirements, development, build, test, deployment, and portfolio management tools for simplifying rapid response to change

The highly customizable architecture of Rational ClearQuest makes it an ideal tool for any organization to adapt their existing process.

ALM Packages for Rational ClearQuest

Rational ClearQuest 7.1.0.0 includes a ready-to-use ALM solution that provides support for managing many of the challenges presented by GDD and ALM.

Packages for download: The ALM Packages for Rational ClearQuest are available for Rational ClearQuest 7.0.1 users to download for free from IBM. The packages can be accessed from the Web at the following address:

`http://www.ibm.com/services/forms/preLogin.do?lang=en_US&source=swg-ratcq`

Registration is required to access the packages. The download includes the ALM Packages for Rational ClearQuest, instructions for applying the packages, a sample database, and three tutorials for use with the sample database. The ClearQuest ALM solution can be used with both ClearQuest versions 7.0.1 and 7.1.

The ALM package provides support for a streamlined, agile application development process that is both role-based and process-driven, as illustrated in Figure 5-3 on page 120. Projects define a context for completing work and can be secured by setting security policies and defining roles. Work can be assigned to team members who are either co-located or distributed. That work is traceable to the original request, to the project that implemented the request, and to the build that contains the implementation of the request.

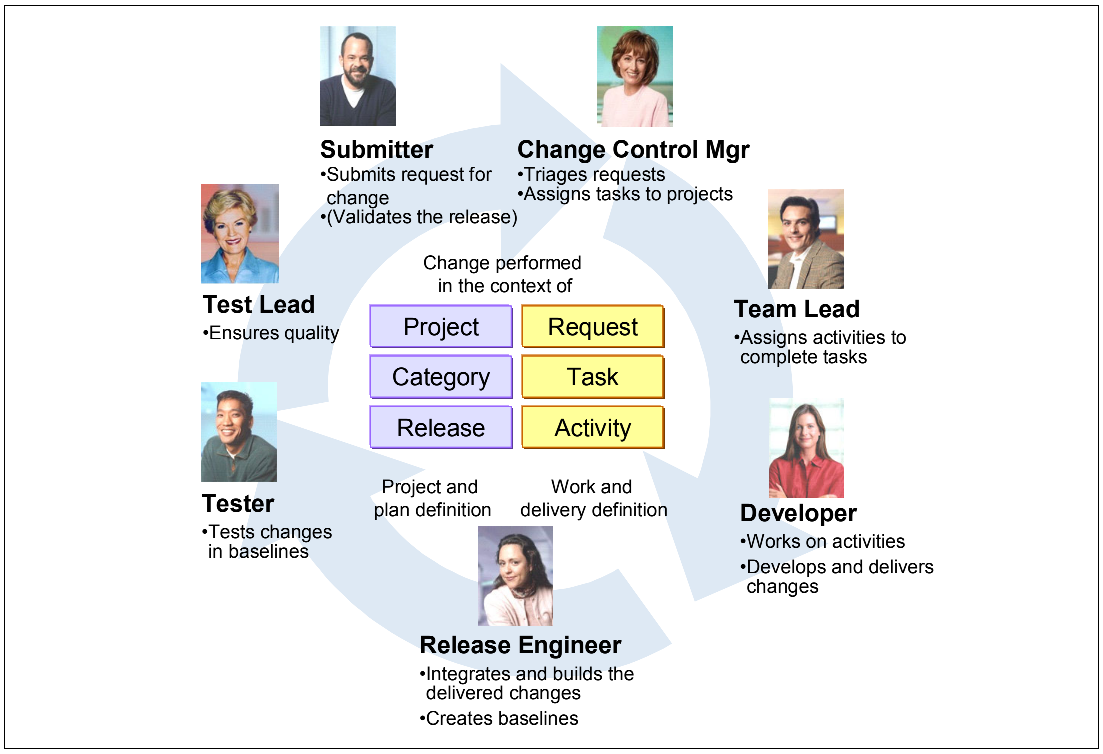

Submitter
•Submits request for change
•(Validates the release)
Change Control Mgr
•Triages requests
•Assigns tasks to projects
Test Lead
•Ensures quality
Team Lead
•Assigns activities to complete tasks
Change performed in the context of
Project
Request
Category
Task
Release
Activity
Project and plan definition
Work and delivery definition
Tester
•Tests changes in baselines
Developer
•Works on activities
•Develops and delivers changes
Release Engineer
•Integrates and builds the delivered changes
•Creates baselines

Figure 5-3 ClearQuest ALM support of role-based development processes

The ALM schema provides a set of records with relationships that help teams manage software development projects. The ALM schema's principle role is to help teams manage the work that is involved in delivering software projects. It provides useful building blocks and a framework that facilitates custom configurations to fit into every enterprise.

There are three essential concepts to understand when working with the ALM schema in ClearQuest:

- Projects provide the context for managing work that is created by the team members. Users are granted access to projects through security policies, and their actions are defined by their role.
- Managing work is enabled in the form of requests, tasks, and activities. The definition of activities and tasks is driven by process definitions, which can change from project team to project team.
- System-wide settings can be defined for projects and managing work. These settings allow for reuse and consistent classification across multiple projects. They can adapt to the enterprise. These settings typically involve a one-time setup or minor changes as the teams grow and evolve with their use of the system. (ClearQuest administrators start here.)

In the reference scenario, we mainly work with requests, tasks, and activities that are used to manage work in the context of an existing project. As shown in Figure 5-4 on page 121, all work starts with a *request*. Requests are planned and implemented with *tasks*, and tasks are completed by using *activities*. When all activities are complete, the task is complete. When the tasks are complete, the request is complete.

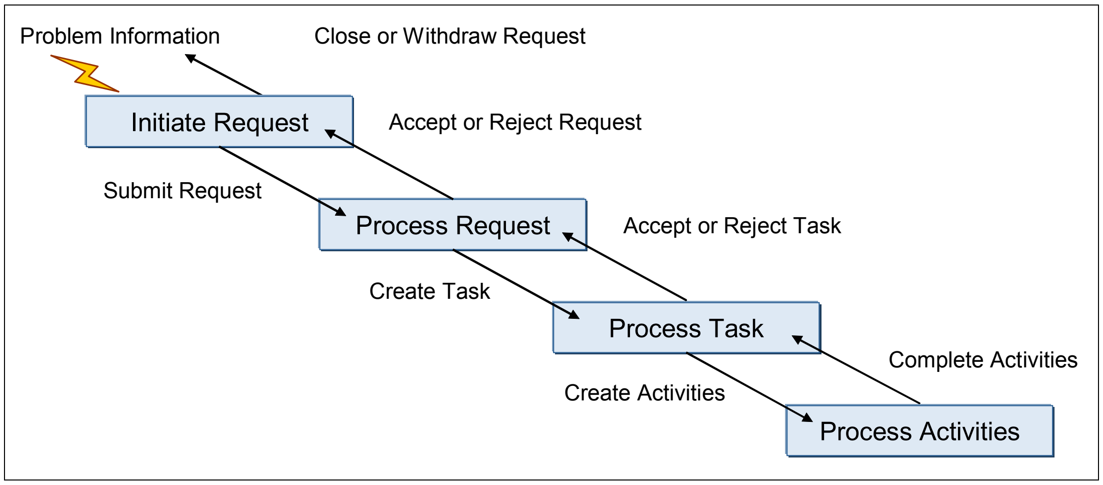


Figure 5-4 The flow of request, task, and activity records

A *request* contains a petition to the team to make a change and includes information that is related to where and how the request was found. It is related to a category that provides further classification for identifying where the need was found, for example, a product, feature, component, or service. The *request type* identifies the nature of the request, for example enhancement, defect, or feature. However, the request can be expanded to cover any type of request for work. By using RUP as an example, you can create a request to "initiate an iteration." A request is owned by the submitter and contains references to all tasks that are associated with it. The tasks contain information about where the request will be fixed.

A *task* represents the work that is needed by a team to complete their part of the request. A task contains all information that is related to where the request will be addressed. It is owned by someone on the project team and represents an aggregation of work spread across multiple team members. In addition, tasks contain references to all activities that are required to complete the task and refer back to the request that it is intended to fulfill.

Lastly, an *activity* is owned by an individual contributor on the project team and represents a single unit of work. It refers back to the task that it helps to complete and can be enabled for Unified Change Management (UCM).

More information: For more information and core concepts about setting up Rational ClearQuest and the ALM schema, see the IBM developerWorks three-part series of articles for "Application lifecycle management with ClearQuest 7.1.0.0." You can find these articles on the Web at the following addressees:

- Part 1

  `http://ltsbwass001.sby.ibm.com/cms/developerworks/rational/library/edge/08/mar08/pampino-pierce/index.html`

- Part 2

  `http://www.ibm.com/developerworks/rational/library/edge/08/apr08/pampino-pierce/index.html`

- Part 3

  `http://www.ibm.com/developerworks/rational/library/edge/08/may08/pampino-pierce/index.html`

5.4 Jazz interoperability

In this act, Rational Team Concert is used by the component team to receive information about and plan for the new request that is submitted in Rational ClearQuest. To facilitate coordination of work between Rational ClearQuest and Rational Team Concert, a *connector* is configured. In a similar manner, Rational Quality Manager is used by the solution test team to plan, manage, and execute the test effort. A connector is configured to facilitate the coordination of work between Rational ClearQuest and Rational Quality Manager.

In this section, we discuss interoperability for the Jazz products. Rational Team Concert provides components, called *connectors*, to integrate into an enterprise scale application life-cycle solution. The following connectors are available:

- ClearQuest Connectors
- ClearCase Connectors

How the component team uses Rational Team Concert is discussed in more detail in 7.3, "Rational Team Concert for agile development" on page 246. How the test team uses Rational Quality Manager is discussed in 11.5, "Rational Quality Manager" on page 425.

5.4.1 ClearQuest Connectors

ClearQuest Connectors, which are available as a part of Rational Team Concert 1.0, provide a flexible infrastructure that allows organizations to synchronize information between Rational ClearQuest records and Rational Team Concert work items. The ClearQuest Connector for Rational Team Concert pragmatically, and with seamless integration, lets project teams work in both tools and share data. Through synchronization operations, the connector maps Rational ClearQuest records, such as ALM Tasks, to Rational Team Concert work items of type Task. When a user creates or modifies a Rational ClearQuest record, the connector creates or modifies a corresponding Rational Team Concert work item. The creation and modification changes also flow from work items to Rational ClearQuest records. The ClearQuest Connector is configurable and provides an elaborate way to map a customized schema, including workflows in Rational ClearQuest with Rational Team Concert work items.

For the scenario in this book, this integration is key to enable the work alignment required for Patricia to fold new requests from Bob into her iteration plans as tasks. The integration is also key for her to assign and align the task to the various contributing component or functional teams in the project.

Project teams on the ClearQuest ALM solution consume the task assignments directly in the Rational ClearQuest environment. Teams that use Rational Team Concert, such as Marco and his team, use the ClearQuest Connector to create synchronized work items in their repository for each assigned task. Marco tracks work by using the synchronized work item. He might optionally do further planning by creating additional child work items to track more sizable tasks. Updates made to the plan by Marco and his team in Rational Team Concert are synchronized back to Rational ClearQuest by the ClearQuest Connector. This provides Patricia with an updated and consolidated view of the component and functional planning for each task that is committed for the project iteration. See Figure 5-5 on page 123.

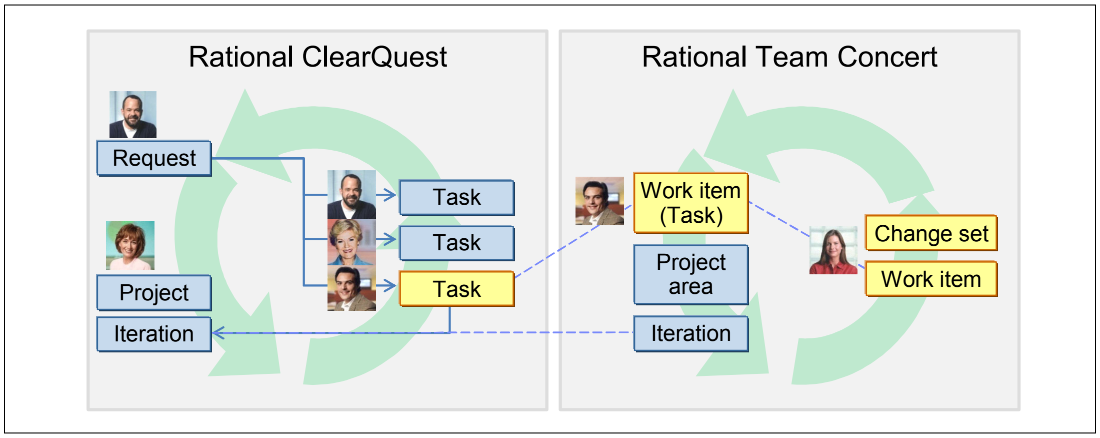


Figure 5-5 ClearQuest Connector for team alignment

A second case of interoperability is the assignment of work. This case occurs when the activities of a task are assigned to the team members for completion. In the case of one or more team members that use Rational Team Concert, synchronized work items are created in their repositories for each assigned activity. An example of this case is Diedrie being assigned the activity to review the outcome of Al's refinement of the architecture, which is required to support Bob's UI branding request. In such a scenario, Patricia, or Al, creates an activity or type of 'Review', and assigns the activity to Diedrie. The activity is created as a synchronized work item in Rational Team Concert. The new work assignment is in Diedrie's plan and the iteration. See Figure 5-6.

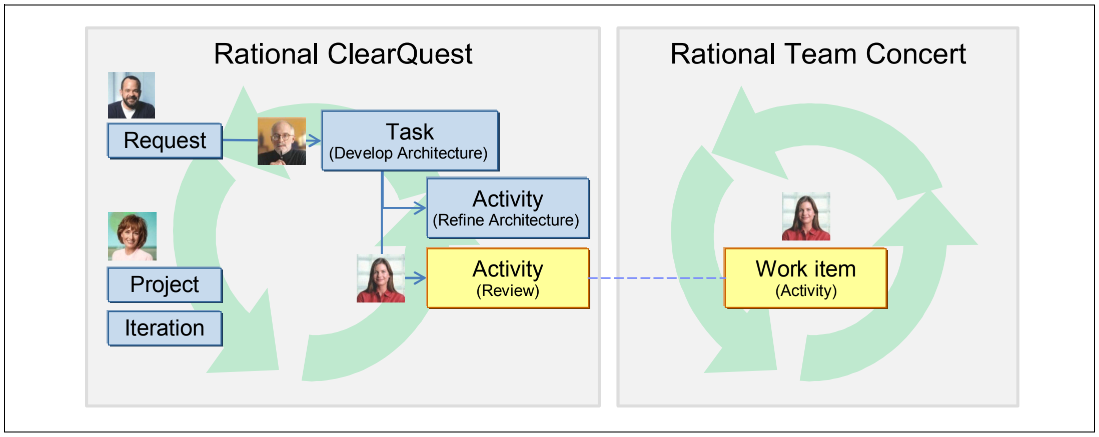


Figure 5-6 Using ClearQuest Connector for work assignments

The scenario in this chapter demonstrates how the team uses the ClearQuest ALM solution and Rational Team Concert to achieve alignment of work. The ClearQuest Connector for Rational Team Concert is an integrated part of the scenario to connect the teams and their repositories. In Part C, "Act 2: Collaborative development" on page 211, the scenario turns its focus to collaborative development by using Rational Team Concert and how the ClearQuest Connector supports project health tracking.

Before projects can use the ClearQuest Connector, they must be configured in the Rational ClearQuest and Rational Team Concert environments, which involves configuring the following items:

- The Rational ClearQuest schema with a ClearQuest Connector package for Rational Team Concert
- The ClearQuest Connector Gateway and Jazz server for Rational Team Concert
- Synchronization rules in Rational Team Concert
- Optionally: The work configurations in Rational ClearQuest and the Rational Team Concert team area process

To configure the ClearQuest Connector for Rational Team Concert, see Appendix B, “Configuring interoperability” on page 565.

5.4.2 ClearCase Connectors

ClearCase Connectors provide seamless interoperation between Rational Team Concert and Rational ClearCase by using ClearCase Synchronized Stream. This capability enables a team that is working in Rational Team Concert to access Eclipse projects that are under Rational ClearCase source control. Alternatively, the team can deliver changes made in Rational Team Concert to Rational ClearCase users through UCM components or versioned object base (VOB) folders. For the scenario in this book, this integration is key to enable a delivery chain for the component team deliveries to solution integration and for Patricia and her leadership team in tracking the work delivery.

For details about ClearCase Synchronized Streams, see “Diedrie delivers daily changes for solution integration” on page 285, in which we discuss the usage of ClearCase Connectors and UCM for the book scenario, and “ClearCase Synchronized Streams” on page 288. For additional information, see the Jazz Web site at the following address:

`http://jazz.net`

5.5 Managing a change request with Rational RequisitePro and Rational ClearQuest

Synopsis: Act 1 of the story exemplifies how the team responds to a new change request that has been submitted by Bob, the product owner. This request is submitted by using ClearQuest ALM, and assets are created to represent planned work, in the form of tasks and activities, which need to be assigned to and worked on by various members of the organization. Patricia, the project manager, uses Rational ClearQuest to triage these assets to Al, the architect; Marco, one of the component leads; and Tammy, the test lead. Additionally, Bob uses Rational RequisitePro and Rational Requirements Composer to do additional requirements definition work and storyboarding. Each team member works on their respective assets and keeps Patricia updated on their progress.

In this section, we follow the referenced scenario to see how the Rational ALM solution is used by various members of the enterprise organization in their daily tasks while working on release two of the Account Opening application and the Credit Check project.

The workflow in Figure 5-7 shows the following actions:

- Bob submits a request.
- Patricia updates the project iteration plan.
- Marco updates the development iteration plan.
- Tammy updates the solution test plan.
- Bob defines and manages the requirements.
- Patricia confirms the project iteration plan.

Act 1 is also described in 4.2, "A reference scenario for responding to a change request" on page 105.

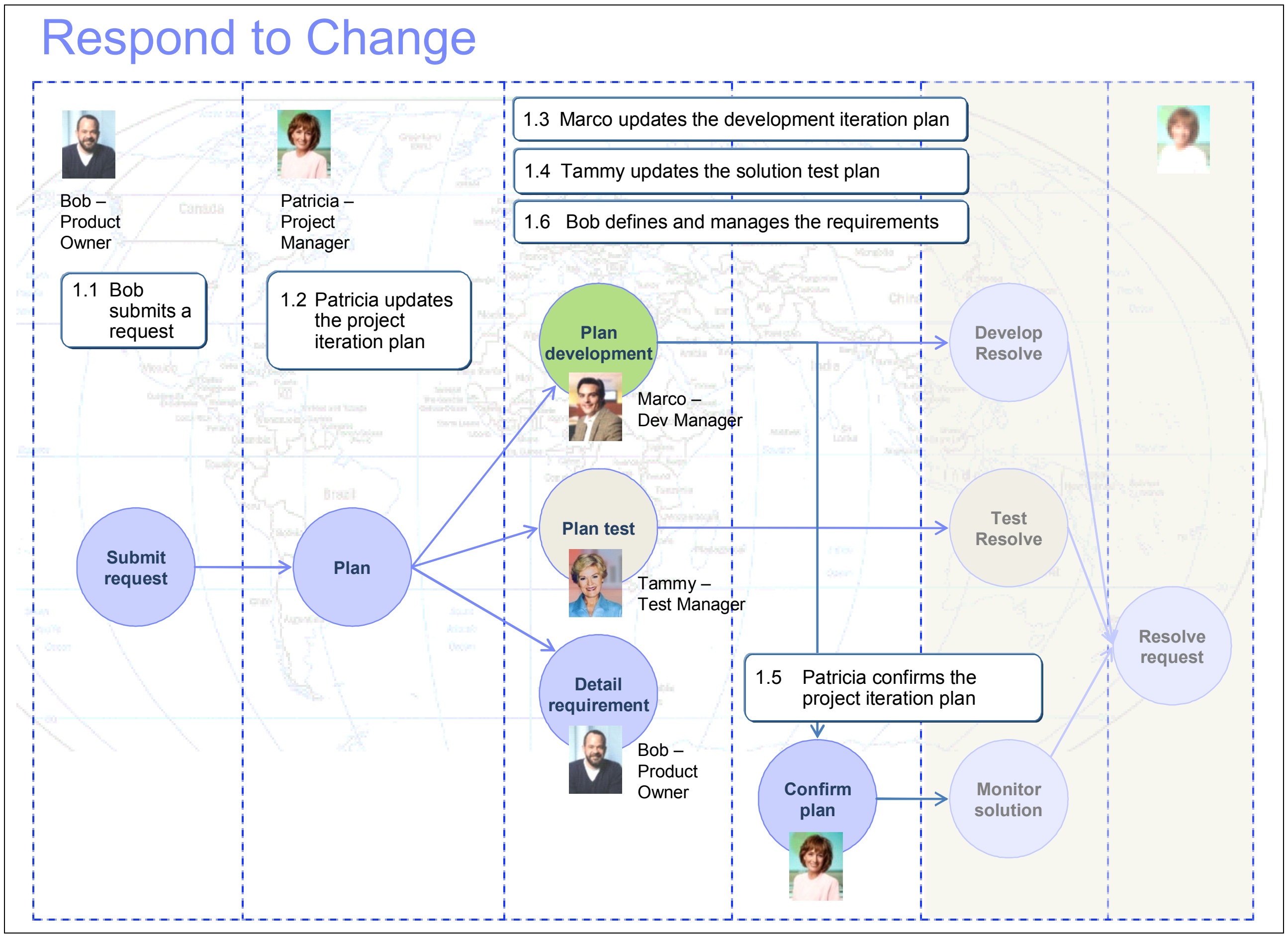

Respond to Change
Bob – Product Owner
Patricia – Project Manager
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements
Plan development
Marco – Dev Manager
Develop Resolve
Submit request
Plan
Plan test
Tammy – Test Manager
Test Resolve
Resolve request
Detail requirement
Bob – Product Owner
1.5 Patricia confirms the project iteration plan
Confirm plan
Monitor solution

Figure 5-7 The flow of steps in Act 1: Responding to a change request

In Act 1, the team creates several new artifacts and establishes traceability to support the life-cycle collaboration. Figure 5-8 shows the life-cycle collaboration artifacts that are established in this act.

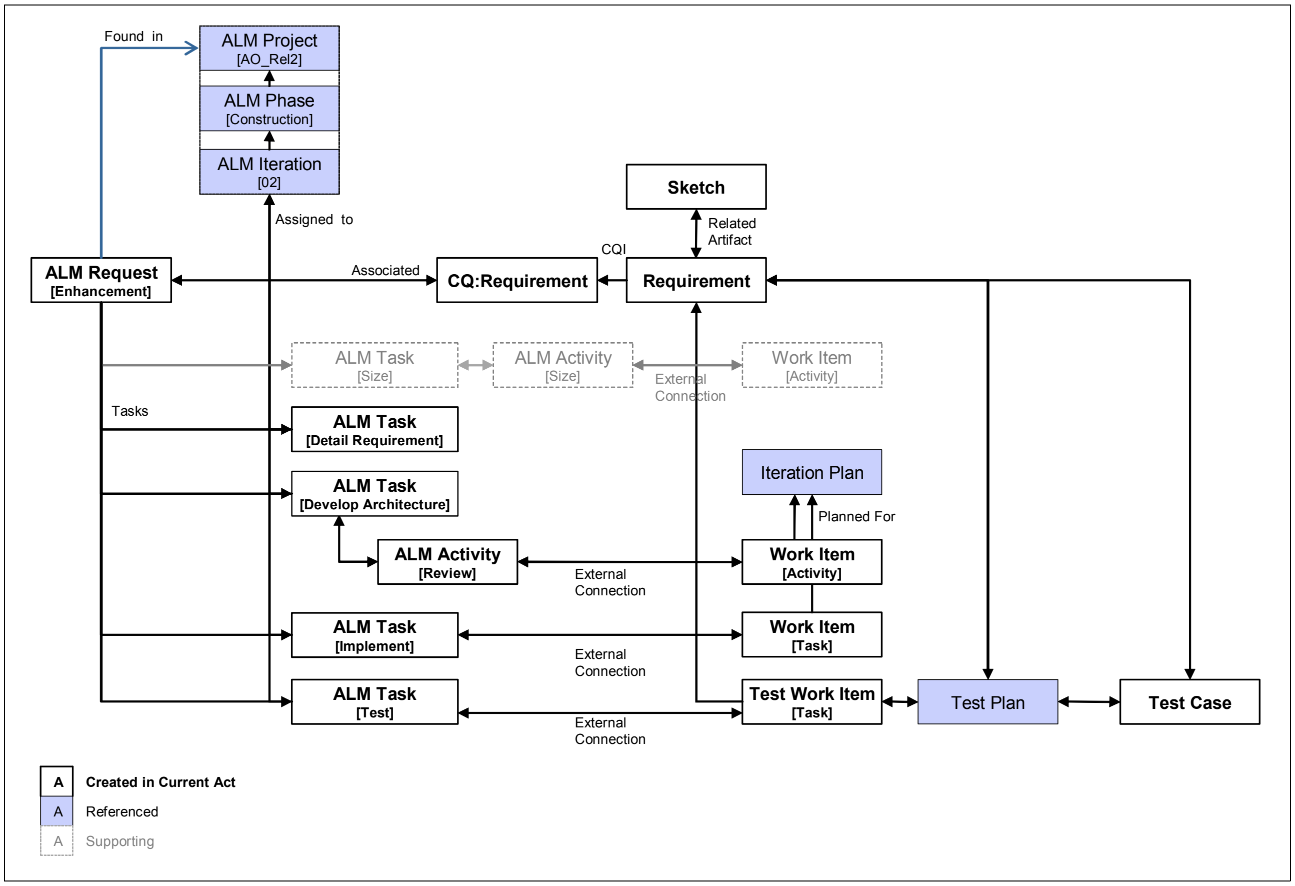


Figure 5-8 Life-cycle collaboration artifacts established in Act 1: Responding to a change request

The following artifacts, which are created in this act, span multiple products. For each artifact, we include the type and the tool in which they are created.

- ALM Project (Rational ClearQuest): The AO_Rel2 project on which the team is working.
- ALM Phase (Rational ClearQuest): The Construction development phase in the Open Unified Process (OpenUP) development process that is used by the project.
- ALM Iteration (Rational ClearQuest): The Construction 02 iteration that the team is currently planning.
- ALM Request (Rational ClearQuest): The release requirements for the project. Requests take the form or defects and enhancements.
- ALM Task (Rational ClearQuest): The committed work, planned for an iteration, to deliver a part of a request.
- ALM Activities (Rational ClearQuest): Work managed as UCM activities or Rational Team Concert work items.
- Work Item (Rational Team Concert): Work planned for an iteration and monitored in Rational Team Concert.
- Iteration Plan (Rational Team Concert): Contained work for an iteration.
- Work item (Rational Quality Manager): Test work planned and monitored in Rational Quality Manager.

- Test Plan (Rational Quality Manager): Test plans in Rational Quality Manager.
- Test Case (Rational Quality Manager): Test cases in Rational Quality Manager
- Requirement (Rational RequisitePro): Managed requirements that define and elaborate on stakeholder requests.
- Requirement (Rational ClearQuest): Record supporting traceability between ALM assets and Rational RequisitePro.
- Sketch (Rational Requirements Composer): Asset that provides requirements definitions and elaboration.
- Reusable Asset (Rational Asset Manager): Managed asset for reuse by a development organization, for example.

5.5.1 Bob submits a request

Synopsis: Bob determines that the UI branding for the Credit Check application does not conform to the organization's standards and submits a new change request.

This scene has the task of submitting an ALMRequest record.

Figure 5-9 highlights the workflow.

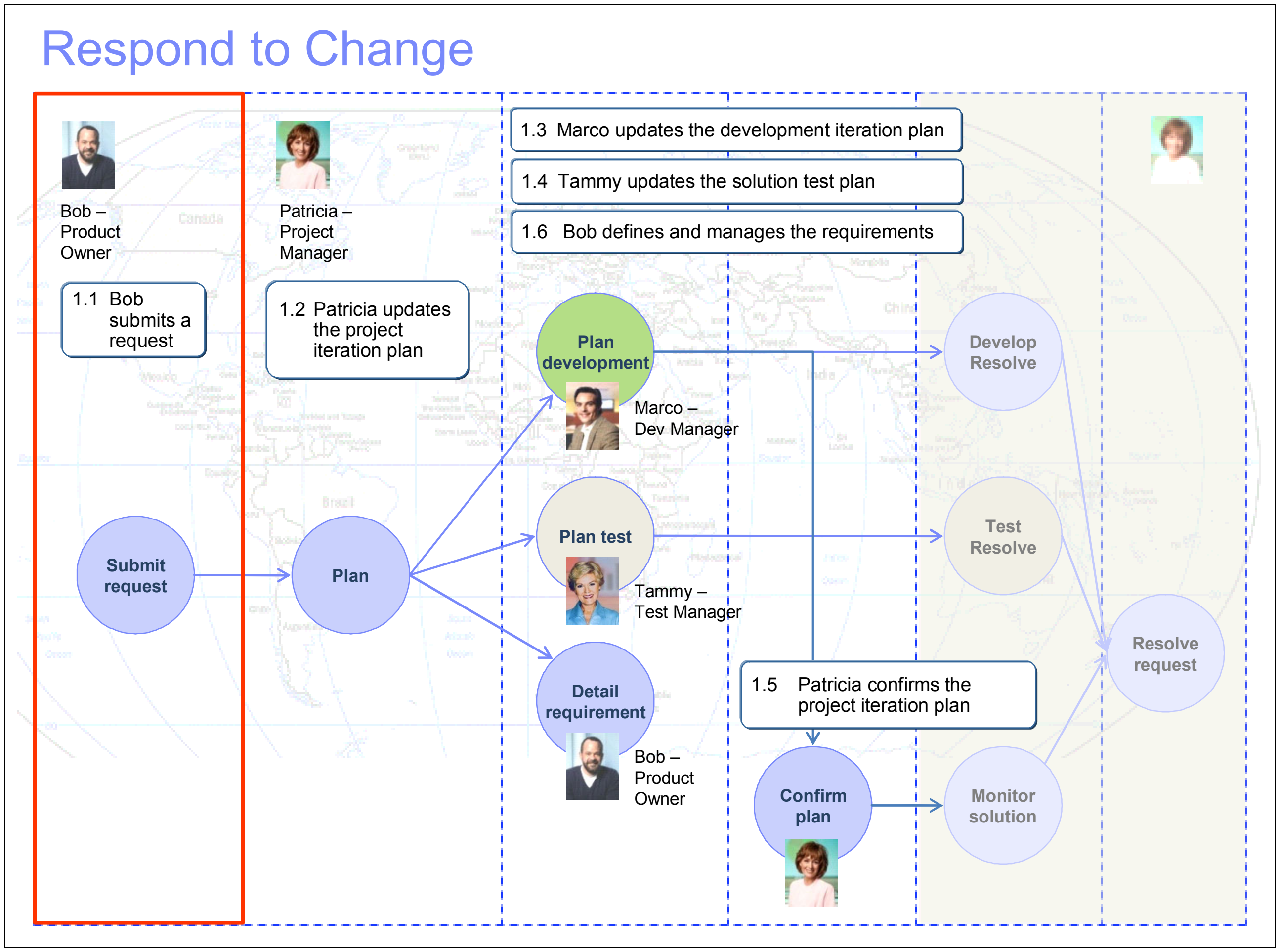

Respond to Change
Bob – Product Owner
1.1 Bob submits a request
Submit request
Patricia – Project Manager
1.2 Patricia updates the project iteration plan
Plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements
Plan development
Marco – Dev Manager
Plan test
Tammy – Test Manager
Detail requirement
Bob – Product Owner
1.5 Patricia confirms the project iteration plan
Confirm plan
Develop Resolve
Test Resolve
Monitor solution
Resolve request

Figure 5-9 Workflow for submitting a change request

Figure 5-10 shows the artifacts that are created.

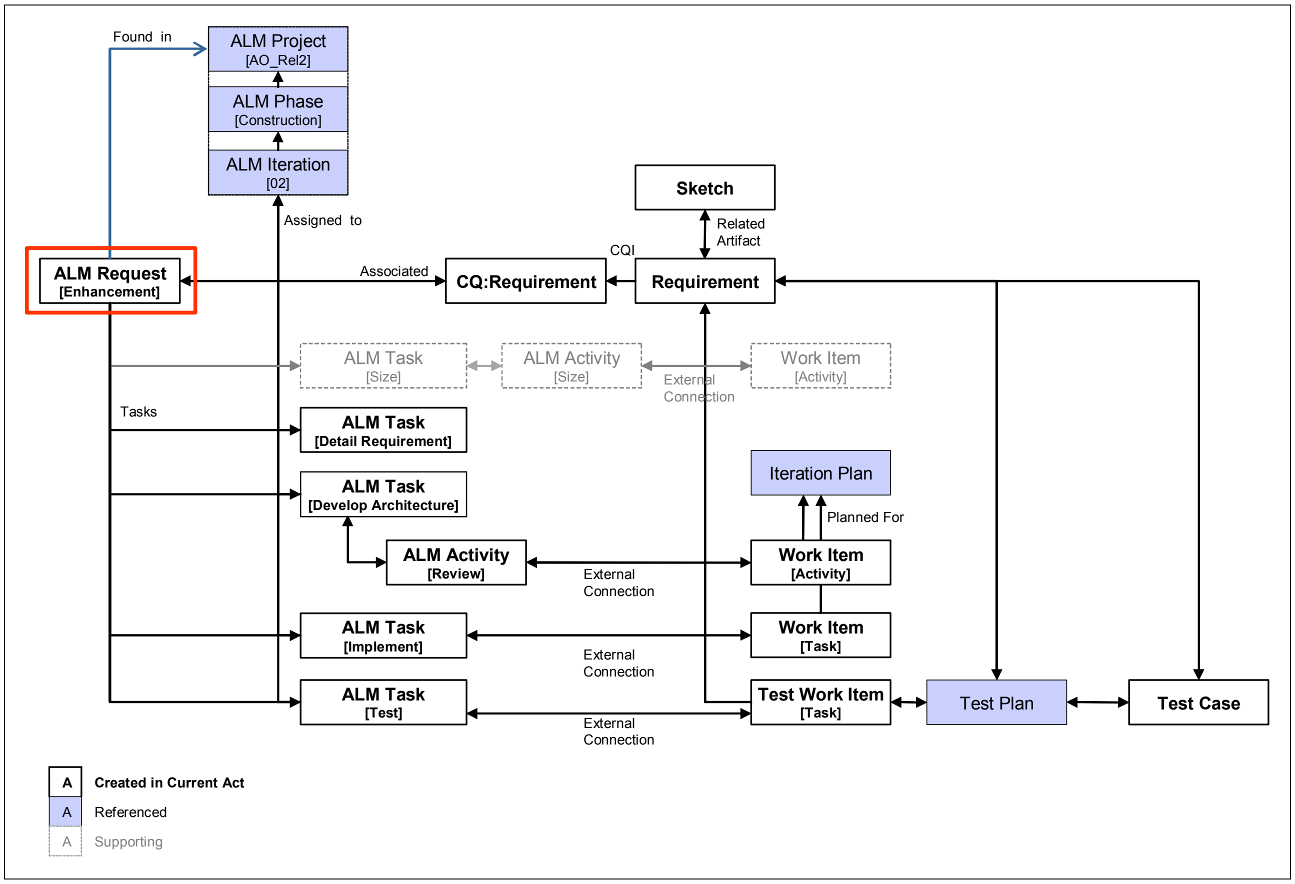


Figure 5-10 Artifacts created

Submitting an ALMRequest record

Goal: The goal is to keep track of change requests by submitting them to a centralized repository.

Bob, the product owner uses the Rational ClearQuest Web interface to accomplish his work. In the scenario, Bob responds to a UI branding change that needs to be submitted and tracked. He logs into Rational ClearQuest and submits a new record of type ALMRequest as explained in the following steps and shown in Figure 5-11 on page 130:

1. He browses to the Rational ClearQuest Web URL.
2. He enters the Rational ClearQuest login user name and password, selects a valid schema repository name, and clicks **Connect**.
3. He clicks the record type list and selects **ALMRequest**.

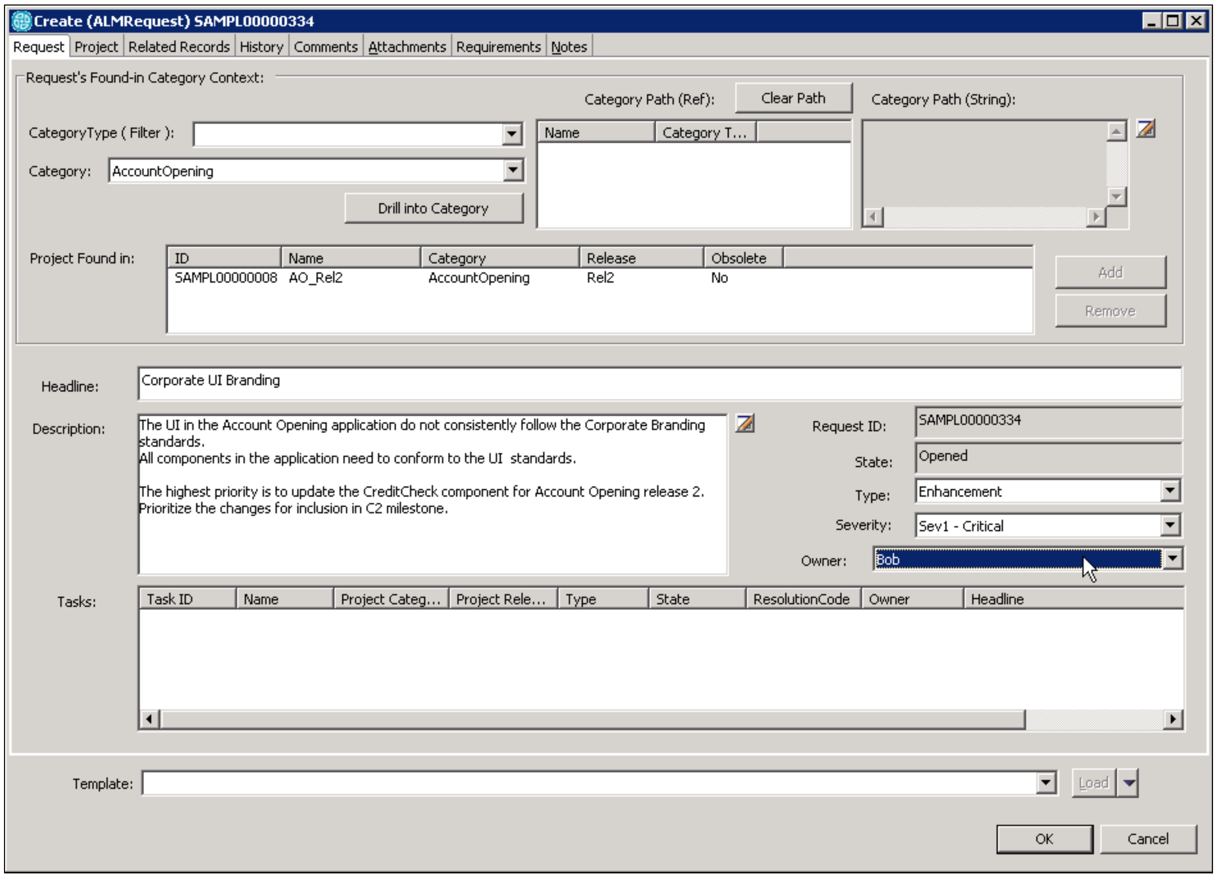
4. He completes all mandatory and optional fields:
   a. For Category, he chooses **AccountOpening**. The Project field is automatically populated with **AO_Rel2**.

   Tip: ALMCategory records can optionally identify a current project for that category. In this scenario, the AccountOpening category record identifies AO_Rel2 as its current project. When submitting a request, selecting the AccountOpening value for the Category field populates the Project Found In field with the current project from the ALMCategory record. Setting the current project on the category reduces the amount of information needed by a submitter and reduces the number clicks for submitting a request.

   b. For Type, he chooses **Enhancement**.
   c. For Severity, he chooses **Sev1 - Critical**.

   Tip: The packages also provide hooks that provide choice list hooks for other fields on the record such as Type and Severity. The items in the choice list are derived from the project in the Project Found In field. These lists are blank until a Project Found In field is identified on the record. This allows project teams to use different values without having to modify the Rational ClearQuest schema. These configurations are defined in the ClearQuest ALM work configurations that are discussed in 5.8.3, “Configuring Rational ClearQuest and the ALM schema” on page 194.

   d. For Owner, he chooses **Bob**.
   e. Optional: He switches to the **Project** tab and adds Patricia to the Notify List.
   f. He clicks **OK** to save the record.


Create (ALMRequest) SAMPL00000334

Request | Project | Related Records | History | Comments | Attachments | Requirements | Notes

Request's Found-in Category Context:

CategoryType (Filter):

Category: AccountOpening

Category Path (Ref):

Name	Category T...

Category Path (String):

Project Found in:

ID	Name	Category	Release	Obsolete
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	No

Headline: Corporate UI Branding

Description: The UI in the Account Opening application do not consistently follow the Corporate Branding standards.
All components in the application need to conform to the UI standards.

The highest priority is to update the CreditCheck component for Account Opening release 2.
Prioritize the changes for inclusion in C2 milestone.

Request ID: SAMPL00000334

State: Opened

Type: Enhancement

Severity: Sev1 - Critical

Owner: Bob

Tasks:

Task ID	Name	Project Categ...	Project Rele...	Type	State	ResolutionCode	Owner	Headline

Template:

Figure 5-11 New ALMRequest record being submitted by ClearQuest

Upon saving this ALMRequest record, Bob is done providing his release requirements to the project team. We now switch to Patricia, the project manager, in the reference scenario.

5.5.2 Patricia updates the project iteration plan

Synopsis: Patricia, the Account Opening project manager, is responsible for the overall project iteration plan and the project team coordination. She and her leadership team are triaging new requests on the Account Opening project. She reviews the UI branding request, which was submitted by Bob, and agrees that the work has a high priority and must be completed in this iteration. She realizes this request impacts her project plan, the CreditCheck component plan, and the test plan. She must create tasks to plan, assign, and align the work across the team in order to implement the request.

Patricia assesses the properties of the request, such as size, priority, and risk. She ascertains that the depth of this request is not difficult. As a result, no type of formalized sizing activities must occur by her team leads.

Patricia updates her project plan by creating and assigning new tasks for the impacted team leads to plan. She assigns a task to Marco for him to plan his implementation, a task to Tammy for her to update her test plan, and a task to Bob to define the requirements in further detail.

This scene involves the following tasks:

- Updating the project iteration plan
- Assigning work

Figure 5-12 highlights the workflow.

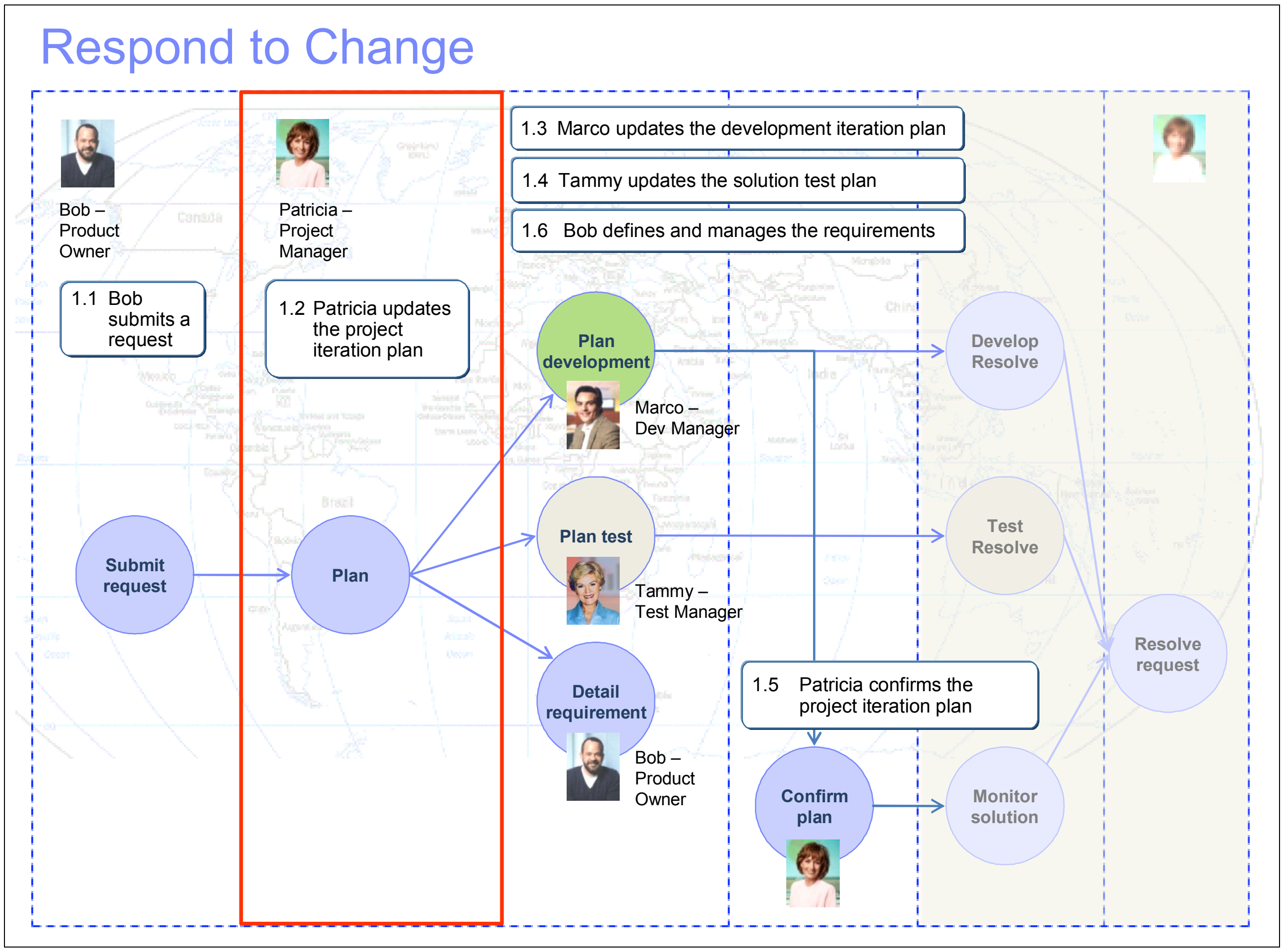

Respond to Change
Bob – Product Owner
Patricia – Project Manager
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements
Plan development
Marco – Dev Manager
Plan test
Tammy – Test Manager
Detail requirement
Bob – Product Owner
Submit request
Plan
Develop Resolve
Test Resolve
Resolve request
1.5 Patricia confirms the project iteration plan
Confirm plan
Monitor solution

Figure 5-12 Patricia updating the project iteration plan

Figure 5-13 shows the artifacts that are created.

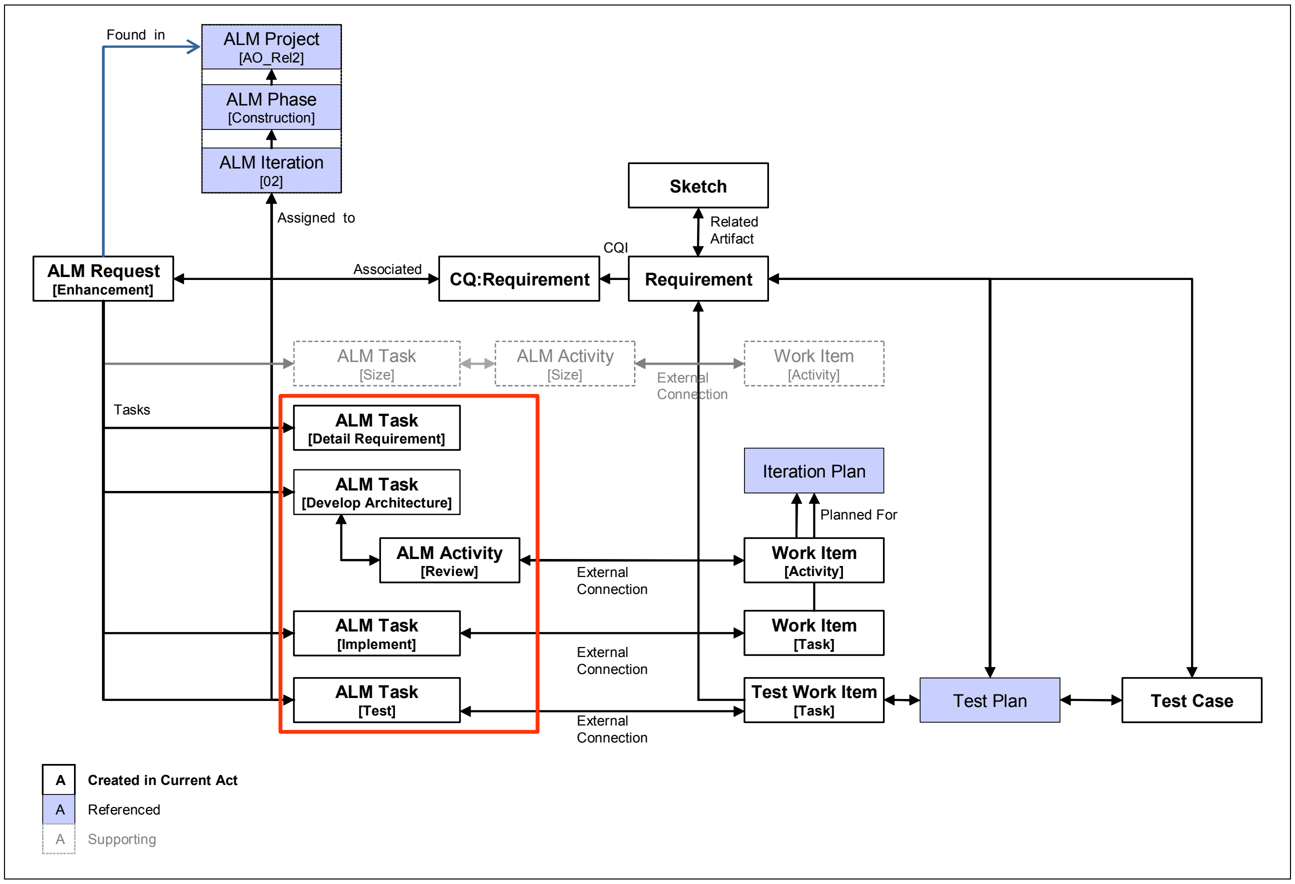


Figure 5-13 Artifacts created

Triaging ALMRequest records

Goal: The goal is to update the iteration plan and phase for the request record that is submitted.

Patricia receives an e-mail notification that a new ALMRequest has been assigned to the project. The e-mail contains record details and the record ID for the ALMRequest that Bob submitted to the project. Patricia also triages new submitted requests for the AccountOpening application and the current AO_Rel2 project on a regular basis.

To triage new requests, Patricia performs the following steps:

1. She opens her ClearQuest Eclipse (or Web) client and logs in.
2. She runs the **Public Queries** → **ALM** → **My Project** → **Triage**. For the dynamic filters, she chooses the **AccountOpening** category or the **AO_Rel2** project. This query can also be copied to her Personal Queries folder and further customized.
3. She clicks the new ALMRequest that Bob submitted to view the record form.

Patricia reads the details and determines that her team does not need to do any sizing work to assess whether the change will fit in the scope of the second construction iteration. She simply updates the iteration plan to reflect this.

Creating ALMTask records for iteration planning

Goal: The goal is to create the additional assets that are needed for planning the work to implement the change request.

Patricia now creates new ALMTask records to ensure that the teams start to plan the work for the iteration.

To plan work, Patricia performs the following steps:

1. She opens the ALMRequest record for viewing.
2. She clicks the **Utility** button and selects the **CreateTask** action (Figure 5-14 on page 134).

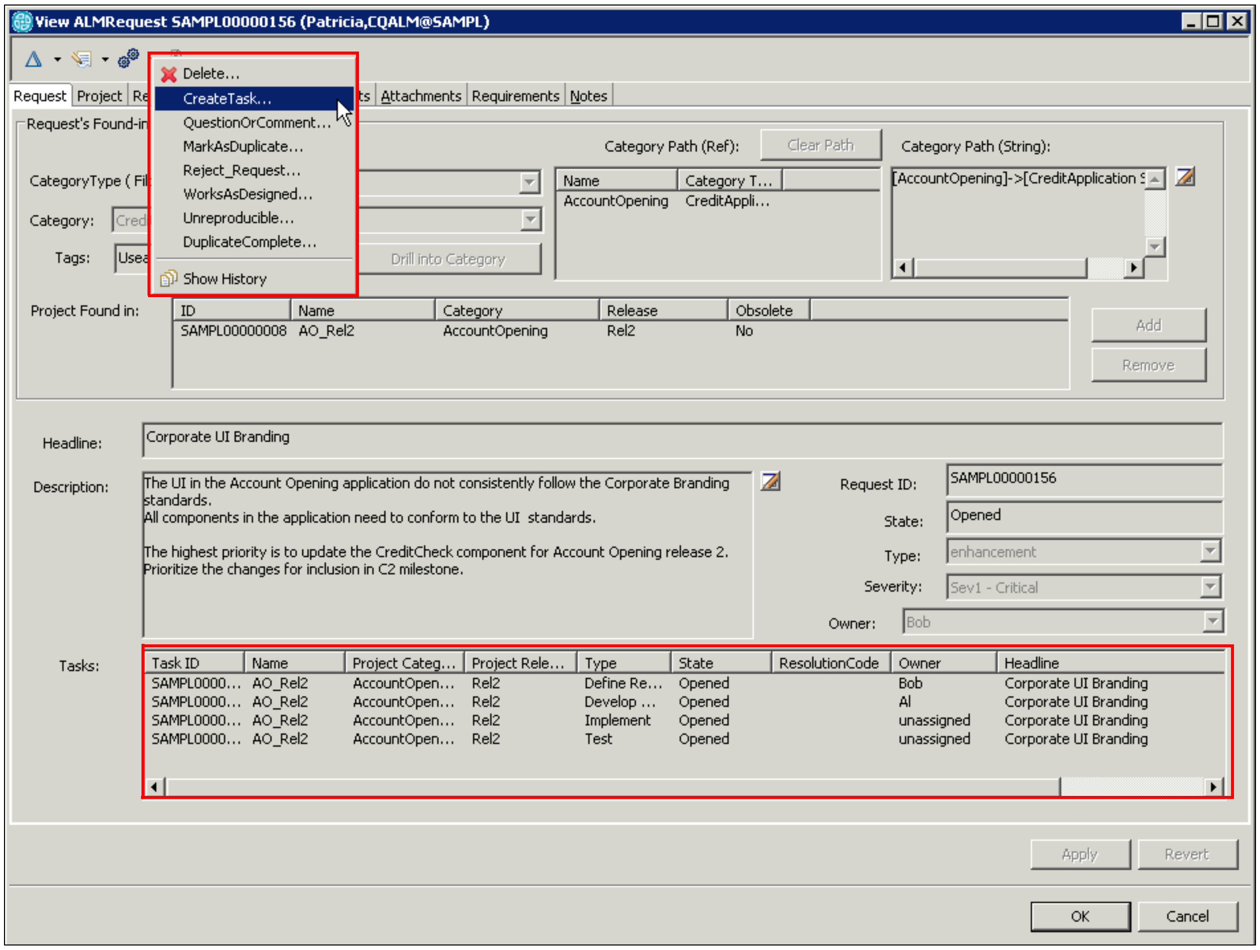
   This action creates new ALMTask records that are required to deliver on the request. The project uses OpenUP, and the created tasks specify work related to requirements, architecture, implementation, and testing.

   After the ALMRequest record is successfully updated, the editor is closed. She reopens the ALMRequest record to view and browse the new tasks.
3. Patricia views the new tasks that are added to the request.

   Four tasks are created. The first task, which is assigned to Bob, is of type *Define Requirements*. The second task, which is assigned to Al, is of type *Develop Architecture*. There is one implementation task and one test task, each of which is unassigned. Patricia focuses on the two unassigned tasks. She intends to assign the implementation of the request to Marco's team and the test task to Tammy's team.

To assign the implementation task, Patricia performs the following steps:

1. She double-clicks the newly created ALMTask record of type **Implement** from the Tasks field to open it for viewing.
2. She clicks the **Modify** button on the record form to put the ALMTask record in a modifiable state.
3. She completes all mandatory and optional fields:
   a. On the Description field, she adds optional instructions for the implementation.
   b. For Priority, she chooses **Prio1 - Urgent**.
   c. For Owner, she chooses **Marco**.
   d. She switches to the **Project** tab and validates that the task has been planned for the current iteration, that is phase Construction and iteration 02. To modify the plan, she chooses from the Phase Assigned To and Iteration Assigned To fields.
4. She clicks the **Apply** button to save her changes.



CategoryType (Fil

Category: Credi

Tags: Usea

Name	Category T...
AccountOpening	CreditAppli...

Category Path (String): [AccountOpening]->[CreditApplication S

Project Found in:

ID	Name	Category	Release	Obsolete
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	No

Headline: Corporate UI Branding

Description: The UI in the Account Opening application do not consistently follow the Corporate Branding standards.
All components in the application need to conform to the UI standards.

The highest priority is to update the CreditCheck component for Account Opening release 2.
Prioritize the changes for inclusion in C2 milestone.

Request ID:	SAMPL00000156
State:	Opened
Type:	enhancement
Severity:	Sev1 - Critical
Owner:	Bob

Tasks:

Task ID	Name	Project Categ...	Project Rele...	Type	State	ResolutionCode	Owner	Headline
SAMPL0000...	AO_Rel2	AccountOpen...	Rel2	Define Re...	Opened		Bob	Corporate UI Branding
SAMPL0000...	AO_Rel2	AccountOpen...	Rel2	Develop ...	Opened		Al	Corporate UI Branding
SAMPL0000...	AO_Rel2	AccountOpen...	Rel2	Implement	Opened		unassigned	Corporate UI Branding
SAMPL0000...	AO_Rel2	AccountOpen...	Rel2	Test	Opened		unassigned	Corporate UI Branding

Figure 5-14 CreateTask action creating new ALMTask records

To assign the test task, Patricia performs the following steps:

1. She double-clicks the newly created ALMTask record of type **Test** from the Tasks field to open it for viewing.
2. She clicks the **Modify** button on the record form to put the ALMTask record in a modifiable state and completes all mandatory and optional fields:

   a. In the Description field, she adds optional instructions for the test planning.
   a. For Priority, she chooses **Prio1 - Urgent**.
   b. For Owner, she chooses **Tammy**.
   c. For phase and iteration, she chooses **Construction** and **02**.

3. She clicks the **Apply** button to save her changes.

The Define Requirements and Develop Architecture tasks have already been assigned to Bob and Al as part of the default role in the OpenUP and project work configurations. Patricia can optionally open the records and provide additional documentation, or set notification, to Bob and Al in their task records.

Creating an ALMActivity record for reviews

Goal: The goal is to create the additional assets that are needed for assigning work to team members.

Patricia want to ensure that any architectural changes from the UI branding are reviewed and approved by the component team. To ensure this process, she open the Develop Architecture task that is assigned to Al. She then adds a review task that is assigned to Diedrie.

To create and assign an Activity, Patricia performs the following steps:

1. She double-clicks the ALMTask of type **Develop Architecture** that is assigned to Al.
2. She clicks the **Utility** button and selects the **CreateActivity** action. One or more ALMActivities are added to the ALMTask.
3. She double-clicks the ALMActivity of type **Outline Architecture**, and updates the following fields:
   a. She updates the Type field to **Review**.
   b. In the Owner field, she assigns the activity to Diedrie.
   c. In the Description field, she adds optional instructions for review and approval.
   d. Optional: On the **Related Records** tab, she adds a reference to the ALMActivity of type **Refine Architecture**.
4. She clicks **OK** to save the changes.

Upon completion of assigning the ALMTask records, Patricia has completed the triage process, and each member of the team can begin to do iteration planning. Both Marco and Tammy are notified that they have new work assigned to them as news feeds or e-mail, or by using the query capabilities in their respective tools.

5.5.3 Marco updates the development iteration plan

Synopsis: Marco has received new implementation work for the change request and needs to plan the work. He updates the iteration plan with the new work.

This scene involves the following tasks:

- Reviewing, planning, and assigning work related to Bob’s request
- Forming, organizing, and balancing the iteration plan

Figure 5-15 highlights the workflow.

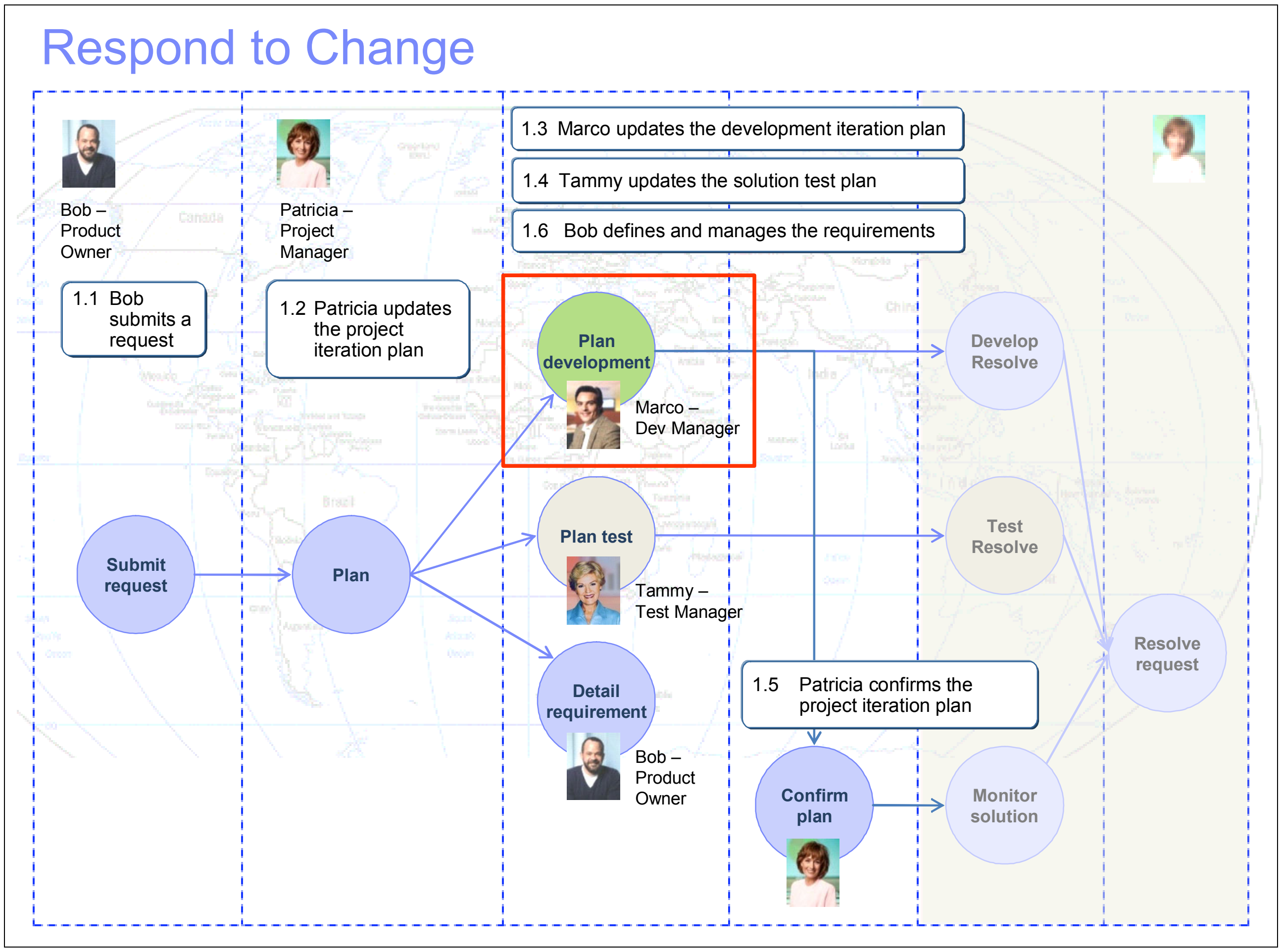

Respond to Change
Bob – Product Owner
Patricia – Project Manager
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements
Plan development
Marco – Dev Manager
Develop Resolve
Submit request
Plan
Plan test
Tammy – Test Manager
Test Resolve
Resolve request
Detail requirement
Bob – Product Owner
1.5 Patricia confirms the project iteration plan
Confirm plan
Monitor solution

Figure 5-15 Marco updating the iteration plan

Figure 5-16 shows the artifacts that are created.

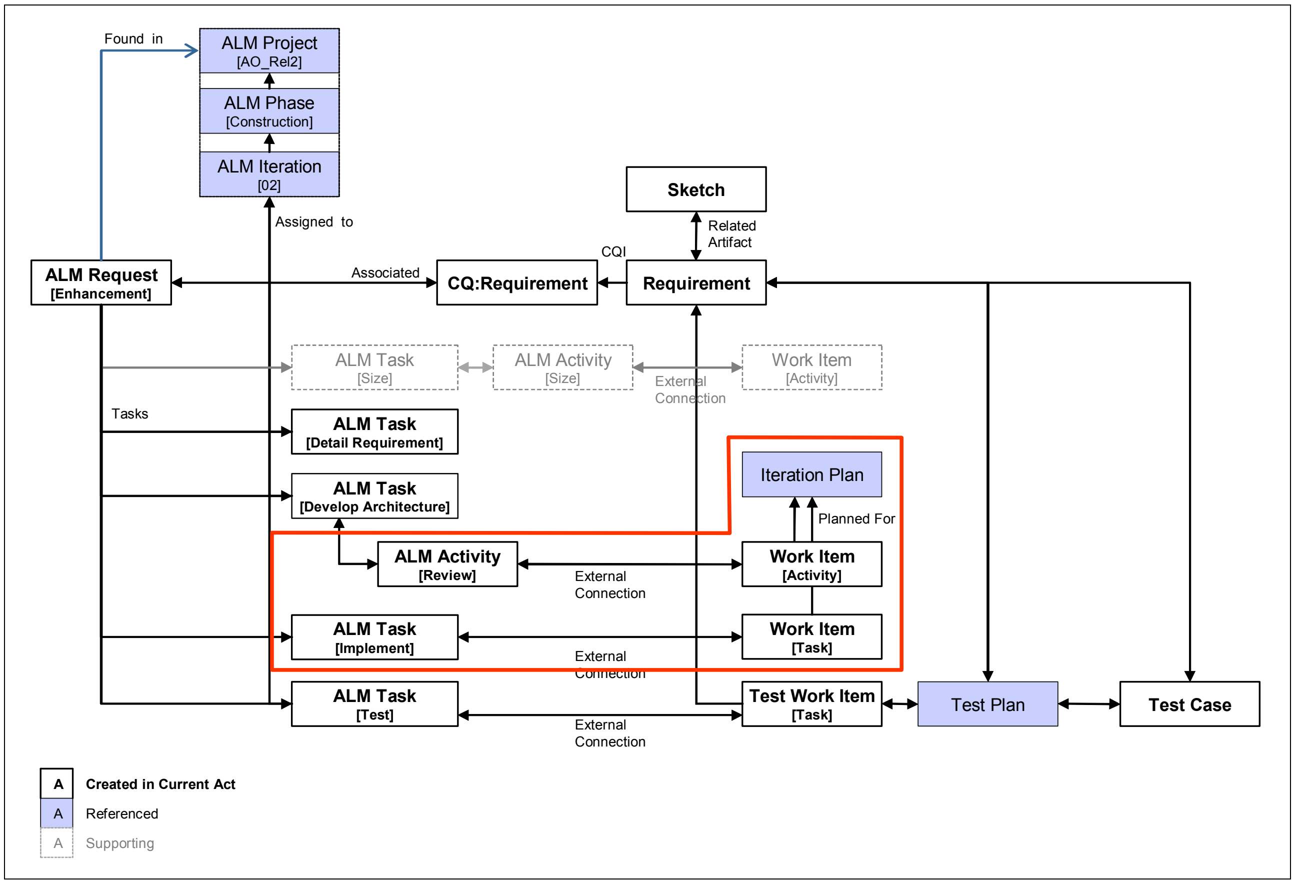


Figure 5-16 Artifacts created

Updating the iteration plan for the work item

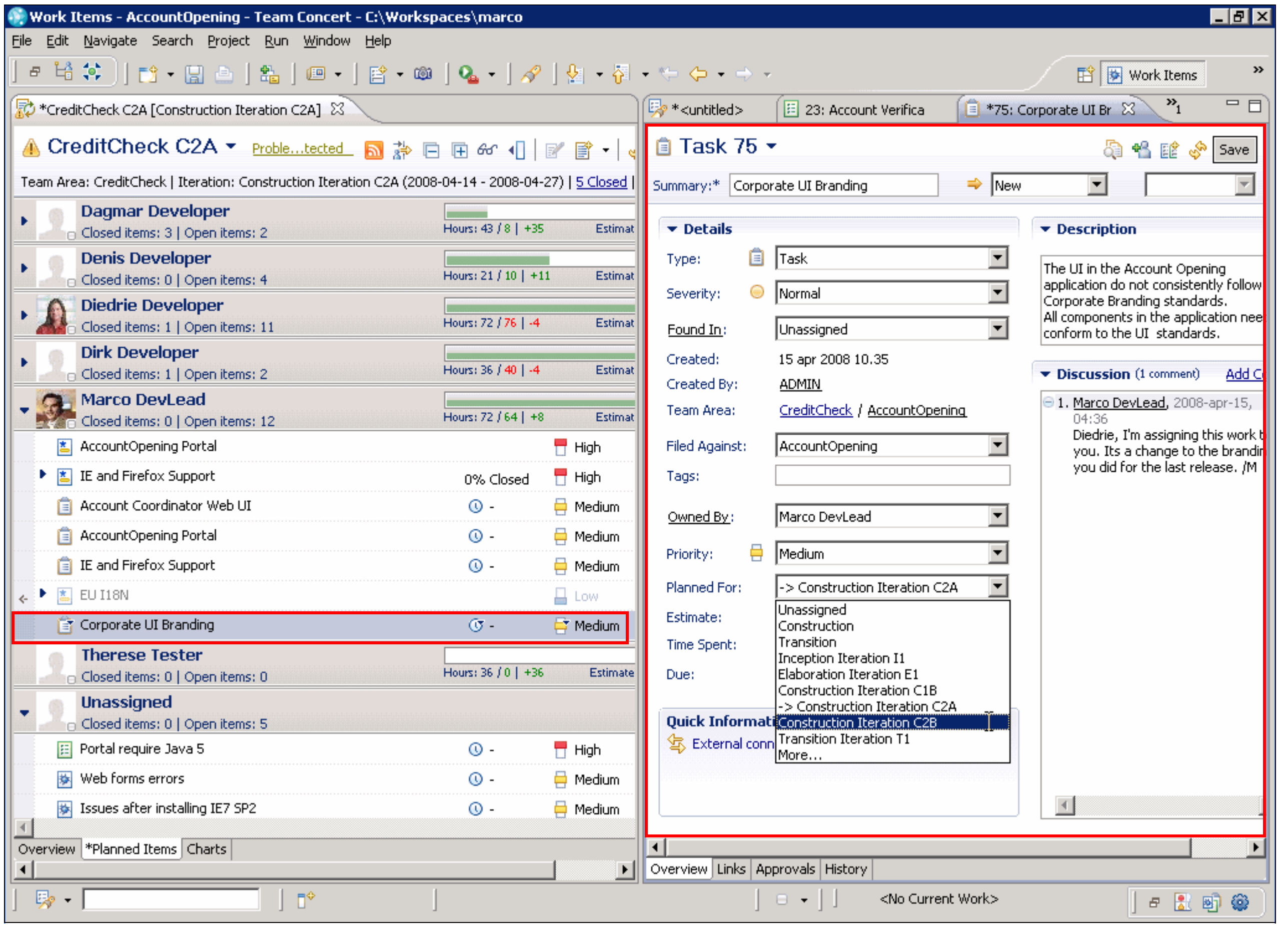
Goal: The goal is to update the iteration plan and assign the new work item which has been assigned to the team.

Marco is triaging new work for the current iteration. He opens Rational Team Concert and can see that a new work item is assigned to him in the iteration plan. He is also notified of this in the RSS feeds event log.

To triage new work items, Marco performs the following steps (see Figure 5-17 on page 138):

1. He opens the current iteration plan by selecting **AccountOpening** → **Plans** → **Construction Iteration C2A** → **CreditCheck C2A [Construction Iteration C2A]** in the Team Artifacts view.
2. He looks for new work items that are assigned to himself or are *Unassigned*.
3. He opens the new work item.

4. In the Work Item editor:
   a. He clicks the Planned For list for the work item and selects the **Construction Iteration C2B** value.
   b. He clicks the Priority list for the work item and selects the **Medium** value.
   c. He clicks **Owner** and assigns the work item to **Diedrie**.
   d. He creates an entry in the Discussions field to document the triage decision and to provide guidance to Diedrie.
5. He clicks the **Save** button to apply the changes to the work item.


Work Items - AccountOpening - Team Concert - C:\Workspaces\marco
File Edit Navigate Search Project Run Window Help
Work Items
*CreditCheck C2A [Construction Iteration C2A]
*<untitled>
23: Account Verifica
*75: Corporate UI Br
Overview
*Planned Items
Charts
Links
Approvals
History
<No Current Work>

CreditCheck C2A Proble...tected

Team Area: CreditCheck | Iteration: Construction Iteration C2A (2008-04-14 - 2008-04-27) | 5 Closed |

Dagmar Developer Closed items: 3 | Open items: 2		Hours: 43 / 8 | +35	Estimat
Denis Developer Closed items: 0 | Open items: 4		Hours: 21 / 10 | +11	Estimat
Diedrie Developer Closed items: 1 | Open items: 11		Hours: 72 / 76 | -4	Estimat
Dirk Developer Closed items: 1 | Open items: 2		Hours: 36 / 40 | -4	Estimat
Marco DevLead Closed items: 0 | Open items: 12		Hours: 72 / 64 | +8	Estimat
AccountOpening Portal			High
IE and Firefox Support	0% Closed		High
Account Coordinator Web UI	-		Medium
AccountOpening Portal	-		Medium
IE and Firefox Support	-		Medium
EU I18N			Low
Corporate UI Branding	-		Medium
Therese Tester Closed items: 0 | Open items: 0		Hours: 36 / 0 | +36	Estimate
Unassigned Closed items: 0 | Open items: 5			
Portal require Java 5	-		High
Web forms errors	-		Medium
Issues after installing IE7 SP2	-		Medium

Task 75 Save

Summary:* Corporate UI Branding New

Details

Type:	Task
Severity:	Normal
Found In:	Unassigned
Created:	15 apr 2008 10.35
Created By:	ADMIN
Team Area:	CreditCheck / AccountOpening
Filed Against:	AccountOpening
Tags:	
Owned By:	Marco DevLead
Priority:	Medium
Planned For:	-> Construction Iteration C2A
Estimate:	
Time Spent:	
Due:	

Planned For list: Unassigned, Construction, Transition, Inception Iteration I1, Elaboration Iteration E1, Construction Iteration C1B, -> Construction Iteration C2A, Construction Iteration C2B, Transition Iteration T1, More...

Quick Informati

External conn

Description

The UI in the Account Opening application do not consistently follow Corporate Branding standards. All components in the application nee conform to the UI standards.

Discussion (1 comment) Add C

1. Marco DevLead, 2008-apr-15, 04:36
Diedrie, I'm assigning this work t you. Its a change to the brandi you did for the last release. /M

Figure 5-17 Marco triaging and assigning the UI branding work item to Diedrie

These steps exemplify one way to assign work to a team member. Other options are to drag work items in the iteration plan or to right-click a work item and select **Assign to Owner**.

For information about additional tasks that Marco and his team must do and the artifacts that are created, see 7.3, “Rational Team Concert for agile development” on page 246.

5.5.4 Tammy updates the solution test plan

Synopsis: Tammy receives a task from Patricia indicating that there is a new UI change request. She reviews the task and updates the solution test plan to ensure that the UI will be tested by her team.

This scene involves the following tasks:

- Reviewing the work related to Bob's request
- Updating the test plan
- Reviewing the test plan
- Submitting a request to configure lab resources
- Updating and closing the task

Figure 5-18 highlights the workflow.

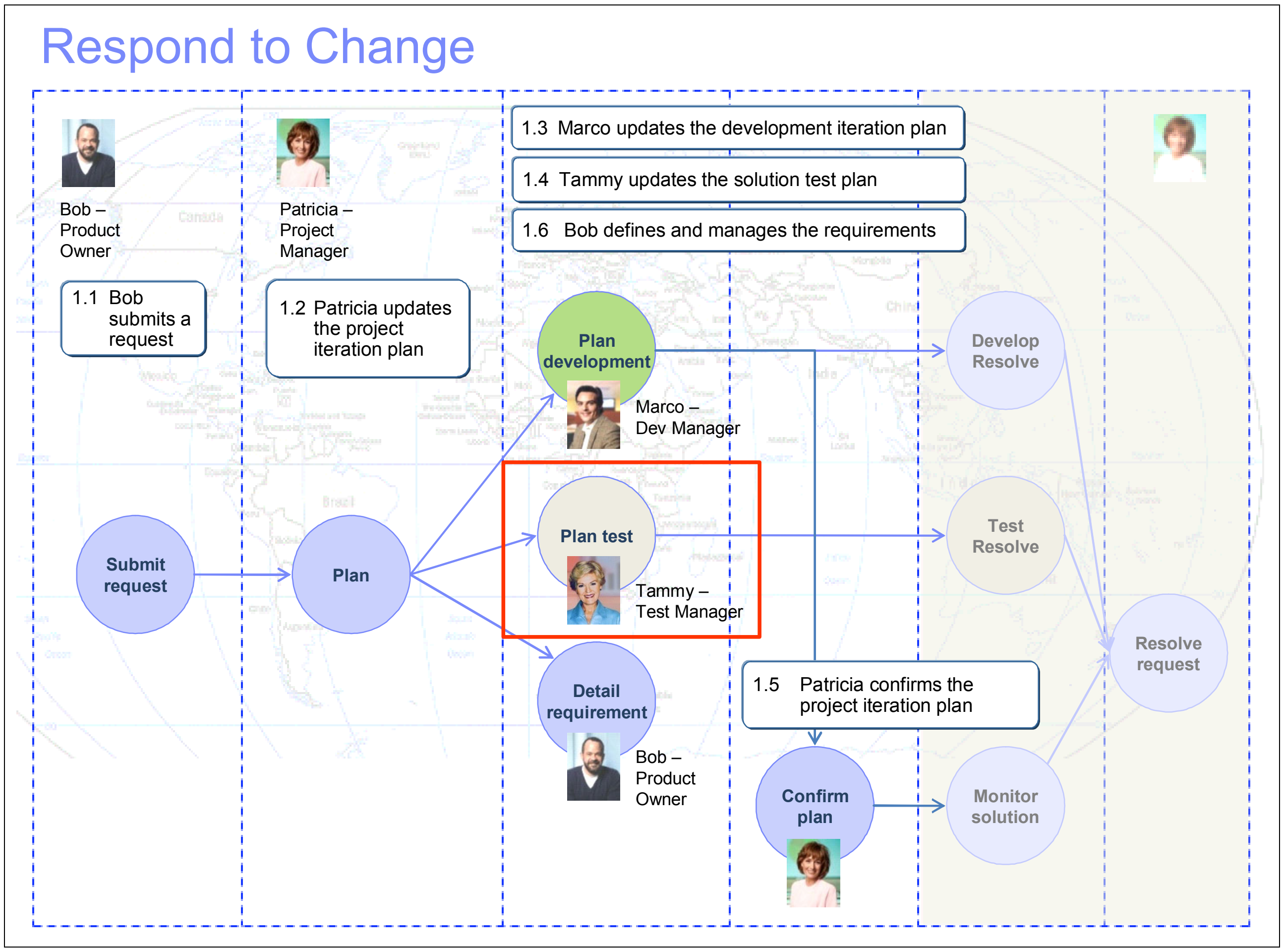

Respond to Change
Bob – Product Owner
Patricia – Project Manager
1.1 Bob submits a request
1.2 Patricia updates the project iteration plan
1.3 Marco updates the development iteration plan
1.4 Tammy updates the solution test plan
1.6 Bob defines and manages the requirements
Plan development
Marco – Dev Manager
Develop Resolve
Submit request
Plan
Plan test
Tammy – Test Manager
Test Resolve
Resolve request
Detail requirement
Bob – Product Owner
1.5 Patricia confirms the project iteration plan
Confirm plan
Monitor solution

Figure 5-18 Tammy updating the solution test plan

Figure 5-19 shows the artifacts that are created.

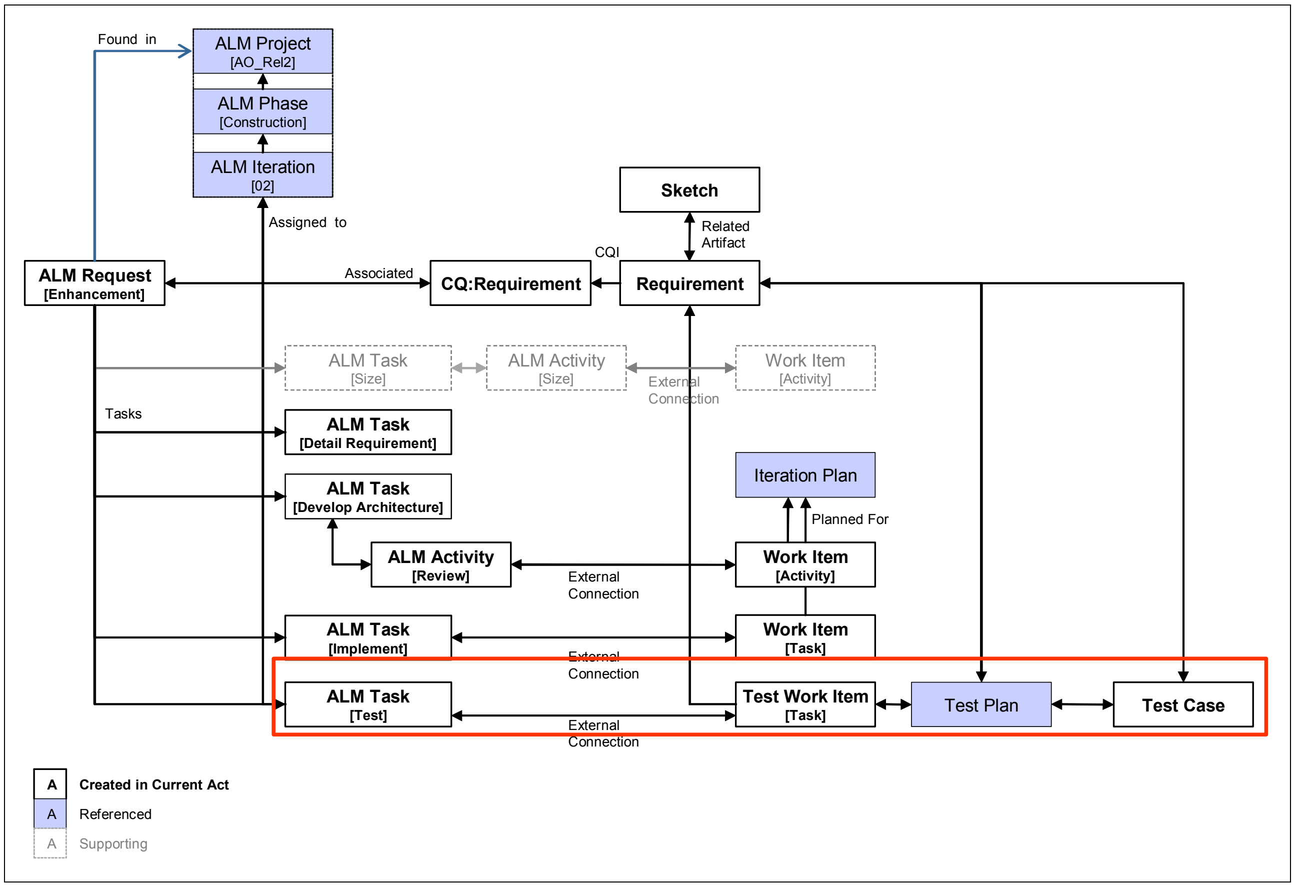


Figure 5-19 Artifacts created

Reviewing the work related to Bob's request

Tammy logs into Rational Quality Manager. From the dashboard, she sees that a new task is assigned to her. She is also notified of this in the RSS feeds event log.

To review the task, Tammy performs the following steps:

1. She clicks the **Home** tab to view her dashboard.
2. In the My Tasks viewlet, she identifies the new task from Patricia (Figure 5-20).

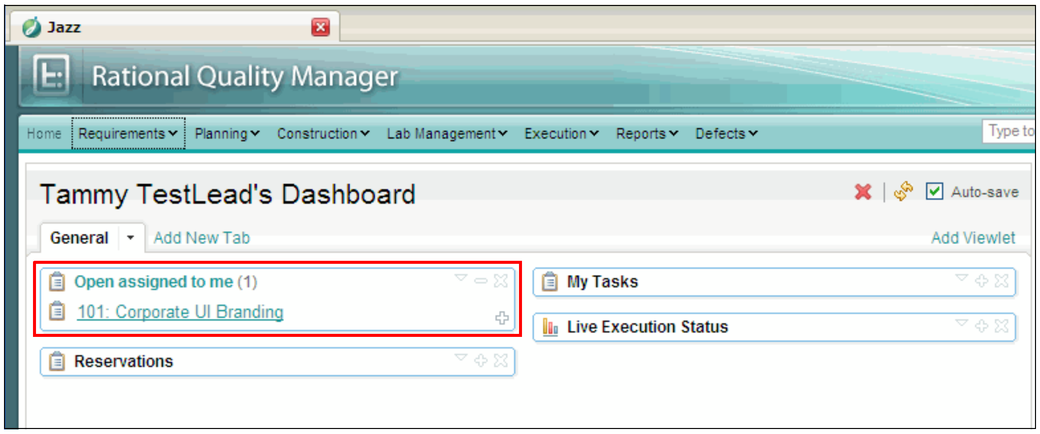
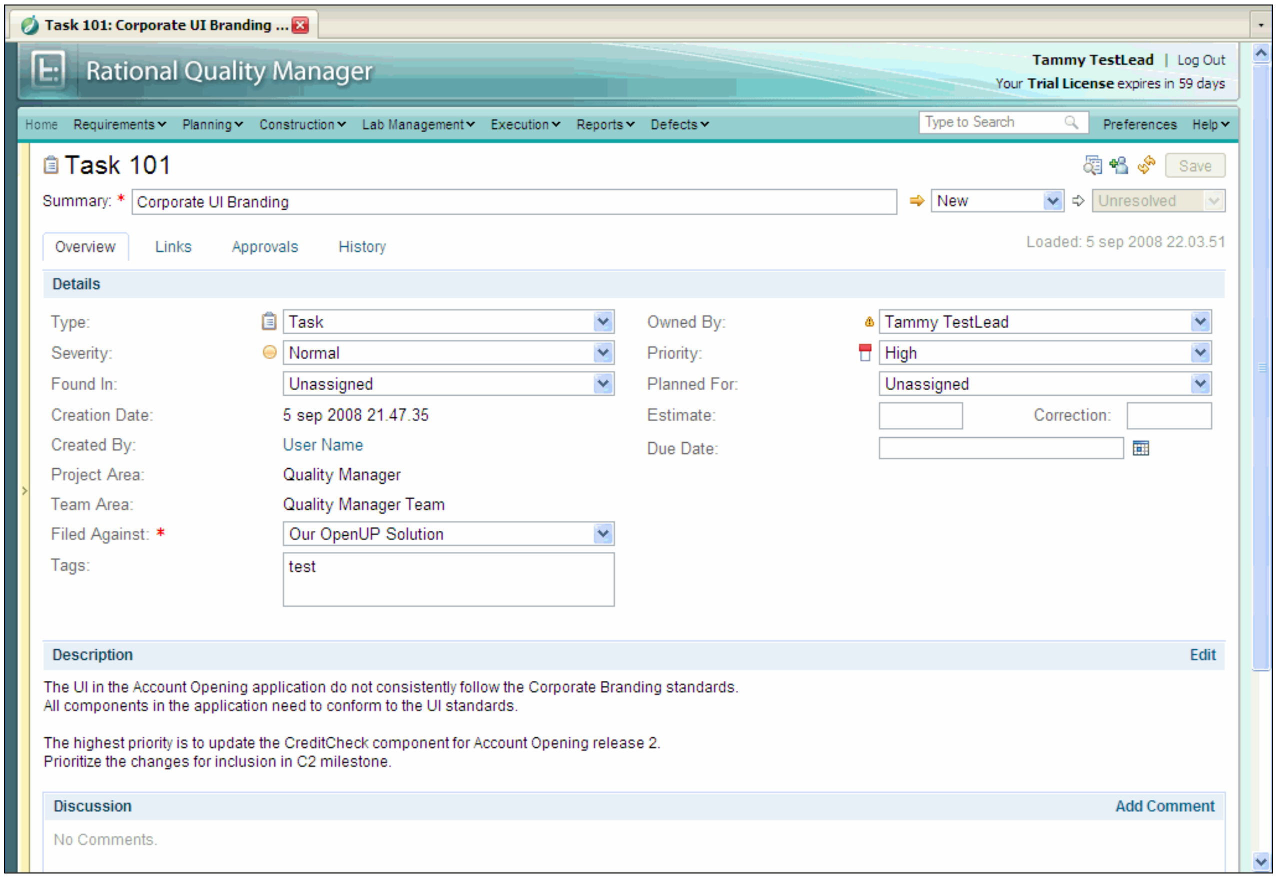


Figure 5-20 A dashboard viewlet showing tasks

3. She clicks the **101: Corporate UI Branding** task to review and edit its contents, which are shown in Figure 5-21.


Task 101: Corporate UI Branding ...
Rational Quality Manager
Tammy TestLead | Log Out
Your Trial License expires in 59 days
Home Requirements Planning Construction Lab Management Execution Reports Defects
Type to Search
Preferences Help
Save
Overview Links Approvals History
Edit
Add Comment

Task 101

Summary: * Corporate UI Branding | New | Unresolved

Loaded: 5 sep 2008 22.03.51

Details

Type:	Task	Owned By:	Tammy TestLead
Severity:	Normal	Priority:	High
Found In:	Unassigned	Planned For:	Unassigned
Creation Date:	5 sep 2008 21.47.35	Estimate:	Correction:
Created By:	User Name	Due Date:	
Project Area:	Quality Manager		
Team Area:	Quality Manager Team		
Filed Against: *	Our OpenUP Solution		
Tags:	test		

Description

The UI in the Account Opening application do not consistently follow the Corporate Branding standards.
All components in the application need to conform to the UI standards.

The highest priority is to update the CreditCheck component for Account Opening release 2.
Prioritize the changes for inclusion in C2 milestone.

Discussion

No Comments.

Figure 5-21 Viewing a task in Rational Quality Manager

Updating the solution test plan

Goal: The goal is to update the solution test plan and assign the new work item that has been assigned to the team.

Tammy must update her test plan to ensure that the team tests the new request. To do so, she adds a test case and updates the sizing and exit criteria in her test plan. She performs the following steps:

1. She locates and opens the test plan for the Account Opening project. In the left navigation menu, she hovers the mouse pointer over **Planning** and selects **My Test Plans** from the menu (Figure 5-22).

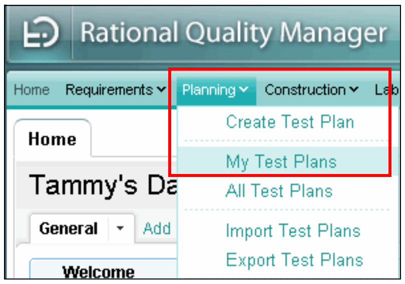


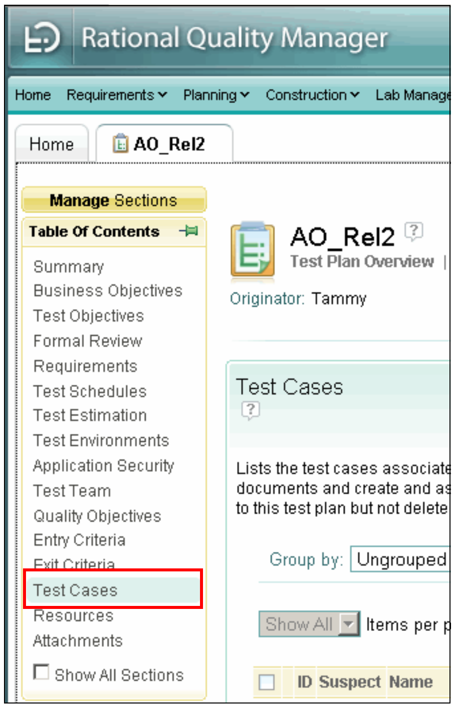
Figure 5-22 Running the 'My Test Plans' query

A query runs to locate all of Tammy's test plans. Then a new tab is added to Tammy's workspace with the list of available test plans.

2. She clicks the **Account Opening** test plan to open it. At this point, she has little information other than a new enhancement request is added to the iteration. Her goal is to ensure that it will be tested by adding placeholders to the test plan.

3. She adds and assigns a test case to the test plan.

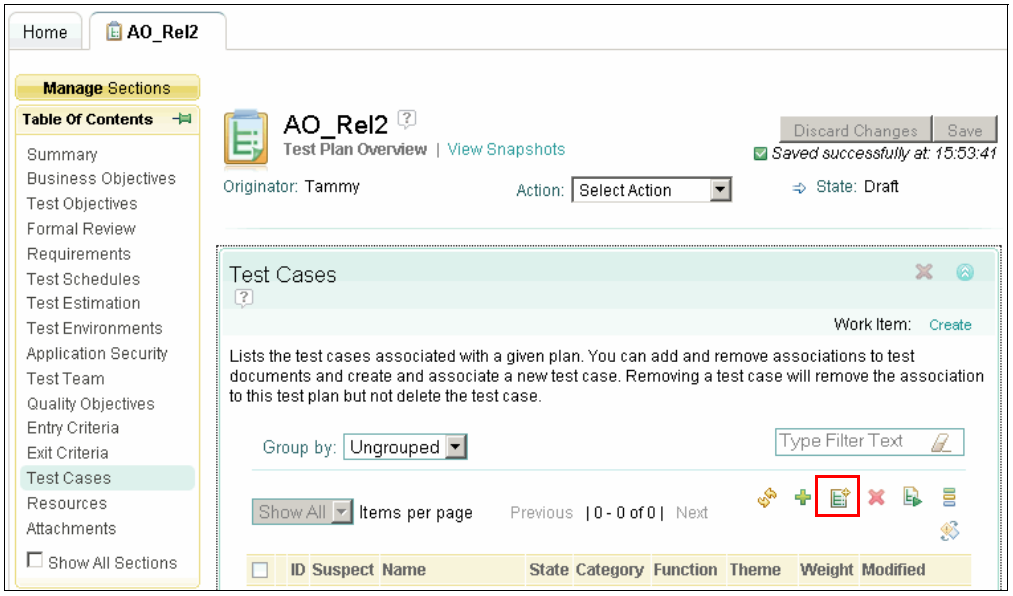
   a. In the Test Plan Table of Contents, she clicks **Test Cases** (Figure 5-23).



Lists the test cases associate
documents and create and as
to this test plan but not delete

Figure 5-23 Test Cases menu item in the Test Plan Table of Contents

b. Tammy clicks the **Add Test Case** button as highlighted in Figure 5-24. The test cases included in this test plan are visible.



Lists the test cases associated with a given plan. You can add and remove associations to test documents and create and associate a new test case. Removing a test case will remove the association to this test plan but not delete the test case.

Figure 5-24 Adding a new test case in the test plan

c. In the New Test Case window, Tammy enters the following details to capture the test case. Figure 5-25 shows the original version of this window before she enters the details.

i. For Name, she types `UI Corporate Branding`.

ii. For Description, she pastes the description from the task that Patricia assigned into the Description field.

iii. For Weight, she enters `1`.

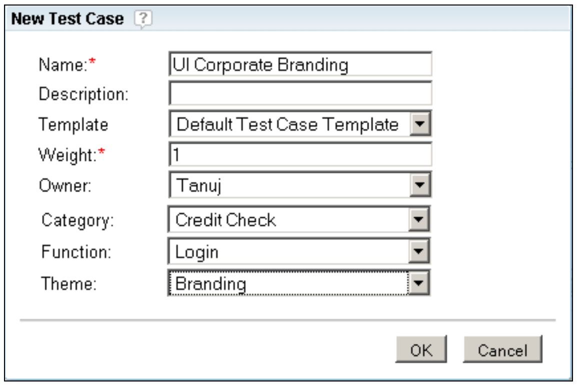
iv. For Owner, she selects **Tanuj**.

v. For Theme, she selects **Branding**.

vi. For Category, she selects **Credit Check**.

vii. For Function, she selects **Login**.

She clicks **Save** to create the test case.



New Test Case

Name:*	UI Corporate Branding
Description:	
Template	Default Test Case Template
Weight:*	1
Owner:	Tanuj
Category:	Credit Check
Function:	Login
Theme:	Branding

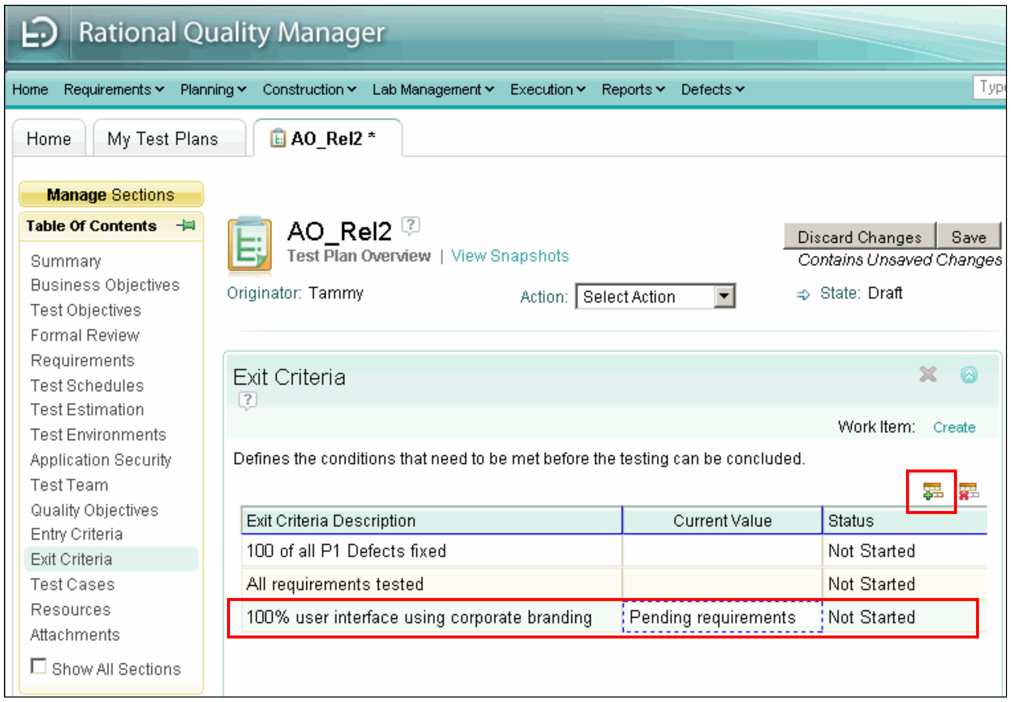
Figure 5-25 New Test Case window

The window closes, and the test case is added to the plan. Note that the test case is listed in plain text and is not a link.

4. She saves the test plan by clicking the **Save** button in the upper right corner of the Test Plan tab. The new test case becomes a linked item in the test plan.

Note: In Act 4, Tammy updates the requirements in the test plan by linking to the requirements that Bob creates at the end of this act. She also creates a link between the requirement and the test case that she just created.

5. Tammy updates the exit criteria to ensure that the team does not release the software without implementing and testing this change.
   a. She clicks the **Account Opening Test Plan** tab to view the test plan.
   b. In the Table of Contents, she clicks **Exit Criteria** (Figure 5-26). The exit criteria for this plan is now visible.
   c. She clicks the **Add Row** button in the Exit Criteria user interface, which is highlighted with red square on the right side in Figure 5-26.



Originator: Tammy
Action: Select Action
State: Draft

Defines the conditions that need to be met before the testing can be concluded.

Exit Criteria Description	Current Value	Status
100 of all P1 Defects fixed		Not Started
All requirements tested		Not Started
100% user interface using corporate branding	Pending requirements	Not Started

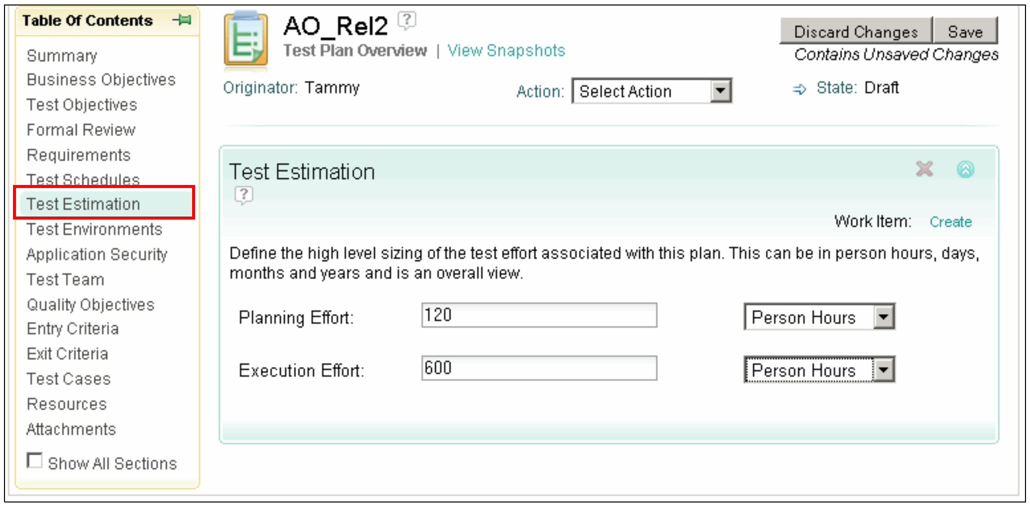
Figure 5-26 Exit Criteria managed in a test plan

   d. For Exit Criteria, she enters the following values:
      i. In the row that is added with the text preselected, Tammy types `100% user interface using corporate branding`.
      ii. In the Current Value field, she types `Pending requirements`.
      iii. She leaves the Status field set as `Not Started`.
   e. Tammy clicks the **Save** button on the test plan. Tammy now has the following exit criteria in her test plan:
      - 100% of all Priority 1 Defects fixed
      - 100% requirements test coverage
      - 100% user interface using the corporate branding

Exit Criteria comes into play later in the scenario as Tammy monitors quality in 11.5.5, “Monitoring quality” on page 455, and in the final act when the team determines that the iteration is complete in 13.3.2, “The team leads assess their exit criteria” on page 504.

The new request adds additional work for her team. Therefore, Tammy reviews the estimate that she has provided for this iteration:

1. While viewing the Account Opening test plan, in Table of Contents, Tammy clicks **Test Estimation** (Figure 5-27).



Originator: Tammy

State: Draft

Test Estimation

Define the high level sizing of the test effort associated with this plan. This can be in person hours, days, months and years and is an overall view.

Planning Effort:	120	Person Hours
Execution Effort:	600	Person Hours

Figure 5-27 Adding sizing estimates to a test plan

The new request is a simple user interface test to confirm that the correct branding elements are used. This single request requires that the test team perform the following actions:

– Create the test cases.
– Determine how many test execution work items to run.
– Execute the test execution items.
– Evaluate the results and manage defects.
– Possibly update the test environments that are required for testing. For example, a new request might require adding another browser version to the testing matrix.

Tammy is using person hours to track the test effort and sees that her test architect has already sized this test effort. With a team of six people working on a four-week iteration, they estimate that they can have a total of 720 person hours (6 people x 30 hours a week x 4 weeks). Tammy and her team use 30 hours instead of 40 hours in their estimate to allow for 10 hours a week to account for e-mail, meetings, and other tasks outside of the test effort.

The planning estimate is currently 120 person hours. This estimates 20 hours per person for a six person team. Tammy includes test planning and test-case construction as part of the planning effort.

The execution effort is currently 600 person hours, which estimates 100 hours per person over the course of the four-week iteration.

2. Taking all of this into account, Tammy adds two hours to the Planning Effort field by changing the value to `122`.

3. Tammy adds two hours to the Execution Effort field by changing the value to `602`.

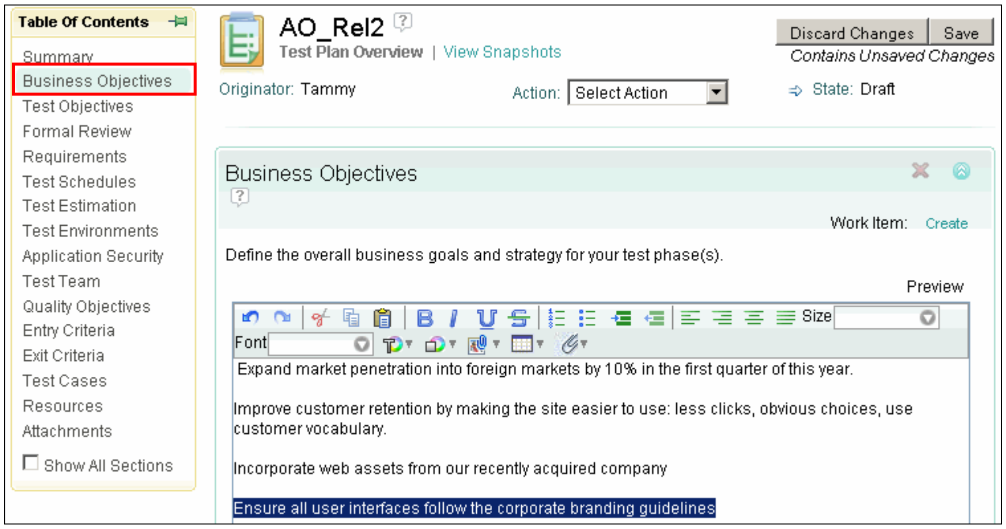
These estimates push the sizing effort beyond their target by a small amount. Tammy decides that the team can absorb this small change. Had the request required a larger amount of work, Tammy would have to reconfigure her plan to accommodate the request or work with the Patricia, Marco, and the other development leads to manage the scope of the iteration.

Reviewing the test plan

This is the second iteration for the project. Therefore, much of the test plan was completed in the first iteration. At this point, Tammy reviews the test plan to ensure that all changes for the second iteration have been addressed in the plan.

Tammy reviews the following in the test plan Table of Contents:

1. She clicks **Summary** and sees that there is no major change in the project that requires an update to the project summary.
2. She clicks **Business Objectives**. She decides to update the business objectives to include the objective of a re-branded user interface. After all, the solution is incorporating components from an acquired company, and all user interface elements must be updated to comply with their corporate standards.
   a. She reviews the business objectives and realizes that they do not reflect the need to re-brand the user interface.
   b. She clicks **Edit** to open the rich text editor.
   c. She adds a line that identifies the objective of a re-branded user interface (Figure 5-28).



Define the overall business goals and strategy for your test phase(s).

Expand market penetration into foreign markets by 10% in the first quarter of this year.

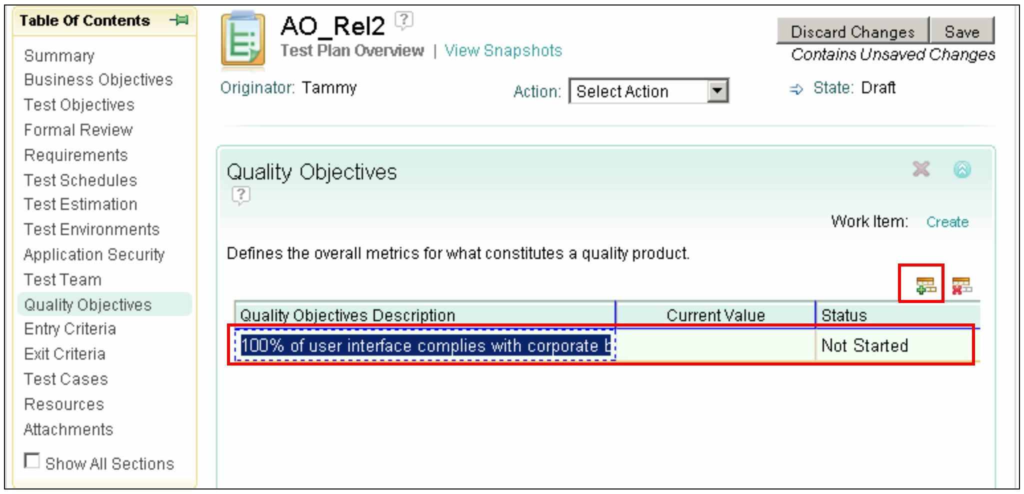
Improve customer retention by making the site easier to use: less clicks, obvious choices, use customer vocabulary.

Incorporate web assets from our recently acquired company

Ensure all user interfaces follow the corporate branding guidelines

Figure 5-28 Updating the Business Objectives in the test plan

   d. She clicks **Save**.
3. She click **Test Objectives** and reviews them. She realizes the shared objective of a re-branded UI. She repeats the steps from the Business Objectives section of the test plan.
4. She clicks **Quality Objectives** and reviews the Quality Objectives section of the test plan.
   a. She clicks **Quality Objectives** in the Table of Contents for the test plan.
   b. She decides to add the re-branded UI as a quality goal. She clicks the **Add Row** button and types a new quality goal of “`100% of user interface complies with corporate brand`” as shown in Figure 5-29 on page 147.



Defines the overall metrics for what constitutes a quality product.

Quality Objectives Description	Current Value	Status
100% of user interface complies with corporate b		Not Started

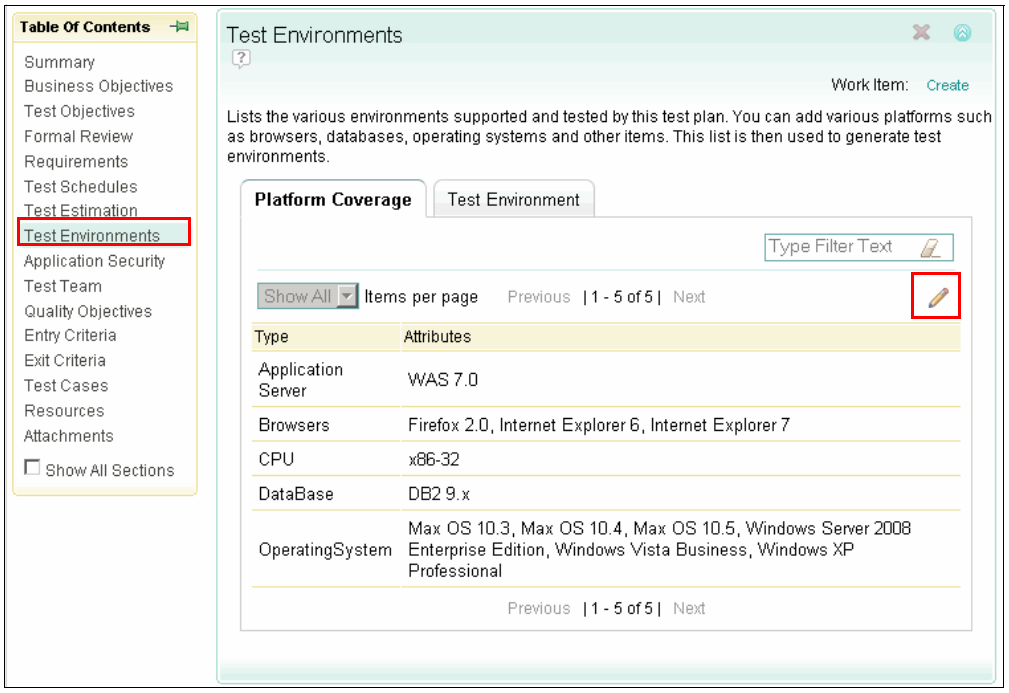
Figure 5-29   Updated Quality Objectives

c. She clicks **Save** on the test plan.

5. She skips **Reviews and Approvals** since there are none for this test plan, and she skips **Requirements** because she just reviewed them.

6. Tammy confirms her execution environments by clicking **Test Environments** in the Table of Contents. In the Test Environments pane, she can add platforms such as browsers, databases, operating systems, and other items. This list is then used to generate test configurations for use in test execution.

   a. She reviews the contents of the **Platform Coverage** tab (Figure 5-30). She uses the Platform Coverage tab to create a nonbinding list of platforms that she intends to cover.



Lists the various environments supported and tested by this test plan. You can add various platforms such as browsers, databases, operating systems and other items. This list is then used to generate test environments.

Type	Attributes
Application Server	WAS 7.0
Browsers	Firefox 2.0, Internet Explorer 6, Internet Explorer 7
CPU	x86-32
DataBase	DB2 9.x
OperatingSystem	Max OS 10.3, Max OS 10.4, Max OS 10.5, Windows Server 2008 Enterprise Edition, Windows Vista Business, Windows XP Professional

Figure 5-30   Platform coverage planned by using the Environments section in the test plan

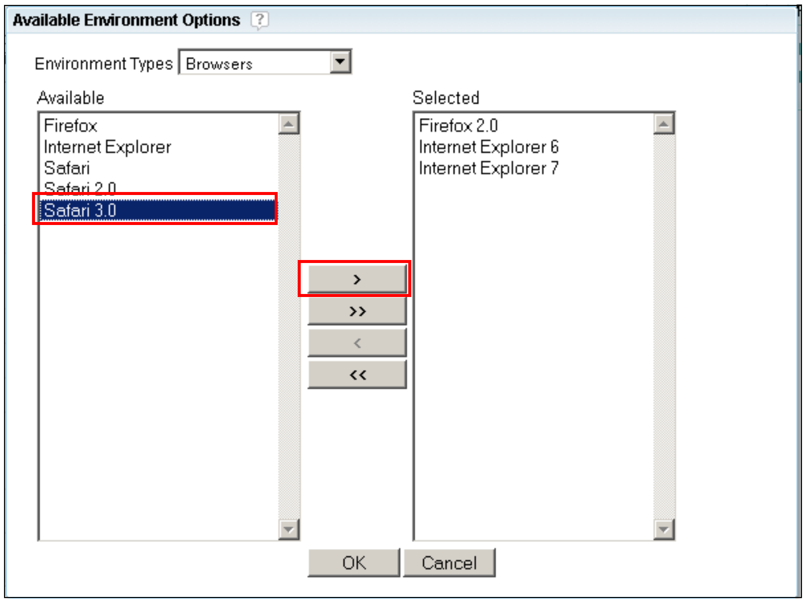
b. In reviewing the list, Tammy decides to update the browsers to use for testing the Web interface. She clicks the **Add Platforms to be covered** button, which is represented by the pencil icon.

c. In the **Available Environment Options** window (Figure 5-31):

   i. From the Environment Types list, she chooses **Browsers**. The window updates with a list of available browsers.

   ii. From the Available list, she selects **Safari 3.0** and clicks the **>** button to move her choice to the **Selected** list.

   iii. She clicks **OK**. The platform coverage has now been updated. When the team creates test execution work items, this new browser version is taken into account.



Available: Firefox, Internet Explorer, Safari, Safari 2.0, Safari 3.0

Selected: Firefox 2.0, Internet Explorer 6, Internet Explorer 7

Figure 5-31 Updating Test Environments in the test plan

d. She clicks **Save**.

7. She clicks the **Test Environments** tab and clicks the **Generate Test Environments** button which is highlighted in Figure 5-32 to generate the test environments for this iteration. The Test Environments tab describes the actual combinations of attributes that will be used for testing. The test environments are associated with an actual test execution.

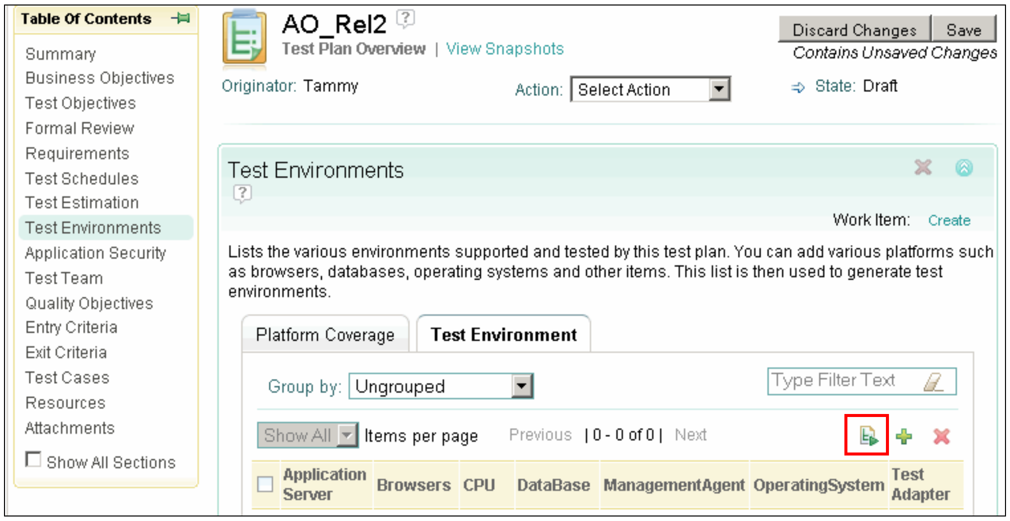


Table Of Contents

Summary
Business Objectives
Test Objectives
Formal Review
Requirements
Test Schedules
Test Estimation
Test Environments
Application Security
Test Team
Quality Objectives
Entry Criteria
Exit Criteria
Test Cases
Resources
Attachments
☐ Show All Sections

AO_Rel2
Test Plan Overview | View Snapshots
Discard Changes | Save
Contains Unsaved Changes
Originator: Tammy
Action: Select Action
State: Draft

Test Environments
Work Item: Create

Lists the various environments supported and tested by this test plan. You can add various platforms such as browsers, databases, operating systems and other items. This list is then used to generate test environments.

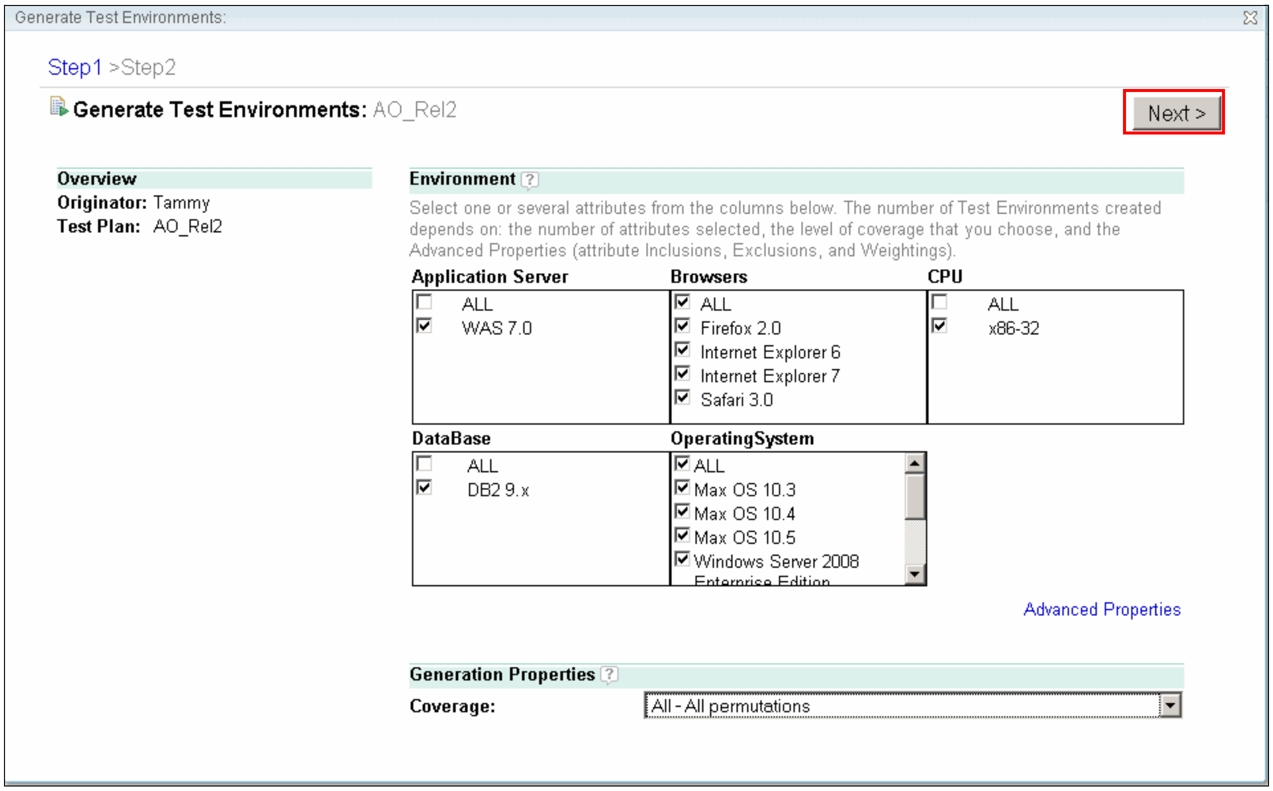
Platform Coverage | Test Environment

Group by: Ungrouped
Type Filter Text
Show All | Items per page | Previous | 0 - 0 of 0 | Next

☐	Application Server	Browsers	CPU	DataBase	ManagementAgent	OperatingSystem	Test Adapter

Figure 5-32   The Test Environment tab

a. In the Generate Test Environment wizard (Figure 5-33), for Step 1, Tammy selects all of the options. For Coverage, she selects **All - All permutations** and then clicks **Next**.


Generate Test Environments:

Step1 >Step2

Generate Test Environments: AO_Rel2

Next >

Overview
Originator: Tammy
Test Plan: AO_Rel2

Environment

Select one or several attributes from the columns below. The number of Test Environments created depends on: the number of attributes selected, the level of coverage that you choose, and the Advanced Properties (attribute Inclusions, Exclusions, and Weightings).

Application Server
☐ ALL
☑ WAS 7.0

Browsers
☑ ALL
☑ Firefox 2.0
☑ Internet Explorer 6
☑ Internet Explorer 7
☑ Safari 3.0

CPU
☐ ALL
☑ x86-32

DataBase
☐ ALL
☑ DB2 9.x

OperatingSystem
☑ ALL
☑ Max OS 10.3
☑ Max OS 10.4
☑ Max OS 10.5
☑ Windows Server 2008 Enterprise Edition

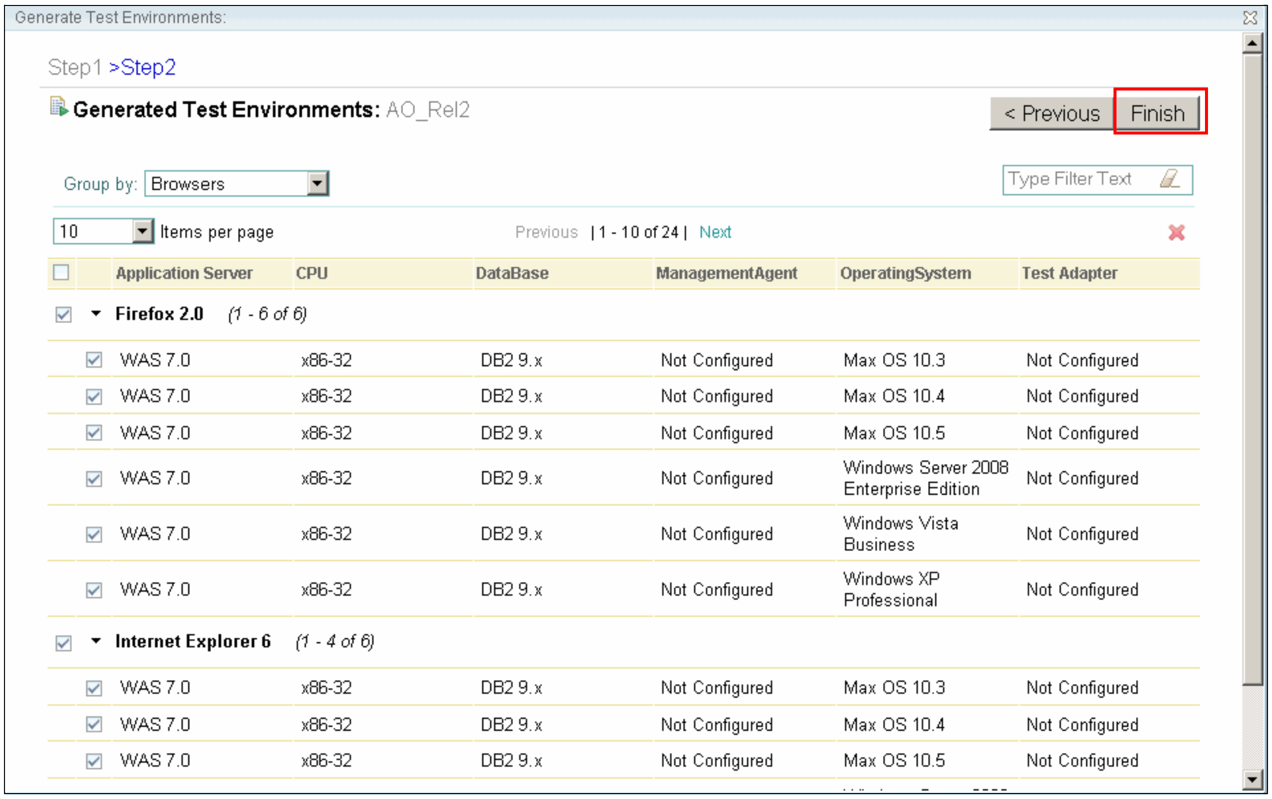
Advanced Properties

Generation Properties
Coverage: All - All permutations

Figure 5-33   Step 1 in generating test environments

b. For Step 2 (Figure 5-34), in which the wizard displays a list of test environments based on the platform support characteristics defined in the test plan, Tammy changes Group by to **Browsers** and views the list of test environments by browser type.

c. Tammy selects each of the test environments that she plans to use in this iteration by clicking the check box next to each configuration. Then she clicks **Finish**.

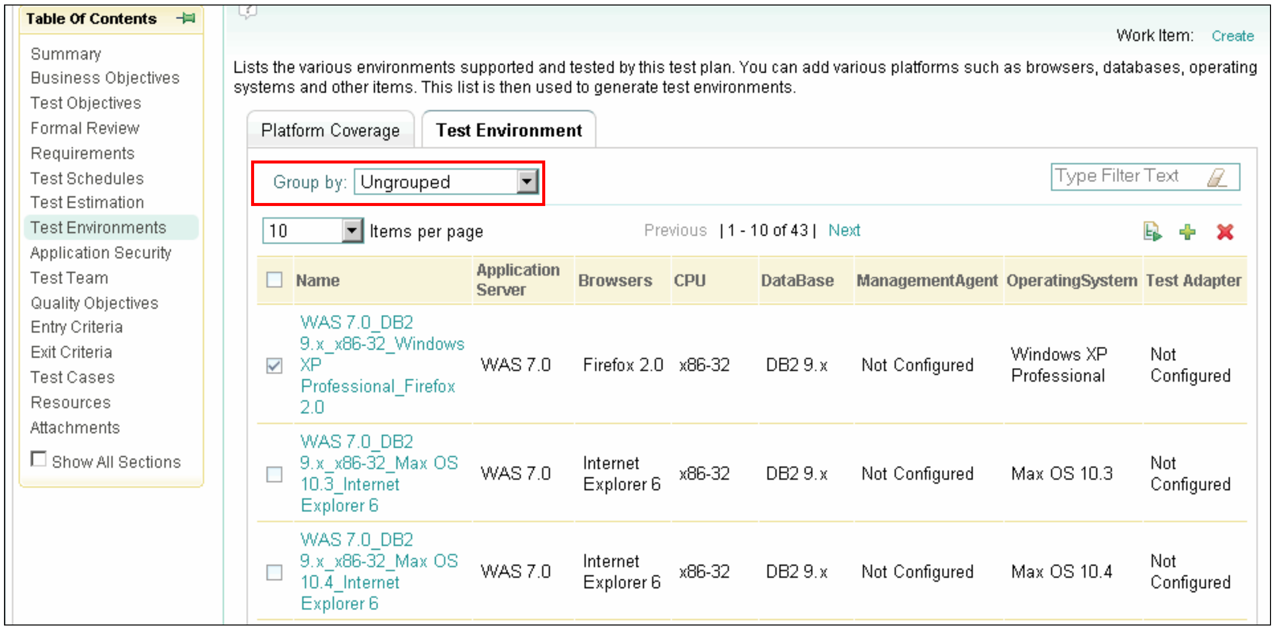


Generate Test Environments:
Step1 >Step2
Generated Test Environments: AO_Rel2
< Previous
Finish
Group by: Browsers
Type Filter Text
10 Items per page
Previous | 1 - 10 of 24 | Next

☐	Application Server	CPU	DataBase	ManagementAgent	OperatingSystem	Test Adapter
✓	**Firefox 2.0** *(1 - 6 of 6)*					
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Max OS 10.3	Not Configured
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Max OS 10.4	Not Configured
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Max OS 10.5	Not Configured
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Windows Server 2008 Enterprise Edition	Not Configured
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Windows Vista Business	Not Configured
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Windows XP Professional	Not Configured
✓	**Internet Explorer 6** *(1 - 4 of 6)*					
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Max OS 10.3	Not Configured
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Max OS 10.4	Not Configured
✓	WAS 7.0	x86-32	DB2 9.x	Not Configured	Max OS 10.5	Not Configured

Figure 5-34 Selecting the Generated Test Environments grouped by browser

The test environments are now added to Tammy's test plan, some of which are shown in Figure 5-35. She can use Group by to group the environments by type, such as by Browser, Application Server, CPU, and so forth.



Lists the various environments supported and tested by this test plan. You can add various platforms such as browsers, databases, operating systems and other items. This list is then used to generate test environments.

Group by: Ungrouped

10 Items per page

	Name	Application Server	Browsers	CPU	DataBase	ManagementAgent	OperatingSystem	Test Adapter
✓	WAS 7.0_DB2 9.x_x86-32_Windows XP Professional_Firefox 2.0	WAS 7.0	Firefox 2.0	x86-32	DB2 9.x	Not Configured	Windows XP Professional	Not Configured
	WAS 7.0_DB2 9.x_x86-32_Max OS 10.3_Internet Explorer 6	WAS 7.0	Internet Explorer 6	x86-32	DB2 9.x	Not Configured	Max OS 10.3	Not Configured
	WAS 7.0_DB2 9.x_x86-32_Max OS 10.4_Internet Explorer 6	WAS 7.0	Internet Explorer 6	x86-32	DB2 9.x	Not Configured	Max OS 10.4	Not Configured

Figure 5-35 Test environments added to the test plan

Note: In this scenario, Tammy generated the test environments. This approach makes sense if you are just getting started with Rational Quality Manager or you know that the test environments do not yet exist in the system. An alternative is to add existing environments into the plan. To add existing test environments to a test plan:

1. In an open test plan, click **Test Environments**.
2. Click the **Add Existing Test Environments** icon.
3. In the window that opens, select one or several test environments and click **OK**.

Tammy is done reviewing and updating her test plan. Next she decides to confirm that the lab resources are reserved for this test effort.

Confirming the lab resources

Tammy decides to make one final check by reviewing the reserved lab resources. She wants to ensure that the servers that are required for this test effort are available and properly configured to support this iteration.

1. From the Lab Management menu, Tammy selects **Create Request** (Figure 5-36).

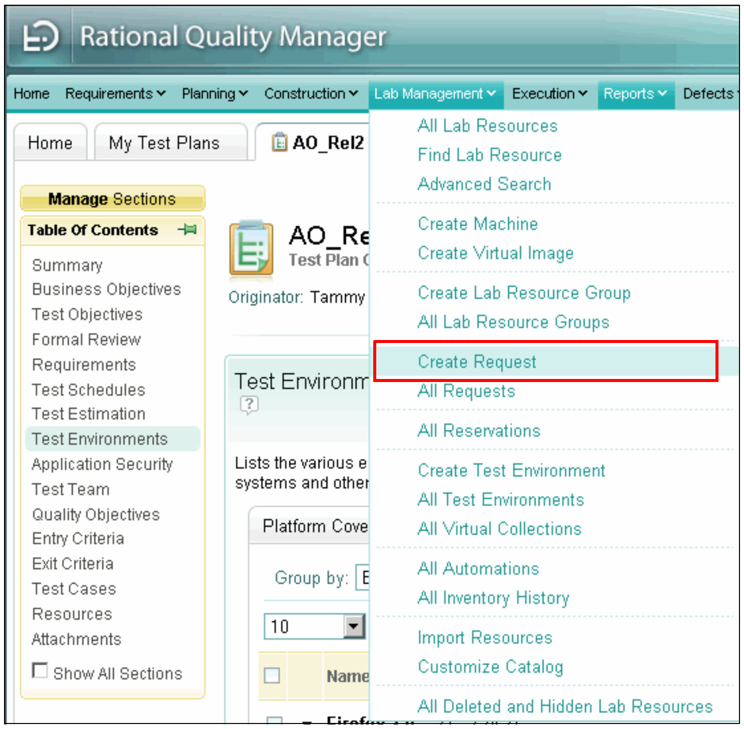


Figure 5-36 Choosing New Request from the Lab Management menu

2. On the Create Request page (Figure 5-37 on page 153):

   a. In the Summary field, she types a descriptive heading, such as “`Macintosh running Safari needed`.” This field is required. The rest of the fields on the Create Request page are optional.

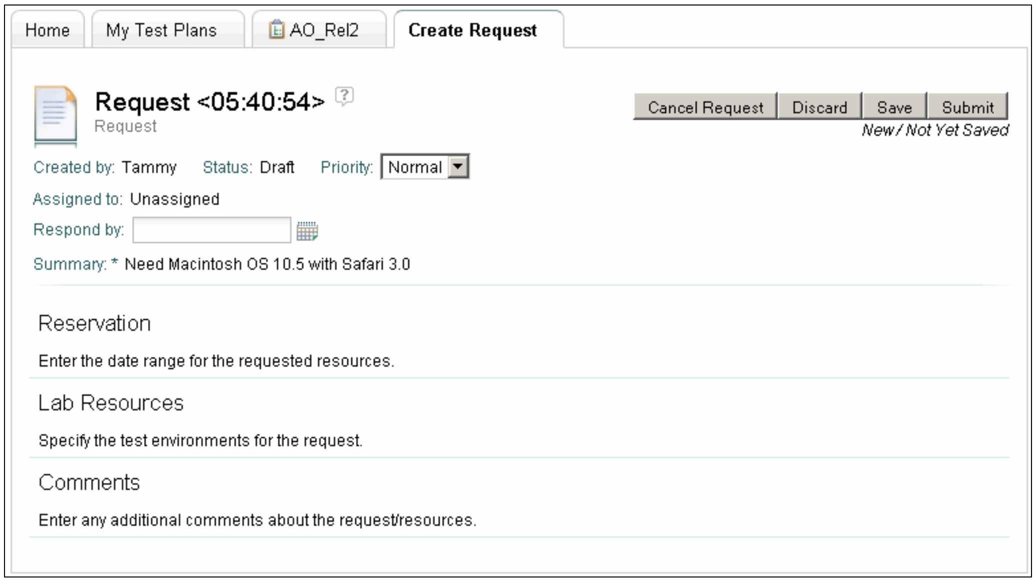
   b. In the Priority field, for Priority (of the request), she selects **Normal**.

   c. In the Respond by field, shes sets the date to be one week from now. This is the time by which she needs a response from the lab manager about this request.

   d. In the Reservation section, she provides the start and end dates and times for which she needs the lab resource. For example, she wants this resource for the last two weeks of the current iteration.

   e. In the Comments field, she types a detailed description of the lab resource request. In this case, she is requesting a new configuration, which includes an Apple Macintosh computer running Safari as the client, which is connected to a server configuration that involves an application server and a database.

   f. In the Lab Resources section, she can enter a new lab resource description or select one from the test environments.



Request <05:40:54>
Request
New / Not Yet Saved
Created by: Tammy Status: Draft Priority: Normal
Assigned to: Unassigned
Respond by:
Summary: * Need Macintosh OS 10.5 with Safari 3.0

Reservation
Enter the date range for the requested resources.

Lab Resources
Specify the test environments for the request.

Comments
Enter any additional comments about the request/resources.

Figure 5-37 Creating a request for a lab resource

g. Because Tammy created test environments in “Updating the solution test plan” on page 141, she can now use the environments to request a lab resource by clicking the **Add from Test Environment button**, which is highlighted in Figure 5-38.

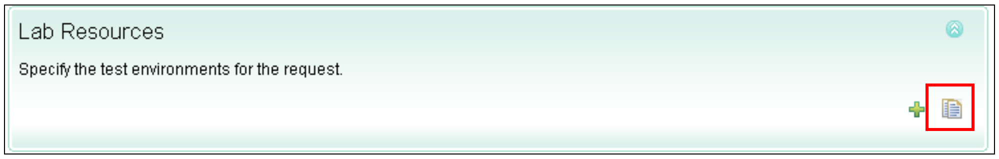
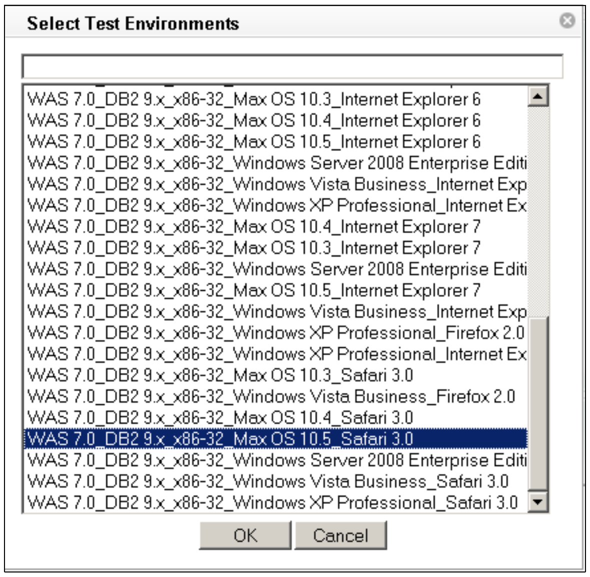

Lab Resources
Specify the test environments for the request.

Figure 5-38 Adding a Lab Resource description from existing test environments

i. In the Select Test Environments window (Figure 5-39), Tammy selects one more environments, and chooses **OK**.


Select Test Environments

```
WAS 7.0_DB2 9.x_x86-32_Max OS 10.3_Internet Explorer 6
WAS 7.0_DB2 9.x_x86-32_Max OS 10.4_Internet Explorer 6
WAS 7.0_DB2 9.x_x86-32_Max OS 10.5_Internet Explorer 6
WAS 7.0_DB2 9.x_x86-32_Windows Server 2008 Enterprise Editi
WAS 7.0_DB2 9.x_x86-32_Windows Vista Business_Internet Exp
WAS 7.0_DB2 9.x_x86-32_Windows XP Professional_Internet Ex
WAS 7.0_DB2 9.x_x86-32_Max OS 10.4_Internet Explorer 7
WAS 7.0_DB2 9.x_x86-32_Max OS 10.3_Internet Explorer 7
WAS 7.0_DB2 9.x_x86-32_Windows Server 2008 Enterprise Editi
WAS 7.0_DB2 9.x_x86-32_Max OS 10.5_Internet Explorer 7
WAS 7.0_DB2 9.x_x86-32_Windows Vista Business_Internet Exp
WAS 7.0_DB2 9.x_x86-32_Windows XP Professional_Firefox 2.0
WAS 7.0_DB2 9.x_x86-32_Windows XP Professional_Internet Ex
WAS 7.0_DB2 9.x_x86-32_Max OS 10.3_Safari 3.0
WAS 7.0_DB2 9.x_x86-32_Windows Vista Business_Firefox 2.0
WAS 7.0_DB2 9.x_x86-32_Max OS 10.4_Safari 3.0
WAS 7.0_DB2 9.x_x86-32_Max OS 10.5_Safari 3.0
WAS 7.0_DB2 9.x_x86-32_Windows Server 2008 Enterprise Editi
WAS 7.0_DB2 9.x_x86-32_Windows Vista Business_Safari 3.0
WAS 7.0_DB2 9.x_x86-32_Windows XP Professional_Safari 3.0
```

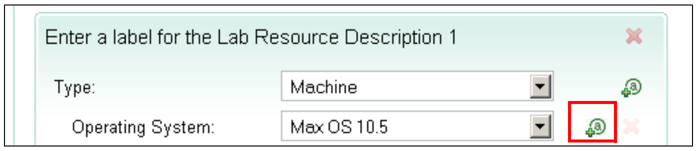
Figure 5-39 Changing the operating system in a configuration request

The environments with all of their attributes are added in the Lab Resources section of the request.

ii. Tammy removes any unneeded attributes, such as Application Server, from the request by clicking the red X next to the attribute.

Since this is a client machine, her plan is to request a single client running Mac OS 10.5 and three browser types: Microsoft Internet Explorer, Apple Safari, and Mozilla Firefox.

iii. She clicks the **Add Criteria** button next to Operating System (Figure 5-40).



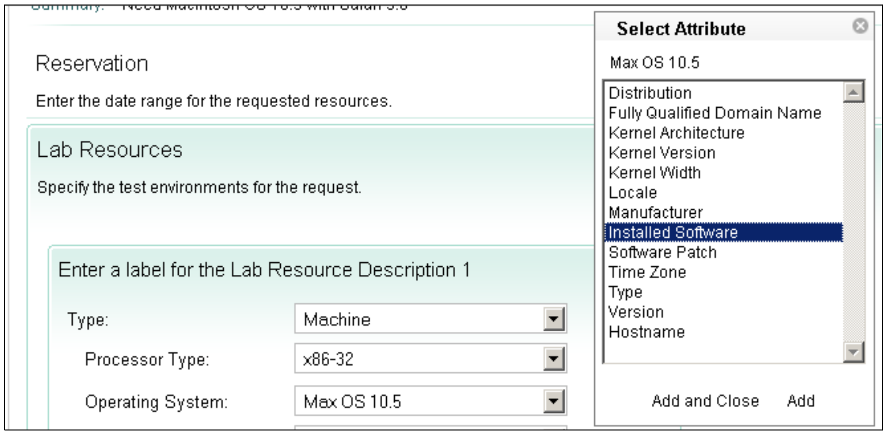
Enter a label for the Lab Resource Description 1

Type: Machine

Operating System: Max OS 10.5

Figure 5-40 The Add Criteria button

iv. In the Select Attribute window (Figure 5-41), she selects **Installed Software** and clicks the **Add** button twice.



Reservation

Enter the date range for the requested resources.

Lab Resources

Specify the test environments for the request.

Enter a label for the Lab Resource Description 1

Type: Machine

Processor Type: x86-32

Operating System: Max OS 10.5

Select Attribute

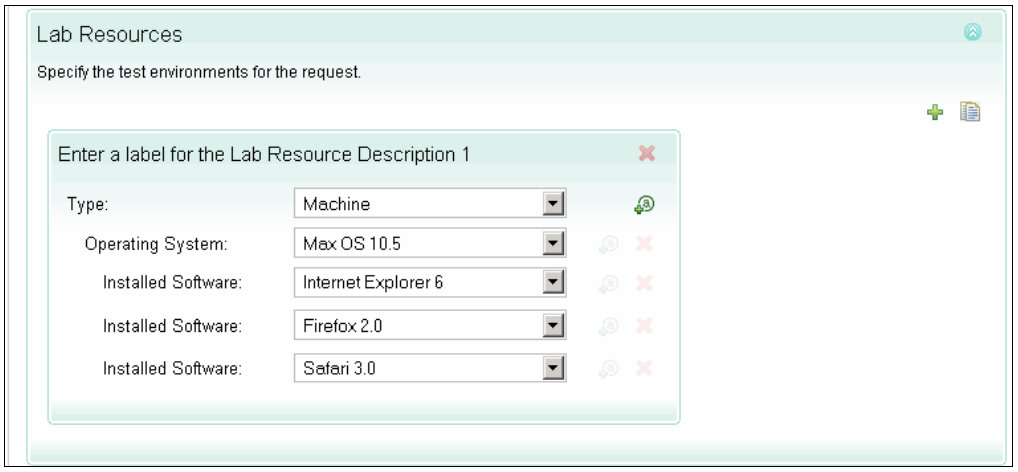
Max OS 10.5

Distribution
Fully Qualified Domain Name
Kernel Architecture
Kernel Version
Kernel Width
Locale
Manufacturer
Installed Software
Software Patch
Time Zone
Type
Version
Hostname

Add and Close Add

Figure 5-41 Adding attributes to the Lab Resources criteria

v. She selects the browser version for each of the Installed Software criteria. Her final request lists an operating system and three criteria for installed software (Figure 5-42).



Lab Resources

Specify the test environments for the request.

Enter a label for the Lab Resource Description 1

Type: Machine

Operating System: Max OS 10.5

Installed Software: Internet Explorer 6

Installed Software: Firefox 2.0

Installed Software: Safari 3.0

Figure 5-42 A lab resource described with attributes

h. When she is satisfied, she clicks **Submit** to send her request to the lab manager. Alternatively, she can do one of the following steps:

   - To save the request as a draft without submitting it, she can click **Save**.
   - To clear the form, she can click **Reset**.
   - To cancel the request, she can click **Cancel**.

The request appears in her dashboard under “My Requests”. It is also listed when she selects All Requests from the Lab Management menu in the left panel.

Behind the scenes (not covered in this book), Larry the lab manager prepares the configuration for Tammy and notifies her when it is available.

Updating and closing the task

Tammy has completed all of the work that is required to absorb the UI branding request into the current iteration. She now closes the task that Patricia assigned to her:

1. She clicks the **Home** tab to view the dashboard.
2. She goes to **My Tasks** viewlet.
3. She clicks the **UI Corporate Branding** task (Figure 5-20 on page 140).
4. In the Work Item editor (Figure 5-21):
   a. She creates an entry in the Discussions field to document changes that she has made to the plan.
   b. She sets the state to **Complete**.
   c. She clicks the **Save** button to apply the changes to the work item. The information that Tammy has added to work item is transferred to Patricia in Rational ClearQuest where she can confirm the project iteration plan.

Tammy and her team begin constructing the test cases and executions for this iteration. Their work continues in Act 4, which is covered in 10.2, “A reference scenario for managing quality” on page 397.

5.5.5 Patricia confirms the project iteration plan

During the iteration planning, Patricia monitors the alignment of work for the ALMTasks that has been committed to the project iteration plan. Patricia uses a set of queries for this purpose:

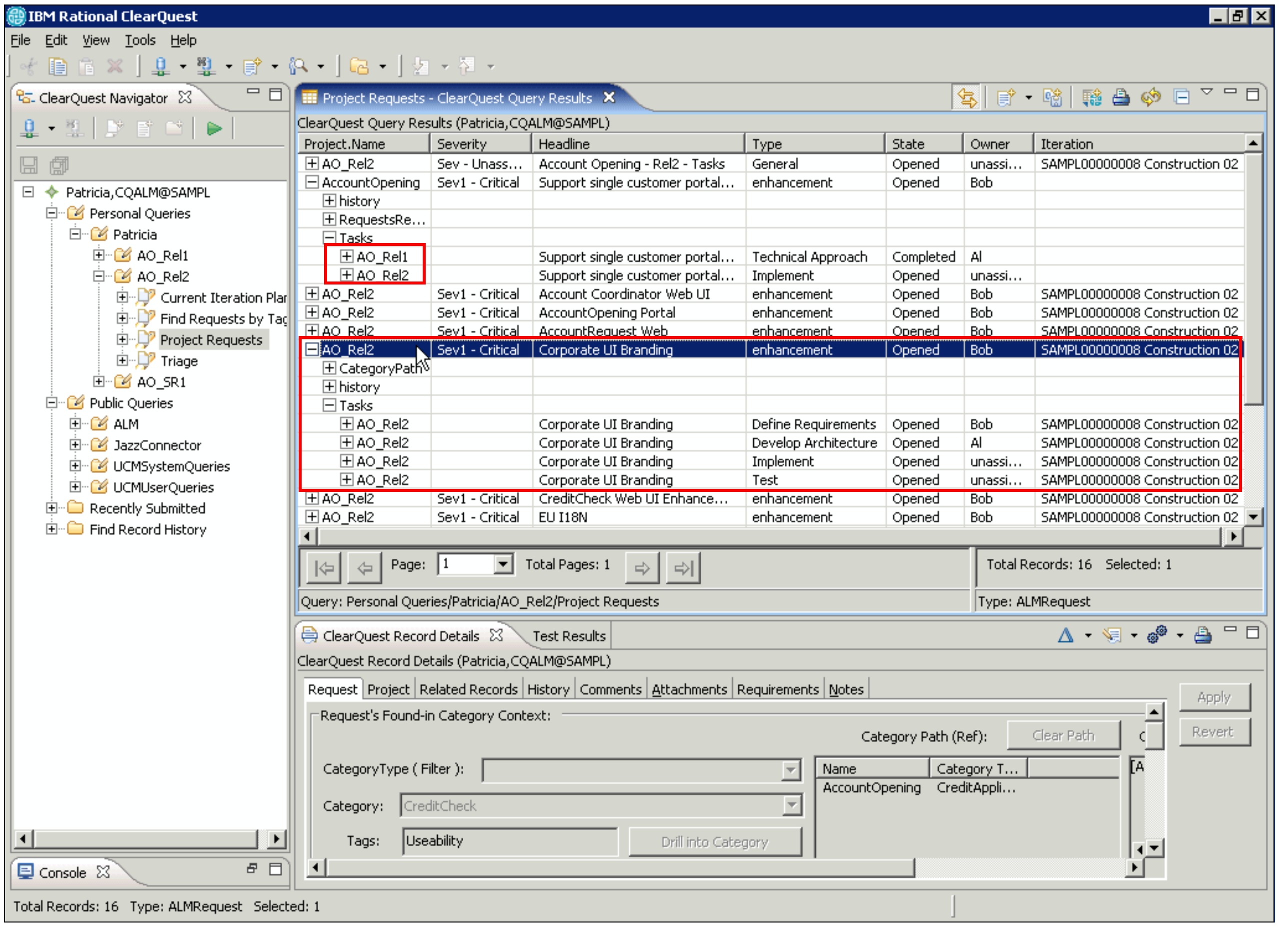
- Project requests

  Patricia runs a query for all ALMRequests that have ALMTasks planned for the current project. By using this query, she can track all release requirements for the project. Figure 5-43 on page 156 shows that the Corporate UI Branding request is planned to be fully delivered in the current project. Figure 5-43 also shows a request for an Account Verification Customer Portal. Note that the technical approach of this request was delivered in an earlier project, AO_Rel1, while the implementation is still unassigned and will be planned for the current project AO_Rel2.

- Current iteration plan

  Patricia runs a query for all ALMTasks in the current iteration, for example, Construction 02 of the AO_Rel2 project. By using this query, she can track all tasks that are currently being planned for current iteration. For teams that perform iteration planning by using ClearQuest ALM, she can drill into the status of the ALMActivities.

Patricia runs her Project Requests query and confirms that all tasks are assigned to the team and that no team is rejecting work activities.


ClearQuest Query Results (Patricia,CQALM@SAMPL)

Project.Name	Severity	Headline	Type	State	Owner	Iteration
AO_Rel2	Sev - Unass…	Account Opening - Rel2 - Tasks	General	Opened	unassi…	SAMPL00000008 Construction 02
AccountOpening	Sev1 - Critical	Support single customer portal…	enhancement	Opened	Bob	
history						
RequestsRe…						
Tasks						
AO_Rel1		Support single customer portal…	Technical Approach	Completed	Al	
AO_Rel2		Support single customer portal…	Implement	Opened	unassi…	
AO_Rel2	Sev1 - Critical	Account Coordinator Web UI	enhancement	Opened	Bob	SAMPL00000008 Construction 02
AO_Rel2	Sev1 - Critical	AccountOpening Portal	enhancement	Opened	Bob	SAMPL00000008 Construction 02
AO_Rel2	Sev1 - Critical	AccountRequest Web	enhancement	Opened	Bob	SAMPL00000008 Construction 02
AO_Rel2	Sev1 - Critical	Corporate UI Branding	enhancement	Opened	Bob	SAMPL00000008 Construction 02
CategoryPath						
history						
Tasks						
AO_Rel2		Corporate UI Branding	Define Requirements	Opened	Bob	SAMPL00000008 Construction 02
AO_Rel2		Corporate UI Branding	Develop Architecture	Opened	Al	SAMPL00000008 Construction 02
AO_Rel2		Corporate UI Branding	Implement	Opened	unassi…	SAMPL00000008 Construction 02
AO_Rel2		Corporate UI Branding	Test	Opened	unassi…	SAMPL00000008 Construction 02
AO_Rel2	Sev1 - Critical	CreditCheck Web UI Enhance…	enhancement	Opened	Bob	SAMPL00000008 Construction 02
AO_Rel2	Sev1 - Critical	EU I18N	enhancement	Opened	Bob	SAMPL00000008 Construction 02

Page: 1 Total Pages: 1 Total Records: 16 Selected: 1

Query: Personal Queries/Patricia/AO_Rel2/Project Requests Type: ALMRequest

ClearQuest Record Details (Patricia,CQALM@SAMPL)

Request's Found-in Category Context:

Category Path (Ref):

CategoryType (Filter):

Category: CreditCheck

Tags: Useability

Name	Category T…
AccountOpening	CreditAppli…

Total Records: 16 Type: ALMRequest Selected: 1

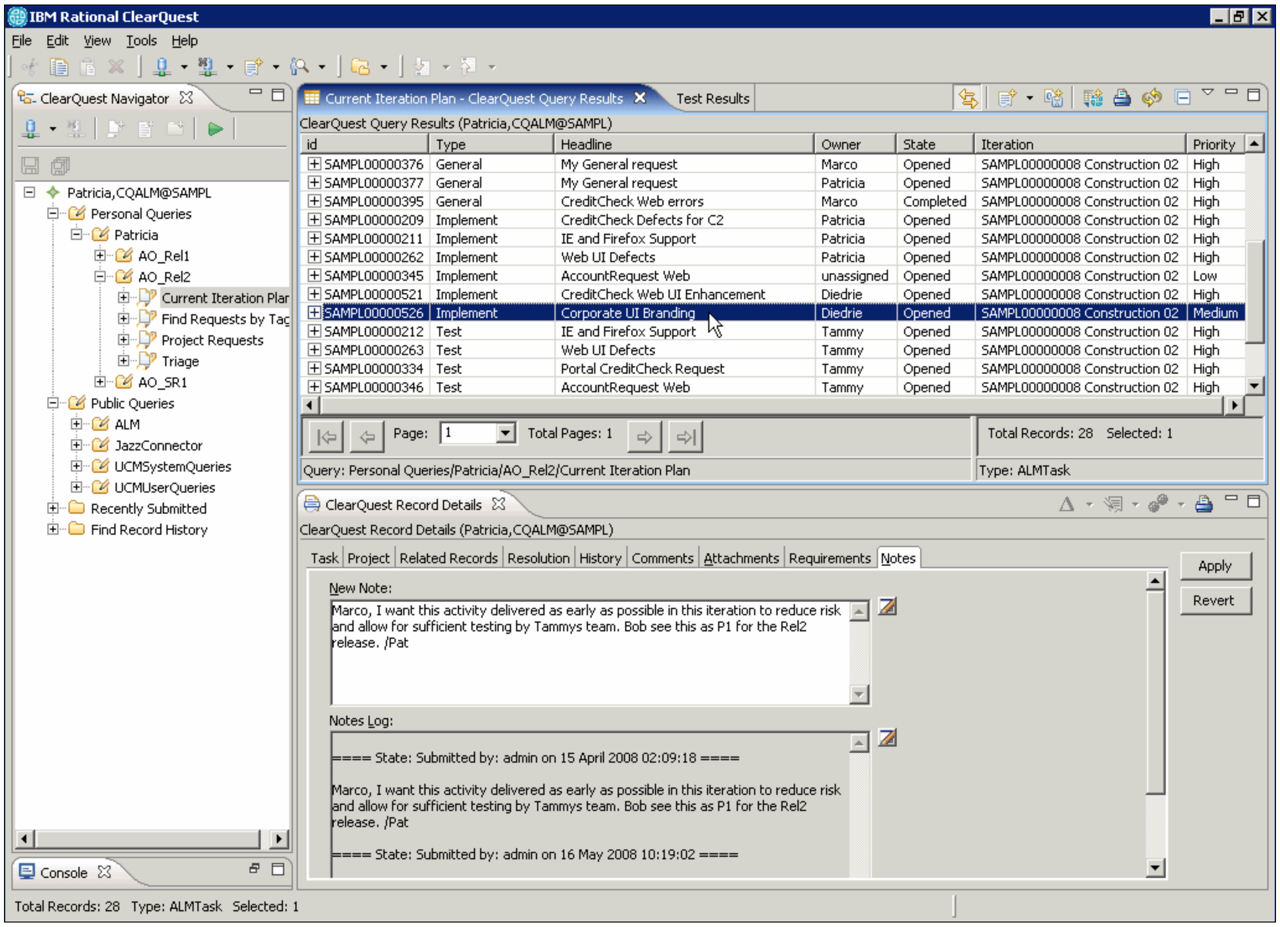
Figure 5-43 Patricia confirming the requests that have work planned in the current project

Patricia runs her Current Iteration query and confirms that all tasks are planned by the teams and align with the project plan. She notices that the CreditCheck team has de-prioritized Bob's UI branding request (Figure 5-44 on page 157). The work has been set as medium priority, with a due date at the end of the Construction 02 iteration. This makes Patricia concerned. She opens the activity to discuss the plan with Marco and his team.

To align the plans, Patricia does the following actions:

1. She opens the UI Branding Activity that is assigned to Diedrie.
2. She clicks the **Notes** tab and reads the discussions. She confirms that the team is planning to complete this work in the C2B iteration and that the priority of the change has been misinterpreted.
3. She adds a new discussion entry and requests that this work is moved to the C2A iteration and underscores that Bob regards this as a high priority for the release. She saves the changes to the record.

As shown in Figure 5-44, Patricia confirms the alignment of the project plan and the team iteration plans. She identifies the late planned delivery of the Corporate UI Branding activity and collaborates with Marco to align the plans.



ClearQuest Query Results (Patricia,CQALM@SAMPL)

id	Type	Headline	Owner	State	Iteration	Priority
SAMPL00000376	General	My General request	Marco	Opened	SAMPL00000008 Construction 02	High
SAMPL00000377	General	My General request	Patricia	Opened	SAMPL00000008 Construction 02	High
SAMPL00000395	General	CreditCheck Web errors	Marco	Completed	SAMPL00000008 Construction 02	High
SAMPL00000209	Implement	CreditCheck Defects for C2	Patricia	Opened	SAMPL00000008 Construction 02	High
SAMPL00000211	Implement	IE and Firefox Support	Patricia	Opened	SAMPL00000008 Construction 02	High
SAMPL00000262	Implement	Web UI Defects	Patricia	Opened	SAMPL00000008 Construction 02	High
SAMPL00000345	Implement	AccountRequest Web	unassigned	Opened	SAMPL00000008 Construction 02	Low
SAMPL00000521	Implement	CreditCheck Web UI Enhancement	Diedrie	Opened	SAMPL00000008 Construction 02	High
SAMPL00000526	Implement	Corporate UI Branding	Diedrie	Opened	SAMPL00000008 Construction 02	Medium
SAMPL00000212	Test	IE and Firefox Support	Tammy	Opened	SAMPL00000008 Construction 02	High
SAMPL00000263	Test	Web UI Defects	Tammy	Opened	SAMPL00000008 Construction 02	High
SAMPL00000334	Test	Portal CreditCheck Request	Tammy	Opened	SAMPL00000008 Construction 02	High
SAMPL00000346	Test	AccountRequest Web	Tammy	Opened	SAMPL00000008 Construction 02	High

Page: 1 Total Pages: 1 Total Records: 28 Selected: 1

Query: Personal Queries/Patricia/AO_Rel2/Current Iteration Plan Type: ALMTask

ClearQuest Record Details (Patricia,CQALM@SAMPL)

New Note:

Marco, I want this activity delivered as early as possible in this iteration to reduce risk and allow for sufficient testing by Tammys team. Bob see this as P1 for the Rel2 release. /Pat

Notes Log:

==== State: Submitted by: admin on 15 April 2008 02:09:18 ====

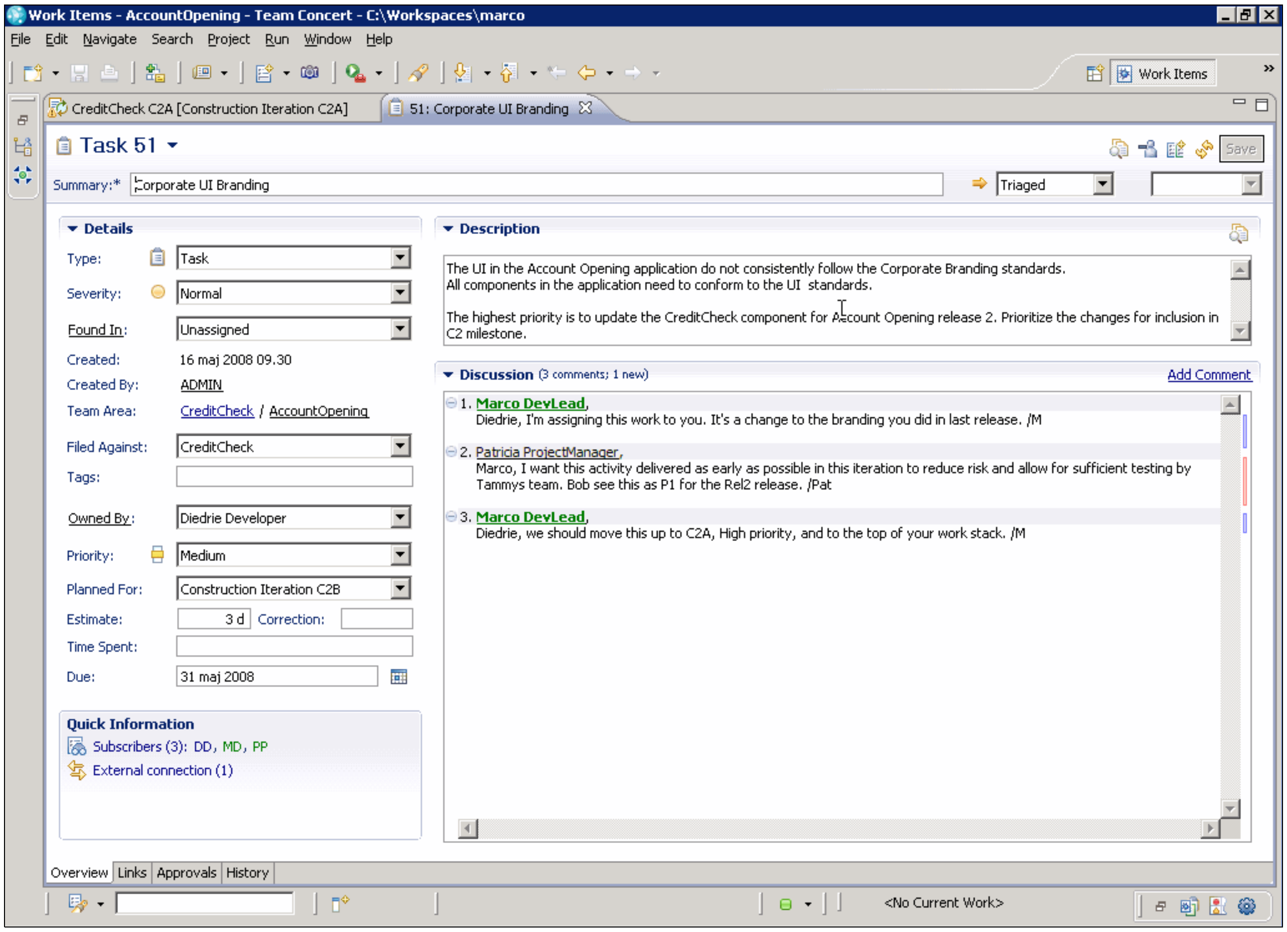
Marco, I want this activity delivered as early as possible in this iteration to reduce risk and allow for sufficient testing by Tammys team. Bob see this as P1 for the Rel2 release. /Pat

==== State: Submitted by: admin on 16 May 2008 10:19:02 ====

Total Records: 28 Type: ALMTask Selected: 1

Figure 5-44 Patricia confirming the alignment of the project plan and the team iteration plans

Marco and Diedrie receive an event notification of the changes to the UI branding work item, and the team discusses Patricia's view of the case (Figure 5-45). Marco and Diedrie update the Work Item priority, Planned For, and Due Date properties to align the iteration plan.


Work Items - AccountOpening - Team Concert - C:\Workspaces\marco
File Edit Navigate Search Project Run Window Help
Work Items
CreditCheck C2A [Construction Iteration C2A]
51: Corporate UI Branding
Save
Overview Links Approvals History
<No Current Work>

Task 51

Summary:* Corporate UI Branding | Triaged

Details

Type:	Task
Severity:	Normal
Found In:	Unassigned
Created:	16 maj 2008 09.30
Created By:	ADMIN
Team Area:	CreditCheck / AccountOpening
Filed Against:	CreditCheck
Tags:	
Owned By:	Diedrie Developer
Priority:	Medium
Planned For:	Construction Iteration C2B
Estimate:	3 d Correction:
Time Spent:	
Due:	31 maj 2008

Quick Information

Subscribers (3): DD, MD, PP

External connection (1)

Description

The UI in the Account Opening application do not consistently follow the Corporate Branding standards.
All components in the application need to conform to the UI standards.

The highest priority is to update the CreditCheck component for Account Opening release 2. Prioritize the changes for inclusion in C2 milestone.

Discussion (3 comments; 1 new) Add Comment

1. Marco DevLead,
Diedrie, I'm assigning this work to you. It's a change to the branding you did in last release. /M

2. Patricia ProjectManager,
Marco, I want this activity delivered as early as possible in this iteration to reduce risk and allow for sufficient testing by Tammys team. Bob see this as P1 for the Rel2 release. /Pat

3. Marco DevLead,
Diedrie, we should move this up to C2A, High priority, and to the top of your work stack. /M

Figure 5-45 Patricia, Marco, and Diedrie collaborating on the alignment of the iteration plan

5.5.6 Bob defines and manages the requirements

Synopsis: In this section, Bob defines and manages requirements. He is notified that he has been assigned an activity by Patricia to define the detailed requirements for the UI branding request that he recently created.

This scene involves the following actions:

- Defining the stakeholder request
- Applying additional definition techniques by using Rational Requirements Composer
- Performing technical refinement of the requirements in Rational RequisitePro
- Linking the new requirements to the ALMRequest

Figure 5-46 highlights the workflow.

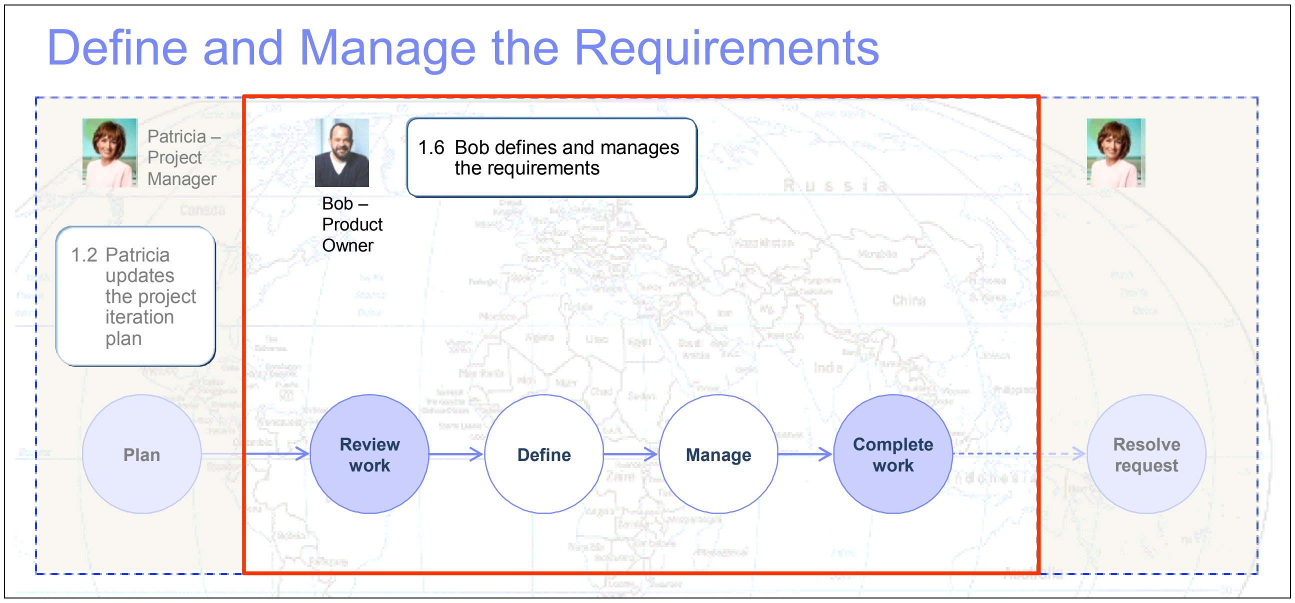

Define and Manage the Requirements
Patricia – Project Manager
1.2 Patricia updates the project iteration plan
Bob – Product Owner
1.6 Bob defines and manages the requirements
Plan
Review work
Define
Manage
Complete work
Resolve request

Figure 5-46 Requirements definition

Figure 5-47 shows the artifacts that are created.

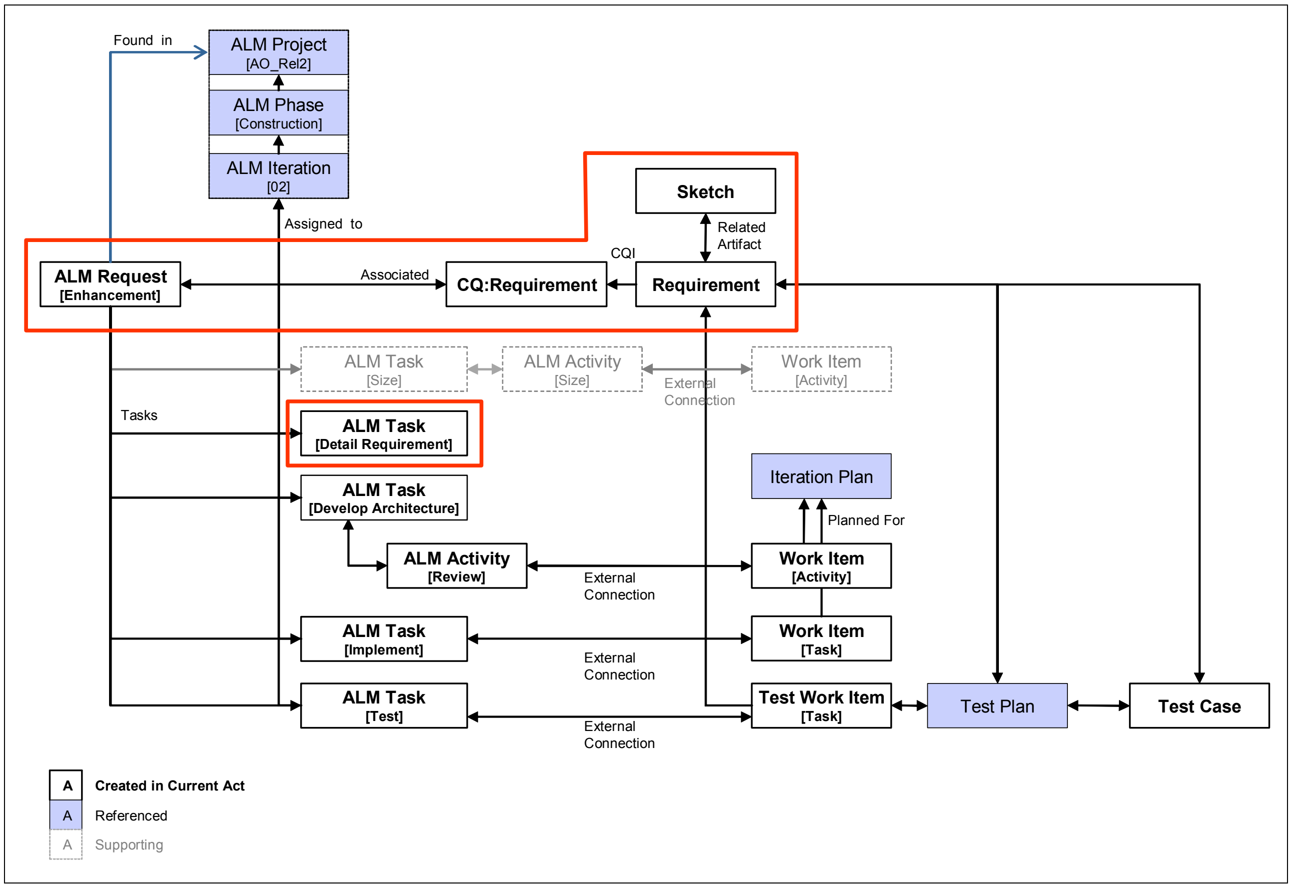


Figure 5-47 Artifacts that are created

Opening the ALMTask to define the requirements

Bob, the product owner, gets an e-mail notification that he has been assigned ownership of an ALMTask record. He logs into the Rational ClearQuest Web interface to get more details. He performs the following steps:

1. Bob browses to the Rational ClearQuest Web URL.
2. He enters the Rational ClearQuest login user name and password. He selects a valid schema repository name from the list and clicks **Connect**.
3. He types the record ID in the query workspace navigator search box and selects the **By ID** radio button. Then he clicks **Search**.
4. After retrieving the record, he determines that further requirement work must be done and transitions the task to an Opened state to let Patricia know that he is working on it:
   a. He clicks the **Open** button to transition the record to an Opened state.
   b. He clicks the **Submit** button to apply the changes.

Defining the stakeholder request

Goal: The goal is to review the defined requirement activities and ensure that initial requests that are captured in ClearQuest are reflective in RequisitePro.

As a requirements best practice, it is helpful to ensure that more detailed requirements are traced back to the original request or need statement. This step is important because the solution team can ensure that application functions that are delivered “resolve the original need statement.” As defined in 5.5.1, “Bob submits a request” on page 127, Bob begins the requirements process by initiating a new request in Rational ClearQuest. Bob ensures that this information is captured in Rational RequisitePro. The intent of including this information in Rational RequisitePro is to reflect traceability across the life cycle and identify that the original need statement as captured in the request. Requirements that fulfill that need statement are managed in Rational RequisitePro.

Bob is now ready to reflect that the requirement is being defined in Rational RequisitePro. He logs into Rational RequisitePro and submits a new requirement of type Stakeholder Request:

1. Bob browses to the Rational RequisitePro Web URL.
2. He enters the Rational RequisitePro login user name and password and selects a valid Rational RequisitePro project in Project. Then he clicks **Login**.
3. He creates a stakeholder request requirement in the stakeholder request package by selecting **Create Requirement** (Figure 5-48).

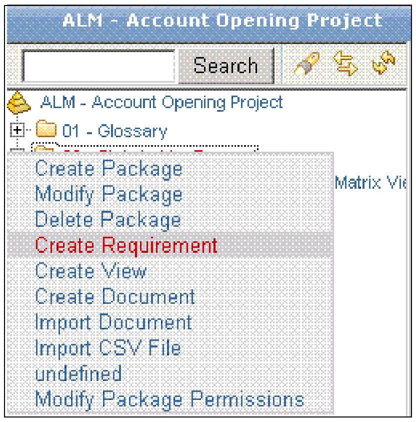
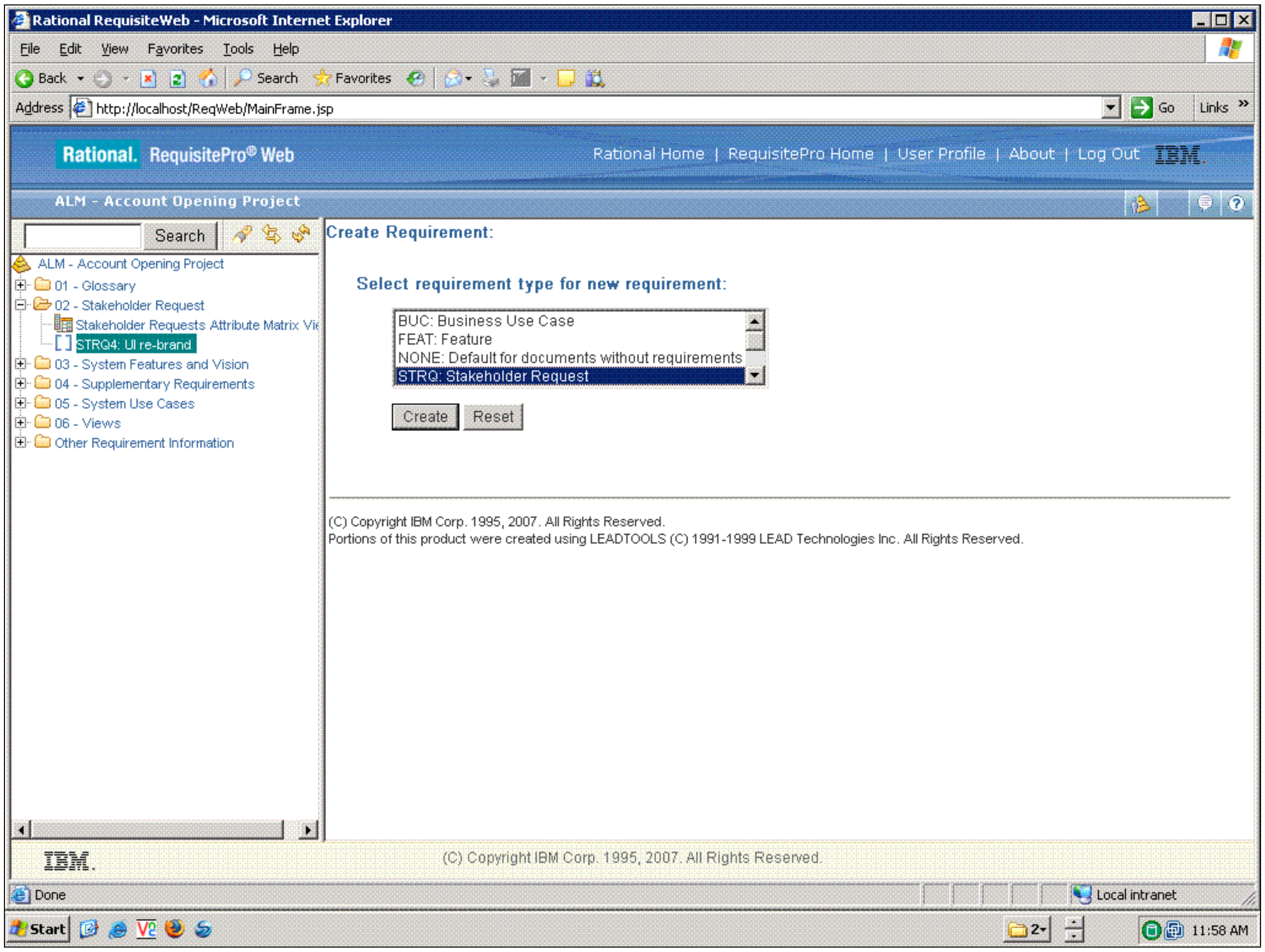


Figure 5-48 Selecting Create Requirement to create a stakeholder request requirement

4. In the Create Requirement pane, he creates a requirement by selecting **STRQ: Stakeholder Request** (Figure 5-49). Then he clicks **Create**.

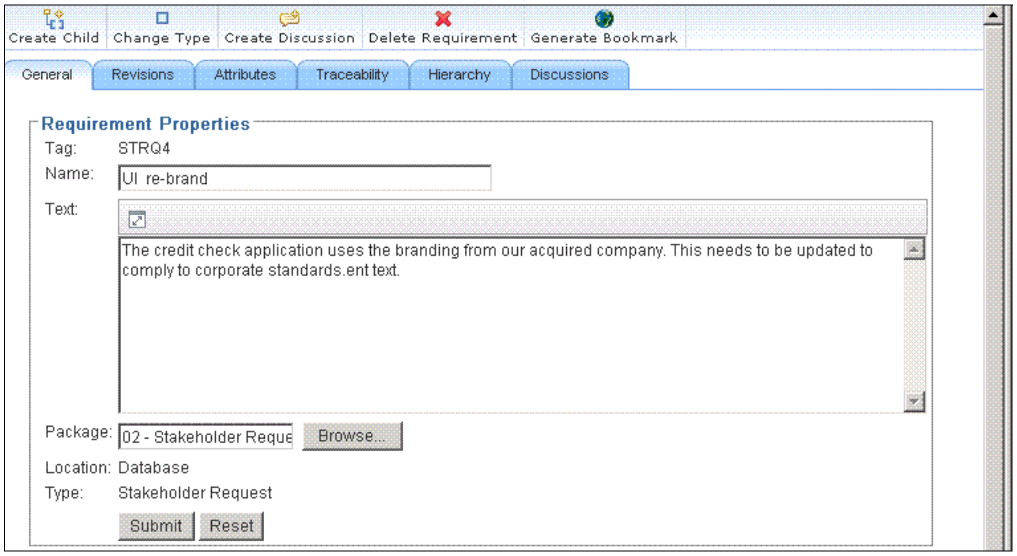

Rational RequisiteWeb - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/ReqWeb/MainFrame.jsp
Rational. RequisitePro® Web
Rational Home | RequisitePro Home | User Profile | About | Log Out
ALM - Account Opening Project
Search
ALM - Account Opening Project
01 - Glossary
02 - Stakeholder Request
Stakeholder Requests Attribute Matrix Vi
STRQ4: UI re-brand
03 - System Features and Vision
04 - Supplementary Requirements
05 - System Use Cases
06 - Views
Other Requirement Information
Create Requirement:
Select requirement type for new requirement:
BUC: Business Use Case
FEAT: Feature
NONE: Default for documents without requirements
STRQ: Stakeholder Request
Create
Reset
Done
Local intranet
11:58 AM

(C) Copyright IBM Corp. 1995, 2007. All Rights Reserved.
Portions of this product were created using LEADTOOLS (C) 1991-1999 LEAD Technologies Inc. All Rights Reserved.

(C) Copyright IBM Corp. 1995, 2007. All Rights Reserved.

Figure 5-49 New stakeholder request

5. In the Requirement Properties pane (Figure 5-50), he provides a short description of the request in the Name field and a more detailed description in the description field of the need statement.



Tag: STRQ4
Name: UI re-brand
Text: The credit check application uses the branding from our acquired company. This needs to be updated to comply to corporate standards.ent text.
Package: 02 - Stakeholder Reque
Location: Database
Type: Stakeholder Request

Figure 5-50 Stakeholder request description

6. He clicks the **Attributes** tab and provides pertinent attribute information such as Priority.

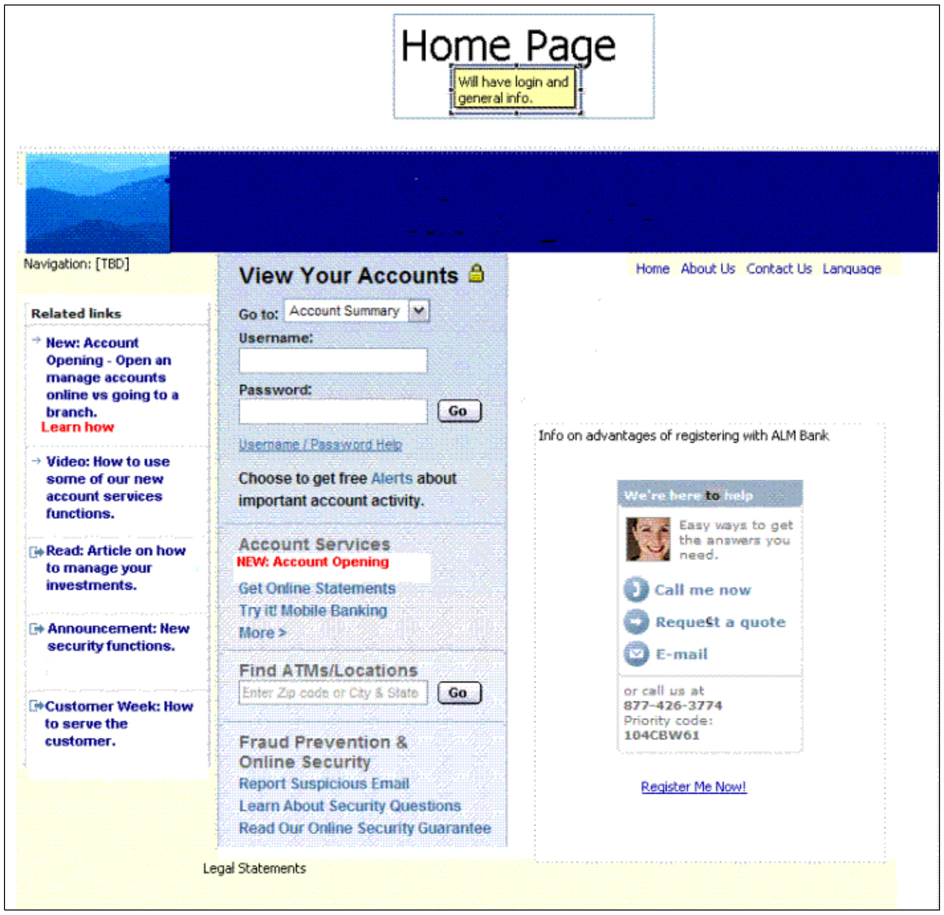
Although the request provides the need statement and business problem, it does not provide enough detail for a team to leverage the requirement information to implement the request. As a result, Bob decides that it is pertinent to conduct additional requirements definition activities in Rational Requirements Composer. The artifacts that he generates help him manage requirements in Rational RequisitePro later.

Additional definition techniques by using Rational Requirements Composer

Goal: Additional definition techniques, such as application sketches and storyboards, help to convey additional information about the business and provide context for further requirement activities. An application sketch is created and requirements are referenced for items in the sketch.

Bob decides to conduct sketching techniques in Rational Requirements Composer to define the flow of the Account Opening Login window to entail the UI branding changes. He creates the sketch, marks elements on the sketch as requirements, and makes it available to his peers for commenting:

1. Bob marks the logo emblem on the sketch as a requirement (Figure 5-51).


Home Page

Will have login and general info.

Navigation: [TBD]

Home About Us Contact Us Language

Related links

New: Account Opening - Open an manage accounts online vs going to a branch. Learn how

Video: How to use some of our new account services functions.

Read: Article on how to manage your investments.

Announcement: New security functions.

Customer Week: How to serve the customer.

View Your Accounts

Go to: Account Summary

Username:

Password: Go

Username / Password Help

Choose to get free Alerts about important account activity.

Account Services

NEW: Account Opening

Get Online Statements

Try it! Mobile Banking

More >

Find ATMs/Locations

Enter Zip code or City & State Go

Fraud Prevention & Online Security

Report Suspicious Email

Learn About Security Questions

Read Our Online Security Guarantee

Info on advantages of registering with ALM Bank

We're here to help

Easy ways to get the answers you need.

Call me now

Request a quote

E-mail

or call us at 877-426-3774

Priority code: 104CBW61

Register Me Now!

Legal Statements

Figure 5-51   Bank Sign On application sketch

2. He selects the element to mark as a requirement and chooses **Mark as Requirement** (Figure 5-52).

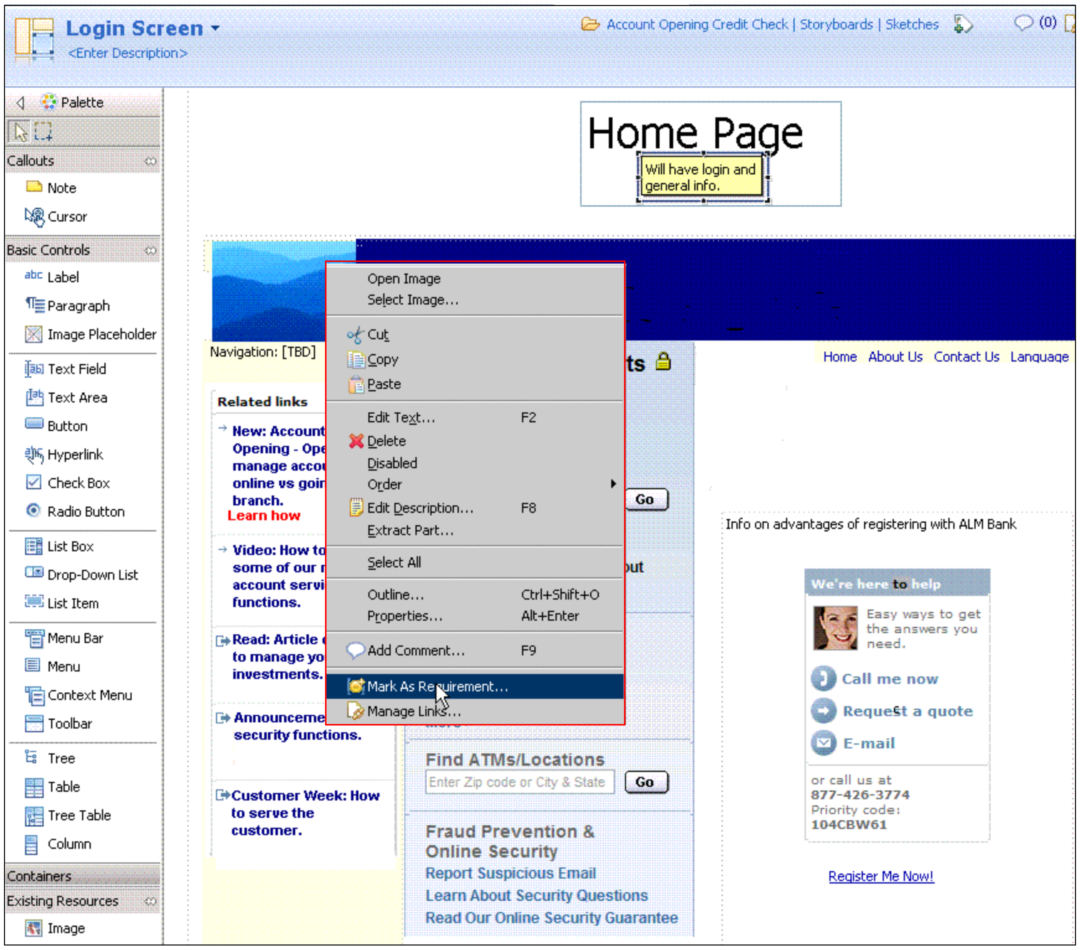
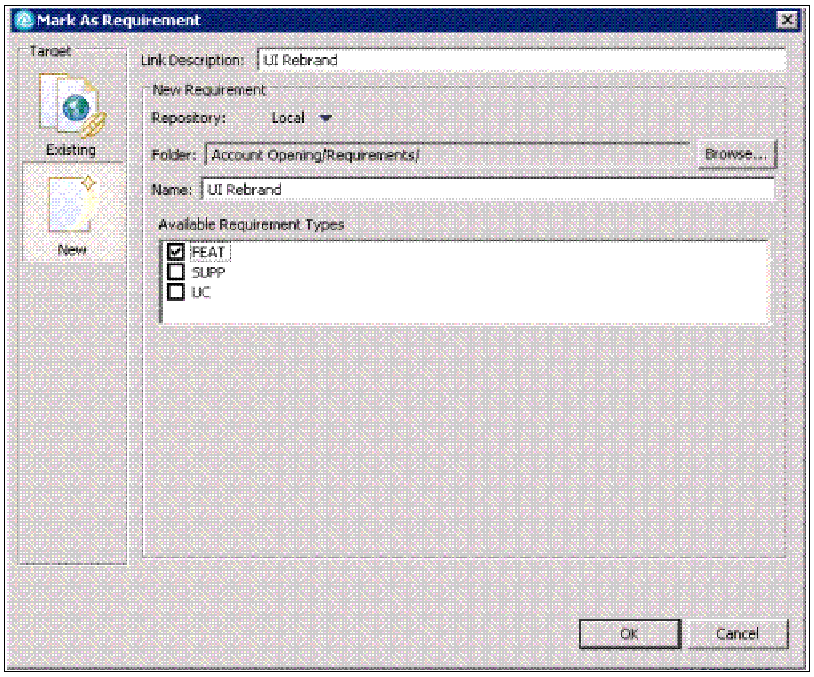


Figure 5-52 Marking a requirement

3. In the Mark As Requirement window (Figure 5-53), he provides a link description and name and chooses the type of requirement. He selects the attributes and clicks **OK** to create the requirement.



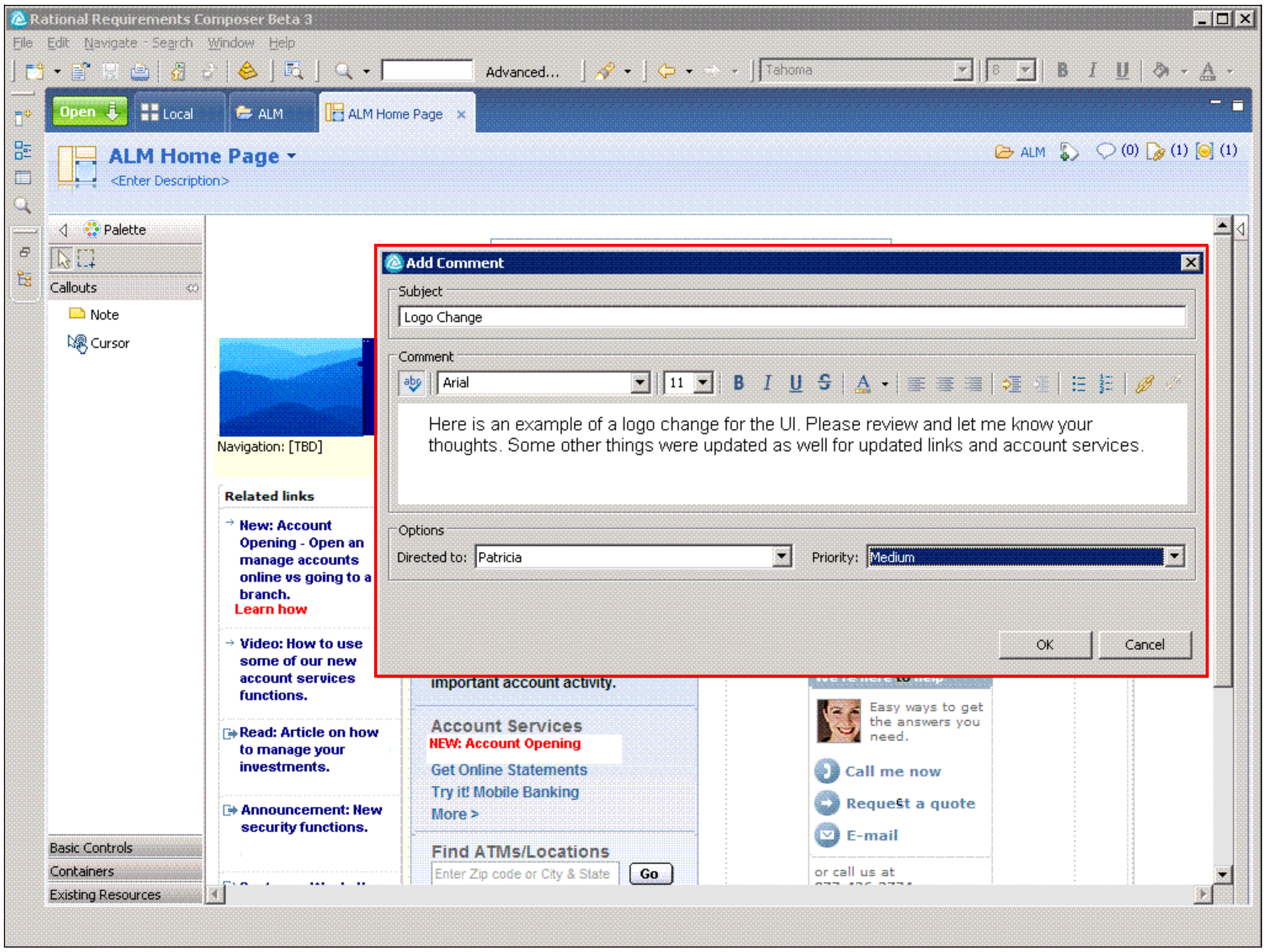
Link Description: UI Rebrand
Repository: Local
Folder: Account Opening/Requirements/
Name: UI Rebrand
Available Requirement Types
- [x] FEAT
- [ ] SUPP
- [ ] UC

Figure 5-53 Defining the requirement type

After the requirement is created, it is marked with an indicator that a requirement is present for that portion of the sketch.

4. Bob informs his peers that he has created a new sketch and requests feedback to ensure that the logo change will address the UI branding requirement:

   a. With the logo selected, he requests feedback on the comment (Figure 5-54).



Subject: Logo Change

Comment: Here is an example of a logo change for the UI. Please review and let me know your thoughts. Some other things were updated as well for updated links and account services.

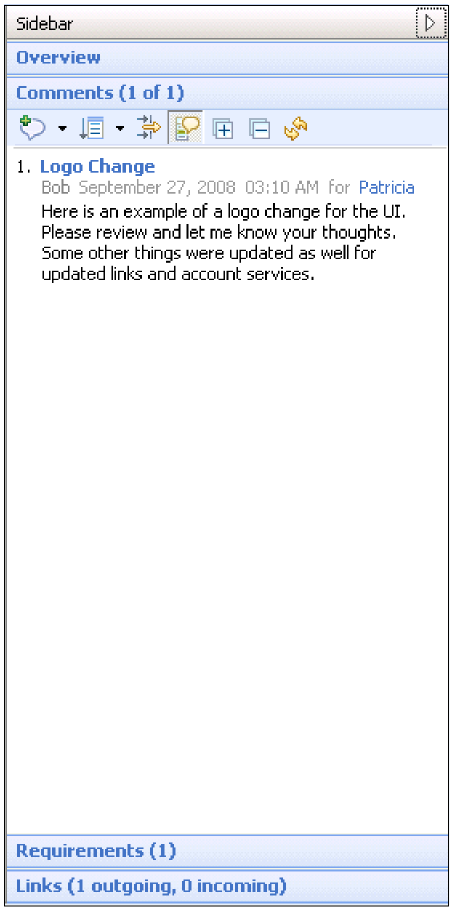
Options

Directed to: Patricia

Priority: Medium

Figure 5-54 UI re-brand sketch comments

The comment is displayed in the list of comments for that sketch (Figure 5-55).



1. Logo Change
Bob September 27, 2008 03:10 AM for Patricia
Here is an example of a logo change for the UI. Please review and let me know your thoughts. Some other things were updated as well for updated links and account services.

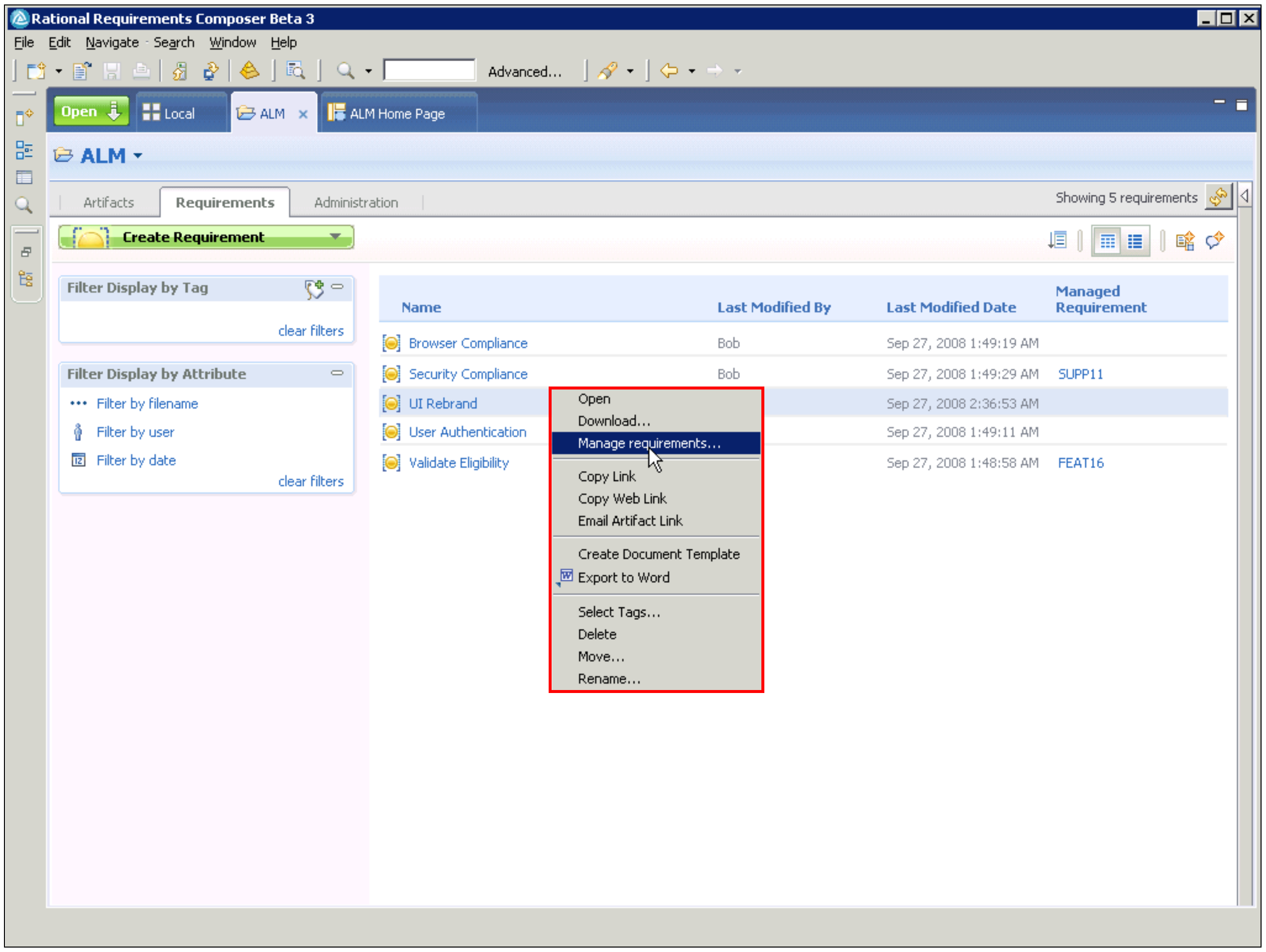
Figure 5-55 Comments list

b. Peer respond to the comment as necessary and then close the comment thread by selecting the **Resolve** option.

Managing the requirements

Now that there is an agreement on the UI changes, Bob is ready to capture more detailed requirements information in Rational RequisitePro. These activities, which are performed in Rational Requirements Composer, help set the context for the original need statement that he captured as a stakeholder request. He uses the requirement information that is captured for the logo that reflects UI branding in Rational RequisitePro Composer to create detailed requirements in Rational RequisitePro. He does this by using the Manage Requirements function as explained in the following steps:

1. Bob selects the requirement, which is named **UI Rebrand**, right-clicks, and selects selects the **Manage requirements** option (Figure 5-56). Choose **Add Requirement to RequisitePro**.



Name	Last Modified By	Last Modified Date	Managed Requirement
Browser Compliance	Bob	Sep 27, 2008 1:49:19 AM	
Security Compliance	Bob	Sep 27, 2008 1:49:29 AM	SUPP11
UI Rebrand		Sep 27, 2008 2:36:53 AM	
User Authentication		Sep 27, 2008 1:49:11 AM	
Validate Eligibility		Sep 27, 2008 1:48:58 AM	FEAT16

Figure 5-56 Selecting the Manage requirements option

2. In the Create Requirements in RequisitePro - Select Requirements window (Figure 5-57), he selects the requirement to manage, which is the UI re-brand in this example, and clicks **Next**.

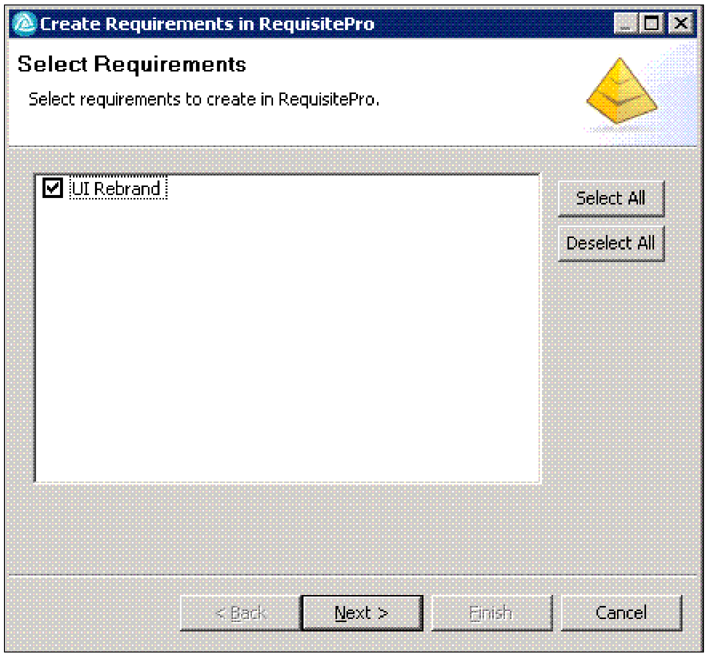
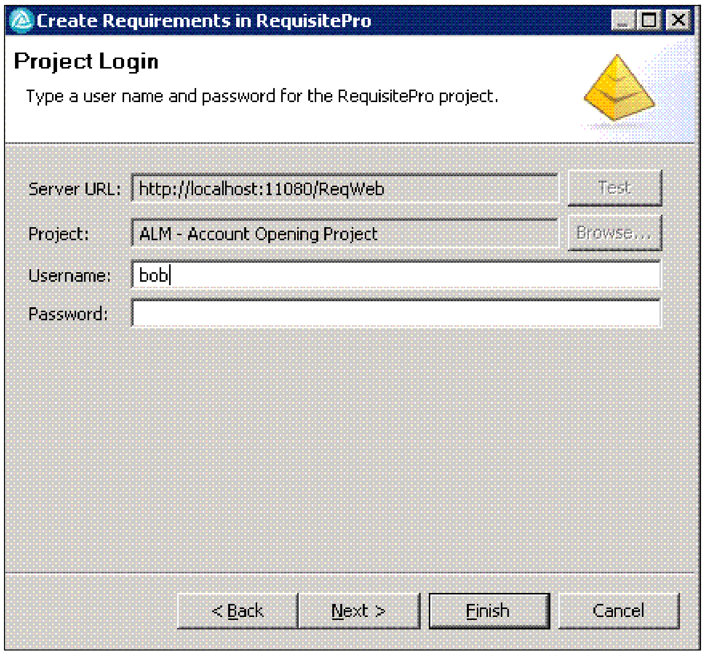

Create Requirements in RequisitePro
Select Requirements
Select requirements to create in RequisitePro.
☑ UI Rebrand
Select All
Deselect All
< Back
Next >
Finish
Cancel

Figure 5-57 Choosing the requirement to manage in RequisitePro

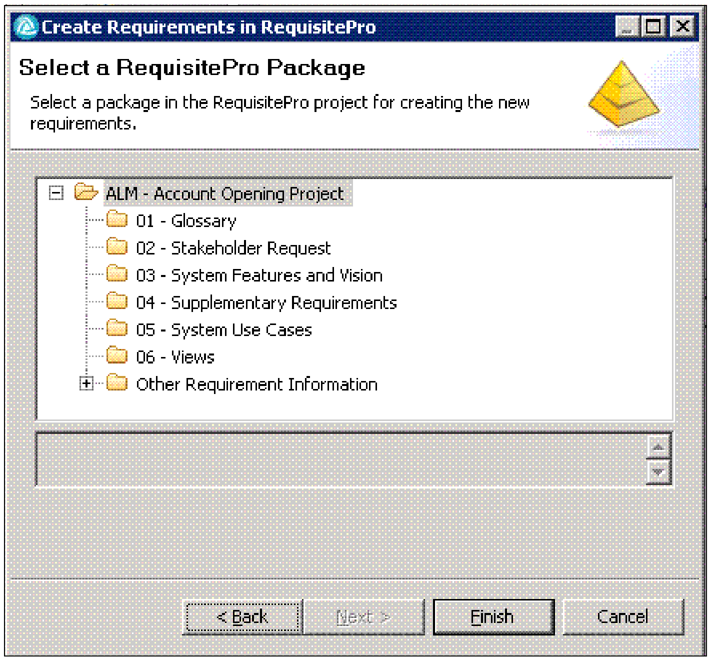
3. In the Project Login window (Figure 5-58), he logs in to the RequisitePro project and clicks **Finish** to complete the manage process.


Create Requirements in RequisitePro
Project Login
Type a user name and password for the RequisitePro project.
Test
Browse...
< Back
Next >
Finish
Cancel

Server URL: http://localhost:11080/ReqWeb
Project: ALM - Account Opening Project
Username: bob
Password:

Figure 5-58 RequisitePro login information

4. In the Select a RequisitePro Package window (Figure 5-59), he chooses the appropriate Rational RequisitePro package to define the requirement. In this example, a Feature requirement is created. Bob selects the **System Features and Vision** package and then clicks **Finish**.

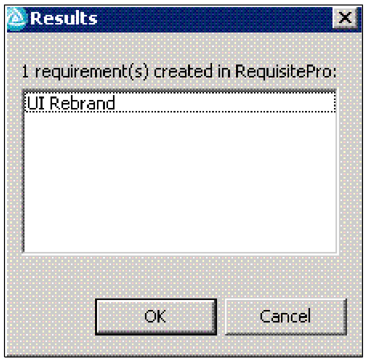

Create Requirements in RequisitePro
Select a RequisitePro Package
< Back
Next >
Finish
Cancel

Select a package in the RequisitePro project for creating the new requirements.

- ALM - Account Opening Project
  - 01 - Glossary
  - 02 - Stakeholder Request
  - 03 - System Features and Vision
  - 04 - Supplementary Requirements
  - 05 - System Use Cases
  - 06 - Views
  - Other Requirement Information

Figure 5-59 Selecting the RequisitePro requirement package

5. In the Results window (Figure 5-60), which indicates that a requirement has been created, he clicks **OK**.

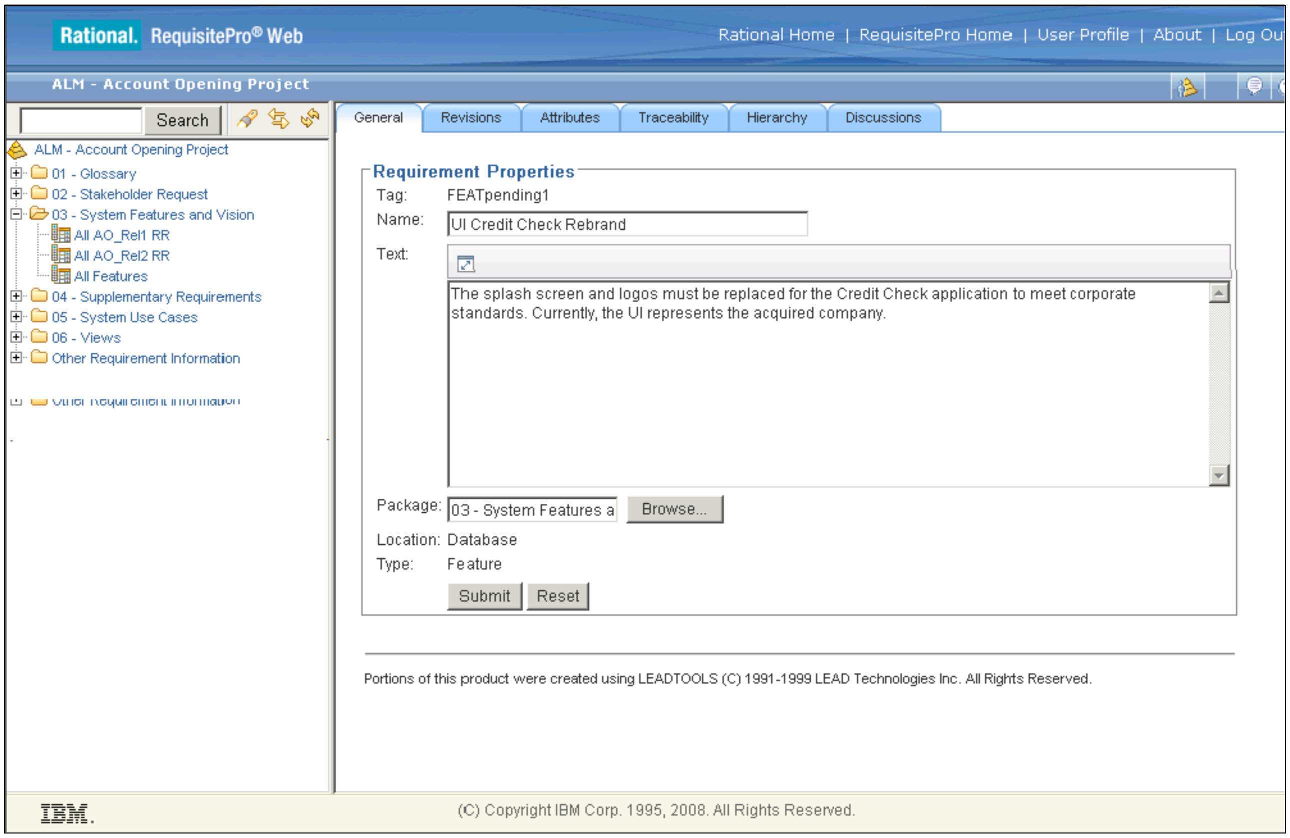


1 requirement(s) created in RequisitePro:

UI Rebrand

Figure 5-60 Requirement created

6. Bob reviews the requirement (Figure 5-61) to ensure that it is created in the System Features and Vision package. The requirement is read only because it contains rich text, and the Location field defines that it is referenced from a Rational Requirements Composer requirement.


Rational. RequisitePro® Web
Rational Home | RequisitePro Home | User Profile | About | Log Ou
ALM - Account Opening Project
Search
General
Revisions
Attributes
Traceability
Hierarchy
Discussions
ALM - Account Opening Project
01 - Glossary
02 - Stakeholder Request
03 - System Features and Vision
All AO_Rel1 RR
All AO_Rel2 RR
All Features
04 - Supplementary Requirements
05 - System Use Cases
06 - Views
Other Requirement Information
Browse...
Submit
Reset
IBM.

Requirement Properties

Tag:	FEATpending1
Name:	UI Credit Check Rebrand
Text:	The splash screen and logos must be replaced for the Credit Check application to meet corporate standards. Currently, the UI represents the acquired company.
Package:	03 - System Features a
Location:	Database
Type:	Feature

Portions of this product were created using LEADTOOLS (C) 1991-1999 LEAD Technologies Inc. All Rights Reserved.

(C) Copyright IBM Corp. 1995, 2008. All Rights Reserved.

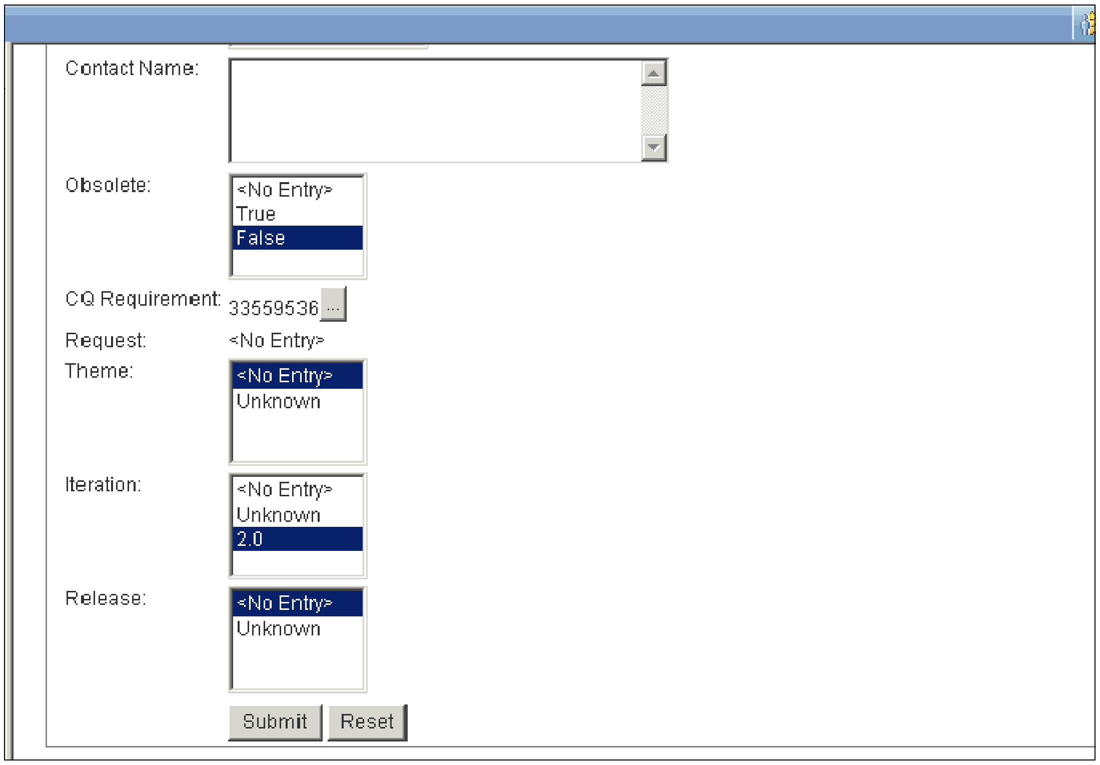
Figure 5-61 RequisitePro requirement created from managed Rational Requirements Composer requirement

Technical refinement of requirements in Rational RequisitePro

Goal: The goal is to provide attribute information to categorize requirements and identify pertinent traceability relationships between requirements.

The newly created feature requirement is now displayed in Rational RequisitePro. Bob also completes pertinent attributes, defines the iteration, and traces the feature back to the original stakeholder request in Rational RequisitePro. This step shows how the request was fulfilled.

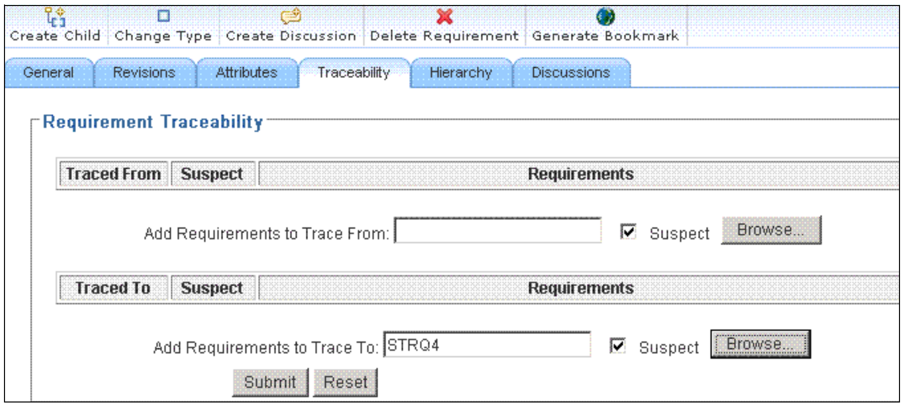
1. Bob provides information to help categorize the requirement. This information can help with project planning and so on. He clicks the **Attribute** tab and categorizes the requirements by using attribute information (Figure 5-62).



Contact Name:	
Obsolete:	<No Entry> / True / [✓ False]
CQ Requirement:	33559536
Request:	<No Entry>
Theme:	[✓ <No Entry>] / Unknown
Iteration:	<No Entry> / Unknown / [✓ 2.0]
Release:	[✓ <No Entry>] / Unknown

Figure 5-62 Requirement Attribute Information

2. Bob traces a feature requirement back to the stakeholder request. He clicks the **Traceability** tab to set pertinent trace relationships (Figure 5-63). As mentioned previously, as technical refinement of requirements occurs, pertinent traceability information is defined.



Traced From	Suspect	Requirements

Add Requirements to Trace From: [ ] ☑ Suspect

Traced To	Suspect	Requirements

Add Requirements to Trace To: STRQ4 ☑ Suspect

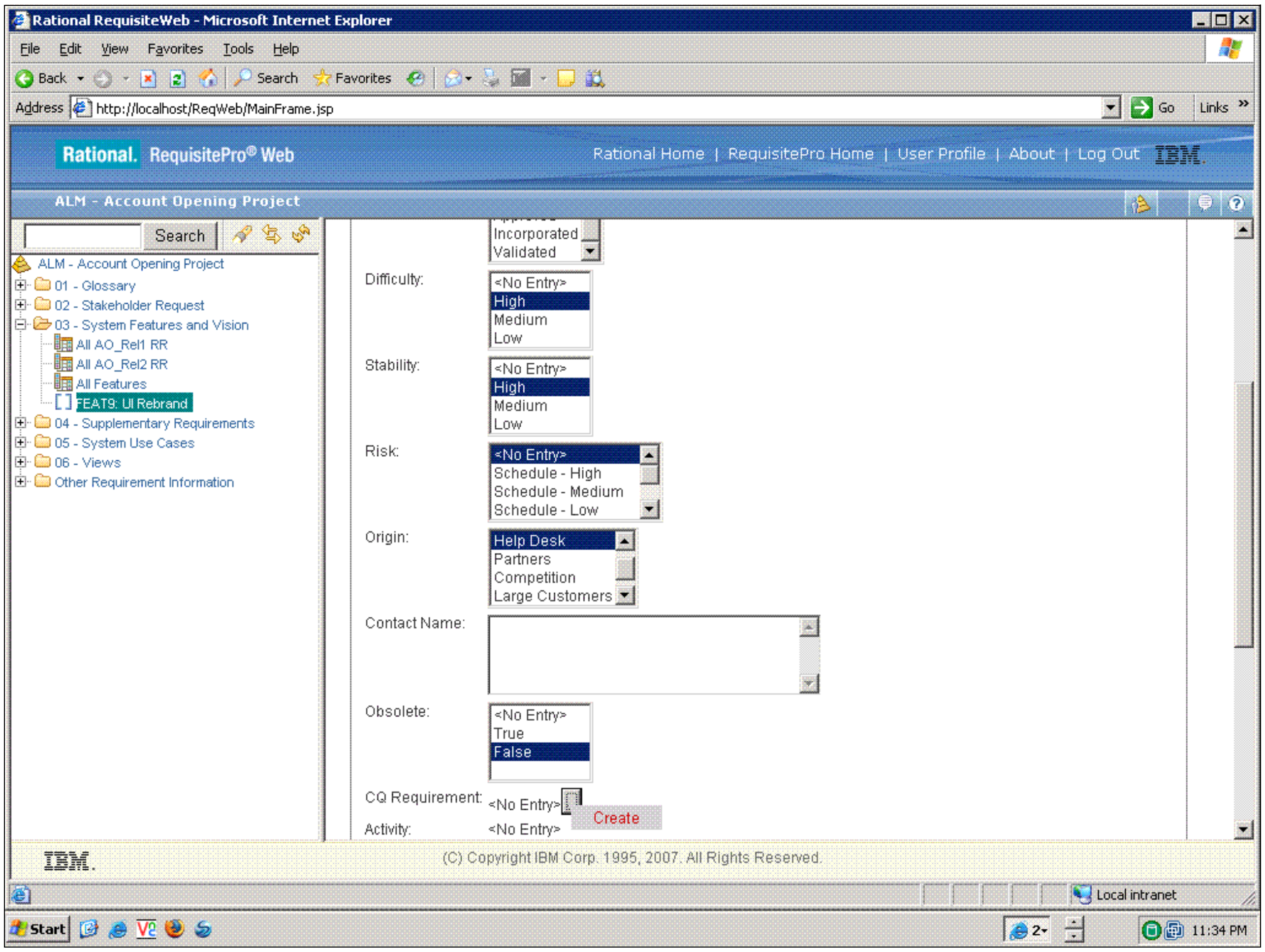
Figure 5-63 Requirement Traceability information

Creating a Rational ClearQuest requirement from Rational RequisitePro

After Bob creates the feature requirement, he establishes traceability to the request in Rational ClearQuest. This step is performed so that there might be an integrated "closed loop process" in Rational ClearQuest and Rational RequisitePro. The traceability is established by a Rational ClearQuest requirement that allows records to be linked between Rational ClearQuest and Rational RequisitePro, showing the requirement that fulfills the original request that is submitted in Rational ClearQuest.

Bob completes this step by creating a ClearQuest requirement attribute:

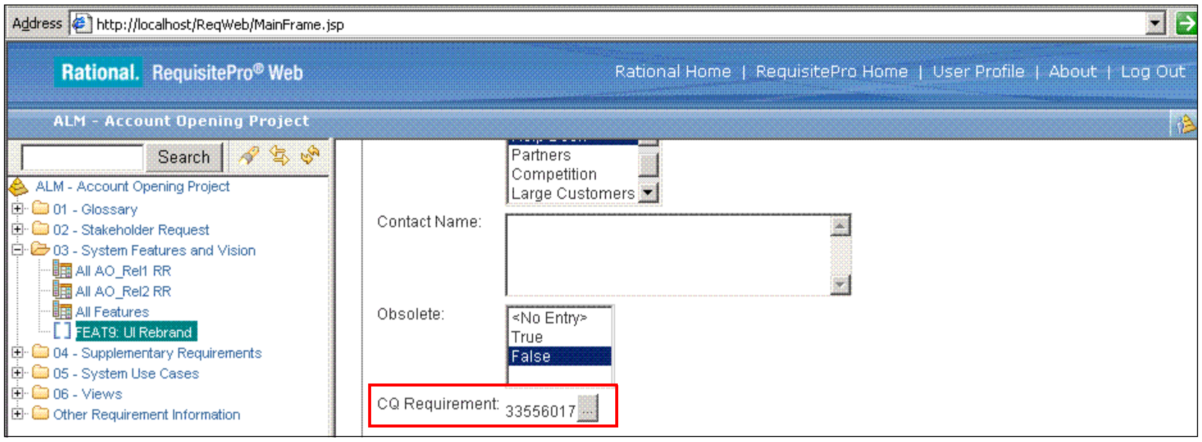
1. Bob navigates to the **Attributes** tab and selects the ClearQuest **Requirement** attribute.
2. He right-clicks the attribute and selects **Create** (Figure 5-64).


Rational RequisiteWeb - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/ReqWeb/MainFrame.jsp
Go Links
Rational. RequisitePro® Web
Rational Home | RequisitePro Home | User Profile | About | Log Out
ALM - Account Opening Project
Search
ALM - Account Opening Project
01 - Glossary
02 - Stakeholder Request
03 - System Features and Vision
All AO_Rel1 RR
All AO_Rel2 RR
All Features
FEAT9: UI Rebrand
04 - Supplementary Requirements
05 - System Use Cases
06 - Views
Other Requirement Information
(C) Copyright IBM Corp. 1995, 2007. All Rights Reserved.
Local intranet
Start
11:34 PM

	Incorporated, Validated
Difficulty:	<No Entry>, [✓ High], Medium, Low
Stability:	<No Entry>, [✓ High], Medium, Low
Risk:	[✓ <No Entry>], Schedule - High, Schedule - Medium, Schedule - Low
Origin:	[✓ Help Desk], Partners, Competition, Large Customers
Contact Name:	
Obsolete:	<No Entry>, True, [✓ False]
CQ Requirement:	<No Entry> Create
Activity:	<No Entry>

Figure 5-64   Creation of a Rational ClearQuest requirement from Rational RequisitePro

3. He logs in to Rational ClearQuest in the window that opens. He provides the login and password and selects the appropriate schema and database. The Rational ClearQuest unique ID for the requirement is now shown (Figure 5-65).


Address http://localhost/ReqWeb/MainFrame.jsp

Rational. RequisitePro® Web
Rational Home | RequisitePro Home | User Profile | About | Log Out
ALM - Account Opening Project
Search
ALM - Account Opening Project
01 - Glossary
02 - Stakeholder Request
03 - System Features and Vision
All AO_Rel1 RR
All AO_Rel2 RR
All Features
FEAT9: UI Rebrand
04 - Supplementary Requirements
05 - System Use Cases
06 - Views
Other Requirement Information
Partners
Competition
Large Customers
<No Entry>
True

Contact Name:

Obsolete: False

CQ Requirement: 33556017

Figure 5-65   ClearQuest Requirement unique ID

The requirement can then be linked to the Rational ClearQuest Define Requirements activity as explained in the following section.

Linking the requirement to an activity

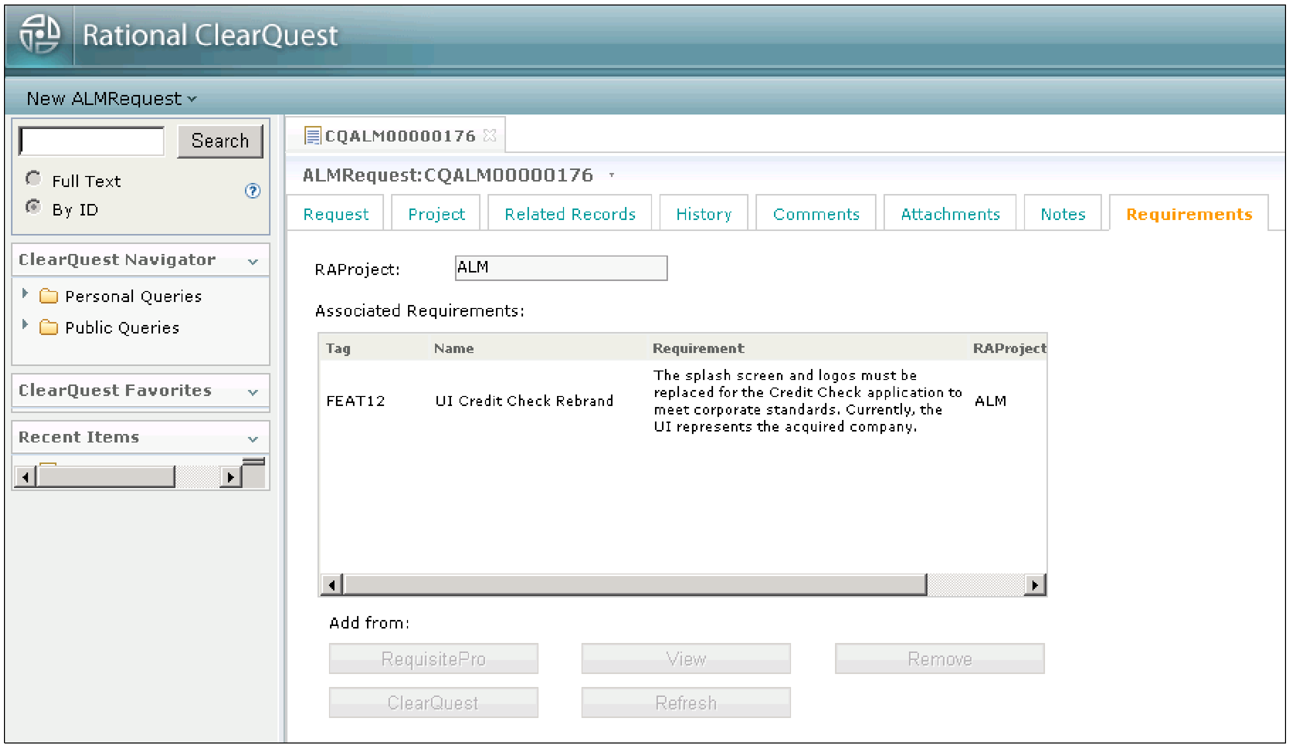
Goal: The goal is to establish traceability between the requirements definition work that was just created to the existing work activities that are associated with the change request.

Upon completion of Bob’s requirement work in Rational RequisitePro and Rational Requirements Composer, he wants to link the requirement, which was created in RequisitePro and pushed over to ClearQuest, to the original ALMRequest record for traceability purposes.

He perform the following steps:

1. Bob browses to the ClearQuest Web URL.
2. He enters the ClearQuest login user name and password, selects a valid schema repository name from the list and clicks **Connect**.
3. He opens the ALMTask that he is working on by searching its record ID or running the **All Tasks** → **Run** query and browsing the search result.
4. He navigates to the **Related Records** tab and double-clicks the UI branding request in the Request list.
5. In the ALMRequest, he navigates to the **Requirements** tab and clicks the **Modify** button to place the record in a modifiable state.
6. From the RAProject list, he selects **AccountOpening**.
7. He clicks the **ClearQuest** button on the form.
8. He clicks the **Search** button in the new window to query all requirement records.
9. He selects the desired requirement from the list and clicks **OK**.

After the requirement is associated with the activity, it is listed on the Requirements tab as shown in Figure 5-66.



RAProject: ALM

Associated Requirements:

Tag	Name	Requirement	RAProject
FEAT12	UI Credit Check Rebrand	The splash screen and logos must be replaced for the Credit Check application to meet corporate standards. Currently, the UI represents the acquired company.	ALM

Figure 5-66 Linked requirement record ALMRequestRecord

The association can also be reviewed by using Rational RequisitePro. Figure 5-67 shows the traceability association in Rational RequisitePro.

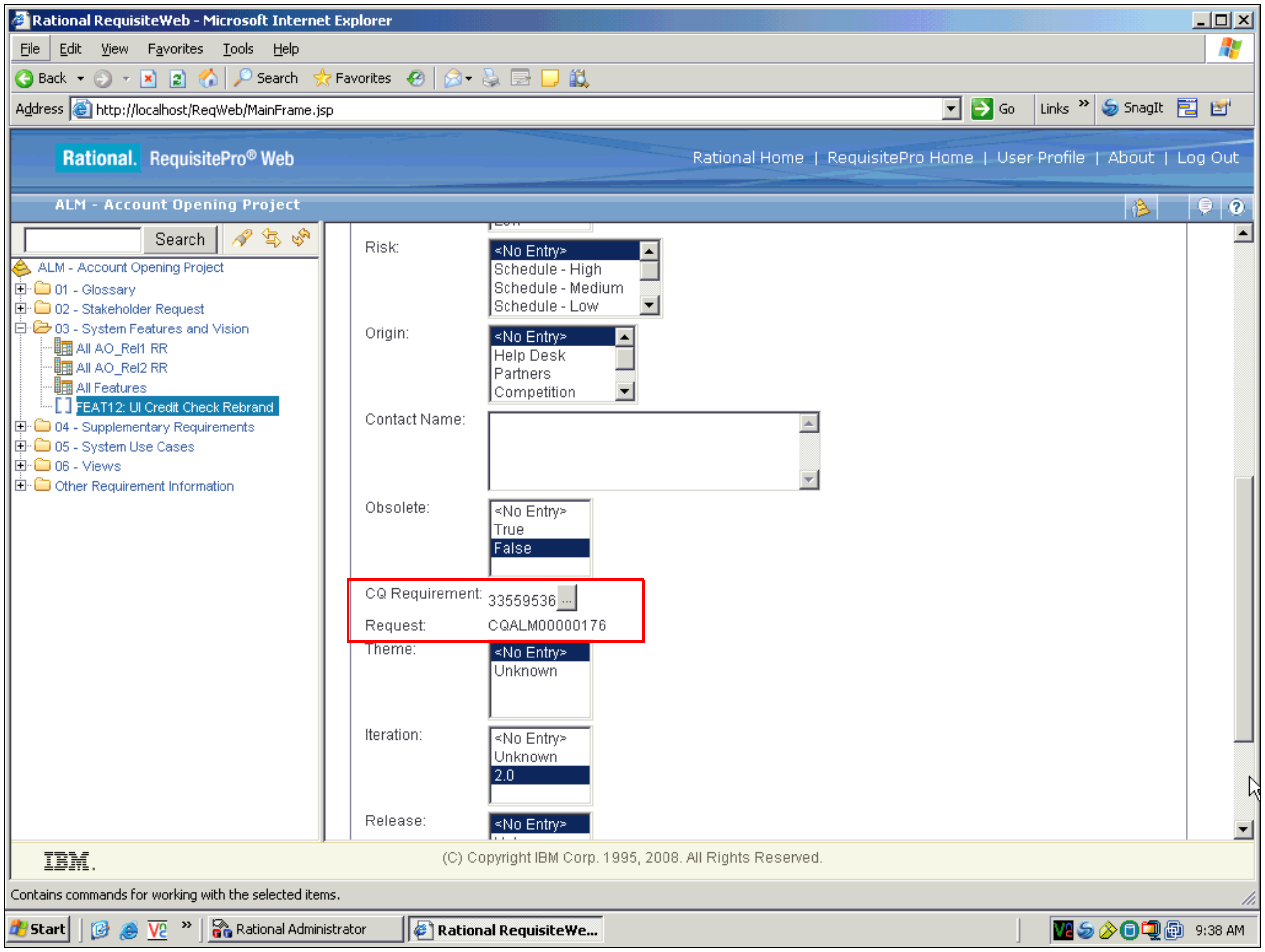

Rational RequisiteWeb - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/ReqWeb/MainFrame.jsp
Go Links SnagIt
Rational. RequisitePro® Web
Rational Home | RequisitePro Home | User Profile | About | Log Out
ALM - Account Opening Project
Search
ALM - Account Opening Project
01 - Glossary
02 - Stakeholder Request
03 - System Features and Vision
All AO_Rel1 RR
All AO_Rel2 RR
All Features
FEAT12: UI Credit Check Rebrand
04 - Supplementary Requirements
05 - System Use Cases
06 - Views
Other Requirement Information
Risk:
<No Entry>
Schedule - High
Schedule - Medium
Schedule - Low
Origin:
<No Entry>
Help Desk
Partners
Competition
Contact Name:
Obsolete:
<No Entry>
True
False
CQ Requirement: 33559536
Request: CQALM00000176
Theme:
<No Entry>
Unknown
Iteration:
<No Entry>
Unknown
2.0
Release:
<No Entry>
IBM.
(C) Copyright IBM Corp. 1995, 2008. All Rights Reserved.
Contains commands for working with the selected items.
Start
Rational Administrator
Rational RequisiteWe...
9:38 AM

Figure 5-67 Rational ClearQuest design requirement activity in Rational RequisitePro

After saving the record Bob's work is complete. He can now transition the record into a Complete state by using the complete action and add any additional notes to the record for Patricia and the rest of his team:

1. Bob clicks the **Complete** button.
2. He provides any necessary information or notes to the description.
3. He clicks the **Submit** button to save the changes.

5.6 Life-cycle collaboration

In this scenario, the actors not only created new assets, but relied on the assets of their team members who play various roles on the team. In addition, the work produced by these team members will impact future scenes of the storyboard. We look at the following life-cycle assets that are used by the characters in this act:

- ALM Project (Rational ClearQuest)
- ALM Phase (Rational ClearQuest)
- ALM Iteration (Rational ClearQuest)
- ALM Request (Rational ClearQuest)

- ALM Task (Rational ClearQuest)
- ALM Activities (Rational ClearQuest)
- Work Item (Rational Team Concert)
- Iteration Plan (Rational Team Concert)
- Work item (Rational Quality Manager)
- Test Plan (Rational Quality Manager)
- Test Case (Rational Quality Manager)
- Exit Criteria (Rational Quality Manager)
- Test Environment (Rational Quality Manager)
- Lab Request (Rational Quality Manager)
- Requirement (Rational RequisitePro)
- Requirement (Rational ClearQuest)
- Sketch (Rational Requirements Composer)

5.7 Planning and measuring success in change management

In this section, we describe how projects plan and measure success in change management by using the Rational ALM solution.

5.7.1 Reporting with ClearQuest ALM

Rational ClearQuest supports customized searches in the repository by using queries. The ClearQuest ALM contains a set of preconfigured queries to be used by the preconfigured project roles in the ALM schema, which are development leads, developers, test leads, and testers. Utility queries are also supported for typical change management activities, such as triaging incoming requests, identifying completed tasks, and identifying duplicate requests. The ClearQuest ALM queries are available in the **Public Queries** → **ALM** → **My Project** query folder.

ClearQuest ALM also contains a set of queries to search for most all ALM record types. These general queries are ideal when maintaining repository information or as templates for personal queries after adding an additional filter. The general queries are available in the **Public Queries** → **ALM** → **General** query folder.

For convenience, a set of commonly used queries can be copied into the Personal Queries and configured for the current project and current iteration. To copy a query:

1. Select the query to copy in the Rational ClearQuest Navigator and choose **Copy**.
2. Select the **Personal Queries** element and choose **New Folder**.
3. Rename the query to the name of the project or iteration, for example, “Current Iteration.”
4. Select the new folder and choose **Paste**.
5. Select the new query and choose **Edit**.
6. Modify the filter values to match the project and category in the current iteration by using the Query Editor wizard. Add additional filters as needed, for example to identify records of a specific phase, iteration, owner, or state.

5.7.2 Reporting with the Business Intelligence Reporting Tool

The collaborative development blueprint identifies team health and reporting as one of the cornerstones in successful change management. More and more organizations today have diverse sets of stakeholders that require different views of change information in order to support their needs for project health. To satisfy this need, flexible enterprise reporting options are required. Truly flexible enterprise reporting must be capable of gathering, joining, and presenting multiple data sets from multiple data sources.

The Business Intelligence Reporting Tool (BIRT) Eclipse project was established to satisfy the need for software delivery reporting. BIRT is an Eclipse technology-based open source reporting project. To learn more about this project, see the Web site at the following address:

`http://www.eclipse.org/birt/phoenix`

Open, flexible software delivery reporting with BIRT

BIRT reporting is based on data pull rather than a data set pushing to a report. This model allows you to query on multiple data sets and data sources, as well as join the data sets as you like. BIRT allows you to create rich variety of reports:

- Lists

  Lists are the simplest reports to create. For example, a quality management team can create a simple list report that lists all defects that have been resolved in a particular build. They can enhance this report and group the resolved defects by component. They can also provide summary information such as the total number of resolved defects for each component.

- Charts

  By using charts, you can aggregate numeric data and display it visually. With BIRT, you can create a rich set of charts including pie charts, line, bar charts, and many more. Charts can support events that allow you to “drill down” into the data to get a more specific view. For example, you might have a pie chart that displays defects across the various states (submitted, open, resolved, tested, closed, and so on) for a particular iteration. You can then drill down into the particular state to get a list of all the related defects.

- Crosstabs

  Crosstabs or matrixes are like data presented in tables or spreadsheets. They display data in two dimensions. For example, you might have a crosstab that shows all the open work items for each member of your project team by state (submitted, open, resolved, tested, closed, and so on).

- Letters and documents

  Textual documents are easy to create with BIRT. Textual documents can include multiple report elements such as a letter to a particular set of stakeholders that includes various data sets. Stakeholders particularly want to know about lists of defects, enhancements, and features that were implemented that they had requested, along with a list of related changes implemented that they might be interested in.

- Compound reports

  Compound reports can bring together multiple reporting elements to present a “whole story.” For example, you might have a milestone report that displays the requirements that have been implemented for the milestone, confirmed severe/critical defects, or return on investment (ROI) for work items that were implemented.

Enterprise change management reporting must be flexible enough to meet the needs of various stakeholders. For team members who are involved in application delivery, access to reporting within their local client is important. Local access to report authoring and execution within their software delivery environment allows them to create adhoc reports to answer tactical questions. Rational ClearQuest provides BIRT Designer within the Rational ClearQuest Eclipse client, which gives these team members a flexible reporting solution to meet their needs.

Additionally, stakeholders that may not operate within Eclipse also need access to reports. Rational ClearQuest provides an enterprise reporting server where reports can be executed and displayed via a standard Internet browser to meet these stakeholder needs.

Also, customers can embed Rational ClearQuest BIRT reports into their applications by using the BIRT API. This increases the ability for the organization to use Rational ClearQuest reports in various contexts throughout the organization.

Creating a Rational ClearQuest report with BIRT

The Rational ClearQuest Eclipse client includes the BIRT Designer, which allows you to create BIRT reports from your Eclipse client as illustrated in Figure 5-68.

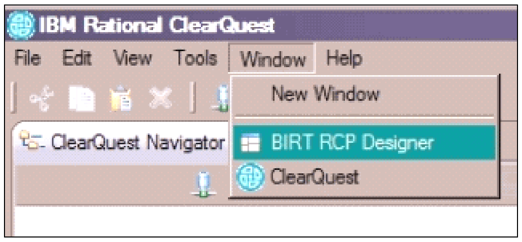


Figure 5-68 Report design with the Rational ClearQuest client for Eclipse

Switching to the BIRT RCP Designer places you in the BIRT perspective, allowing you to create new reports, libraries, or templates (Figure 5-69).

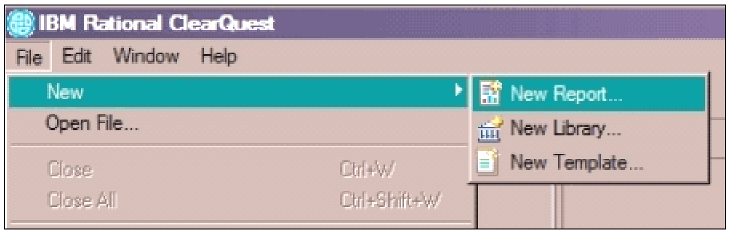
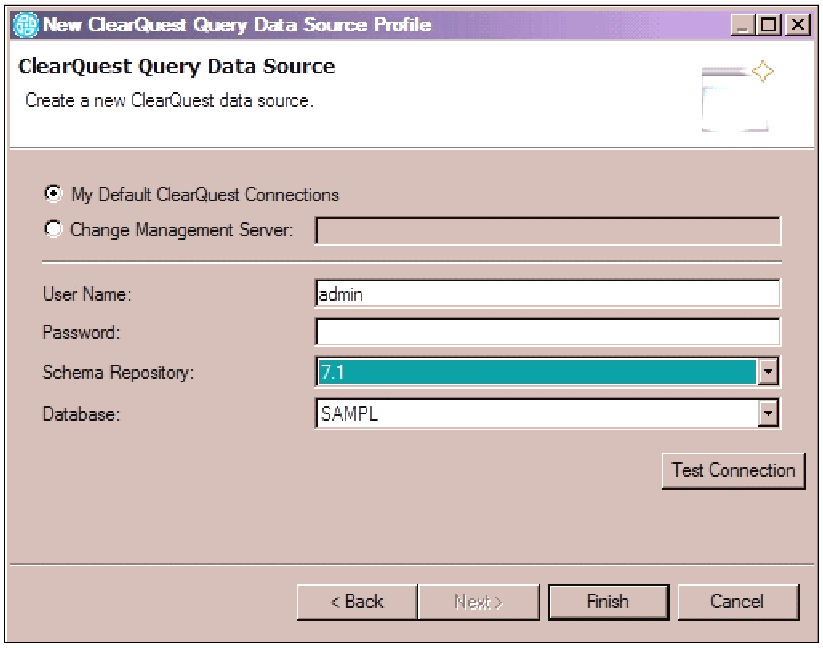


Figure 5-69 Reports, report libraries, report templates created with the Rational ClearQuest BIRT client

The fundamental building blocks of a report with BIRT are data sources and data sets.

BIRT accesses the Rational ClearQuest databases by using the Rational ClearQuest-provided Open Data Access (ODA) model. By using this method of access to Rational ClearQuest instead of directly providing access to the databases via JDBC™ or SQL, the Rational ClearQuest security model is preserved and you do not have to provide database IDs and passwords freely to anyone who needs to develop reports.

Data sources essentially map to Rational ClearQuest user databases (Figure 5-70). These are the databases that Rational ClearQuest uses to store change data.


New ClearQuest Query Data Source Profile

ClearQuest Query Data Source

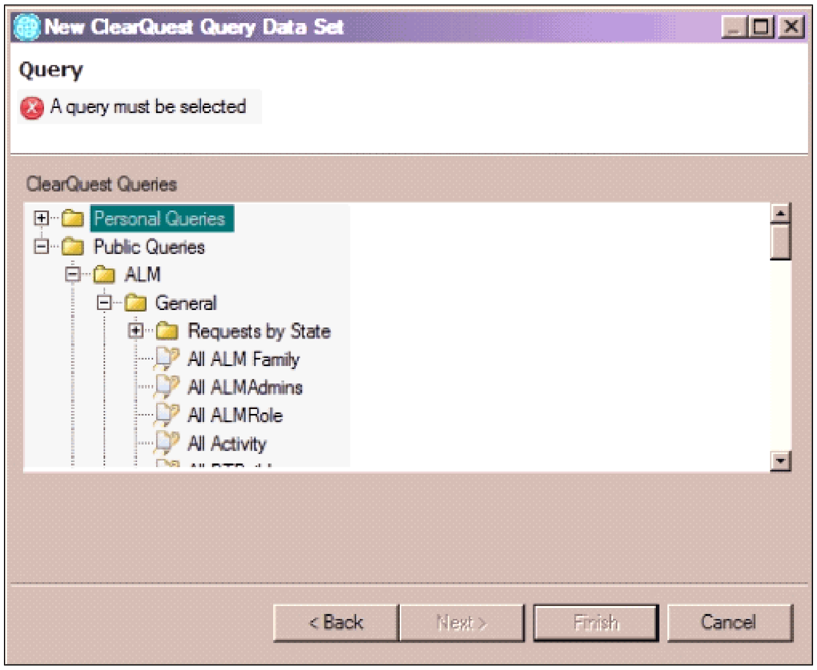
Create a new ClearQuest data source.

(●) My Default ClearQuest Connections
( ) Change Management Server:

User Name: admin
Password:
Schema Repository: 7.1
Database: SAMPL

Figure 5-70 Rational ClearQuest data sources to preserve the Rational ClearQuest security model

Data sets are analogous to Rational ClearQuest built-in queries. You must create Rational ClearQuest queries for the data sets that you will use in your BIRT reports (Figure 5-71).


New ClearQuest Query Data Set

Query

A query must be selected

ClearQuest Queries

- Personal Queries
- Public Queries
  - ALM
    - General
      - Requests by State
      - All ALM Family
      - All ALMAdmins
      - All ALMRole
      - All Activity

Figure 5-71 Creating Rational ClearQuest queries for each Rational ClearQuest data set you will use

After you create your data sources and data sets, they become a part of the arsenal of data that you can use to create reports. Your data palette can include multiple data sources and data sets, with which you can create powerful, complex reports that pull data from multiple Rational ClearQuest user databases and queries (Figure 5-72).

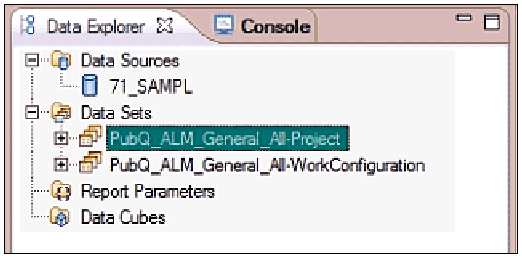


Figure 5-72   Data sources and data sets, fundamental building blocks for reports

The report designer (Figure 5-73) provides drag-and-drop controls to make it easy to create a rich variety of reports. This feature greatly simplifies and accelerates the design and development process for reporting, making it possible to generate adhoc reports to get tactical answers without a significant investment of time or effort.

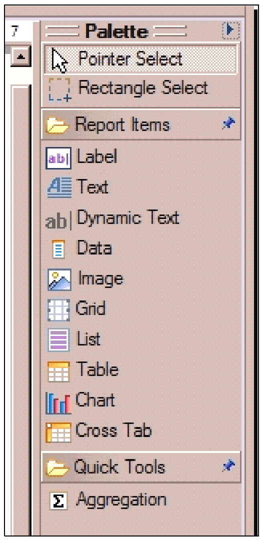


Figure 5-73   Report palette for dragging reporting elements for fast report creation

One of the most powerful enterprise reporting capabilities with Rational ClearQuest BIRT is the focus on reuse that is built into the product. Report libraries can be created to store common reporting elements such as data sources and data sets (Figure 5-74).

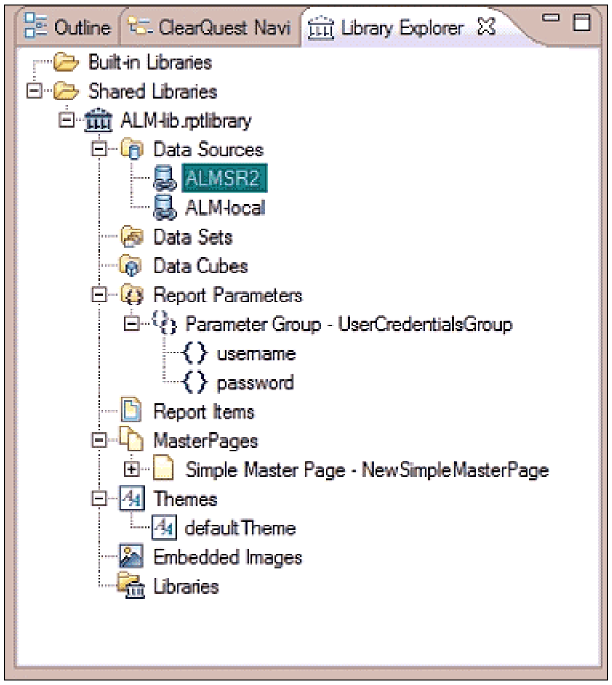


Figure 5-74 Libraries making reuse simple and powerful across the enterprise

Additionally, report templates provide the ability to save common report designs so that they can be easily used and reused with different data sources or data sets. Report templates provide the power to create reporting formats and styles that meet the specific needs of various stakeholders. You can then repeatedly use the templates to provide various report data in a consistent style and format that meets the needs of your stakeholders.

5.7.3 Reporting team health with Rational ClearQuest and BIRT

In this section, we discuss the use of Rational ClearQuest and BIRT to report team health.

Measuring flow: Finding bottlenecks in change management

Measuring flow can help you find and diagnose bottlenecks in your change management process. *Flow* for your change management process can be defined as the progression of change requests through a workflow from creation to completion.

Measuring flow can help you demonstrate performance against service-level agreements (SLAs). It can also help you measure the impact of changes to your change management process (Figure 5-75).

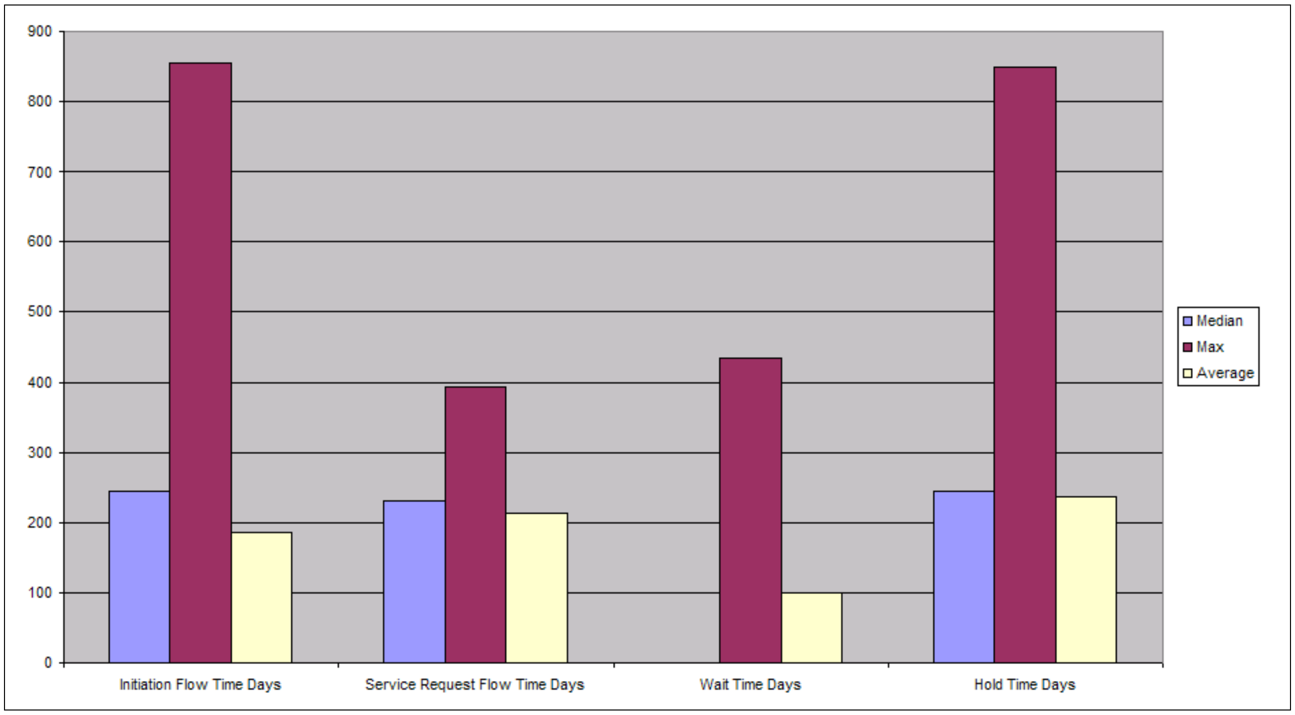


Figure 5-75 Example of measuring flow in the change management process

Managing iterations

As discussed in “Creating ALMTask records for iteration planning” on page 133, ClearQuest ALM allows tasks can be assigned to iterations of a project. By doing this, the project team can balance their workload across iterations. Additionally, charts, such as the one in Figure 5-76 on page 184, can be created to see how the work is distributed across the team. This insight helps project managers, such as Patricia, to spread the workload evenly across the team members and avoid “critical path” situations. Such charts also help the project manager to ensure that all work is assigned.

In Figure 5-76, there are five tasks in the “No value” column. No value means that they are not assigned and can act as a trigger to the project manager that some work is slipping through the cracks. Also note that the Construction phase is evenly balanced across each team member, while the workload shifts in the Transition phase as expected. By running charts such as this, project managers can govern their project more effectively, ensure that all work is assigned and everyone is active, and prevent overwhelming individual team members with too much work.

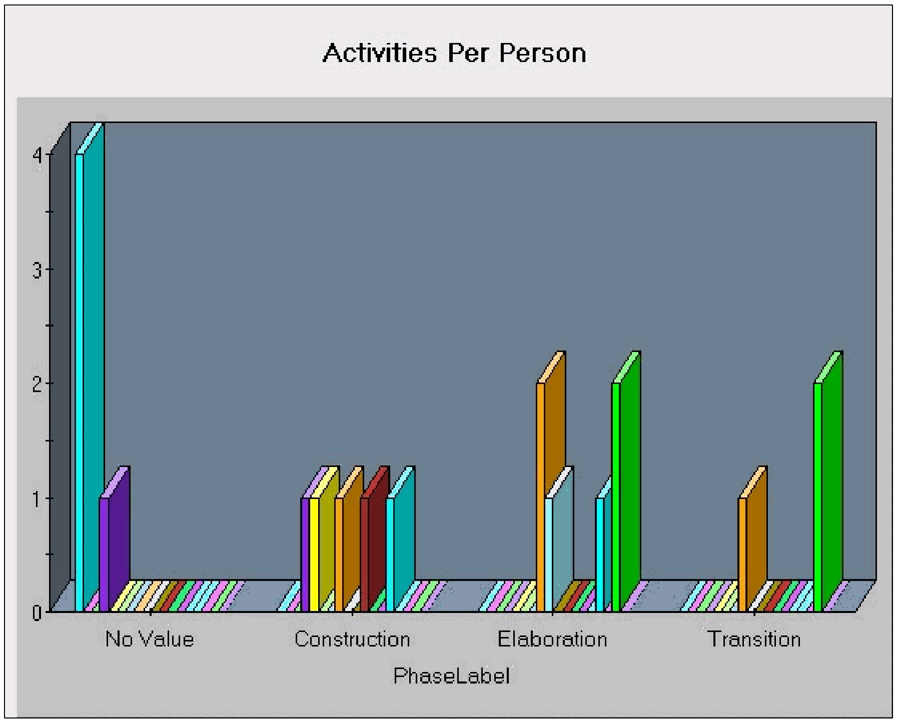
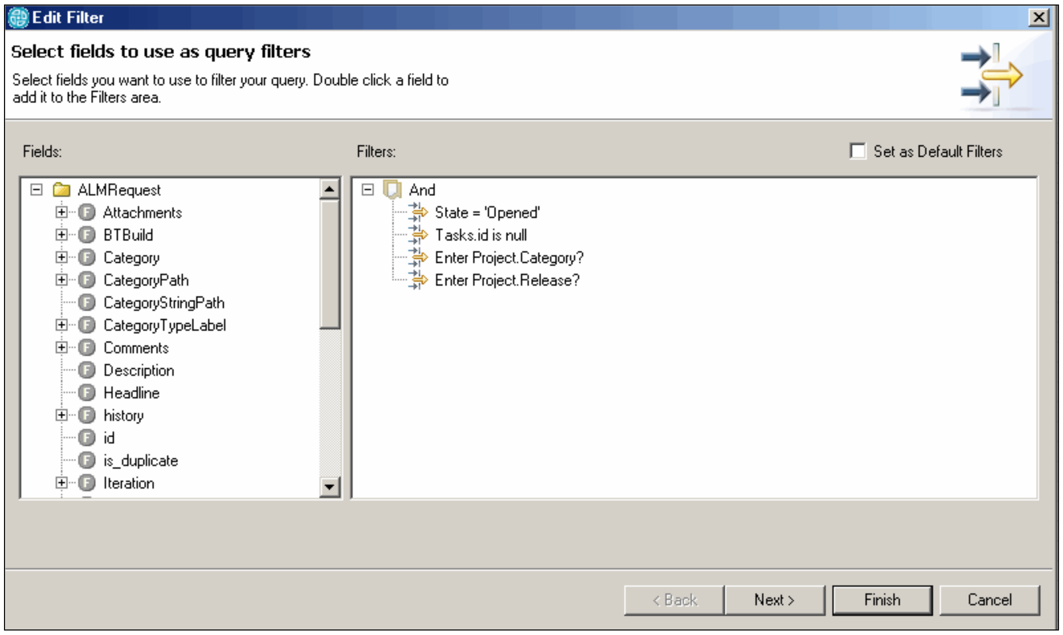


Figure 5-76   Example of chart of load balancing over project phases

Managing new requests

In her role as project manager and triage administrator, Patricia has to periodically query for newly submitted requests and determine whether to accept or reject them. To access the triage queue for an ALM project, Patricia can use the preconfigured **Public Queries** → **ALM** → **General** → **Find Request by Project** query. This query finds all requests for a project. She can modify this query to only find all requests in the Open state with no tasks assigned as shown in Figure 5-77.



Select fields to use as query filters

Select fields you want to use to filter your query. Double click a field to add it to the Filters area.

And
State = 'Opened'
Tasks.id is null
Enter Project.Category?
Enter Project.Release?

Figure 5-77   Defining a query to triage incoming requests

To understand the rate of incoming and closed requests, Patricia can also monitor a chart that shows the count of open versus closed requests in the current project. See 5.5.2, “Patricia updates the project iteration plan” on page 130, which exemplifies the actions that Patricia takes when acting on new requests.

Managing tasks

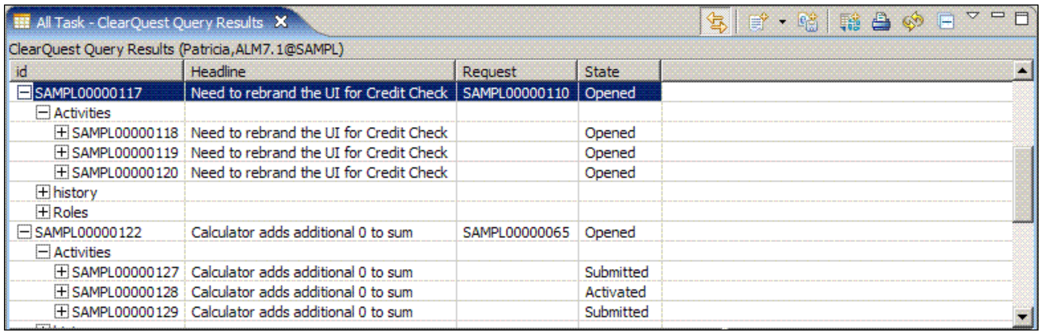
In her role as project manager, Patricia must periodically measure the progress and health of the project iteration. The component development leads also monitor health, but with a focus on their areas of responsibilities. Patricia must also validate the teams’ progressions of the roll-up of tasks into the requests that are committed for the current iteration.

Patricia must monitor the following key situations and metrics, among others:

- Identify tasks that are not planned for execution by the owning team.
- Identify tasks that are slipping or are not worked on and progressing to Active state.
- Identify tasks with all completed activities that should transition to a Completed state.
- Identify requests with all completed tasks that should transition to a Completed state.
- Identify tasks that are impacted by change to requests, for example, withdrawing planned work that relates to requests that are being rejected or withdrawn.

It is a best practice to configure the commonly used queries in the Public or Private queries folders. Roles that manage multiple projects or iterations can organize the specific queries in subfolders as exemplified in 5.7.1, “Reporting with ClearQuest ALM” on page 177.

The state and progress of a request, a task, and activities are monitored by using the general queries and applying filters on a project, phase, and iteration. As shown in Figure 5-78, additional information, such as the state of contained items, can be retrieved by expanding items in the query.



ClearQuest Query Results (Patricia,ALM7.1@SAMPL)

id	Headline	Request	State
SAMPL00000117	Need to rebrand the UI for Credit Check	SAMPL00000110	Opened
Activities			
SAMPL00000118	Need to rebrand the UI for Credit Check		Opened
SAMPL00000119	Need to rebrand the UI for Credit Check		Opened
SAMPL00000120	Need to rebrand the UI for Credit Check		Opened
history			
Roles			
SAMPL00000122	Calculator adds additional 0 to sum	SAMPL00000065	Opened
Activities			
SAMPL00000127	Calculator adds additional 0 to sum		Submitted
SAMPL00000128	Calculator adds additional 0 to sum		Activated
SAMPL00000129	Calculator adds additional 0 to sum		Submitted

Figure 5-78 Monitoring the state of tasks and contained activities

Unaligned work

During project iteration planning, Patricia must validate that all component and functional teams are planning to complete all assigned work within the scope of the iteration. By running a query that selects all tasks that are assigned to the current iteration, and comparing the Iteration field, she can identify any work that is not aligned with her project iteration plan.

In 5.5.5, “Patricia confirms the project iteration plan” on page 155, we illustrate the actions that Patricia takes when ensuring the alignment of work.

5.8 Reference architecture and configuration

In this section, we describe how Rational RequisitePro, Rational Requirements Composer, and Rational ClearQuest are used in this act of the story. We also explain how they fit into the overall ALM architecture and how they are configured.

5.8.1 Fitting into the enterprise ALM solution

In this section, we discuss pertinent information about the architecture for the tools that are used in the requirements definition and management space for Rational RequisitePro, Rational Requirements Composer, and Rational ClearQuest.

Scenario considerations

As shown in Figure 5-79, Rational Requirements Composer, Rational RequisitePro, and Rational ClearQuest play an integral role in the success of the scenario.

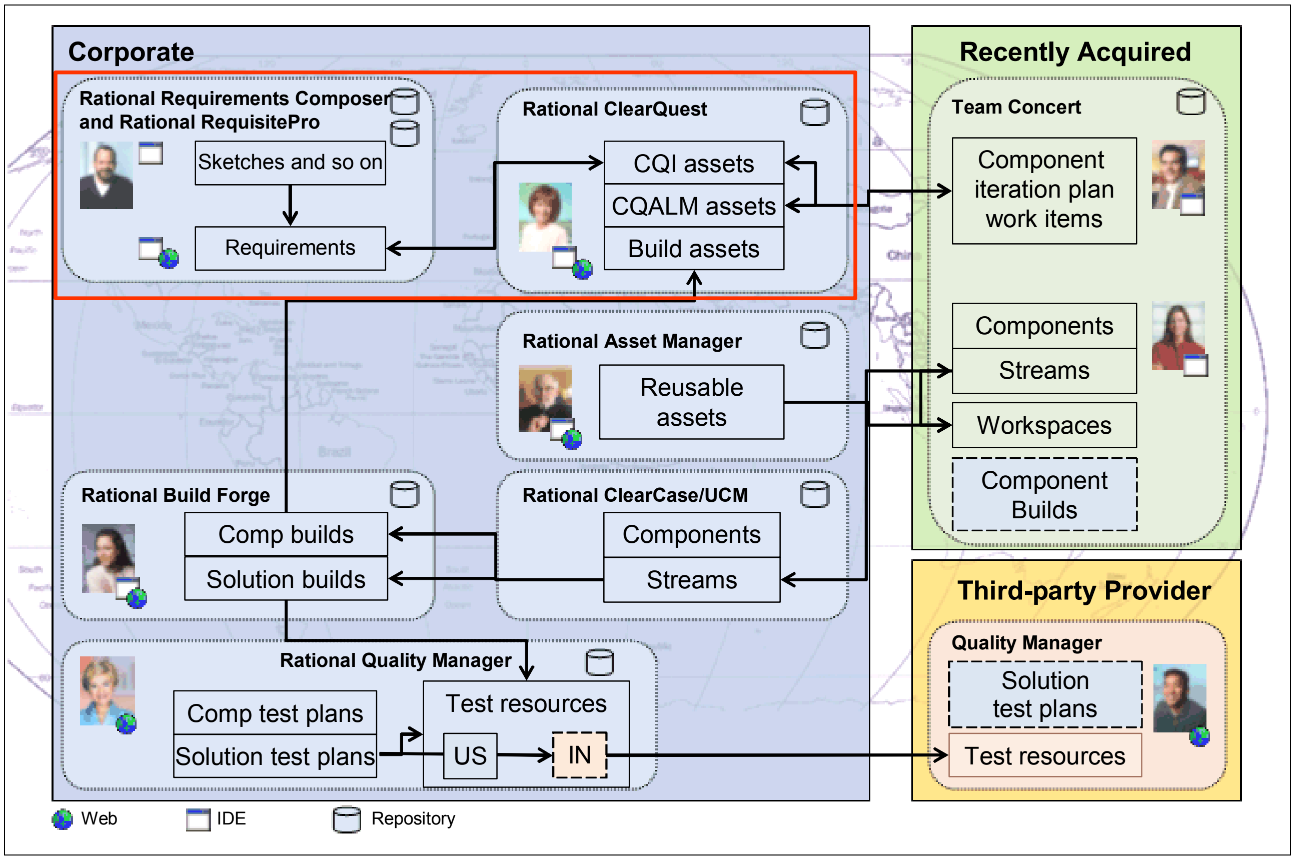


Figure 5-79 Team interoperative repository configuration

Requirements information provides a foundation for other software development activities that occur later in the life cycle.

Rational RequisitePro provides a mechanism for managing requirements across the life cycle. Information regarding the original request that Bob captured in Rational ClearQuest is available in Rational RequisitePro. Basic information from the original need statement captured in Rational ClearQuest is available in Rational RequisitePro. Often there is not enough detail to implement the request. As a result, further requirements definition activities must occur. These elicitation activities provide additional information so that a requirement can be detailed. The elicitation type activities are captured in Rational Requirements Composer. Artifacts that are captured in Rational Requirements Composer help drive the requirements definition process. Requirements content can be flushed out through activities to generate business process sketches, use cases, interface flows, sketches, and storyboards regarding the UI branding for Account Opening as shown in Figure 5-80.

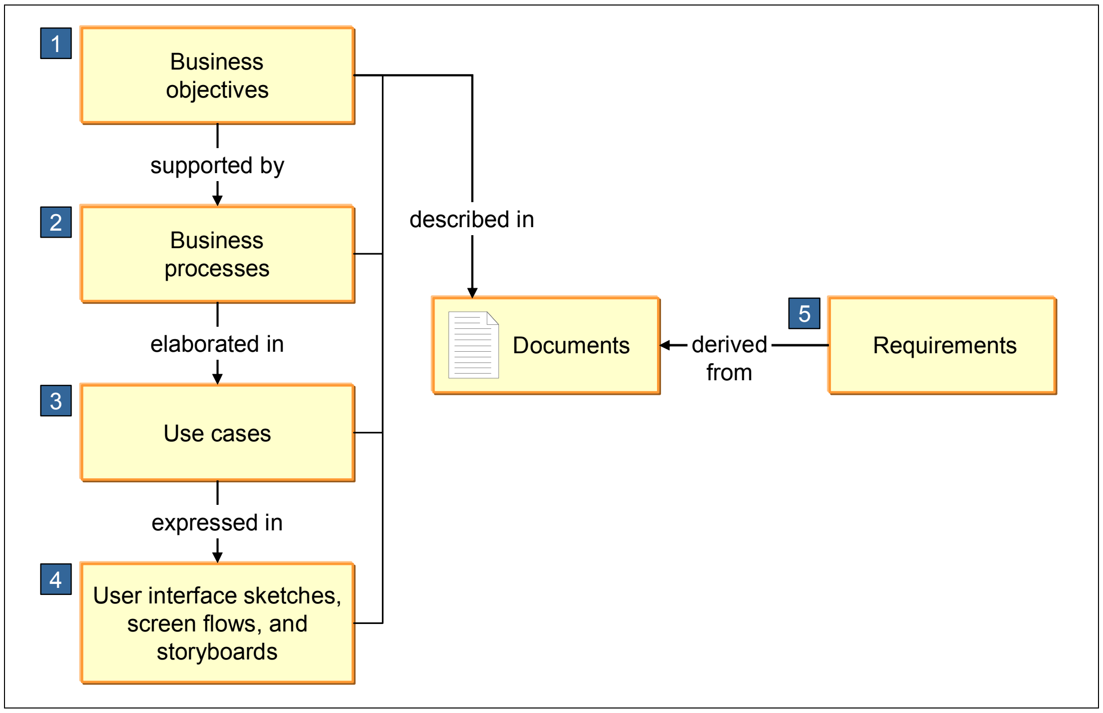


Figure 5-80 Requirements definition in a business context

The Rational Requirements Composer UI branding storyboard linked to the managed requirement in Rational RequisitePro. The detailed requirement information is available for reference to the solution and component team across the life cycle. This information can be used by Marco's component team in Rational Team Concert by using the ClearQuest Connector. Additionally, the requirements information can be used by Tammy in Rational Quality Manager to aid in the generation of test plans and test cases by using the Rational Quality Manager and Rational Team Concert integration.

Usage model: Rational RequisitePro and Rational Requirements Composer

In this sample setup, the solution team resides in the United States (U.S.), where the component team is in Europe. The Rational RequisitePro Web components and database are hosted in the U.S. Bob, the product owner, conducts requirements management activities by using the Rational RequisitePro Web interface. He performs requirements definition and elicitation techniques with the Rational Requirements Composer client that is installed locally on his computer. Bob points his Rational Requirements Composer client to the Rational Requirements Composer server that is installed on a separate server. Bob uses both tools in day-to-day activities for requirements definition and management.

In this scenario, both the Rational Requirements Composer client and the Rational Requirements Composer server reside on the same server. The integration between the two tools remains the same if Rational Requirements Composer client points to the Rational Requirements Composer server, provided that the Account Opening RequisitePro project is properly shared and authenticated.

Rational ClearQuest and the ALM schema

Rational ClearQuest and the ALM schema also play an integral role in the success of the scenario, as illustrated in Figure 5-81. All parts of the organization interface with Rational ClearQuest at one point or another throughout the life cycle of this application. Rational ClearQuest acts as a central hub between requirements gathering in Rational RequisitePro and Rational RequistePro Composer, Rational Team Concert, and the newly acquired agile team in Europe. It also acts as the central hub for assets for the release and test teams that are using Rational Build Forge and Rational Quality Manager.

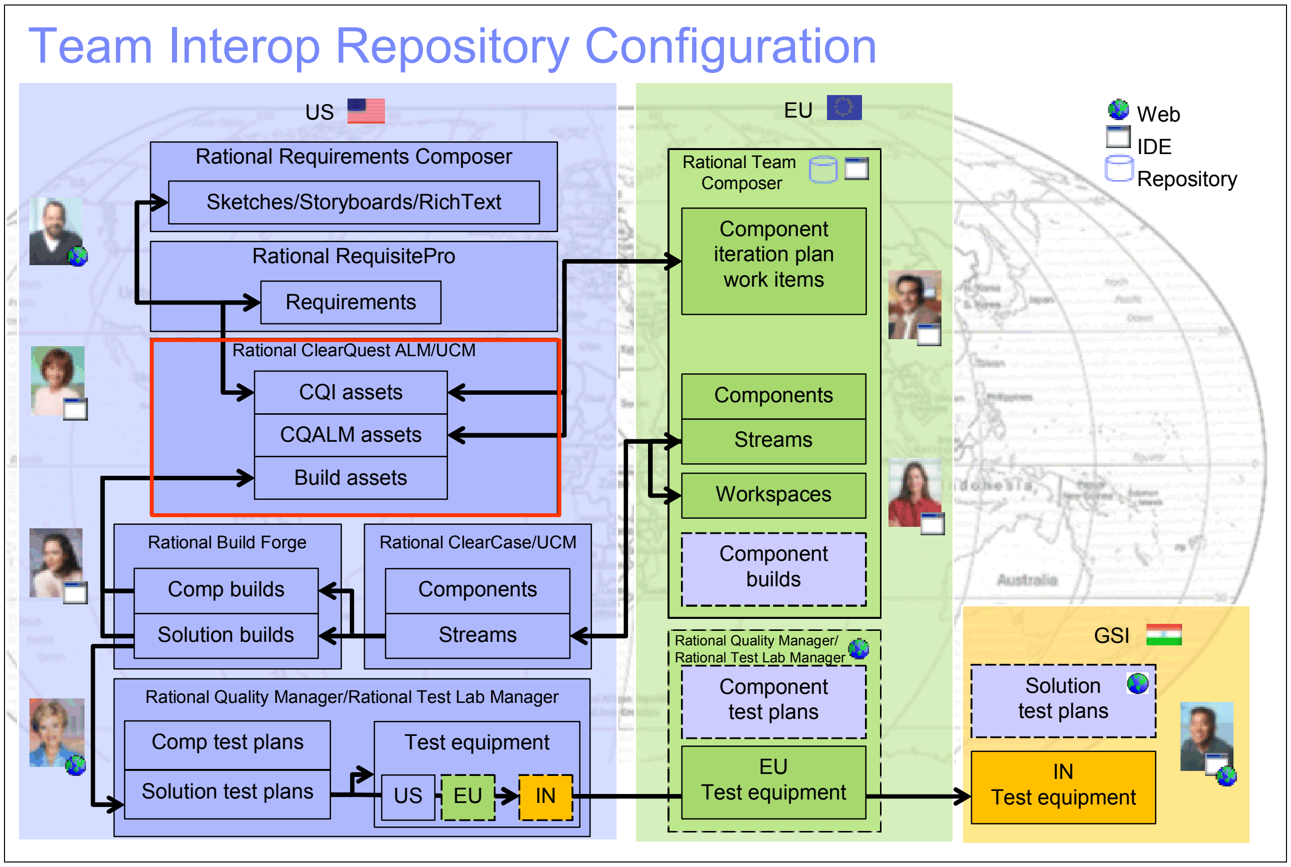


Figure 5-81 Rational ClearQuest acting as a central hub for team interoperability

In this sample setup, the Rational ClearQuest Web components are hosted on a server in the U.S. The back-end database server is also hosted on a machine in the U.S. Both servers must be on their own dedicated machines for optimal performance. Bob, the product owner, connects to Rational ClearQuest by using the Web interface, while Patricia, the project manager, uses the Eclipse interface that is installed locally on her computer. Both clients can provide parity in a majority of tasks that either user will use in their day-to-day activities.

In the reference scenario, Rational ClearQuest MultiSite was not used, but the Rational ClearQuest database can be in a replicated environment based on the team structure.

Rational RequisitePro integration to Rational ClearQuest remains the same as in a multisite environment. This is true provided that the Rational RequisitePro project is properly shared by using Universal Naming Convention (UNC) paths over the network and remote Rational ClearQuest clients can connect to this network share and communicate with the Rational RequisitePro databases. Rational ClearQuest record mastership remains at the site where the Rational RequisitePro project is registered to the Rational ClearQuest database.

Requirement information is synchronized later from the local replica to the remote replica by using normal Rational ClearQuest MultiSite synchronization.

The Rational ClearCase/UCM integration also operates in a multisite environment. For more information regarding the integration, see the Redbooks publication *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM*, SG24-6399. This Redbooks publication also explains how to do parallel development in geographically distributed locations by using Rational ClearCase and Rational ClearQuest MultiSite and includes detailed procedures for planning and implementing MultiSite for a UCM environment.

5.8.2 Configuring Rational RequisitePro and Rational Requirements Composer

In this section, we explain how Rational RequisitePro and Rational Requirements Composer were configured for the scenario in this book.

Rational RequisitePro

Rational RequisitePro projects are created from Rational RequisitePro project templates. A project template is a “boilerplate” that identifies the logical organizational structure of the project, the metadata. That is, it identifies the kinds of requirements that will be captured in the project, views, security, and so on.

The type of project template that is chosen depends on the requirements methodology that is followed by the team. In our example, we use the Agility at Scale for our requirements methodology. Just enough requirement information is captured to provide the team a foundation by which to build or implement the solution. Based on our flow, we capture the following types of requirements in Rational RequisitePro:

- Stakeholder request

  These requests reflect the initial need statement and business problems that are defined by the stakeholder.

- Feature requirement

  This type of requirement is an externally observable service that is provided by the system that directly fulfills a stakeholder need.

  Agile approach: If you are following more of an agile approach, the need statement requires a bit of refinement. The next level of detail is captured as a user story.

- Use case

  A use case captures requirements as a sequence of actions that a system performs that yields an observable result of value to those interacting with the system.

  Agile approach: If you are following more of an agile approach, use cases are used sparingly and only to provide additional detail to a user story where insufficient. Some of the scenario-type steps might be flushed out during test driven development with JUnit testing.

- Supplementary requirements (supporting requirements)

  These requirements define necessary system quality attributes, such as performance, usability, and reliability, and global functional requirements that are not captured in behavioral requirements artifacts such as use cases.

Agile approach: If you are following more of an agile approach, supporting requirements are used sparingly and only to identify nonfunctional requirements that cannot be described in the context of a user story.

Rational Requirements Composer

Like Rational RequisitePro, Rational Requirements Composer also leverages the concept of "project." Projects currently can be created from the Project Starter template or by using a blank template. The Project Starter template includes the following information (Figure 5-82):

Features	Package that provides the requirement types of a feature.
Glossaries	Package that identifies meaningful terms.
Processes	Package where process models can be stored.
Requirements	Generalized requirements package. (All requirements regardless of type are identified are organized, here, unless specified otherwise.)
Storyboards	Sketches can be stored in this package.
Supplementary	Nonfunctional requirements are stored in this package.
Use Cases	Use cases model information.

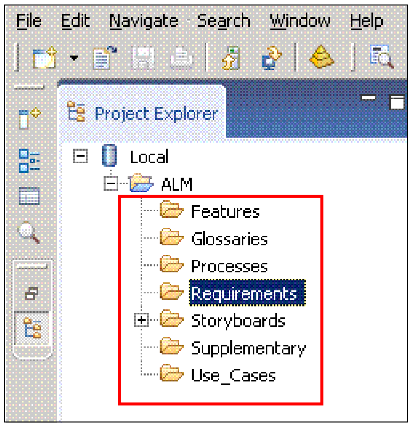
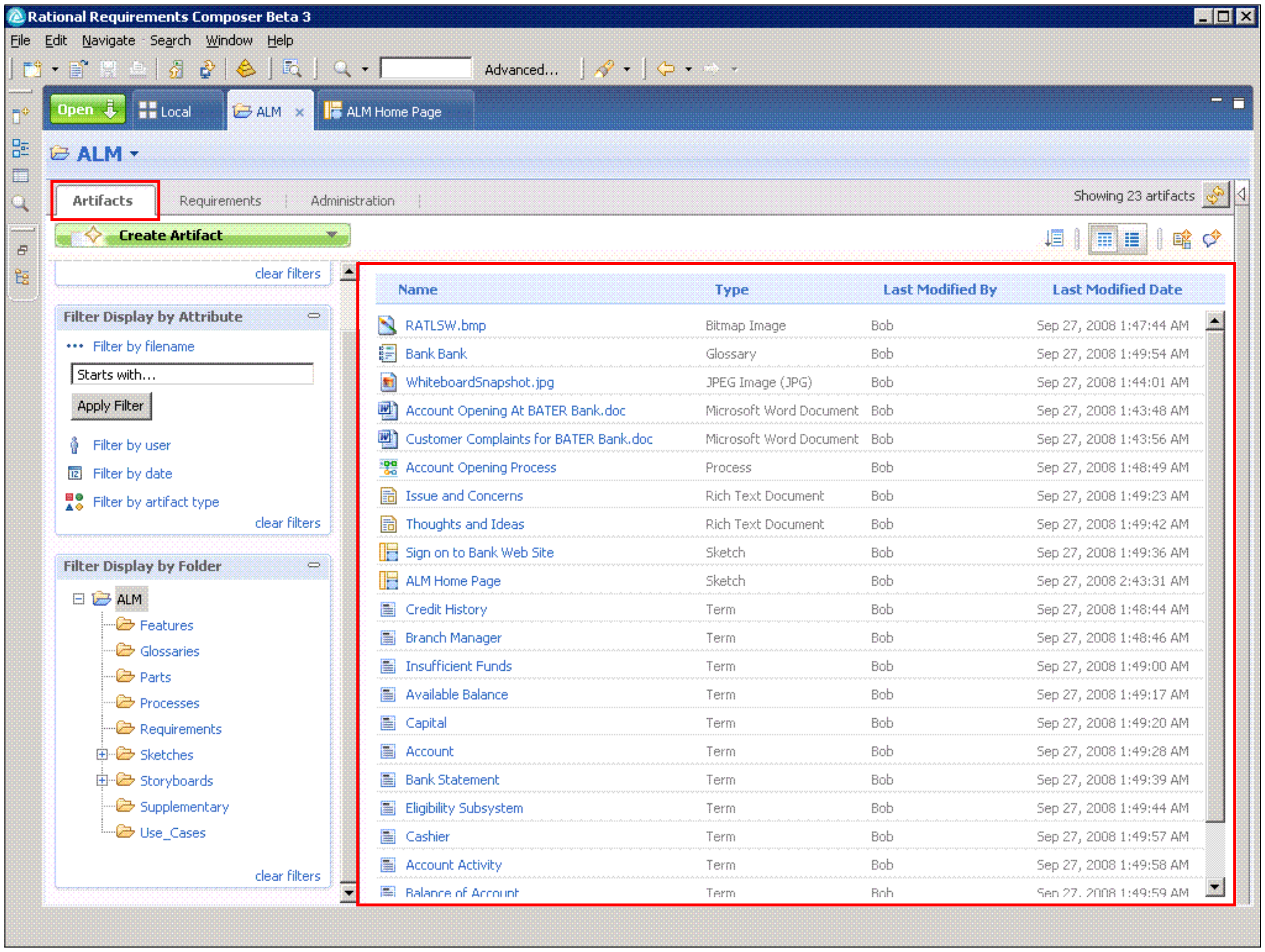


Figure 5-82 Classic project properties of Rational Requirements Composer

Integration points between Rational RequisitePro and Rational Requirements Composer

In this section, we discuss the integration points between Rational RequisitePro and Rational Requirements Composer. To manage requirements in Rational RequisitePro from the artifacts that are created in Rational Requirements Composer, the Rational Requirements Composer project must be integrated with a Rational RequisitePro project:

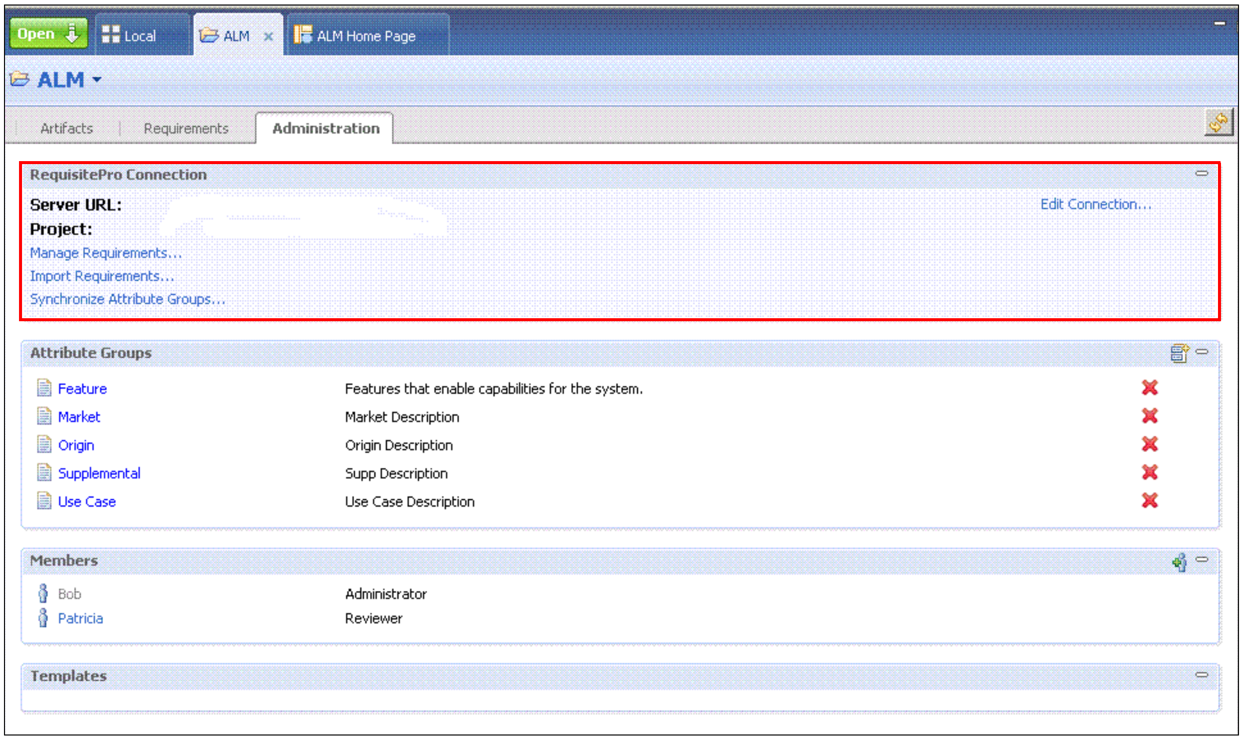
1. Open the ALM project in the Repository Explorer (Figure 5-83).



Name	Type	Last Modified By	Last Modified Date
RATLSW.bmp	Bitmap Image	Bob	Sep 27, 2008 1:47:44 AM
Bank Bank	Glossary	Bob	Sep 27, 2008 1:49:54 AM
WhiteboardSnapshot.jpg	JPEG Image (JPG)	Bob	Sep 27, 2008 1:44:01 AM
Account Opening At BATER Bank.doc	Microsoft Word Document	Bob	Sep 27, 2008 1:43:48 AM
Customer Complaints for BATER Bank.doc	Microsoft Word Document	Bob	Sep 27, 2008 1:43:56 AM
Account Opening Process	Process	Bob	Sep 27, 2008 1:48:49 AM
Issue and Concerns	Rich Text Document	Bob	Sep 27, 2008 1:49:23 AM
Thoughts and Ideas	Rich Text Document	Bob	Sep 27, 2008 1:49:42 AM
Sign on to Bank Web Site	Sketch	Bob	Sep 27, 2008 1:49:36 AM
ALM Home Page	Sketch	Bob	Sep 27, 2008 2:43:31 AM
Credit History	Term	Bob	Sep 27, 2008 1:48:44 AM
Branch Manager	Term	Bob	Sep 27, 2008 1:48:46 AM
Insufficient Funds	Term	Bob	Sep 27, 2008 1:49:00 AM
Available Balance	Term	Bob	Sep 27, 2008 1:49:17 AM
Capital	Term	Bob	Sep 27, 2008 1:49:20 AM
Account	Term	Bob	Sep 27, 2008 1:49:28 AM
Bank Statement	Term	Bob	Sep 27, 2008 1:49:39 AM
Eligibility Subsystem	Term	Bob	Sep 27, 2008 1:49:44 AM
Cashier	Term	Bob	Sep 27, 2008 1:49:57 AM
Account Activity	Term	Bob	Sep 27, 2008 1:49:58 AM
Balance of Account	Term	Bob	Sep 27, 2008 1:49:59 AM

Figure 5-83 Accounting Opening project properties

2. Click the **Administration** tab and click **Edit Connection** to enter the Rational RequisitPro connection information (Figure 5-84).



RequisitePro Connection

Server URL:

Project:

Edit Connection...

Manage Requirements...

Import Requirements...

Synchronize Attribute Groups...

Attribute Groups

Feature	Features that enable capabilities for the system.
Market	Market Description
Origin	Origin Description
Supplemental	Supp Description
Use Case	Use Case Description

Members

Bob	Administrator
Patricia	Reviewer

Templates

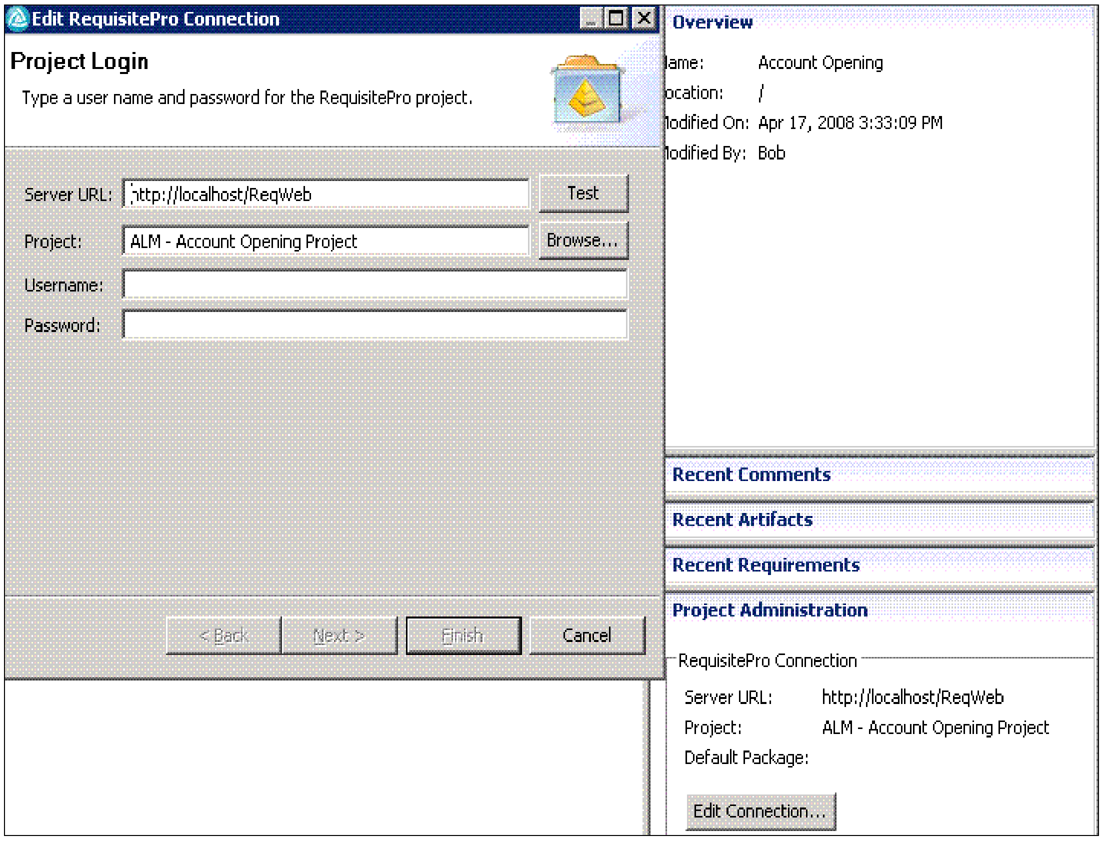
Figure 5-84 Selecting the project administration properties

3. In the Edit RequisitePro Connection window (Figure 5-85):

   a. In the Server URL field, type the URL for RequisiteWeb in the following format:

      `http://<server_name>/ReqWeb`

      The URL is case sensitive. Therefore, you must enter it accordingly.



Server URL: http://localhost/ReqWeb
Project: ALM - Account Opening Project
Username:
Password:

lame: Account Opening
ocation: /
lodified On: Apr 17, 2008 3:33:09 PM
lodified By: Bob

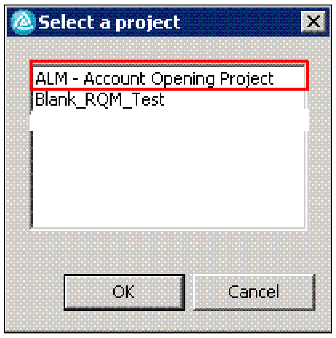
Server URL: http://localhost/ReqWeb
Project: ALM - Account Opening Project
Default Package:

Figure 5-85 Editing the Rational RequsiteWeb connection

   b. For Project, click **Browse** to navigate to the Rational RequisitePro project.

   c. Navigate to the project directory for the project and select the project name to access the project.

   d. After you selecting and it is displayed in the project list box (Figure 5-86), click **OK**.



ALM - Account Opening Project
Blank_RQM_Test

Figure 5-86 Selecting a project

The project name is now populated in the Project field.

e. Provide a user name and a password for the project. Then click **Next**.

f. In the window that lists the packages in the Rational RequisitePro project, select a package that you want to use as the default selection when managing or importing requirements.

g. Click **Finish** to save the connection information.

After you complete these steps, the Rational Requirements Composer and Rational RequisitePro project are integrated and requirements can now be managed between the two applications.

5.8.3 Configuring Rational ClearQuest and the ALM schema

The ClearQuest ALM solution that is exemplified in this book is configured by using the following components:

- The ClearQuest ALM packages
- The ClearQuest ALM OpenUP configuration
- The ClearQuest Connector integration package
- The Rational RequisitePro integration package
- Additional utility packages that are included in Rational ClearQuest

In this section, we discuss how Rational ClearQuest and the ClearQuest ALM solution were configured for the book example. For details about the configuration of interoperability in the Rational ALM solution, see Appendix B, “Configuring interoperability” on page 565.

Beginning with release 7.1.0.0, ClearQuest provides an out-of-the-box schema named “ALM” along with two new packages to help implementing Application Lifecycle Management in projects. The solution can be used out of the box as a new schema, or by adding the required packages to an existing schema.

The ClearQuest schema that used in this book was created by using the ready-to-use ALM schema and the OpenUP configuration. This schema already has applied the main packages, which are necessary for the ALM scenario. If work is being done with an existing ready-to-use schema or one that has been developed from scratch, the following two packages must be applied to the schema:

- ALMWork 1.0
- ALMProject 1.0

The steps to configure the Rational ClearQuest by using the ALM schema is provided in “Configuring the ClearQuest ALM schema” on page 566.

Packages for download: The ALM Packages for Rational ClearQuest are available for Rational ClearQuest 7.0.1 users to download for free from IBM. The packages can be accessed from the Web at the following address:

http://www.ibm.com/services/forms/preLogin.do?lang=en_US&source=swg-ratcq

Registration is required to access the packages. The download includes the ALM Packages for Rational ClearQuest, instructions for applying the packages, a sample database, and three tutorials for use with the sample database. The ClearQuest ALM solution can be used with both ClearQuest versions 7.0.1 and 7.1.

Configuring ClearQuest ALM for OpenUp

The ALM schema provides an OpenUP configuration by using the ClearQuest ALM schema and the system-wide settings. The work configuration and label records in the ClearQuest ALM solution enable process, workflow, and terminology configurations without impacting the underlying schema. Tool administrators or project leads can set policies for standardization, organization, and governance of projects.

In this book, we use the sample work configurations for OpenUP that are provided with ALM Packages for ClearQuest v7.1.

Extending OpenUP for sizing, approvals, and retrospectives

The ClearQuest ALM solution provides a base OpenUP configuration. Extensions to this process were made to support the following extended workflows:

- Request sizing; see 4.3.2, "Sizing requests" on page 113
- Reviews and approvals; see "Patricia approves the release" on page 508
- Retrospectives; see "Retrospectives" on page 518

For details about the ALM schema additions, see the following sections:

- "Configuring Rational ClearQuest for solution delivery" on page 521
- "Adding optional resolution codes" on page 576

Configuring Rational ClearQuest ALM schema for interoperability

In addition to the required ALM packages, additional packages to support interoperability between Rational ClearQuest and other tools, such as Rational RequisitePro, Rational ClearQuest, and Rational Team Concert, were added. Packages that are required by UCM are automatically added when adding the ALMWork 1.0 package and are applied to the ALMActivity record type.

The following packages were added to the ALM schema:

- RequisitePro 1.9 enables integration with Rational RequisitePro.
- JazzInterop 1.0 enables integration with Rational Team Concert.
- Notes 5.1 enables interoperability with discussions in Rational Team Concert.

Steps to configure the ClearQuest ALM schema for interoperability with the Jazz products are provided in "Configuring ClearQuest ALM schema for interoperability" on page 569.

The OpenUP configuration and work configuration require additions to support the parity between the fields in Rational ClearQuest and the Jazz products OpenUP template. Additional extensions are optional but make the integration more flexible:

- Required additions to the OpenUP work configuration in Rational ClearQuest
- Optional additions to the ClearQuest ALM schema

For details about the ALM schema additions, see the following sections:

- "Configuring ClearQuest ALM system-wide settings for interoperability" on page 571
- "Configuring the ClearQuest ALM schema" on page 566

Configuring the JazzInterop package

The JazzInterop package enables record synchronization between work items in Rational Team Concert or Rational Quality Manager and Rational ClearQuest records. In the example in this book, the JazzInterop package was associated with the ALMActivity, ALMTask, ALMProject, ALMCategory, ALMResolutionCodeLabel, and ALMTypeLabel record types.

The JazzInterop package is added to the ALM schema and associated with the ALM types by using the Rational ClearQuest Designer tool.

The details about the configuration, see “Adding packages for the ALM schema” on page 567.

Configuring Jazz interoperability

Rational ClearQuest integration with the Rational Jazz server is provided by the ClearQuest Connector that is provided as a part of the Rational Team Concert and Rational Quality Manager products. ALM interoperability is configured by using the following components:

- ClearQuest Connector gateway that is deployed to the Rational ClearQuest database server or a co-located server
- Synchronization rules that are deployed to the Jazz server

You can find steps to configure the Jazz products for ALM interoperability in “Configuring Jazz repositories for interoperability” on page 583.

Configuring the ClearQuest Connector Gateway

The ClearQuest Connector Gateway performs the synchronization between Rational ClearQuest and the Jazz server. The gateway runs on a stand-alone application server that is provided with the Rational Team Concert and Rational Quality Manager products.

The ClearQuest Connector Gateway is deployed and co-located with the Rational ClearQuest database. The gateway is configured with a property file that specifies the connection information, such as the server addresses and user account information, to Rational ClearQuest and Jazz servers.

For details about the configuration, see “Configuring the ClearQuest Gateway” on page 581.

Configuring ALM synchronization rules

The details of the synchronization between the Rational ClearCase and Jazz repositories are specified in the synchronization rules. These rules specify the how records are paired, the field of the items to synchronize, and details in the mapping of values. The rules also specify the type of extended transformation that is provided by synchronization managers in the ClearQuest Gateway.

The synchronization rules are general in concept, but specific to the ClearQuest schema that is used. The following set of synchronization rules was elaborated and deployed to support the example in this book:

- com.ibm.rational.clearquest.CQALM.ALMTask

  This rule synchronizes ALMTasks and Work Items.
- com.ibm.rational.clearquest.CQALM.ALMActivity

  This rule synchronizes ALMActivities and Work Items.
- com.ibm.rational.clearquest.CQALM.ALMCategory

  This rule creates new categories from ALMCategory records.
- com.ibm.rational.clearquest.CQALM.ALMProject

  This rule resolves references to ALMCategory from ALMActivity and ALMTask records.
- com.ibm.rational.clearquest.CQALM.ALMTypeLabel

  This is a helper rule to synchronize the Priority reference field.

- com.ibm.rational.clearquest.CQALM.ALMResolutionCodeLabel

  This is a helper rule to synchronize the ResolutionCode reference field.
- com.ibm.rational.clearquest.CQALM.Attachments

  This rule manages the creation and synchronization of attachments in the Rational ClearQuest and Rational Team Concert products.
- com.ibm.rational.clearquest.CQALM.users

  This rule manages the creation of new contributors in Rational Team Concert from user records in Rational ClearQuest.

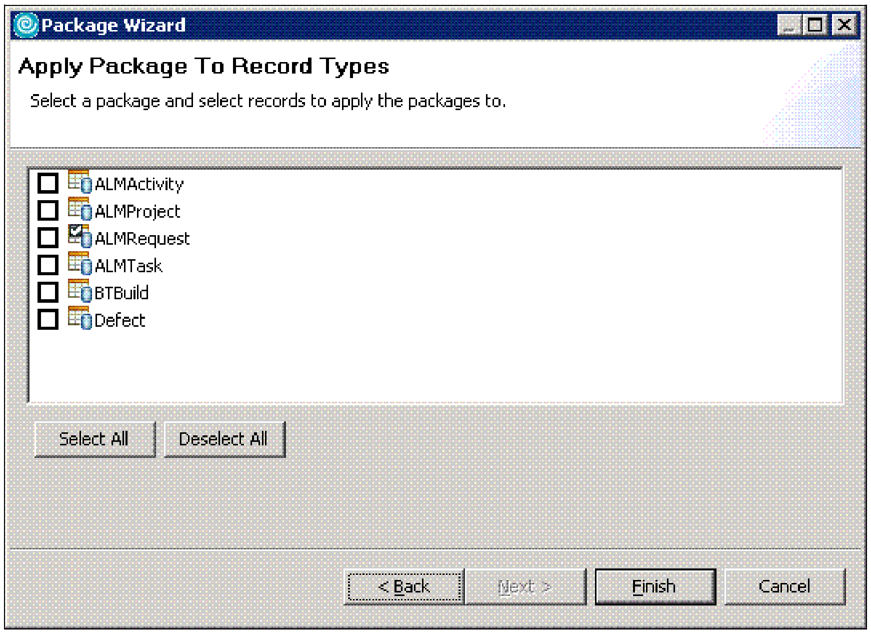
For details about the ALM schema additions, see "Configuring and deploying synchronization rules" on page 585.

Configuring interoperability in Rational RequisitePro

By using the Rational ClearQuest v7.1 Schema Designer, as shown in Figure 5-87, the Rational RequisitePro package was applied the to the following record types in the schema:

- ALMRequest (required)
- ALMActivity (optional)
- ALMTask (optional)
- Any other record type needed for your business logic (optional)

Attention: Apply the Rational RequisitePro package to all types that will support traceability to or from a requirement in Rational RequisitePro. The configuration steps in this book assume that traceability is established to the ALMRequest type.



Package Wizard

Apply Package To Record Types

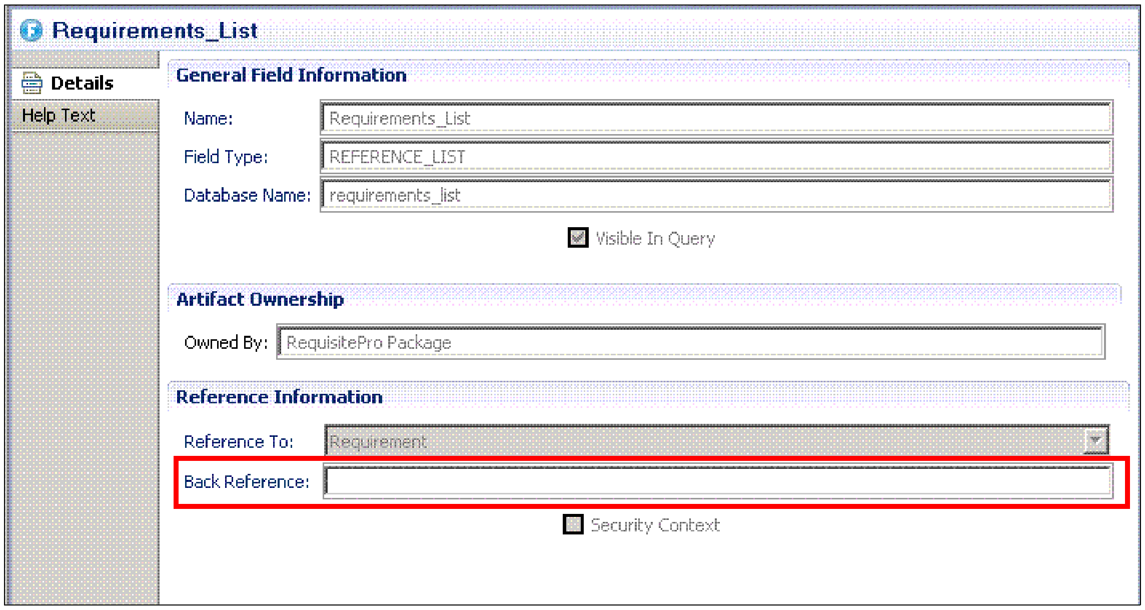
Select a package and select records to apply the packages to.

- [ ] ALMActivity
- [ ] ALMProject
- [ ] ALMRequest
- [ ] ALMTask
- [ ] BTBuild
- [ ] Defect

Figure 5-87 Applying the Rational RequisitePro package to record types

After all the packages are applied, check the schema to make the necessary changes to complete the Rational RequisitePro and Rational ClearQuest integration:

1. Open each record type that has the Rational RequisitePro package applied and browse to the field listing section in the designer.
2. Find the field called **Requirements_List**, which the package added, and for Back Reference (Figure 5-88), add a reference to the field that points back to the requirement record type. In this case, the Back Reference is called `ALMRequests_List`.
3. Repeat this process for each record type that the RequisitePro package added.


Requirements_List

Details
Help Text

General Field Information

Name: Requirements_List
Field Type: REFERENCE_LIST
Database Name: requirements_list
Visible In Query [✓]

Artifact Ownership

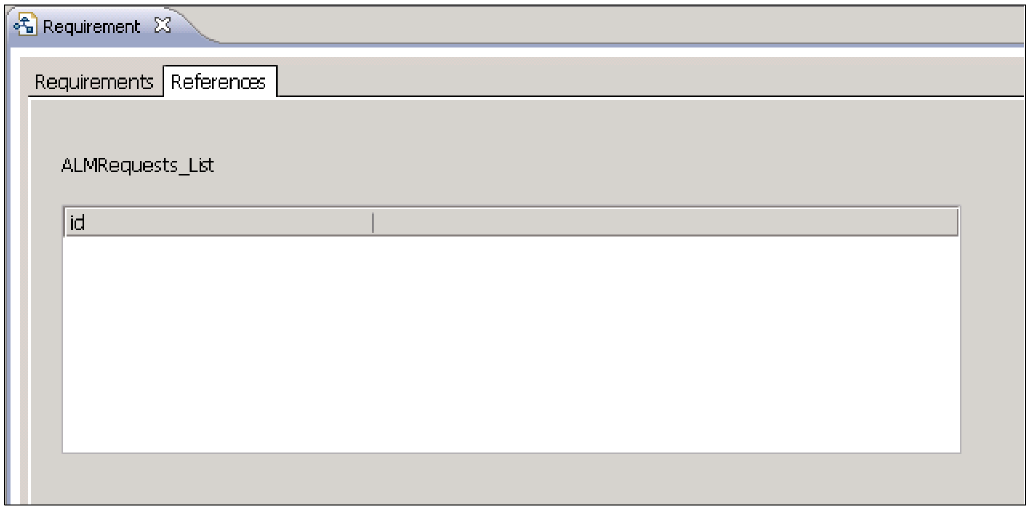
Owned By: RequisitePro Package

Reference Information

Reference To: Requirement
Back Reference:
Security Context [ ]

Figure 5-88 Requirement_List back reference field

After all back references are added, you can optionally choose to add the newly added back reference fields to the requirements record form as shown in Figure 5-89.


Requirement

Requirements | References

ALMRequests_List

id	

Figure 5-89 References tab on the Requirement record type

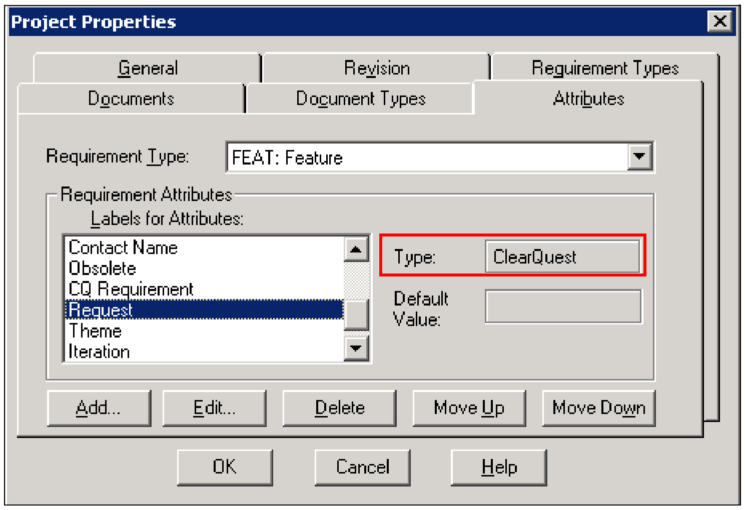
Integration of Rational ClearQuest and Rational RequisitePro

To manage requirements and link them back to their associated Rational ClearQuest assets, you must configure the integration between Rational ClearQuest and Rational RequisitePro. This configuration entails the following setup:

- Pertinent packages must be present in the Rational ClearQuest schema for the Rational RequisitePro and Rational ClearQuest integration.

  Tip: Review the Rational ClearQuest installation guide, which references pertinent packages that should be present for the integration of Rational ClearQuest and Rational RequisitePro.

- The Rational RequisitePro project has attributes that have been created of type "ClearQuest" for the stateful records that will be linked to Rational RequisitePro requirements (Figure 5-90).



Requirement Type: FEAT: Feature

Requirement Attributes

Labels for Attributes:

Contact Name
Obsolete
CQ Requirement
Request
Theme
Iteration

Type: ClearQuest

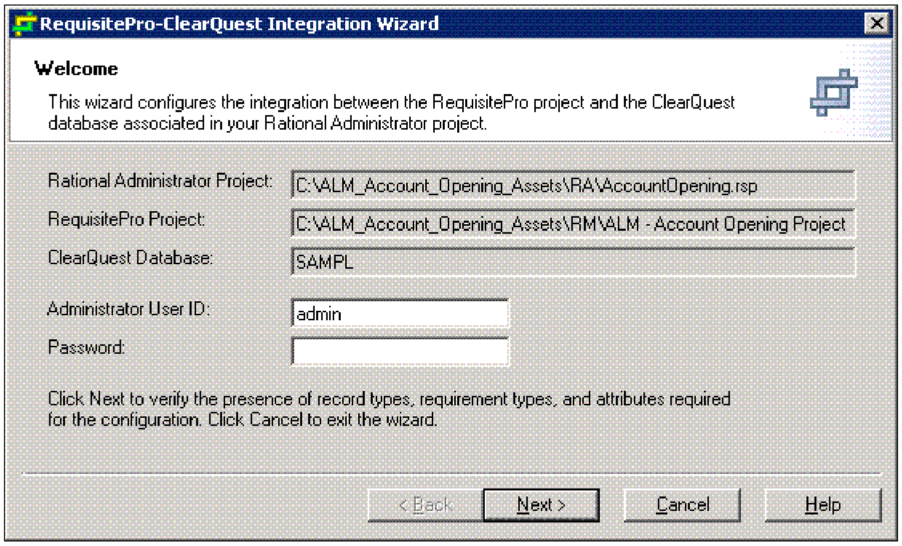
Default Value:

Figure 5-90 Project Properties - ClearQuest integration attributes

- The Rational Administration Wizard must be executed as explained in the remainder of this section.

An administration machine with Rational Administrator, Rational RequisitePro, and Rational ClearQuest must be installed to complete the following required steps:

1. Open Rational Administrator and select the desired project. Right-click the project and select **RequisitePro-ClearQuest Integration**.
2. In the RequisitePro-ClearQuest Integration Wizard window (Figure 5-91), type the Administrator user ID and Password and click **Next**.


RequisitePro-ClearQuest Integration Wizard

Welcome

This wizard configures the integration between the RequisitePro project and the ClearQuest database associated in your Rational Administrator project.

Rational Administrator Project: C:\ALM_Account_Opening_Assets\RA\AccountOpening.rsp

RequisitePro Project: C:\ALM_Account_Opening_Assets\RM\ALM - Account Opening Project

ClearQuest Database: SAMPL

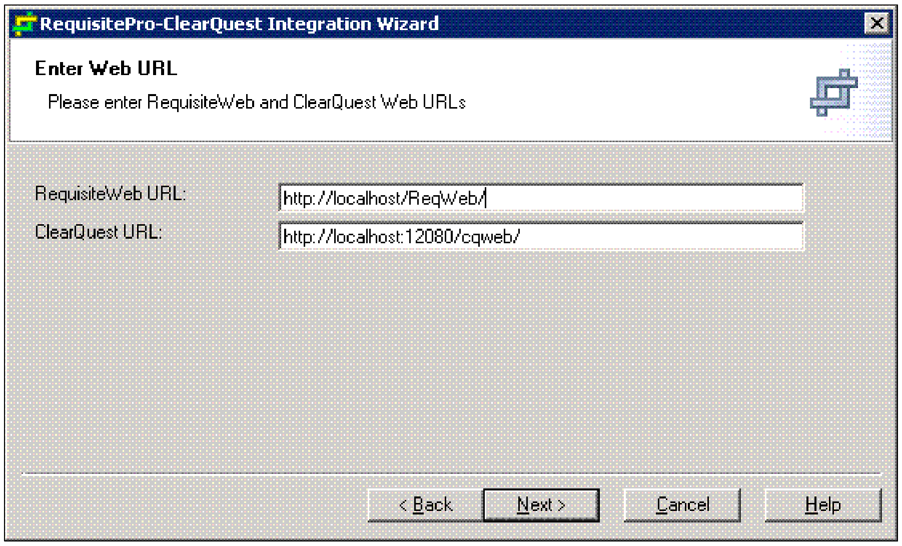
Administrator User ID: admin

Password:

Click Next to verify the presence of record types, requirement types, and attributes required for the configuration. Click Cancel to exit the wizard.

Figure 5-91 RequisitePro-ClearQuest Integration Wizard configuration - Welcome window

3. In the Enter Web URL window (Figure 5-92), type the RequisitePro Web URL and ClearQuest Web URL and click **Next**.


RequisitePro-ClearQuest Integration Wizard

Enter Web URL

Please enter RequisiteWeb and ClearQuest Web URLs

RequisiteWeb URL: http://localhost/ReqWeb/

ClearQuest URL: http://localhost:12080/cqweb/

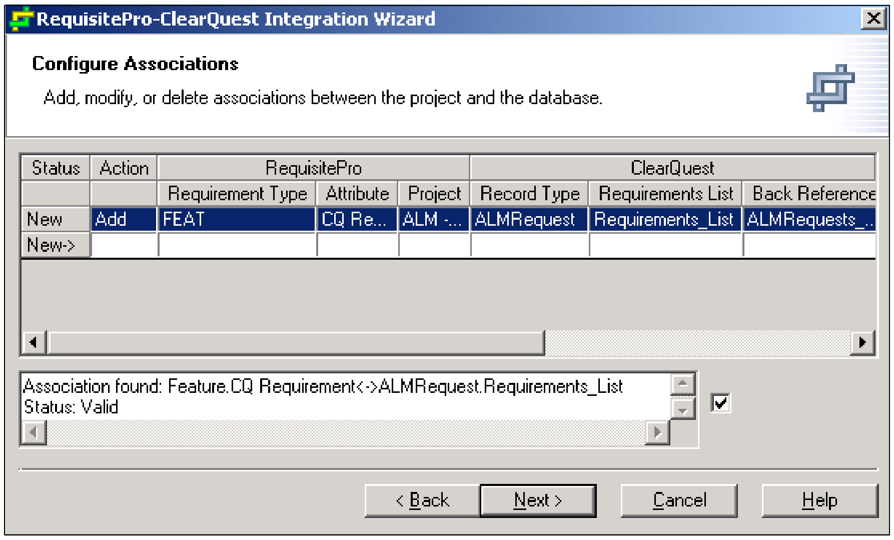
Figure 5-92 RequisitePro-ClearQuest Integration Wizard configuration - Enter Web URL window

4. In the next window, configure the associations by selecting **Add** for the Action column and then completing the following parameters as shown in Figure 5-93 and Figure 5-94:

   – For Requirement Type, type FEAT.
   – For Attribute, type Request.
   – For Project, specify the project name.
   – For RecordType, type ALMRequest.
   – For Requirements List, type the name of the Reference field to Requirements record type.
   – For Back Reference, type the name of the Back Reference field from Requirements to ALMRequest.

   Click **Next** to complete the configuration.

   Figure 5-93 shows the initial parameters.



RequisitePro-ClearQuest Integration Wizard

Configure Associations

Add, modify, or delete associations between the project and the database.

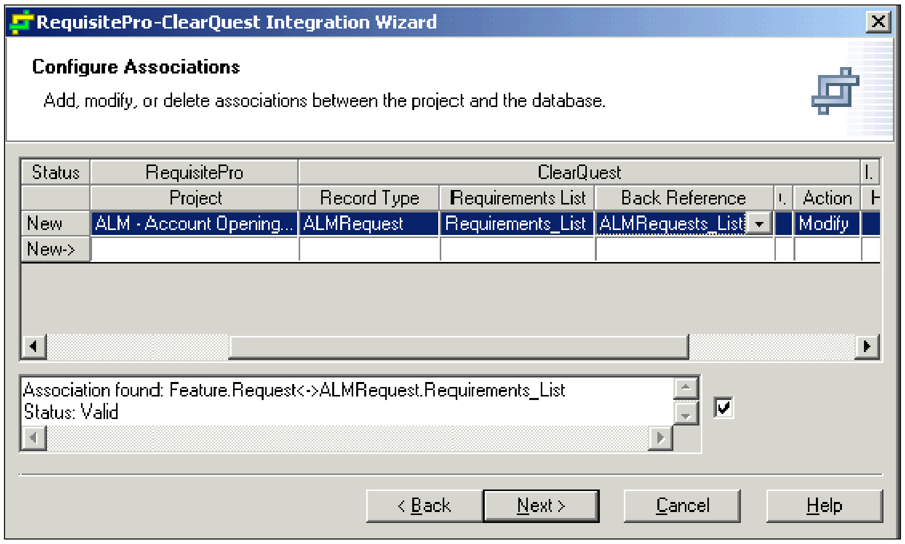
Status	Action	RequisitePro			ClearQuest		
		Requirement Type	Attribute	Project	Record Type	Requirements List	Back Reference
New	Add	FEAT	CQ Re...	ALM -...	ALMRequest	Requirements_List	ALMRequests_..
New->							

Association found: Feature.CQ Requirement<->ALMRequest.Requirements_List
Status: Valid

< Back | Next > | Cancel | Help

Figure 5-93   RequisitePro-ClearQuest integration wizard configuration

Figure 5-94 shows the parameters that are displayed after scrolling to the right.



RequisitePro-ClearQuest Integration Wizard

Configure Associations

Add, modify, or delete associations between the project and the database.

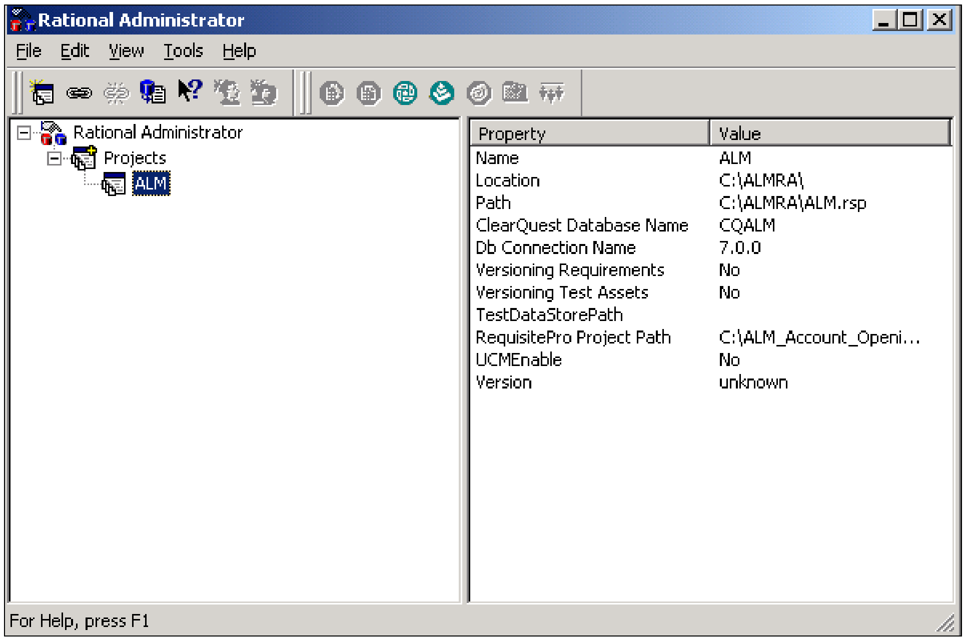
Status	RequisitePro	ClearQuest					I.
	Project	Record Type	Requirements List	Back Reference	I.	Action	H
New	ALM - Account Opening...	ALMRequest	Requirements_List	ALMRequests_List		Modify	
New->							

Association found: Feature.Request<->ALMRequest.Requirements_List
Status: Valid

< Back | Next > | Cancel | Help

Figure 5-94   RequisitePro-ClearQuest Integration Wizard configuration

After the association is complete, the Rational Administrator window (Figure 5-95) opens and shows details of the Rational ClearQuest database and Rational RequisitePro project, which are integrated.



Property	Value
Name	ALM
Location	C:\ALMRA\
Path	C:\ALMRA\ALM.rsp
ClearQuest Database Name	CQALM
Db Connection Name	7.0.0
Versioning Requirements	No
Versioning Test Assets	No
TestDataStorePath	
RequisitePro Project Path	C:\ALM_Account_Openi...
UCMEnable	No
Version	unknown

Figure 5-95 Rational Administrator with configured RequisitePro-ClearQuest integration

Setting up a Rational Team Concert feed to ClearQuest queries

Rational Team Concert has a built-in feed reader so that users can subscribe to feeds from various types of sources. The Rational ClearQuest Web interface supports representing query output in variables formats, such as ATOM and HTML, by using its Rest API.

To configure Rational Team Concert to subscribe to a Rational ClearQuest feed and monitor Rational ClearQuest records:

1. Log in to Rational Team Concert and navigate to the **Team Artifacts** tab.
2. Right-click **Feeds** and select **New Subscription**.
3. Enter the URL in the following format to the Rational ClearQuest Web server that points to the ATOM feed for the query that needs a subscription:

```
http://localhost/cqweb/restapi/CQALM/SAMPL/QUERY/Public
Queries/JazzConnector/AccountOpening/ALMTask.AccountOpening
?format=ATOM&loginId=admin&password=
```

4. Expand the **Feeds** item and double-click the newly created feed to run it. The reader should display the feed as shown in Figure 5-96.

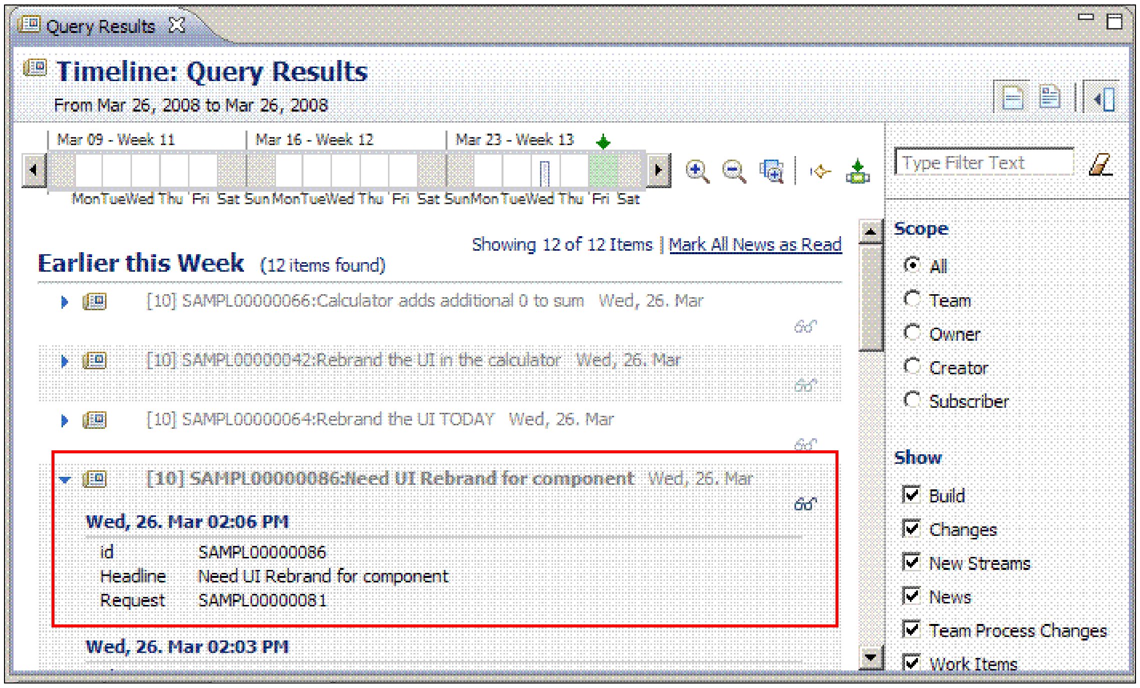

Query Results
Timeline: Query Results
From Mar 26, 2008 to Mar 26, 2008
Mar 09 - Week 11
Mar 16 - Week 12
Mar 23 - Week 13
Type Filter Text
Showing 12 of 12 Items | Mark All News as Read
Earlier this Week (12 items found)
[10] SAMPL00000066:Calculator adds additional 0 to sum Wed, 26. Mar
[10] SAMPL00000042:Rebrand the UI in the calculator Wed, 26. Mar
[10] SAMPL00000064:Rebrand the UI TODAY Wed, 26. Mar
[10] SAMPL00000086:Need UI Rebrand for component Wed, 26. Mar
Wed, 26. Mar 02:06 PM
id SAMPL00000086
Headline Need UI Rebrand for component
Request SAMPL00000081
Wed, 26. Mar 02:03 PM
Scope
All
Team
Owner
Creator
Subscriber
Show
Build
Changes
New Streams
News
Team Process Changes
Work Items

Figure 5-96 ClearQuest ATOM feed in Rational Team Concert

5.9 Problem determination and known workarounds

In this section, we discuss general program determination techniques and must-gather type information that can be used for troubleshooting and contacting support. We cover the following products:

- Rational ClearQuest
- Rational RequisitePro
- Rational RequisitePro Composer

5.9.1 General techniques

In general, using problem determination techniques can greatly reduce the time to solve a problem and help maintain a smoother running environment overall. Understanding the scope and the narrowing down of a problem is always a good first step when tackling any new problem that you might encounter. Formulating a problem statement to understand exactly what the problem is and where it lies in the environment can help do this.

After the problem is defined, you can start to narrow down the scope by asking questions regarding where the problem does not exist in the environment. In addition, questions on when and where the problem exists and does not exist help to further narrow the scope. This also helps to identify where gaps in information exist and where more information needs to be gathered.

Upon completion of these questions and gathering as much data as possible, you can work on creating a list of possible causes that might be the root of the problem. Ordering the list and tackling the most likely cause helps to reduce the time to solve the problem.

The use of self-help resources and support generated content is another way to reduce the time to solve a problem. Visit IBM Rational support's Web site at the following address to search for content on specific products and problems:

`http://www.ibm.com/software/rational/support`

5.9.2 Troubleshooting Rational ClearQuest

While Rational ClearQuest is a flexible tool that allows organizations high amounts of customizations, this customization can ultimately lead to issues with can arise during configuration and deployment. We discuss some of the most common areas when working with Rational ClearQuest and the ALM schema.

Rational ClearQuest packages

Installing and setting up Rational ClearQuest packages can cause conflicts if the schema to which they are applied already have fields with the same name. When applying a package to a record type, you might see the following error message:

```
The Name “Project” already exists
```

This message typically indicates that there is a field in the record type with the same name as one that the package needs to apply. To work around this problem, you must check out the schema and find the field name in question that is causing the conflict as indicated in the error message and rename it to something new. After the field name is changed, apply the package again. It should succeed, provided that there are no other duplicate field names in the schema.

Rational ClearQuest Web client

The Rational ClearQuest Web interface has been redesigned in the 7.1.0.0 release and has a new architecture that uses IBM WebSphere® Application Server and Change Management Server technologies. In general, when working with errors by using the Web interface, its always a best practice to see if the same error exists with the Eclipse interface. This exploration helps to narrow down the scope of the problem greatly. It also allows you to focus on troubleshooting the problem in the Eclipse environment since the architecture is generally less complex to work with.

Performance

The ALM schema is optimally designed to work with the new Web interface. However, should performance issues arise, see the IBM developerWorks article “IBM Rational general schema design performance” for more advice about best practices when working the Rational ClearQuest schema design. You can find the article at the following address:

`http://www.ibm.com/developerworks/rational/library/07/0717_patel/index.html`

Rational ClearQuest Eclipse client

When working with the Rational ClearQuest Eclipse interface, it is best to narrow down the scope of the problem by determining which component might have caused the failure. Errors can occur during one the following major use cases:

- Running queries
- Loading records to view
- Performing actions and state changes
- Running reports

Based on the type of error, there are different techniques that you can use to troubleshoot the issue further. For ALM, specifically parent child links and reference records are used frequently to maintain a hierarchy for project work.

Diagnostics core tracing

Diagnostic tracing is sometimes necessary to understand which component of Rational ClearQuest is failing and helps to identify the correct action that needs to be taken to resolve the problem. Setting up diagnostic tracing is achieved by using one of the following methods:

- ratl_diagnostics table
- Environment variables
- Microsoft Windows registry keys

ratl_diagnostics is a table that can be added to your Rational ClearQuest user database and causes trace information to be generated for any client machine that is connecting to this database. The table is read when the Rational ClearQuest client is launched and writes logging information to a trace file as specified in the table. This option is good when you are unsure about which client is causing the problem and cannot readily reproduce the problem for any single client machine.

Important: Be advised that this option can cause a lot of overhead because every client that accesses the database will generate trace information, which can quickly grow in size based on transaction load.

To create the table, use the PDSQL utility that is in the \<install dir>\ClearQuest directory of your administration machine and run the SQL commands in Example 5-1 to generate the required tables.

Example 5-1 SQL script to generate the ratl_diagnostic table

```
drop table ratl_diagnostics;
create table ratl_diagnostics (
    diag_name SQL_VARCHAR(16),
    diag_value SQL_VARCHAR(255));
```

After the table is created, use the SQL commands in Example 5-2 to populate them with the necessary values.

Example 5-2 SQL statements to insert diagnostic information for tracing

```
insert into ratl_diagnostics (diag_name,diag_value) values ('Trace',‘SQL')
insert into ratl_diagnostics (diag_name,diag_value) values ('Output','ODS')

To update an existing trace:
update ratl_diagnostics set diag_value = 'SQL=2' where diag_name = 'Trace‘

To turn all tracing off:
update ratl_diagnostics set diag_value = '' where diag_name = 'Output‘
```

To troubleshoot the most common types of problems, use the trace keys in Example 5-3. For a full listing of trace keys and their meanings, contact IBM Rational Technical Support.

Example 5-3 Trace keys for troubleshooting Rational ClearQuest

```
TRACE = SQL=2;THROW;DB_CONNECT=2;SESSION;EDIT;RESULTSET;API;VBASIC;PERL;PACKAGES
```

An environment variable can also be set up on individual client machines for problem determination. You can set variables for either Windows or UNIX to help generate diagnostic logs. On Windows, you set the variables as shown in Example 5-4, which generates the output to c:\trace.log.

Example 5-4 Variables on Windows

```
set CQ_DIAG_TRACE=Throw;Db_Connect=2;SQL=2;API
set CQ_DIAG_REPORT=MESSAGE_INFO=0x70B
set CQ_DIAG_OUTPUT=c:\trace.log
```

On UNIX, you can set the variables as shown in Example 5-5, which generates the output in file trace.log.

Example 5-5 Variables on UNIX

```
setenv  CQ_DIAG_TRACE Throw;Db_Connect=2;SQL=2;API
setenv  CQ_DIAG_REPORT MESSAGE_INFO=0x70B
setenv  CQ_DIAG_OUTPUT trace.log
```

Finally diagnostic tracing can be done by using the Windows registry of the client machine where the problem occurs. As shown in Example 5-6 you can add the following registry key to your client system to generate trace output. Be advised to back up your registry before you attempt to make any modifications.

Example 5-6 Rational ClearQuest registry keys to enable trace output

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Rational Software\ClearQuest\Diagnostic]
"Trace"="SQL=2;THROW;DB_CONNECT=2;SESSION;EDIT;RESULTSET;API;VBASIC;PERL;PACKAGES"
"Report"="MESSAGE_INFO=0X70B;DIAG_FLAGS=-1"
"Output"="C:\\temp\\cq_diagnotics.log"
```

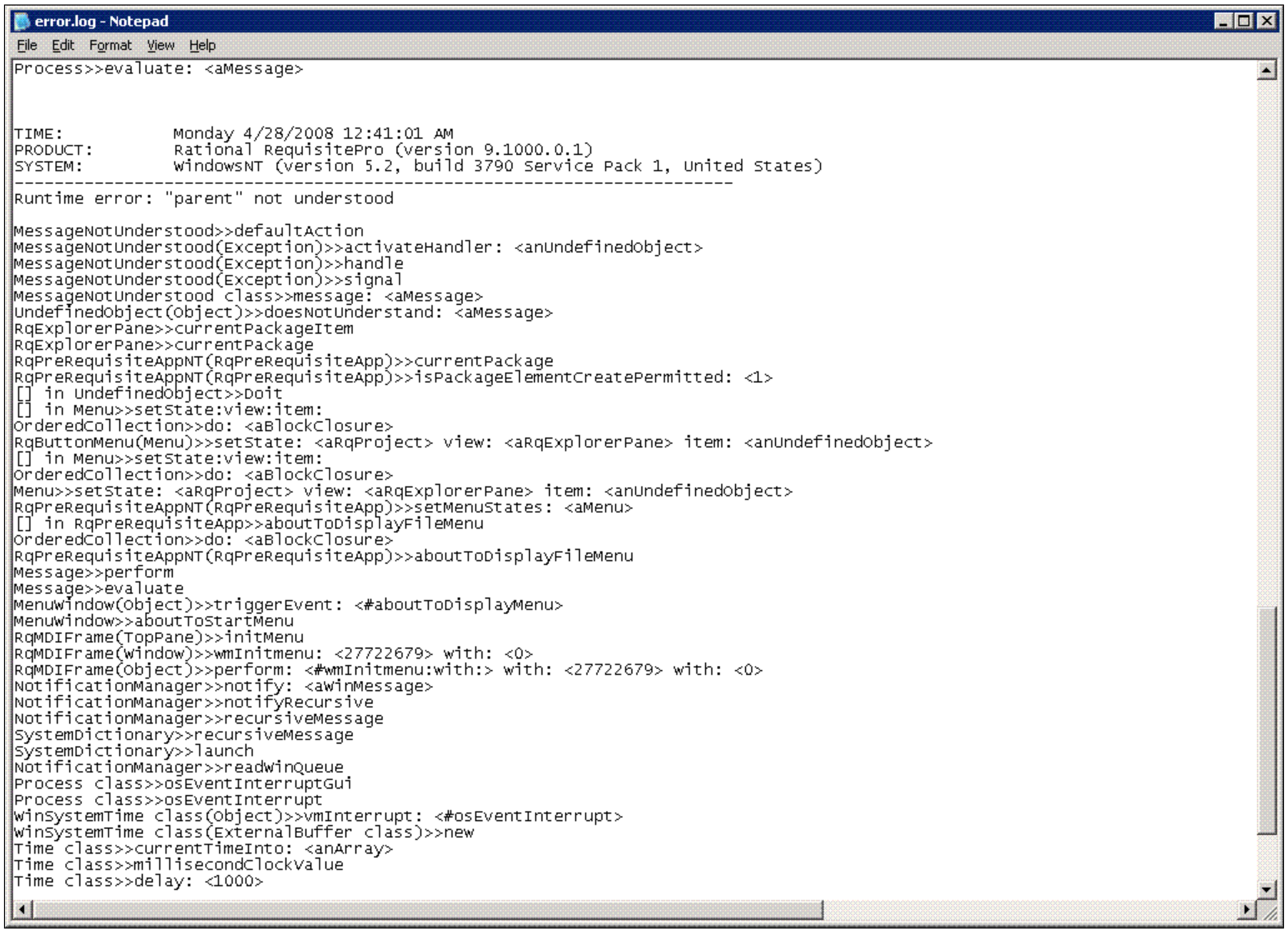
5.9.3 Troubleshooting Rational RequisitePro

Rational RequisitePro users can leverage the Web interface or thick client for which there are various troubleshooting techniques. We describe some high-level considerations for logging, and performance and integration considerations.

Rational RequisitePro client logging

Logging information for Rational RequisitePro is stored in the <install directory>\IBM Rational\RequisitePro\bin\error.log. This file can be viewed in a text editor and includes error messages that are thrown by using the thick client sorted-by date. The log file is helpful in identifying error messages that are received from the thick client.

Figure 5-97 shows a sample of the error.log file.

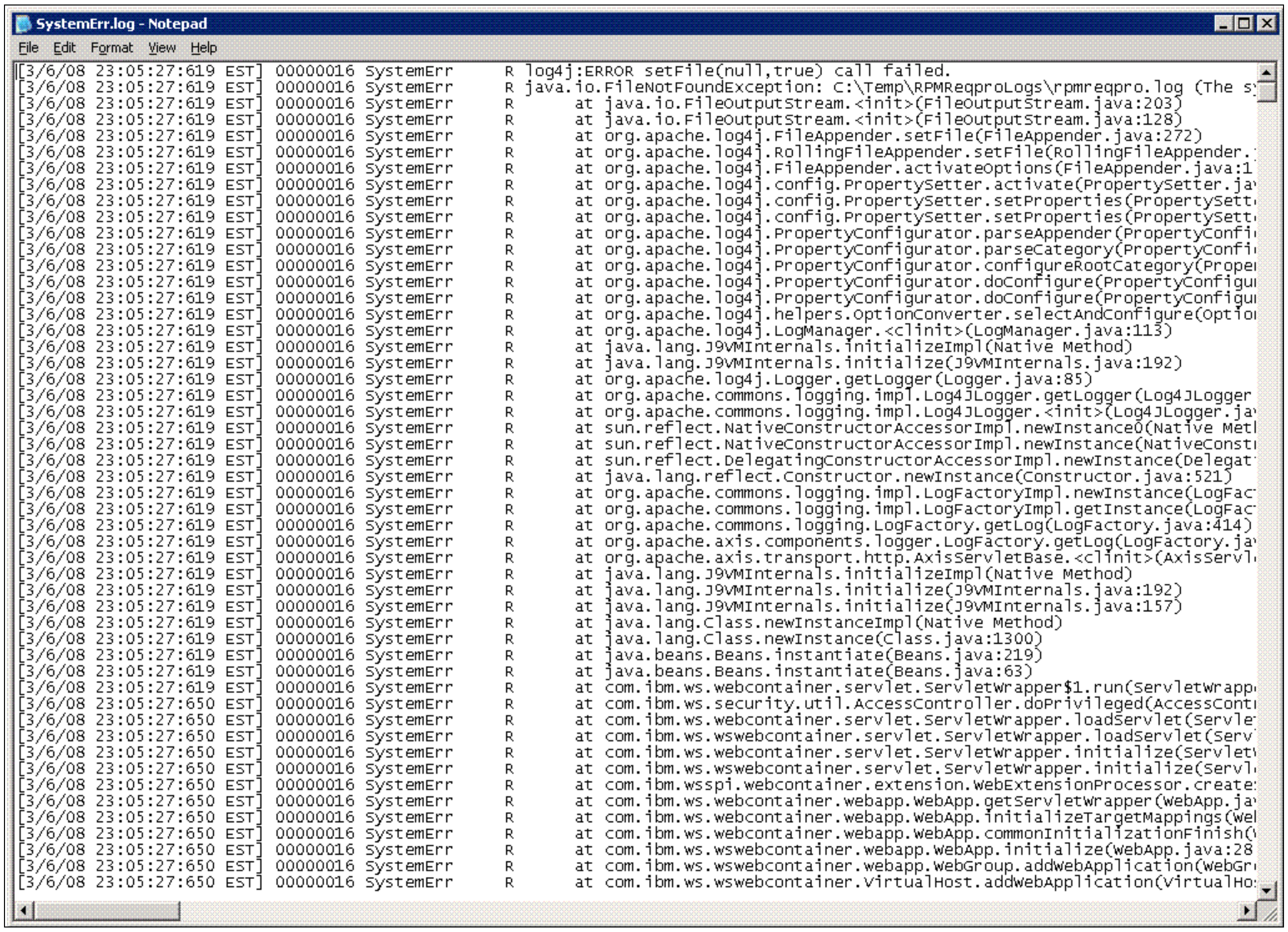


```
Process>>evaluate: <aMessage>



TIME:           Monday 4/28/2008 12:41:01 AM
PRODUCT:        Rational RequisitePro (version 9.1000.0.1)
SYSTEM:         WindowsNT (version 5.2, build 3790 Service Pack 1, United States)
------------------------------------------------------------------------
Runtime error: "parent" not understood

MessageNotUnderstood>>defaultAction
MessageNotUnderstood(Exception)>>activateHandler: <anUndefinedObject>
MessageNotUnderstood(Exception)>>handle
MessageNotUnderstood(Exception)>>signal
MessageNotUnderstood class>>message: <aMessage>
UndefinedObject(Object)>>doesNotUnderstand: <aMessage>
RqExplorerPane>>currentPackageItem
RqExplorerPane>>currentPackage
RqPreRequisiteAppNT(RqPreRequisiteApp)>>currentPackage
RqPreRequisiteAppNT(RqPreRequisiteApp)>>isPackageElementCreatePermitted: <1>
[] in UndefinedObject>>Doit
[] in Menu>>setState:view:item:
OrderedCollection>>do: <aBlockClosure>
RqButtonMenu(Menu)>>setState: <aRqProject> view: <aRqExplorerPane> item: <anUndefinedObject>
[] in Menu>>setState:view:item:
OrderedCollection>>do: <aBlockClosure>
Menu>>setState: <aRqProject> view: <aRqExplorerPane> item: <anUndefinedObject>
RqPreRequisiteAppNT(RqPreRequisiteApp)>>setMenuStates: <aMenu>
[] in RqPreRequisiteApp>>aboutToDisplayFileMenu
OrderedCollection>>do: <aBlockClosure>
RqPreRequisiteAppNT(RqPreRequisiteApp)>>aboutToDisplayFileMenu
Message>>perform
Message>>evaluate
MenuWindow(Object)>>triggerEvent: <#aboutToDisplayMenu>
MenuWindow>>aboutToStartMenu
RqMDIFrame(TopPane)>>initMenu
RqMDIFrame(Window)>>wmInitmenu: <27722679> with: <0>
RqMDIFrame(Object)>>perform: <#wmInitmenu:with:> with: <27722679> with: <0>
NotificationManager>>notify: <aWinMessage>
NotificationManager>>notifyRecursive
NotificationManager>>recursiveMessage
SystemDictionary>>recursiveMessage
SystemDictionary>>launch
NotificationManager>>readWinQueue
Process class>>osEventInterruptGui
Process class>>osEventInterrupt
WinSystemTime class(Object)>>vmInterrupt: <#osEventInterrupt>
WinSystemTime class(ExternalBuffer class)>>new
Time class>>currentTimeInto: <anArray>
Time class>>millisecondClockValue
Time class>>delay: <1000>
```

Figure 5-97 Rational RequisitePro error log example

Rational RequisitePro Web

Rational RequisiteWeb is powered with the Rational Web platform, which uses a combination of the IBM HTTP Server and WebSphere. The logging information that is captured for RequisiteWeb is stored in the profile directory for WebSphere. This directory contains various log files, including the system out log, for example (Figure 5-98). The logs are in <install directory>\IBM Rational\common\rwp\EmbeddedExpress\profiles\profile2\logs.



```
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R log4j:ERROR setFile(null,true) call failed.
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R java.io.FileNotFoundException: C:\Temp\RPMReqproLogs\rpmreqpro.log (The s
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.io.FileOutputStream.<init>(FileOutputStream.java:203)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.io.FileOutputStream.<init>(FileOutputStream.java:128)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.FileAppender.setFile(FileAppender.java:272)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.RollingFileAppender.setFile(RollingFileAppender.
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.FileAppender.activateOptions(FileAppender.java:1
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.config.PropertySetter.activate(PropertySetter.ja
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.config.PropertySetter.setProperties(PropertySett
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.config.PropertySetter.setProperties(PropertySett
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.PropertyConfigurator.parseAppender(PropertyConfi
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.PropertyConfigurator.parseCategory(PropertyConfi
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.PropertyConfigurator.configureRootCategory(Prope
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.PropertyConfigurator.doConfigure(PropertyConfigu
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.PropertyConfigurator.doConfigure(PropertyConfigu
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.helpers.OptionConverter.selectAndConfigure(Optio
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.LogManager.<clinit>(LogManager.java:113)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.J9VMInternals.initializeImpl(Native Method)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.J9VMInternals.initialize(J9VMInternals.java:192)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.log4j.Logger.getLogger(Logger.java:85)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.commons.logging.impl.Log4JLogger.getLogger(Log4JLogger
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.commons.logging.impl.Log4JLogger.<init>(Log4JLogger.ja
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Met
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConst
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Delegat
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.reflect.Constructor.newInstance(Constructor.java:521)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.commons.logging.impl.LogFactoryImpl.newInstance(LogFac
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.commons.logging.impl.LogFactoryImpl.getInstance(LogFac
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.commons.logging.LogFactory.getLog(LogFactory.java:414)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.axis.components.logger.LogFactory.getLog(LogFactory.ja
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at org.apache.axis.transport.http.AxisServletBase.<clinit>(AxisServl
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.J9VMInternals.initializeImpl(Native Method)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.J9VMInternals.initialize(J9VMInternals.java:192)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.J9VMInternals.initialize(J9VMInternals.java:157)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.Class.newInstanceImpl(Native Method)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.lang.Class.newInstance(Class.java:1300)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.beans.Beans.instantiate(Beans.java:219)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at java.beans.Beans.instantiate(Beans.java:63)
[3/6/08 23:05:27:619 EST] 00000016 SystemErr     R       at com.ibm.ws.webcontainer.servlet.ServletWrapper$1.run(ServletWrapp
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.security.util.AccessController.doPrivileged(AccessCont
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.webcontainer.servlet.ServletWrapper.loadServlet(Servle
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.wswebcontainer.servlet.ServletWrapper.loadServlet(Serv
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.webcontainer.servlet.ServletWrapper.initialize(Servlet
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.wswebcontainer.servlet.ServletWrapper.initialize(Servl
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.wsspi.webcontainer.extension.WebExtensionProcessor.create
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.webcontainer.webapp.WebApp.getServletWrapper(WebApp.ja
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.webcontainer.webapp.WebApp.initializeTargetMappings(We
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.webcontainer.webapp.WebApp.commonInitializationFinish(
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.wswebcontainer.webapp.WebApp.initialize(WebApp.java:28
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.wswebcontainer.webapp.WebGroup.addWebApplication(WebGr
[3/6/08 23:05:27:650 EST] 00000016 SystemErr     R       at com.ibm.ws.wswebcontainer.VirtualHost.addWebApplication(VirtualHo
```

Figure 5-98 RequisitePro Web interface log example

Rational RequisitePro performance considerations

There are many factors for performance considerations. Some of these factors include usage model for a team, size of a project, distributed nature of the team, and so on. All of these factors cannot be described in detail in the context of this book, nor is it the focus. Good sources for this information include technotes and *Global Development and Delivery in Practice: Experiences of the IBM Rational India Lab*, SG24-7424.

Rational RequisitePro Web performance considerations

Because Rational RequisitePro Web uses the Rational Web platform as its Web server and servlet engine, many of the performance considerations depend upon this technology. The main consideration for Rational RequisitePro Web is the total concurrent users. As this number becomes significant, such as 100 or more users, administrators must consider load balancing the server. Additionally, compression for the HTTP server and adjusting the thread pool size can also improve performance in data representation with the Web interface.

Integration considerations for Rational RequisitePro and Rational ClearQuest

Information regarding the configuration requirements for this integration is included in both the Rational ClearQuest and Rational RequisitePro installation guides as well as in “Rational RequisitePro and Rational ClearQuest integration” in the information center at the following address:

https://publib.boulder.ibm.com/infocenter/cqhelp/v7r1m0/index.jsp?topic=/com.ibm.rational.clearquest.integrations.doc/topics/cqint_reqpro/c_reqpro_cq_integ.htm

Consider the following main points:

- The Rational ClearQuest schema must have the Rational RequisitePro package applied to enable the integration. Review the sections that discuss various packages in the Rational ClearQuest installation guide.
- The Rational RequisitePro project must include attributes of the Rational ClearQuest integration that reflect stateless records that exist in Rational ClearQuest.
- The Rational Administrator project must be accessible by both Rational ClearQuest and Rational RequisitePro because it serves as a hub for the integration.

Part C

Act 2: Collaborative development

In this part, we introduce and describe the second act in the storyboard, which involves a component team that is responsible for implementing the change. In Chapter 6, “An agile team implements a change” on page 213, we provide information about collaborative development as it relates to the scenario. Then in Chapter 7, “Rational Team Concert for collaborative development” on page 231, we provide details about the IBM Rational products that are used to support this act of the storyboard.

Role-based guide: To understand how the content in this part applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

6

An agile team implements a change

In this chapter, we provide an overview of collaborative development and a reference scenario for how it can be applied by an agile component team that is working within a larger enterprise solution team.

We include the following sections in this chapter:

We also include information about how this scenario relates to the previous scenarios and how it can impact the subsequent scenario in the life cycle.

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

6.1 Introduction to collaborative development

Development is no longer an individual effort performed in isolation. Development has become a team sport where teams of developers, who are often geographically distributed, develop effective software in concert. New collaborative development principles and new development tool solutions are needed to support collaborating with co-located or geographically distributed team members.

In this chapter, we discuss the new market trends in collaborative development and how the IBM Rational Application Lifecycle Management (ALM) solutions, such as Rational Team Concert, which is built on Jazz, make collaborative development easy, productive, and fun.

6.1.1 The changing collaborative development market

The industry shift and development challenges that are discussed in 2.1.1, “Changes in the ALM market” on page 16, have a direct impact on how teams develop software. In this section, we discuss how software development and delivery have moved toward a collaborative development model that supports a more agile way of working.

Changes toward collaborative development

Collaboration is particularly important in the practice of software delivery. After all, software is the product of many conversations. To create software that satisfies the needs of users, many people across the organization and geographic boundaries discuss the needs and approaches to satisfy customer demand. These conversations result in a clear set of requirements that can be implemented by the development team.

During development, the teams continue to collaborate to ensure that the design, prototypes, and final implementation best suit the needs of the stakeholders. A high level of collaboration is critical because the development process remains fundamentally difficult. The difficulty is due, in part, to shifting requirements as the stakeholder needs become clarified with each review and other real-time changes that are needed to provide software that best satisfies their users’ needs. It is also due to the complexity of the products and services that are being delivered. Today’s teams are often geographically or organizationally distributed, adding to the risks and complications in software delivery with different time zones, organizational boundaries, and language barriers.

All of these challenges require development teams to work closer together. Team members need a solution that helps them act in concert regardless of their location or time zone. Team leads seek to monitor progress throughout the development life cycle with teams that are spread around the world.

Collaborative development pragmatically removes several of the barriers that add risk to traditional development teams operating in silos, through the following methods:

- By using a *shared vocabulary*, team members can collaborate in context and jointly use and evolve the vocabulary, rather than form a committee to establish one.
- *Shared measurements* allow transparency to plans, work, and results, so that teams can configure their targets and track health based on real-time metrics.
- *Shared assets* avoid duplication of effort and benefit reuse by seamless access to assets and team members.
- By using *shared practices*, teams can form practices and ensure consistency by right-sized process enactment.
- By ensuring *continuous improvement*, teams can improve practices in small steps.

Changes toward distributed development

In the modern economy, companies are continuing to expand globally by distributing their teams around the world through a variety of means, including offshoring, acquiring, partnering, and outsourcing. As globalization becomes more prevalent, many companies are evolving their approach and practices for distributed development.

Traditionally global organizations assigned product or project ownership by locality, making each location or branch responsible for delivering its own application or project, and each of these sites worked independently. As companies expand their presence around the world, changes are required where multiple sites become a team of teams that contributes to a global delivery chain. Each team might own a module or component that they deliver upstream for integration with components from other locations or companies, culminating in a final solution, application, or product. Teams might belong to the same organization, division, or company, or in some cases, teams cross organizational boundaries.

However, many geographically distributed teams are facing a number of issues and pain points:

- Misunderstood or mismatched processes between teams can lead to mistakes, increased rework, and decreased productivity.
- Cultural issues and language barriers cause delays and affect working relationships.
- Visibility and control of development activities at all sites are challenging. Coordinating work across multiple sites is time consuming.
- Project metrics can be inconsistent or difficult to obtain, making it difficult to measure success.
- Infrastructures and development tools can vary widely due to siloed organizations, acquisitions, and outsourcing, limiting transparency and access.

Collaborative development is one of the identified critical success factors of distributed development because it addresses the following areas:

- Global coordination and oversight
- Well-defined processes and workflow
- Inventory and information management
- Clear and accessible communication
- Flexible and adaptable development infrastructure

Changes toward agile development

Agile principles are successfully used by many organizations. The variations in detail and approach are many. More organizations are attempting to grow with agile principles by allowing smaller agile teams to contribute to the larger enterprise software supply chain, often in geographically distributed projects.

IBM Rational customers look to Rational for recommendations on the Agility-at-Scale method. This guidance is gaining in importance as enterprises seek to adopt and improve on agile principles. See “Approaches to Agility at Scale” on page 43, which elaborates on the following key Agility-at-Scale success factors:

- Constant collaboration
- Iterative development
- Agility of small teams
- Adoption of frequent builds
- Adoption of frequent integration testing
- Actively engaged stakeholders

Adopting some form of the agile method is no longer a question of “if” but more a question of “how” and “when.” Teams are exploring which method makes the most sense for a particular project given the people and circumstances surrounding the project.

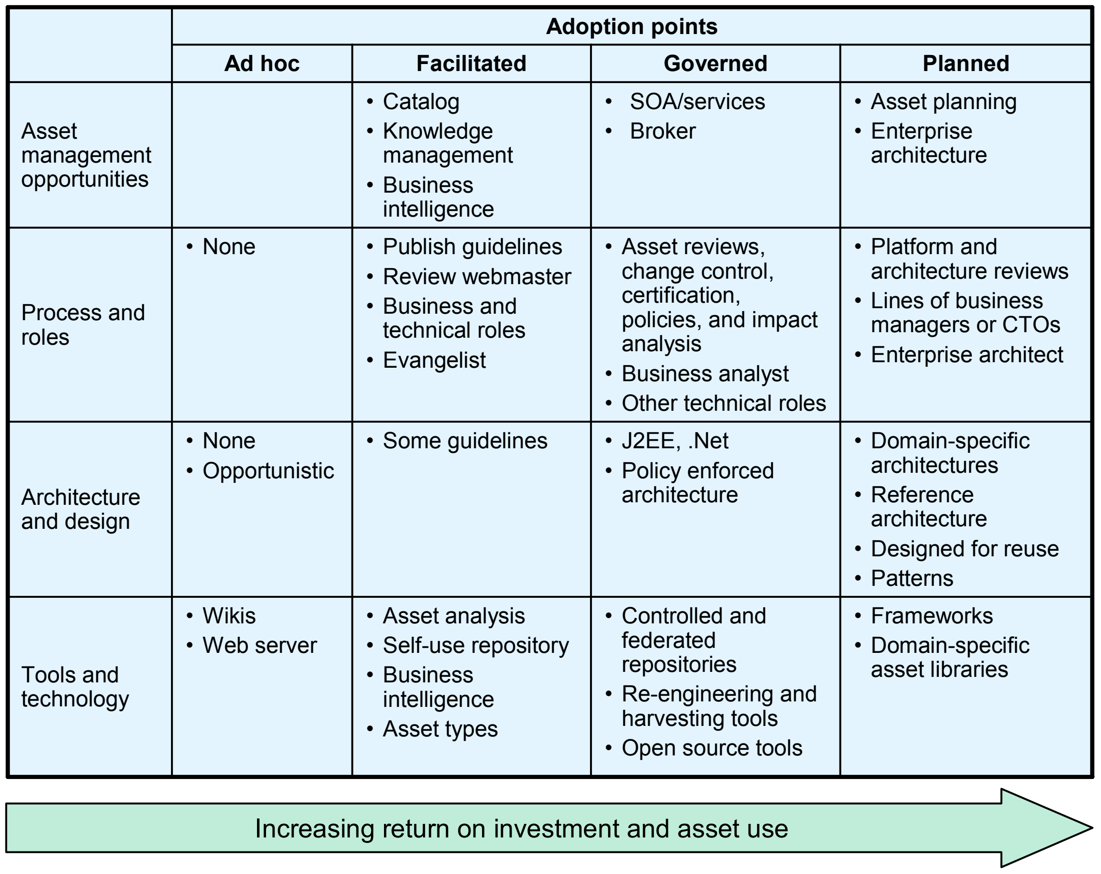
Changes toward asset management

Historically enterprises have viewed asset management as a reuse-focused activity. However, modern enterprises look to asset management as a way to address challenges with communicating knowledge, managing delivery and governance of their assets, and understanding the impact of changes for their asset planning. The core need to reuse still exists, but many enterprises manage assets not to be reused, but to be measured and reported, as well as a means to disseminate knowledge and affect decision making.

To determine the approach to managing your assets, consider the following key items:

- Examine the pain points.
- Describe the asset management scenarios to address the pain points.
- Select your asset management adoption point.

For example, the enterprise might determine that a major pain point is that they do not have a single point of entry to understand what their assets are, across the various development artifacts. This lack of information affects their ability to both disseminate knowledge and understand asset relationships. By using the image in Figure 6-1, the enterprise can adopt asset management in the “Facilitated” column, creating a repository that serves as a catalog and information disseminator.



	Adoption points			
	Ad hoc	Facilitated	Governed	Planned
Asset management opportunities		• Catalog • Knowledge management • Business intelligence	• SOA/services • Broker	• Asset planning • Enterprise architecture
Process and roles	• None	• Publish guidelines • Review webmaster • Business and technical roles • Evangelist	• Asset reviews, change control, certification, policies, and impact analysis • Business analyst • Other technical roles	• Platform and architecture reviews • Lines of business managers or CTOs • Enterprise architect
Architecture and design	• None • Opportunistic	• Some guidelines	• J2EE, .Net • Policy enforced architecture	• Domain-specific architectures • Reference architecture • Designed for reuse • Patterns
Tools and technology	• Wikis • Web server	• Asset analysis • Self-use repository • Business intelligence • Asset types	• Controlled and federated repositories • Re-engineering and harvesting tools • Open source tools	• Frameworks • Domain-specific asset libraries

Figure 6-1 Asset management adoption points

The enterprise might have several asset management adoption points under way across various teams or groups to address their respective issues.

6.1.2 Collaborative development blueprint

The goal of a Collaborative Application Lifecycle Management (CALM) solution is to streamline a team's ability to develop a release of software. Collaborative development, as part of CALM, enables a team to effectively develop and deliver software solutions.

To address the needs of the collaborative development market, Rational has produced and delivered on a collaborative development blueprint as illustrated in Figure 6-2. Rational Team Concert, which is built on Jazz products, is a new addition to the Rational portfolio, which delivers on the collaborative development blueprint. The Jazz products both define a vision for the way products can integrate to support collaborative development and provide a technology platform to deliver on this vision.

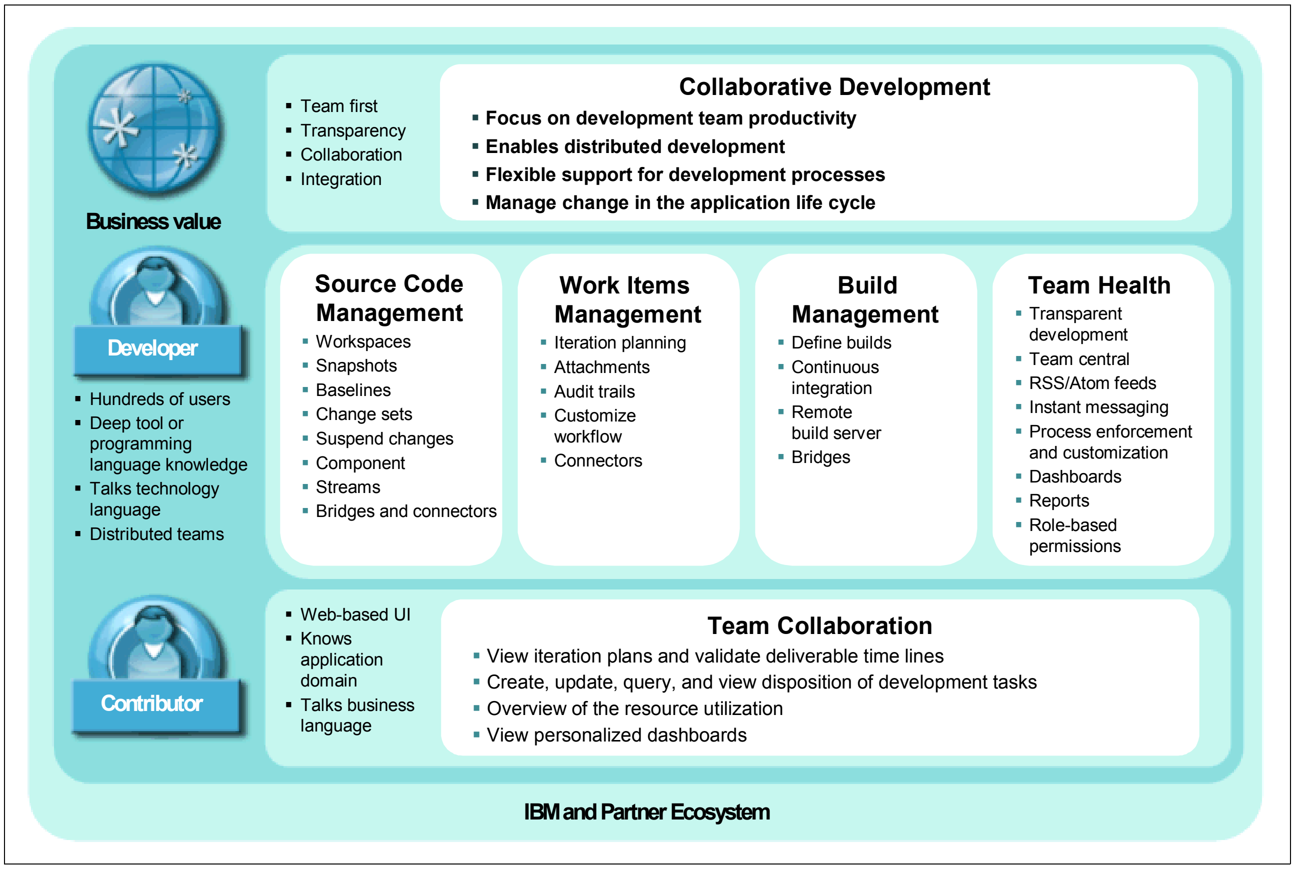


Figure 6-2 Blueprint for collaborative development

Collaborative development using Rational Team Concert

Collaborative development coordinates the disciplines of source code management, work items management, build management, and team health. A collaborative development system provides a repository for managing and organizing the development effort. By placing all assets in the collaborative development system, the development team has a clearer indicator of their progress. Source code change sets are linked to work items. Change sets are tracked in each build. Work item status, quantity, and distribution across team members, along with build performance metrics, contribute to understanding the team health.

Collaborative development is enabled in Rational Team Concert, which is built on the Jazz platform. The Jazz platform is a scalable, extensible team collaboration platform for integrating work across the phases of the development life cycle. The Jazz platform helps teams build software more effectively while making the software development activity more productive and enjoyable.

The Jazz platform provides real-time collaboration that is delivered in context. Accurate project health information is drawn directly from actual work, rather than from time-consuming progress reports. Traceability and auditability are automated by managing all artifacts and their inter-relationships across the life cycle. Instant messaging and other types of collaboration are integrated to support communication and presence awareness, and RSS feeds enable everyone to be informed of all events. The Jazz platform is built on Eclipse and other open technologies and serves as a foundation for the Rational Software Delivery Platform. In addition, the Jazz products are open to the IBM partner ecosystem to extend the platform.

The Jazz platform also supports Agility-at-Scale development practices to larger distributed teams as discussed in 2.2.3, “Scaling agile methods” on page 38. Agile development works particularly well for smaller, co-located teams. Rational Team Concert, which is built on the Jazz platform, provides a centralized platform for such teams. The transparency and visibility that Jazz provides, and the interoperability with enterprise-level change management, means that the advantages of agile methods can be extended to larger, distributed teams in an enterprise.

Rational Team Concert for IBM System z and System i extend the core capabilities for Collaborative Development to IBM z/OS® and IBM i development.

Source code management

Teams organize their source code, documents, and other assets by using the source control capabilities. The source code management capabilities in Rational Team Concert provide change-flow management to facilitate the sharing of controlled assets, retain a history of changes made to these assets, and enable simultaneous development of multiple versions of shared assets. With these capabilities, teams can work on several development lines at the same time.

The source code management capability builds on the principles of components, streams, change sets, and workspaces that provide collaborative and flexible usage models for teams to develop, integrate, and deliver complex applications. Unique capabilities with server-based workspaces enable users to suspend and resume changes. By doing so, an agile team can rapidly respond to issues and deliver quick fixes, without adding risk and overhead to ongoing development. The ability to exchange and collaborate on change sets lets team members share work, effectively perform code reviews, deliver or withdraw changes, restore code state snapshots, or resolve conflicting changes made by team members.

The source code management in Rational Team Concert provides rich and centralized change management capabilities that are simple to use. It promotes team collaboration for co-located and distributed teams.

Work items management and planning

Collaborative development starts with iteration planning. The Work Items component enables team members to easily see and track work that is assigned to them. The component includes defects that are submitted against the components for which they are responsible. The team can use the Work Items component to plan development work for milestones and obtain metrics that indicate the progress made toward the project development and quality goals.

By the seamless integration of planning and change management in Rational Team Concert, teams can benefit from significant tool automation of the agile collaborative development principles. In Rational Team Concert, the tasks on the project plan are the same as the actual living tasks that the owners are working against. This alleviates the burden of project plan updates from project manager and gives everyone full real-time transparency into the actual progress of tasks against the plan.

The change management process and the management of work items are tightly interlinked and need to be configured to fit the process that is adopted by the development team. The Jazz platform provides process enactment that makes work items aware of the development process. By using Rational Team Concert, teams can deploy ready-to-use development processes, such as Open Unified Process (OpenUP) or The Eclipse Way, or customize the Work Item information model, workflow, and process to fit a specific change management process.

Build management

Build management is more than just a “compile” of a project. Build management involves the entire process of assembling, compiling, verifying, and packaging a solution:

- The right version of the source code must be collected into the build area.
- The application is compiled for one or more operating systems.
- Build verification tests are run.
- The code is statically and dynamically analyzed.
- The build results are captured and staged.
- The entire process must leave an audit trail.
- The team depending on the build must be notified of its availability and state.
- Not at the least, the metrics that are related to build health trends must be collected and made available to the team.

Build management also includes the capability for individual team members to extend the project compilation capabilities in Eclipse with Personal Builds by using the software configuration management (SCM) and build capabilities. With personal builds, developers can run the build integration process in their personal sandbox and validate code stability prior to delivering the changes to integration with the risk of breaking or destabilizing the team build.

The team build capabilities in Rational Team Concert are based on the Jazz platform’s support for the automation, monitoring, and notification of the team builds. The capabilities support a model for representing the team’s build definitions, build engines, build results, and build notifications. The model is designed to support teams by using a range of different build technologies and is configured by teach team to fit their build process requirements.

Team build in Rational Team Concert can be configured to run locally to the repository or distributed by using dedicated build clients. Builds can also be integrated into Rational Build Forge to leverage a central enterprise build function.

Team health

The team health capabilities in Rational Team Concert are based on the Jazz platform’s support for transparency, seamless, and effortless access to assets, notification, reporting, and monitoring capabilities. Team health capabilities enable self-organized agile teams to respond effectively and rapidly to project challenges and changing project needs.

With the transparency of collaborative development by using Rational Team Concert, teams can view the real-time plans and commitments that their team is working on. Notifications keep team members updated on changes that are related to their work. In-context collaboration, instant messaging, and collaborative asset traceability make it easy for teams to share and re-balance work assignments.

To ensure team success, teams need effortless access to metrics, queries, and reports. By integrating health indicators into the various views of the tools, team members and team roles have continuous team health updates at their fingertips. Dashboards provide team-wide information about the health of the project or on individual component teams. Both teams and individuals can configure dashboards by using important health metrics as needed by role or life-cycle phase. Views provided in Rational Team Concert, such as TeamCentral and MyWork, bring the generic dashboard and team transparency capabilities to the fingertips of the practitioner by providing effortless and continuous access to team events and work schedules.

Team collaboration with team contributors

A key capability in collaborative development is to prevent a lock-in of information from external contributors. To respond to customer needs, development teams must ensure continuous stakeholder involvement and guidance. This interaction between the users and the development team drives collaborative development demands on transparency. The product owners, analysts, program office, and other cross-organizational stakeholders need easy access to project information to provide timely input.

External team contributors demand easy access to project health information and assets by using Web 2.0 usage patterns. By providing guest access and integrating with enterprise user authentication services, teams can establish a balance between information security, stakeholder access, and information management.

Rational Team Concert enables stakeholder access to the project plans for development lines, iteration plans, and work items. This access enables stakeholders to validate the timing and scope of project deliverables. Team contributors can also use the Web interface to create, modify, and report on work items. This ability gives the development team early and continuous feedback and contributes to the success and viability of the software that they are producing. Users of the Web interface can also configure individual dashboards as required by the needs of their role. By using the dashboard, users can combine access to work item queries, notifications, and health metrics.

Asset management

Asset management involves the process of planning, developing, governing, and using assets throughout their entire life cycle and across the enterprise. It includes the ability to ensure that assets have the following capabilities:

- Can be quickly found to avoid duplication costs
- Can be consistently reviewed in an efficient manner to ensure usability and alignment with business and technical strategic direction
- Comply with policies such as the usage of open source assets
- Can effectively be brokered between producers and consumers
- Are properly funded based on usage

Rational Asset Manager provides a published asset management repository for technical and business assets. It uses the Reusable Asset Specification to achieve the following goals:

- Allow you to define, create, and modify assets
- Provide asset type specific search and governance
- Measure asset reuse in development
- Handle any kind of asset such as applications, business processes, business rules, components, patterns, services, frameworks, templates, and other user-defined asset types

As such, asset management is more than just a published wiki or Web server of components that teams can use. The primary benefit that it provides is to reduce software development, operations and maintenance costs by improving quality by facilitating the reuse of approved and proven assets. In addition, asset management facilitates asset usage and traceability by integrating with other IBM Rational, WebSphere, Tivoli software development tools, as well as your own tools.

6.2 A reference scenario for collaborative development

In the following section, we provide an overview of the steps taken by the component development team to design, develop, test, and deliver enhancements to the component. In 7.3, "Rational Team Concert for agile development" on page 246, we demonstrate this workflow by using Rational Team Concert.

The scenario in this chapter continues to build on the previous act in this book. In 4.2, "A reference scenario for responding to a change request" on page 105, of Act 1, Patricia and Marco align on an iteration plan that includes the work to re-brand the CreditCheck Web UI. The scenario continues in 8.2, "A reference scenario for enterprise build management" on page 325, of Act 3, where Rebecca integrates the delivered changes to the solution.

In Act 2: Collaborative development, we discuss how Diedrie collaborates with her team to validate the design changes to the component and how she develops, tests, and delivers the changes. Before delivering to the integration stream, she asks Marco to review her changes. Diedrie then monitors that her delivery is successfully incorporated into the solution integration build. This act is illustrated in Figure 6-3.

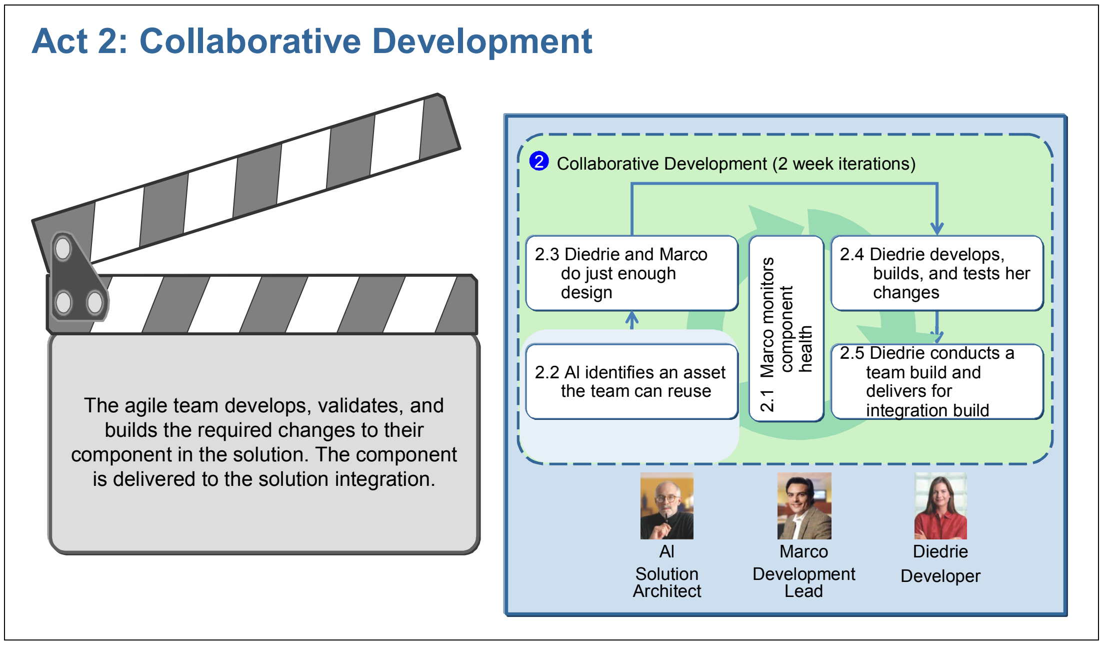

Act 2: Collaborative Development
The agile team develops, validates, and builds the required changes to their component in the solution. The component is delivered to the solution integration.
2 Collaborative Development (2 week iterations)
2.3 Diedrie and Marco do just enough design
2.4 Diedrie develops, builds, and tests her changes
2.1 Marco monitors component health
2.2 Al identifies an asset the team can reuse
2.5 Diedrie conducts a team build and delivers for integration build
Al Solution Architect
Marco Development Lead
Diedrie Developer

Figure 6-3 In Act 2, the component team develops and delivers requested UI branding changes

6.2.1 The actors

This scenario includes the several key actors as described in this section.

Marco is the development lead in the component team that is responsible for the delivery and quality of the Credit Check Component in the Account Opening project. As part of the agile team, he takes a leadership role in the architecture and implementation of the component. He works collaboratively with his team and with Bob, Al, and Tammy to ensure that the teams collectively deliver on the expected release requirements, design principles, and solution quality.

Diedrie is one of the developers on the component team. As an agile developer, her responsibilities span the end-to-end changes from design to test and delivery. She is also supervising the continuous component builds and weekly component deliveries. She implements the UI branding changes that Bob requested in the CreditCheck component.

Al is the software architect for the Account Opening solution, as well as for other solutions within the enterprise. Marco and Diedrie collaborate with Al and request his guidance on reusing the corporate assets for UI branding.

6.2.2 The workflow

In Act 2 of the ALM scenario, we capture the scenes and steps of a single request that is being developed and delivered by Marco and his team. The agile component team collaborates to right size the development effort by reusing assets and developing and validating the required component changes. At the end of the act, the component is delivered to solution integration.

The following tasks, which are shown in Figure 6-4 on page 223, are performed by the agile component team in Act 2: Collaborative development:

- Al identifies an asset that the team can reuse.
- Diedrie and Marco do just enough design.
- Diedrie develops, builds, and tests her changes.
- Diedrie conducts a team build and delivers for the integration build.

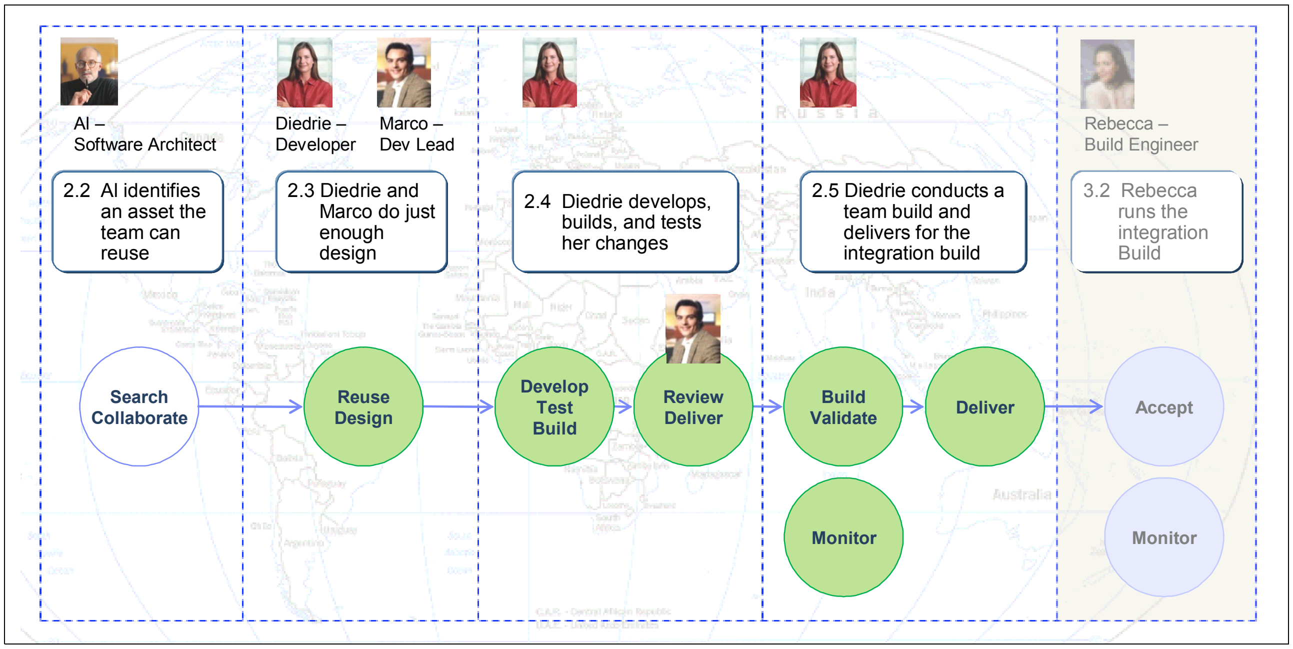

Al – Software Architect
Diedrie – Developer
Marco – Dev Lead
Rebecca – Build Engineer
2.2 Al identifies an asset the team can reuse
2.3 Diedrie and Marco do just enough design
2.4 Diedrie develops, builds, and tests her changes
2.5 Diedrie conducts a team build and delivers for the integration build
3.2 Rebecca runs the integration Build
Search Collaborate
Reuse Design
Develop Test Build
Review Deliver
Build Validate
Deliver
Accept
Monitor
Monitor

Figure 6-4 The flow of steps for Act 2: Collaborative development

6.2.3 Marco monitors component health

Synopsis: Marco conducts daily stand-up meetings with his component team to align the work for the day. The team talks about their work and deliveries in the context of their live iteration plan. Each team member has the opportunity to collaborate with the rest of the team on work progress, change dependencies, and blocking issues.

During the daily stand-up meetings, the team uses the live iteration plan to make tactical changes to improve project health. New work items are triaged or added. Ownership and priorities or work are clarified. Blocking issues are reprioritized.

Diedrie has been assigned the work to re-brand the Credit Check user interface and make other changes. She estimates and schedules the work that is assigned to her for the current iteration.

The workflow in this scene captures how Marco completes the following tasks:

- Collaborates with his team at daily stand-up meetings, regardless of geographic location
- Conducts live collaboration sessions on the component iteration plan
- Enables the team to be self-organizing regarding tasks, work, and assignments

6.2.4 Al identifies an asset the team can use

Synopsis: Diedrie is considering her options on the Corporate UI Branding work that is assigned to her. She needs an advice from Al on a reusable component. She sends Al an instant message, attaches a link to her work item, and asks for his guidance. Al knows that a reusable component is available because he has been helping other teams on the same topic. Sharing this component with Diedrie saves her and the rest of the team development and testing time.

Al logs into the asset repository and starts his search in the Credit Management community. This community is set up to support this business line, and the CreditCheck team has approvals and permission to participate in this community.

Al searches the entries in the Credit Management community and gets several hits on branding, but focuses his attention to the “Re-brand UI Component” asset that has received high scores from other users. After viewing the details of the asset, Al decides to share his discovery with Diedrie and Marco. He collaborates by sending an e-mail that includes a link to the found asset.

Diedrie and Marco receive the e-mail and invite Al to a Web discussion about the topic. Al shares his desktop and helps the team browse the documentation, the design, and the test cases. Diedrie takes notes about the discussion and saves them as an attachment to her UI branding work item. She also saves the link to the reuse component as a related artifact.

The workflow in this scene captures how Al completes the following tasks:

- Uses asset repositories and search to identify a reusable asset
- Uses asset scores to drive adoption decisions
- Collaborates with his team and shares guidance
- Attaches discussions and design decisions in context of the work

6.2.5 Marco and Diedrie do just enough design

Synopsis: Diedrie is ready to start working on the Credit Check UI branding. She opens her work item and changes its state to indicate that work has begun. She creates a new, clean, and updated workspace from the latest component baseline to integrate her changes.

Diedrie then logs into the reuse repository from her development environment, by using the link that Al provided. She reviews the online asset documentation and her notes from her discussion with Al. She then starts importing all related assets into her workspace. Some assets have Eclipse projects in them. Others must be pointed to a target project to be imported.

Diedrie then proceeds and starts prototyping the redesign in one of the Credit Check UI forms. She saves her design changes and a new change set gets created and associated with her work assignment.

To confirm her design strategy, Diedrie invites Marco and Al to a design review of the changes captured in the change set. Marco and Al accept her changes into their sandbox workspaces and start browsing the changes. Both Marco and Al confirm that the design of the changes looks good. To get validation from the stakeholder, Diedrie brings Bob into the discussion and runs a demonstration of the updated UI form. Bob confirms that the corporate brand design and manner of use are now correct.

The workflow in this scene captures how Diedrie completes the following tasks:

- Locates, inspects, and imports the reusable asset from an asset repository
- Uses a sandbox workspace to prototype on her design collaboration with other contributors
- Uses team collaboration to validate her design with architects and stakeholders

6.2.6 Diedrie develops, builds, and tests her changes

Synopsis: Diedrie is now ready to proceed and complete the development of redesigning all UI forms.

The component team is doing test-driven development (TDD). Diedrie iteratively focuses first on the JUnit test that validates her change. She updates and runs her test, and confirms that the test fails. She then iteratively applies the design pattern for the UI changes, builds her changes, and reruns the test. She proceeds iterating until all CreditCheck form tests pass and the code has been updated.

Diedrie is now ready to merge any incoming changes from her teammates. This is a prerequisite to deliver her changes to integration. She accepts all incoming changes and resolves change conflicts. In some cases, she needs to collaborate with other developers in her team to decide on best approaches to merging their changes.

She also needs to run a private component build to validate that nothing is broken in her workspace. As new dependencies are added in the code project, she must update the component build script to reflect these dependencies. After completing the changes to the build script, she requests a private build of her workspace. The build script integrates compilation, build validations tests, and code analysis. She confirms that the build and validation results were successful.

Diedrie is now done with her changes and must complete a code review before delivering her changes to the component integration stream. This review is a practice that the team enforces in their development process. She creates a new review request, attaches the change sets, and submits the request to Marco.

The workflow in this scene captures how Diedrie completes the following tasks:

- Develops her changes in a local workspace
- Merges with changes made by others on her team
- Does test-driven development
- Integrates code review into development practices that are adopted by the team

6.2.7 Diedrie delivers her changes and builds the component

Synopsis: Diedrie is now ready to deliver her changes to the integration stream and resolve her work assignment. She makes sure that her workspace points to the integration stream and then delivers her changes. The delivery includes her changes to the application code, the unit tests, and changes to the build and validation scripts.

To catch any issues with her delivery as soon as possible, she requests a new integration build to be run. She awaits the build result and validates that it completed successfully.

To complete her work, Diedrie opens her work assignment. She adds a summary of comments on the resolution, adds an entry for the milestone release notes, changes the state of her work to resolved, and saves her changes.

The workflow in this scene captures how Diedrie completes the following tasks:

- Delivers all related changes in one atomic operation
- Monitors that her delivery did not break the integration build
- Resolves her work assignment

6.3 Considerations in collaborative development

As discussed in this chapter, collaborative development spans the entire life cycle. The scenario that is included in this book provides a view into the collaborative development discipline, but does not cover every aspect or Rational tool solution configuration. In this section, we discuss additional considerations for collaborative development:

- Considerations in deploying a CALM solution to a small project by using only Rational Team Concert
- Considerations in deploying collaborative development by using the ClearQuest ALM solution and Unified Change Management (UCM) to a component team
- Considerations when scaling the CALM solution to an enterprise scale

For additional considerations about deploying the Rational ALM solution and the Jazz platform, see the following Web pages:

- Rational support

  `http://www.ibm.com/software/rational/support`

- Rational technical resources and best practices for the Rational software platform from developerWorks

  `http://www.ibm.com/developerworks/rational`

- Jazz

  `http://jazz.net`

6.3.1 Lifecycle solution for small teams

In this book, we capture an end-to-end ALM scenario. Chapter 2, "Collaborative Application Lifecycle Management" on page 15, defines ALM, and Chapter 3, "A scenario for CALM" on page 47, describes the scenario that is used in this book.

However, some projects or teams might decide to deploy a reduced CALM solution as appropriate to their practices, maturity, or process needs. The collaborative development blueprint captures the minimal core capabilities in a CALM solution to provide the following features:

- Process enactment and change management for an agile way of working
- SCM support to manage parallel collaborative development of branches or releases
- Build automation and integrated build quality validation
- Team health and reporting to enable team transparency

The scenario that is captured in the Part C, "Act 2: Collaborative development" on page 211, can also be viewed as a stand-alone collaborative development case for a smaller agile team. Some limitations to end-to-end ALM apply:

- Limited requirements management and definition by using work items
- Limited quality management by using build validation with JUnit test cases
- Limited team size or distributed repositories

Configuring Rational Team Concert for CALM

The Eclipse Way process configuration, which is available in Rational Team Concert, provides CALM support scaled for smaller- and medium-sized agile teams. The following considerations apply in short:

- Teams use stories and plan items to capture and organize stakeholder requests.
- Teams use parent-child relationships to organize requirements and work breakdown as work items.
- Teams use customized work items types, and tag clouds, to manage the extended tracking needs of CALM support.
- Teams integrate build scripts with test scripts to continuously manage quality as an integral part of integration. Team members take multiple roles and share development and test responsibilities.
- Teams run a project on a central server and scale multiple projects to multiple servers.

6.3.2 Collaborative development with UCM

Part 2 of our scenario is based on the story where the newly acquired agile component team is using Rational Team Concert and integrations with the ClearQuest ALM solution to deploy a collaborative development environment. In the larger context of the Account Opening release, the project is run by a team of teams. While the agile CreditCheck team is one subteam, there are many other teams. See 3.3.3, “The agile development team” on page 54. Other component teams in the scenario might use Rational ClearCase or Rational ClearQuest with UCM. See Figure 6-5.

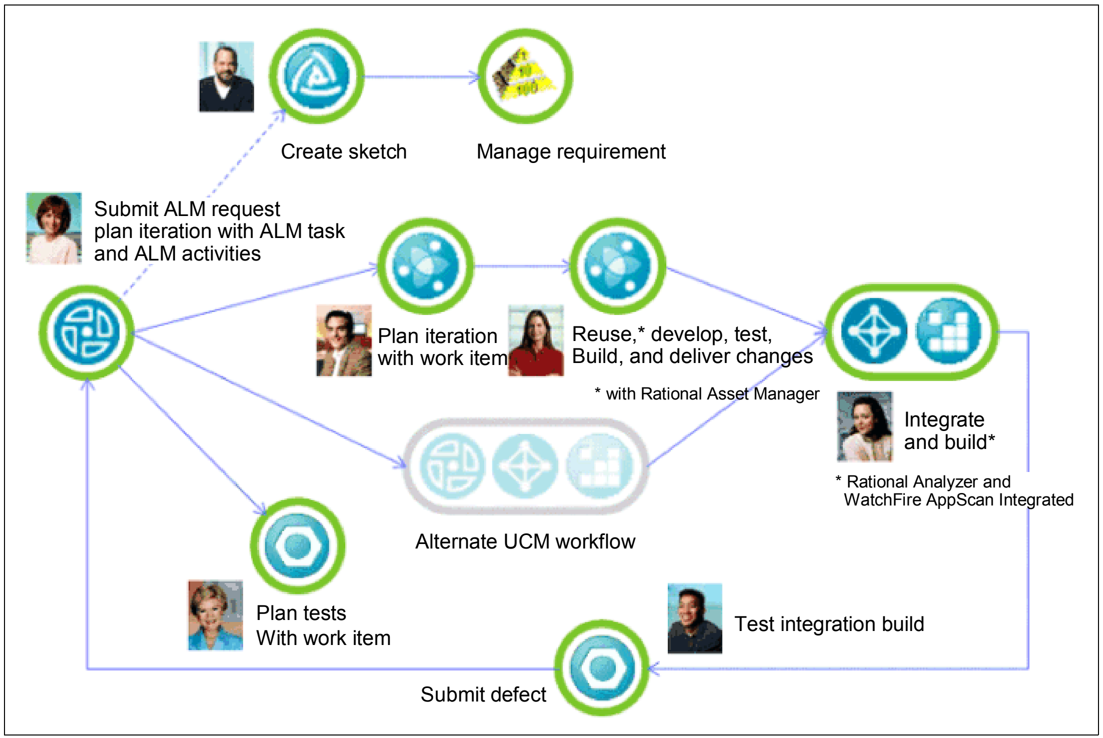


Figure 6-5 UCM workflows used by the teams in the Account Opening project

The details of the UCM workflows are covered briefly in this section as an alternative tool configuration for agile or traditional teams using collaborative development.

Part 2 of our scenario can also be configured by using Rational ClearCase or Rational ClearQuest with UCM. The following considerations apply:

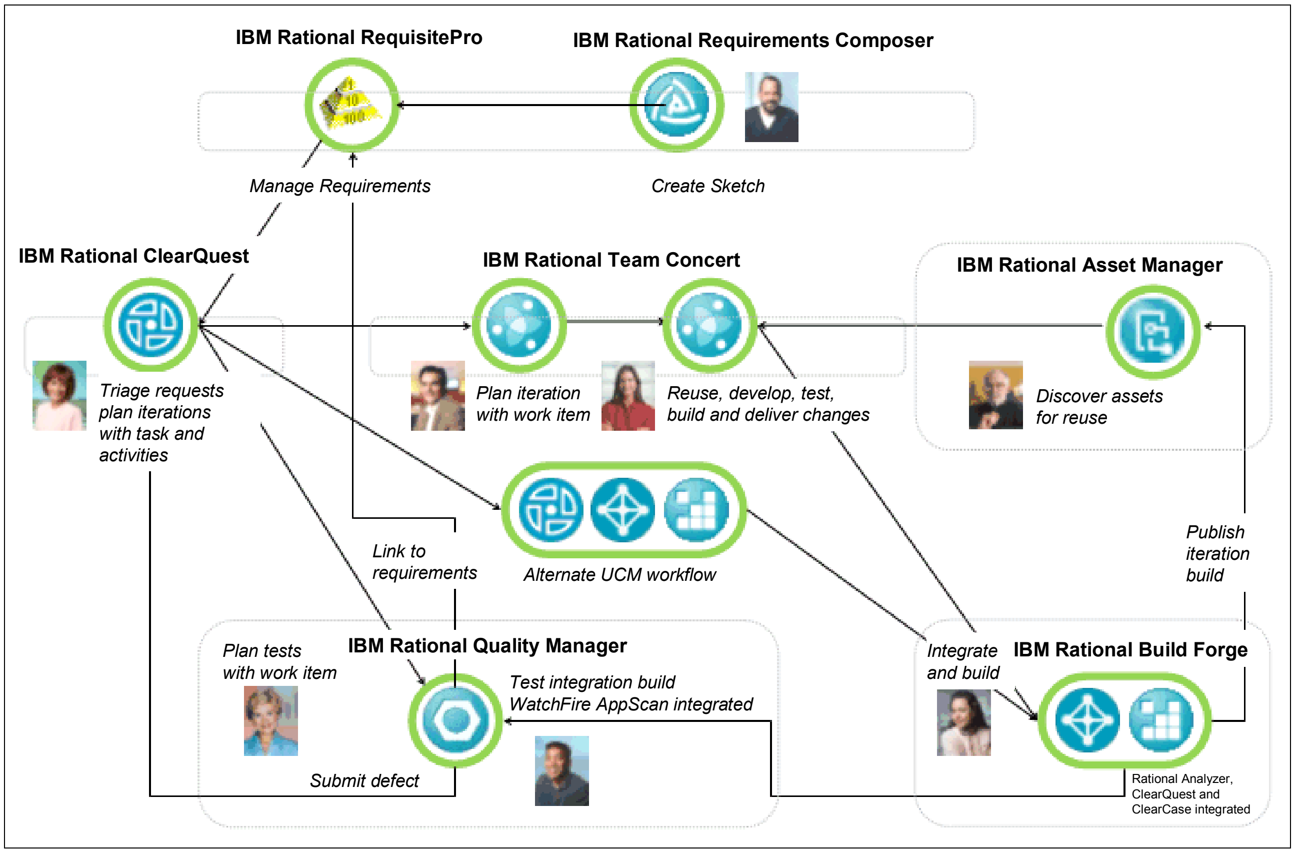
- Component teams that use Rational ClearCase or Rational ClearQuest with UCM can use the central Rational ClearCase or Rational ClearQuest repositories, or use Rational ClearCase or Rational ClearQuest MultiSite for repository replication. The ClearQuest ALM schema has specifically been designed to fully support Rational ClearQuest MultiSite.
- Several considerations apply to how the component team structures their artifacts in ClearQuest ALM. Component teams can have individual iteration schedules or multiple depending projects, or need to individually configure the development process and workflow, which is advised, to create their own ALM project. Component teams are advised to create separate child projects that link to the parent project. In the scenario in this book, each component team links to the parent Account Opening project by using using ClearQuest ALM. Project parent-child relationships are managed on the Project tab of ALM Projects. To create a new child project with a copy of the parent work configurations and iteration plans, use the Project Wizard.
- Component teams that use UCM use UCM-enabled ALM Activity records to schedule, track, and deliver work. UCM Change Sets are associated with the ALM Activities.
- ALM baselines are associated with BT Build records to track delivered UCM, which enables ALM Activities.
- The state changes on ALM Activities propagate to tracking of ALM Tasks and ALM Request for Component and project leads for project health tracking.

6.3.3 Collaborative asset management

The scenario in this book addresses a subset of asset management capabilities as implemented in Rational Asset Manager. For the assets to achieve quality and consumability, there is much collaboration around the development, sign-offs, communication, consumption, measurement, and reporting of the assets. In essence, collaboration is fundamental to asset management. Some of the collaboration capabilities that are necessary include the following components:

Communities	The interested stakeholders, producers, and consumers for a set of assets, with their associated governance.
Asset reviews and approvals	Assets migrate through multiple states with reviews and sign-offs by various teams and individuals.
Subscriptions/notifications	Asset producers, consumers, and other interested parties are notified of the asset's state through out its life cycle.
Rating	Consumers and other interested parties declare their view on the asset.
Discussion forums	Searchable discussions on the assets and their usage experience.
Access rights	Controlling access to the assets.
Impact analysis	Understanding impacted teams, projects, and individuals.
Policy compliance	Automates review processes of assets to avoid costly risks.

Collaboration crosses the boundaries of the development organization into the operations teams. As the asset is approved, it can be published into runtime repositories such as WebSphere Service Registry and Repository (WSRR) and Tivoli Change and Configuration Management Database (CCMDB), providing bi-directional navigation and cross-organizational collaboration (Figure 6-6).



IBM Rational RequisitePro
IBM Rational Requirements Composer
Manage Requirements
Create Sketch
IBM Rational ClearQuest
IBM Rational Team Concert
IBM Rational Asset Manager
Triage requests plan iterations with task and activities
Plan iteration with work item
Reuse, develop, test, build and deliver changes
Discover assets for reuse
Link to requirements
Alternate UCM workflow
Publish iteration build
Plan tests with work item
IBM Rational Quality Manager
Test integration build WatchFire AppScan integrated
Submit defect
Integrate and build
IBM Rational Build Forge
Rational Analyzer, ClearQuest and ClearCase integrated

Figure 6-6 Rational Asset Manager integrating with runtime repositories

7

Rational Team Concert for collaborative development

In this chapter, we provide a detailed demonstration of Act 2: Collaborative development. The purpose of this chapter is to provide a pragmatic demonstration of how the roles in this scenario use Rational Team Concert to accomplish their tasks. This chapter provides the following information:

- An overview of the product features of Rational Team Concert used in the scenario
- A step-by-step demonstration of using Rational Team Concert in the scenario
- A summary of the assets that are created and used by the team
- How to measure success for collaborative development in this scenario
- How the products used fit into a larger enterprise Application Lifecycle Management (ALM) scenario and how they are configured
- Tips and tricks for resolving known problems

Specifically, this chapter includes the following sections:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

7.1 Act 2: Collaborative development

This chapter includes a step-by-step discussion of how the characters in the story complete Act 2 of the storyboard, which is illustrated in Figure 7-1.

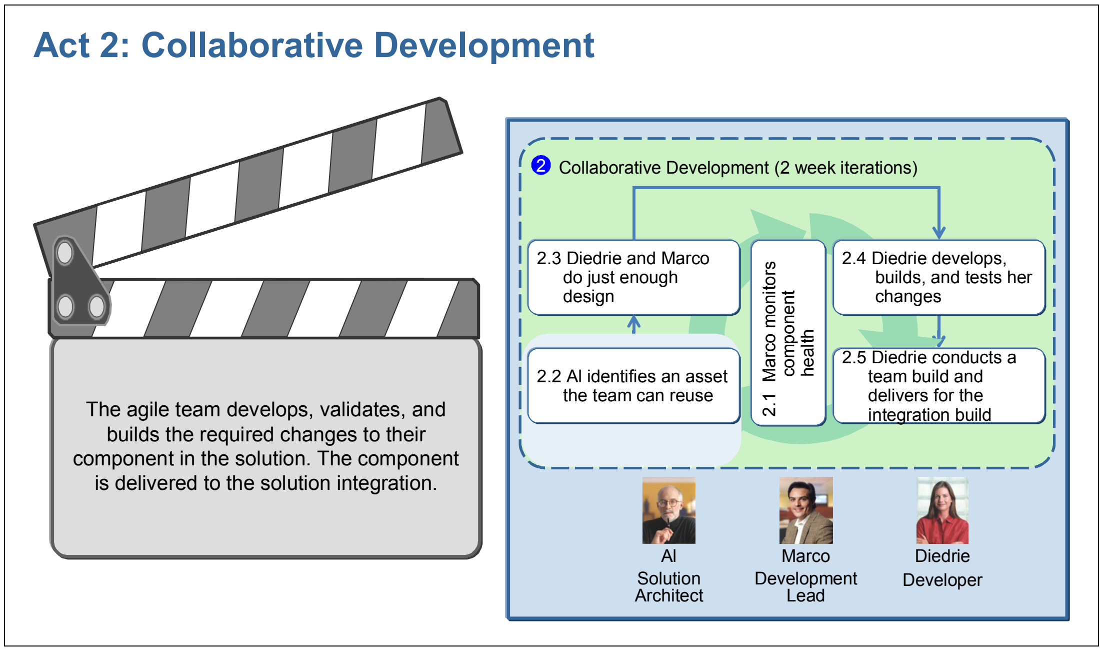

Act 2: Collaborative Development
The agile team develops, validates, and builds the required changes to their component in the solution. The component is delivered to the solution integration.
2 Collaborative Development (2 week iterations)
2.3 Diedrie and Marco do just enough design
2.2 Al identifies an asset the team can reuse
2.1 Marco monitors component health
2.4 Diedrie develops, builds, and tests her changes
2.5 Diedrie conducts a team build and delivers for the integration build
Al
Solution Architect
Marco
Development Lead
Diedrie
Developer

Figure 7-1 Act 2: Collaborative development

This act, as shown in Figure 7-1, consists of the following scenes:

- Marco monitors component health.
- Al identifies an asset that the team can reuse.
- Diedrie and Marco do just enough design.
- Diedrie develops, builds, and tests her changes.
- Diedrie conducts a team build and delivers for an integration build.

Rational Team Concert 1.0 is used in this act. It is integrated into the following products:

- Rational ClearQuest 7.1 and the ClearQuest ALM schema to do enterprise-level change management

  This integration uses the ClearQuest Connector in Rational Team Concert.

- Rational ClearCase 7.1 to do delivery to the solution integration

  This integration uses the ClearCase Connector in Rational Team Concert.

7.2 Overview of Rational Team Concert

Rational Team Concert (Figure 7-2 on page 233) is a Rational product on the Jazz team collaboration platform. It provides the following key capabilities to support collaborative development:

- Integrates seamlessly the development task across the delivery life cycle for the team
- Facilitates team collaboration and coordination and helps the team develop applications more effectively and with less risk

- ► Supports team collaboration across co-located and globally distributed teams
- ► Establishes and maintains traceability and audit trails, and automates bookkeeping so that teams are accountable
- ► Integrates into Eclipse for developers and provides Web access for external contributors
- ► Makes collaborative development more enjoyable

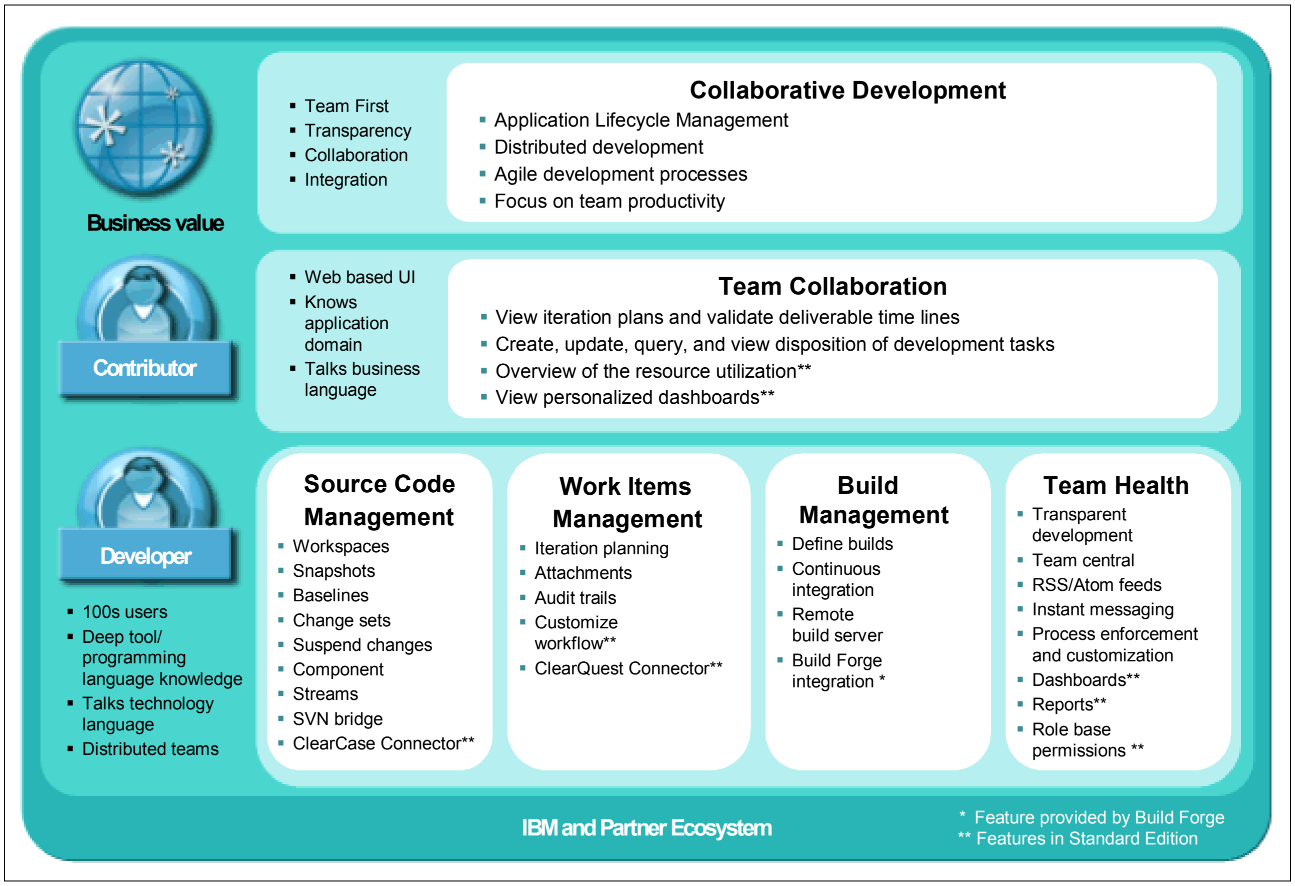


Figure 7-2 Rational Team Concert

Rational Team Concert supports and provides seamlessly integrated workflows, both for the application life-cycle assets managed by the Jazz repository, but also to assets managed by external repositories that are seamlessly integrated into the Jazz platform.

The following ALM domains are supported by Rational Team Concert (Figure 7-2):

- ► Source code management
- ► Work items management, including agile iteration planning
- ► Build management
- ► Team health and collaboration

Rational Team Concert provide integration into Eclipse for developers, and Web access to team collaboration capabilities for external contributors who require easy tool access with minimal setup.

The Jazz platform

The Jazz platform has been referenced in short in previous chapters. This platform is a scalable, extensible team collaboration platform for integrating work across the phases of the development life cycle. The Jazz platform helps teams build software more effectively while making the software development activity more agile, productive, and enjoyable. Jazz is built on Eclipse and other open technologies. It serves as a foundation for the Rational Software Delivery Platform and for a partner ecosystem to extend the platform.

The Jazz platform provides real-time collaboration that is delivered in context. This enables teams to work more closely, effectively, and transparently. Team members can conduct discussions and easily include development artifacts into the discussion, or persist the discussion as an integral part of the artifact.

The Jazz platform also supports simple artifact exchange and collaboration between developers by using higher value assets, such as complete change sets, as compared to the previous more time consuming exchange of source code files or mail with code snippets. Instant messaging and other types of collaboration are integrated to support communication and presence awareness, and RSS feeds enable everybody to be informed of events important to the team or to the individual.

With the Jazz platform, live project health information can be drawn directly from actual work, rather than from time-consuming progress reports. Traceability and auditability are automated by managing all artifacts and their inter-relationships across the life cycle.

The Jazz platform also supports Agility-at-Scale development practices to larger distributed teams as discussed in 2.2.3, “Scaling agile methods” on page 38, and in 6.1.1, “The changing collaborative development market” on page 214. Agile development works particularly well for smaller co-located teams. For larger distributed teams, Rational Team Concert provides a collaborative centralized development platform that enables the Agility-at-Scale method. The transparency and visibility that the Jazz platform provides, and the interoperability with enterprise-level change management, means that the advantages of agile methods and Rational Team Concert can be extended to larger and distributed teams in the enterprise.

Rational Team Concert for IBM System z and System i extends the core capabilities for collaborative development to IBM z/OS and IBM i development.

Scaling to a heterogeneous platform

The Jazz platform provides a range of capabilities to consolidate or support a heterogeneous development platform by using three approaches: import, bridge, and interoperate.

In Rational Team Concert, you can *import* an existing change management solution by using the import capabilities. Existing assets in CVS, Bugzilla, and other products, can be imported and merged with the Jazz repository team artifacts.

Other source code management (SCM) solutions, such as Subversion, can *bridge* into the Jazz environment while the managed artifacts reside in the external repository. Usage of the bridge capabilities requires the original workflows to be used with the external tool. In the case of Subversion, the SCM workflows require usage of a Subversion Eclipse client in Rational Team Concert.

The Jazz platform provides capabilities for external repositories to *interoperate* with the Jazz repository. Usage of such interoperability makes the external tool integrate seamlessly into the Jazz workflows. A user might not even know or care that the artifacts are interoperating with an external source as the tool gestures and ALM workflows are unchanged to regular use. Such interoperability is available for Rational ClearCase, where Unified Change Management (UCM) streams and components are integrated with the Jazz SCM component, and for Rational ClearQuest, where Rational ClearQuest records are displayed on the Jazz platform as plain work items.

Jazz.net

Detailed information about the Jazz platform and Rational Team Concert is beyond the scope of this book. You can find more information, particularly about the following topics, at the following Web site:

`http://jazz.net`

- Access to the Jazz open commercial software development project
- Access to online help information about the Rational Team Concert product
- Technical in-depth material about the Jazz platform
- Access to the global Jazz community
- Guidance about Rational Team Concert deployment and configuration practices
- Samples

Project and team areas

The Jazz platform provides basic concepts to creating and managing projects as illustrated in Figure 7-3.

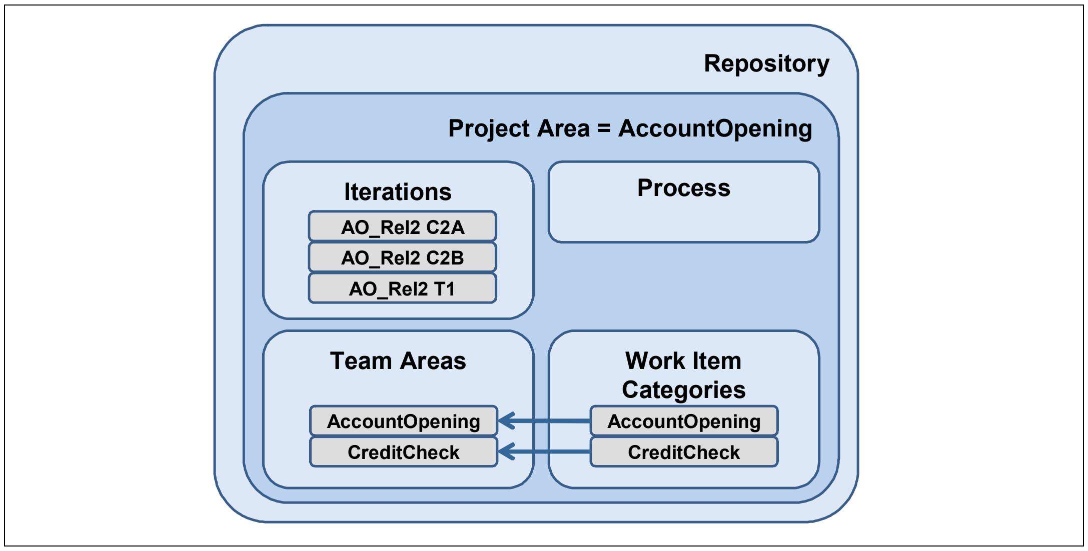


Figure 7-3 Project and team areas in Rational Team Concert

The following structures are used for project and team management in Rational Team Concert:

- Project

  A project area provides definition for the project deliverables, team structure, process, and schedule.

- Team

  A team area manages team membership, roles assignments, and team artifacts.

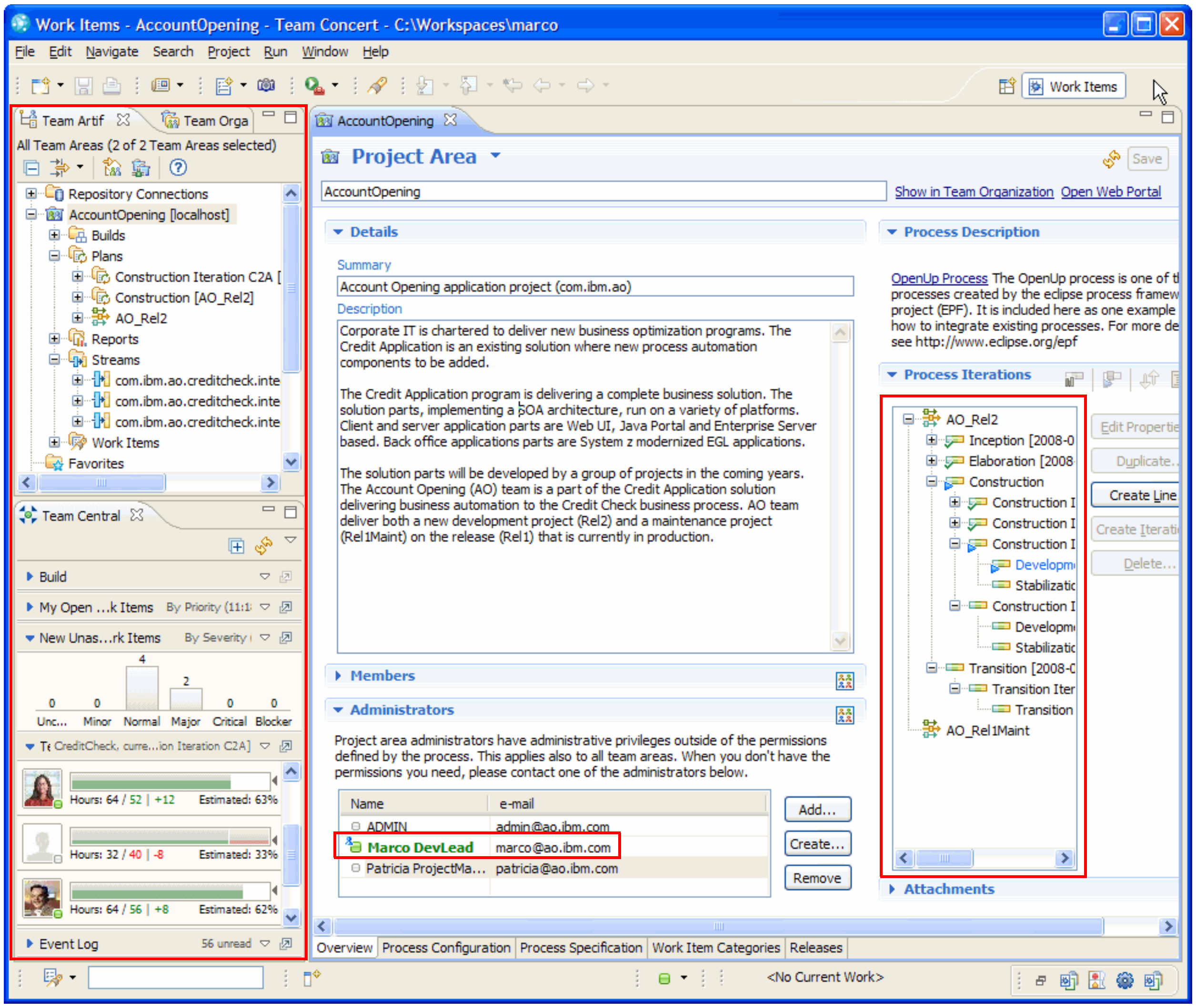
- Category

  Categories group work items by the various components or functional areas of the project and are associated with a team area.

- Team process

  A *process* is the collection of roles, practices, rules, and guidelines that are used to organize and control the flow of work. The Jazz platform is process aware, and the rules of process behavior validate preconditions and provide follow-up actions. A process can also define project reports, queries, and work item types.

The CreditCheck Team Area that brings development team together on collaborative development in Rational Team Concert is shown in Figure 7-4. Some key Team Area properties are highlighted in the figure. For example, the Team Artifacts and Team Central views provide transparency to team health, the team members and their assigned roles, and the iteration structure with the current iteration plan that is selected.


Work Items - AccountOpening - Team Concert - C:\Workspaces\marco
File Edit Navigate Search Project Run Window Help
Work Items
Team Artif
Team Orga
All Team Areas (2 of 2 Team Areas selected)
Repository Connections
AccountOpening [localhost]
Builds
Plans
Construction Iteration C2A [
Construction [AO_Rel2]
AO_Rel2
Reports
Streams
com.ibm.ao.creditcheck.inte
com.ibm.ao.creditcheck.inte
com.ibm.ao.creditcheck.inte
Work Items
Favorites
Team Central
Build
My Open ...k Items By Priority (11:1
New Unas...rk Items By Severity
Unc... Minor Normal Major Critical Blocker
Te CreditCheck, curre...ion Iteration C2A]
Hours: 64 / 52 | +12 Estimated: 63%
Hours: 32 / 40 | -8 Estimated: 33%
Hours: 64 / 56 | +8 Estimated: 62%
Event Log 56 unread
AccountOpening
Project Area
Save
Show in Team Organization Open Web Portal
Details
Summary
Description
Process Description
Process Iterations
AO_Rel2
Inception [2008-0
Elaboration [2008
Construction
Construction I
Construction I
Construction I
Developm
Stabilizatic
Construction I
Developm
Stabilizatic
Transition [2008-(
Transition Iter
Transition
AO_Rel1Maint
Edit Propertie
Duplicate...
Create Line
Create Iterati
Delete...
Members
Administrators
Add...
Create...
Remove
Attachments
Overview Process Configuration Process Specification Work Item Categories Releases
<No Current Work>

AccountOpening

Summary: Account Opening application project (com.ibm.ao)

Description:

Corporate IT is chartered to deliver new business optimization programs. The Credit Application is an existing solution where new process automation components to be added.

The Credit Application program is delivering a complete business solution. The solution parts, implementing a SOA architecture, run on a variety of platforms. Client and server application parts are Web UI, Java Portal and Enterprise Server based. Back office application parts are System z modernized EGL applications.

The solution parts will be developed by a group of projects in the coming years. The Account Opening (AO) team is a part of the Credit Application solution delivering business automation to the Credit Check business process. AO team deliver both a new development project (Rel2) and a maintenance project (Rel1Maint) on the release (Rel1) that is currently in production.

OpenUp Process The OpenUp process is one of t processes created by the eclipse process framew project (EPF). It is included here as one example how to integrate existing processes. For more de see http://www.eclipse.org/epf

Project area administrators have administrative privileges outside of the permissions defined by the process. This applies also to all team areas. When you don't have the permissions you need, please contact one of the administrators below.

Name	e-mail
ADMIN	admin@ao.ibm.com
Marco DevLead	marco@ao.ibm.com
Patricia ProjectMa...	patricia@ao.ibm.com

Figure 7-4 The CreditCheck Team Area in Rational Team Concert

Agile planning

Agile planning principles dictate that “just enough” and “live” planning are to be conducted in order to effectively deliver working software to stakeholders. Teams should not sidetrack with the production of expansive project plans in separate tools or documentation that does not contribute to project delivery. Planning must be directly tied to project execution to become an enabler to the team members and not a burden. This principle applies both at the team level and agile iteration planning, as well as the self-configuring scheduling of work done by each team member.

In collaborative development, planning is an activity that involves the entire team, not only the project manager. The team consumes the release requirements and drives out the scope of the iteration by using stories or themes. The plan realization of the stories becomes live tasks for the team members. Each team member takes ownership of these tasks, or work items, and configures their schedule and work estimations. This collaborative effort makes the teams self-configure their plans.

By the seamless integration of the planning and the change management in Rational Team Concert, teams can benefit from significant tool automation of the agile collaborative development principles. In Rational Team Concert, the tasks of the project plan are the same as the actual living tasks that the owners are working against. This method alleviates the burden of project plan updates from the project manager and gives everyone full real-time transparency into the actual progress of tasks against the plan.

The following artifacts are used for agile planning in Rational Team Concert (see Figure 7-4 on page 236):

- Development line

  A *development line* represents an area of activity within a project that typically has its own objectives, deliverables, team, process, and schedule. Project intervals or phases are defined within a development line and are expressed as a hierarchy of iterations.

- Iteration

  Iterations are defined within a development line and are expressed as a hierarchy of project intervals. Iteration plans are used to express the details of an iteration in terms of work items.

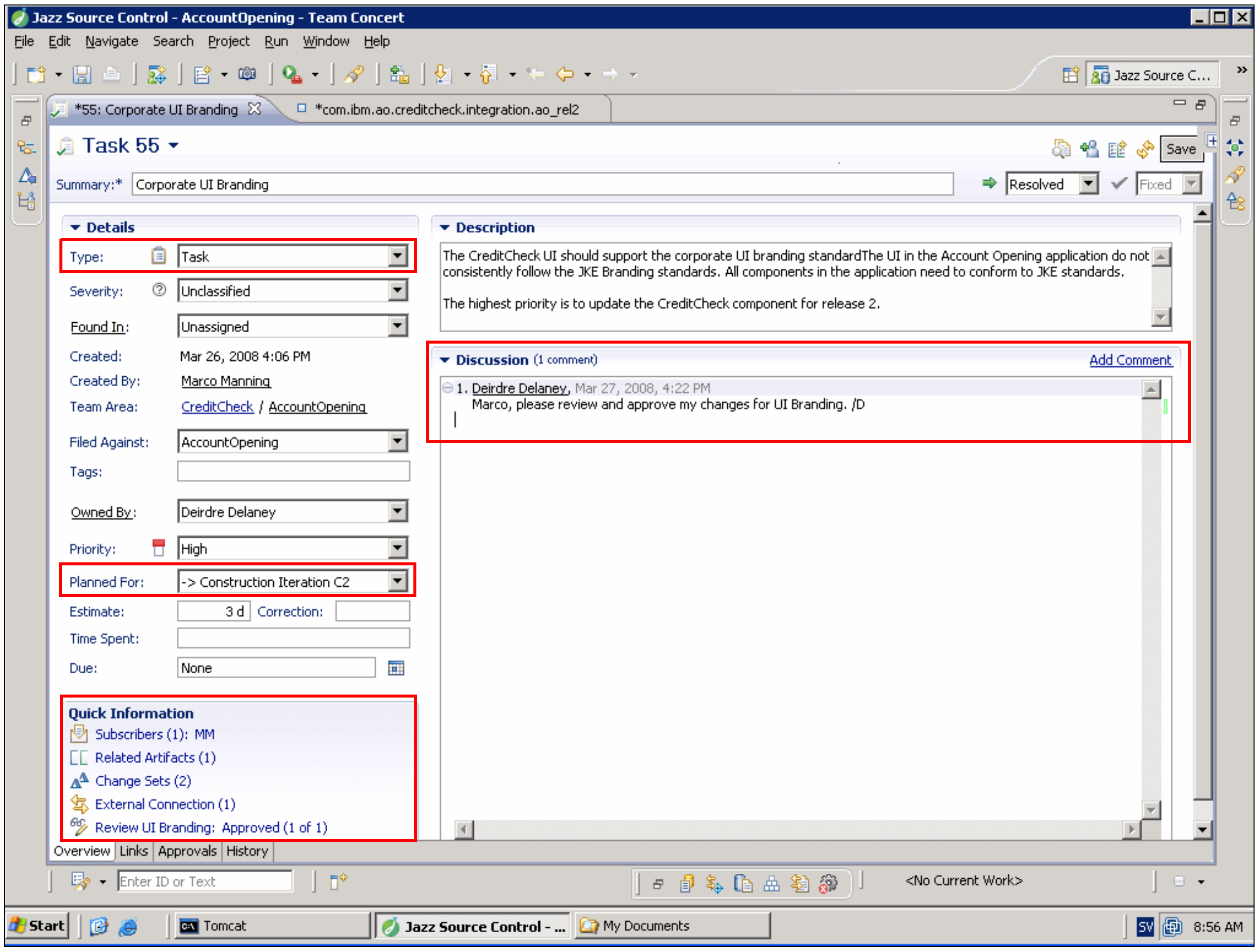
- Work item

  A *work item* is a representation of the tasks and issues that a team needs to address during the development cycle. Work items are key for iteration planning and indicators of the health of the project.

Work items

Work items enable team members to easily see and track work that is assigned to them and defects that are submitted against the components for which they are responsible. The team can use work items to plan development work for milestones and obtain metrics that indicate the progress made toward the project development and quality goals.

Figure 7-5 on page 238 shows an example of a work item in Rational Team Concert. This figure also shows the UI Branding work item that is assigned to Diedrie. Some key work item properties are highlighted, such as the work item type, the iteration plan specifying when work is scheduled, the traceability to other life-cycle collaboration artifacts, and the discussions with other team members.


Jazz Source Control - AccountOpening - Team Concert
File Edit Navigate Search Project Run Window Help
Jazz Source C...
*55: Corporate UI Branding
*com.ibm.ao.creditcheck.integration.ao_rel2
Save
Overview Links Approvals History
Enter ID or Text
<No Current Work>
Start
Tomcat
Jazz Source Control - ...
My Documents
8:56 AM

Task 55

Summary:* Corporate UI Branding

Resolved | Fixed

Details

Type:	Task
Severity:	Unclassified
Found In:	Unassigned
Created:	Mar 26, 2008 4:06 PM
Created By:	Marco Manning
Team Area:	CreditCheck / AccountOpening
Filed Against:	AccountOpening
Tags:	
Owned By:	Deirdre Delaney
Priority:	High
Planned For:	-> Construction Iteration C2
Estimate:	3 d Correction:
Time Spent:	
Due:	None

Quick Information

- Subscribers (1): MM
- Related Artifacts (1)
- Change Sets (2)
- External Connection (1)
- Review UI Branding: Approved (1 of 1)

Description

The CreditCheck UI should support the corporate UI branding standardThe UI in the Account Opening application do not consistently follow the JKE Branding standards. All components in the application need to conform to JKE standards.

The highest priority is to update the CreditCheck component for release 2.

Discussion (1 comment) Add Comment

1. Deirdre Delaney, Mar 27, 2008, 4:22 PM
Marco, please review and approve my changes for UI Branding. /D

Figure 7-5 Example of a work item in Rational Team Concert

Work items in Rational Team Concert are a generic concept that captures multiple change request types. For example, Rational Team Concert ships with the following types:

- Story
- Risk
- Task
- Enhancement
- Defect

The available set of work item types is defined in the development process. The default set of work items is provided by the process template that is chosen at project area creation time. Additional work item types can also be extended by editing the process definition.

Each work item type has a state transition model that defines the states that the work item represents and the actions that users take to move the work item from one state to another. A typical state transition model provides a path from a state of New, to In progress, to Verified, and to Closed.

Just as work items are seamlessly integrated with iteration planning, they also integrate with the other key ALM workflows. That is configuration management, build management, and reporting.

Work item integration with configuration management enables change sets to be associated with a work item and to capture the traceability from a request to the delivery of the resulting change.

Furthermore, work item integration with build management enables the tracking of the work that was completed and delivered into a build, the builds that contain the work item, and the build that the work item is reported against. From the build item, it is also possible to create a new defect and associate the work item with the build. All of these capabilities are key enablers for quality teams in configuring and executing their test plans.

Finally, the capability to report on work item metrics gives the team a wide range of opportunities to characterize and act on project health.

Source configuration management

Teams organize their source code, documents, and other artifacts by using the source control capabilities. The source control management capabilities in Rational Team Concert provide change-flow management to facilitate the sharing of controlled artifacts, retain a history of changes made to these artifacts, and enable simultaneous development of multiple versions of shared artifacts. By doing so, teams can work on several development lines at the same time.

The Rational Team Concert tool uses an SCM model that aligns in concept with earlier Rational solutions. The following key ALM assets are used for configuration management:

- Component

  Artifacts under source control are grouped into components. Therefore, a *component* is a group of versionable artifacts, such as one or more folders and files, that share a common root.

- Stream

  A *stream* is a collection of one or more components. Streams are somewhat analogous to the branches that are found in other source control management systems, but considerably more powerful. Any component in a repository can be included in zero or more streams. A stream represents a single history line of changes for a set of components. That is, the version history of change sets that are delivered to the stream. By using multiple streams, a development organization can work in parallel on projects that use different versions, or a history of change sets, of the same components.

- Change set

  A *change set* is an object that collects a related group of changes to the contents of one or many files.

- Baseline

  A *baseline* is an object that saves the state of a component and provides exactly one consistent version of every artifact in the component. A baseline consists of one or more change sets. Any baseline can be delivered or accepted, which effectively delivers or accepts all of the change sets in it. A baseline saves the state of a component in a workspace so that you can restore that state when needed.

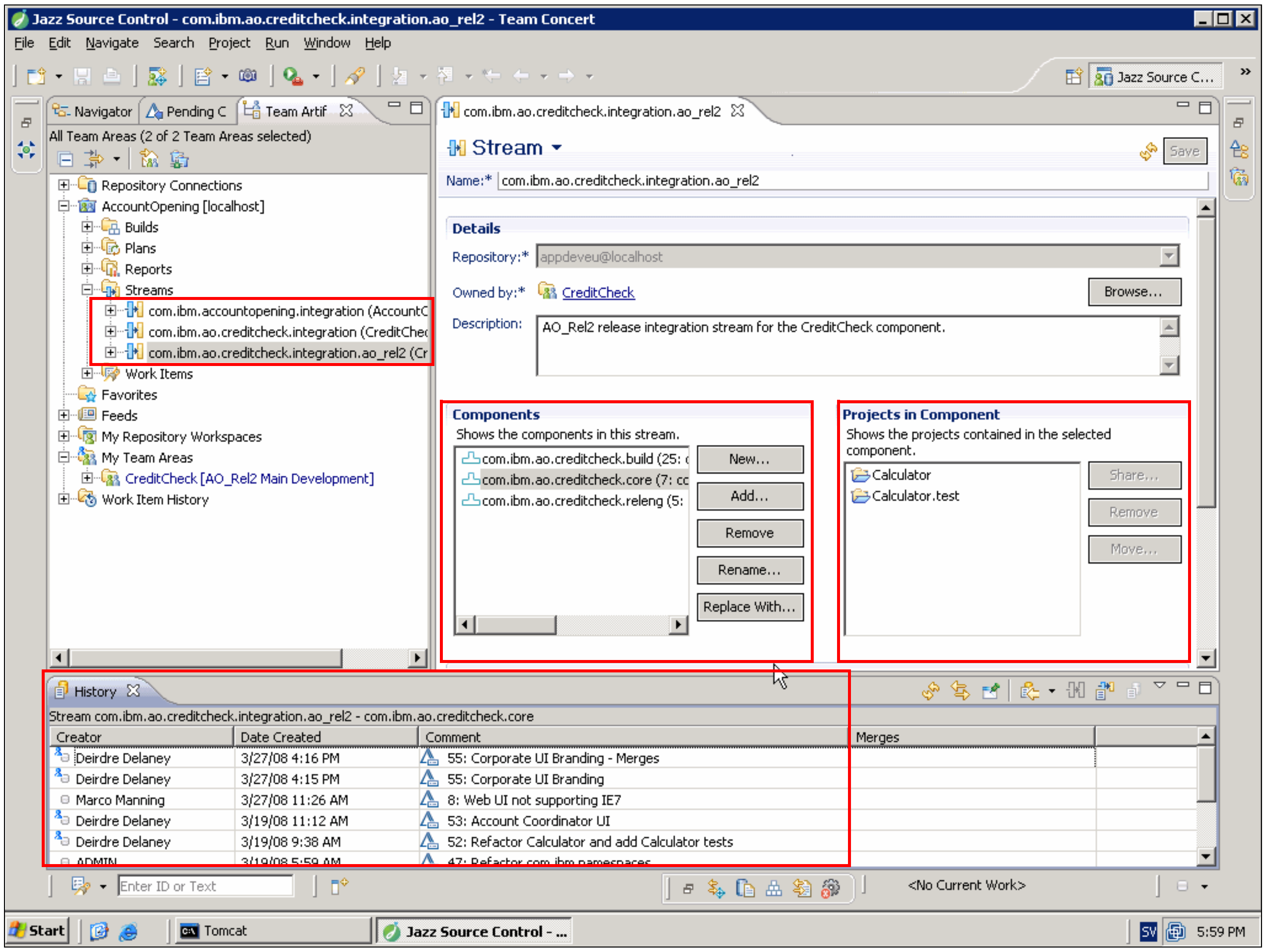
- Snapshot

  A *snapshot* includes one baseline for each component in a workspace or stream.

- Workspace

  A *workspace* is an area where team members view and modify components and their contained versionable artifacts. Workspaces are kept in the server store and can be loaded into a local workspace.

Figure 7-6 shows the AO_Rel2 Integration stream, which governs code changes for the next release that the component team is working on. This figure shows such key stream concepts as the streams that are used by the team, the SCM component contained in the stream, the Eclipse projects contained per SCM component, and the history of change sets that are delivered to the stream.


Jazz Source Control - com.ibm.ao.creditcheck.integration.ao_rel2 - Team Concert

File Edit Navigate Search Project Run Window Help

Navigator | Pending C | Team Artif

All Team Areas (2 of 2 Team Areas selected)

- Repository Connections
- AccountOpening [localhost]
  - Builds
  - Plans
  - Reports
  - Streams
    - com.ibm.accountopening.integration (AccountC
    - com.ibm.ao.creditcheck.integration (CreditChe
    - com.ibm.ao.creditcheck.integration.ao_rel2 (Cr
  - Work Items
- Favorites
- Feeds
- My Repository Workspaces
- My Team Areas
  - CreditCheck [AO_Rel2 Main Development]
- Work Item History

com.ibm.ao.creditcheck.integration.ao_rel2

Stream

Name:* com.ibm.ao.creditcheck.integration.ao_rel2

Details

Repository:* appdeveu@localhost

Owned by:* CreditCheck

Description: AO_Rel2 release integration stream for the CreditCheck component.

Components

Shows the components in this stream.

- com.ibm.ao.creditcheck.build (25: c
- com.ibm.ao.creditcheck.core (7: cc
- com.ibm.ao.creditcheck.releng (5:

Projects in Component

Shows the projects contained in the selected component.

- Calculator
- Calculator.test

History

Stream com.ibm.ao.creditcheck.integration.ao_rel2 - com.ibm.ao.creditcheck.core

Creator	Date Created	Comment	Merges
Deirdre Delaney	3/27/08 4:16 PM	55: Corporate UI Branding - Merges	
Deirdre Delaney	3/27/08 4:15 PM	55: Corporate UI Branding	
Marco Manning	3/27/08 11:26 AM	8: Web UI not supporting IE7	
Deirdre Delaney	3/19/08 11:12 AM	53: Account Coordinator UI	
Deirdre Delaney	3/19/08 9:38 AM	52: Refactor Calculator and add Calculator tests	
ADMIN	3/19/08 5:59 AM	47: Refactor com.ibm.namespaces	

<No Current Work>

Figure 7-6 The AO_Rel2 integration stream in the CreditCheck component

Team build

The build management capabilities in Rational Team Concert are based on the Jazz platform support for the automation, monitoring, and notification of the team's builds. These capabilities support a model that represents the team's build definitions, build engines, and build results. The model is designed to support teams by using a range of different build technologies and is configured by each team to fit their build process requirements.

The following artifacts are used for build management:

- Build engine

  A build engine represents a build system that runs on a build server.

- Build definition

  The build definition contains the details on the build and defines a particular build, such as a weekly project-wide integration build.

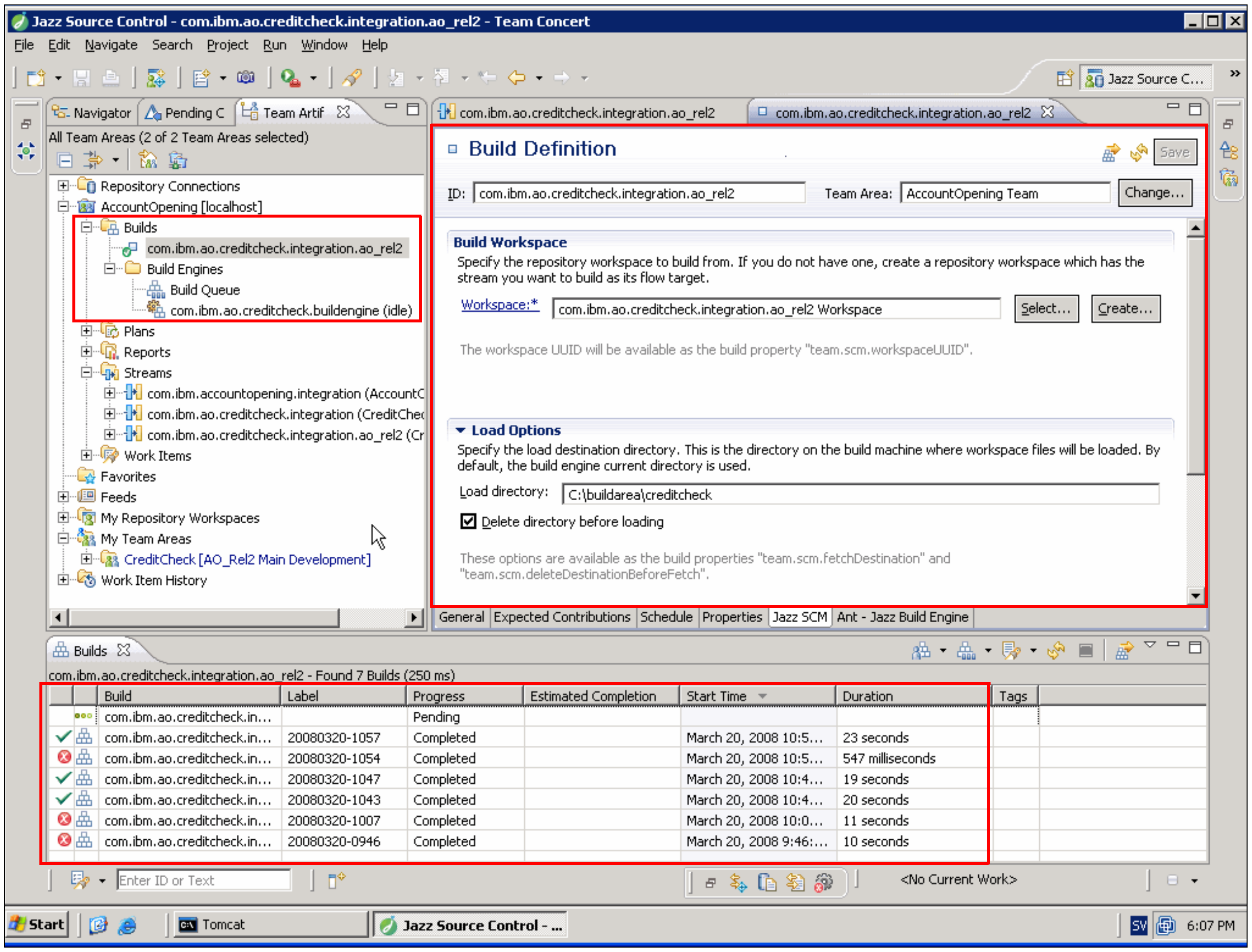
- Build script

  The build.xml script performs the build and is referenced by the build definition.

- Build result

  The build result represents the outputs from a particular run that might contain, for example, compilation, build validation tests, code analysis, and release asset creation.

Figure 7-7 shows the integration build definition that is used by the Credit Check team. This figure shows such key build concepts as the build definitions and build engines that are used by the team, the AO_Rel2 build definition, and the history of the build results for the AO_Rel2 integration and the build validation runs.



Jazz Source Control - com.ibm.ao.creditcheck.integration.ao_rel2 - Team Concert

All Team Areas (2 of 2 Team Areas selected)

- Repository Connections
- AccountOpening [localhost]
  - Builds
    - com.ibm.ao.creditcheck.integration.ao_rel2
    - Build Engines
      - Build Queue
      - com.ibm.ao.creditcheck.buildengine (idle)
  - Plans
  - Reports
  - Streams
    - com.ibm.accountopening.integration (AccountC
    - com.ibm.ao.creditcheck.integration (CreditChec
    - com.ibm.ao.creditcheck.integration.ao_rel2 (Cr
  - Work Items
- Favorites
- Feeds
- My Repository Workspaces
- My Team Areas
  - CreditCheck [AO_Rel2 Main Development]
- Work Item History

Build Definition

ID: com.ibm.ao.creditcheck.integration.ao_rel2 Team Area: AccountOpening Team

Build Workspace

Specify the repository workspace to build from. If you do not have one, create a repository workspace which has the stream you want to build as its flow target.

Workspace:* com.ibm.ao.creditcheck.integration.ao_rel2 Workspace

The workspace UUID will be available as the build property "team.scm.workspaceUUID".

Load Options

Specify the load destination directory. This is the directory on the build machine where workspace files will be loaded. By default, the build engine current directory is used.

Load directory: C:\buildarea\creditcheck

[✓] Delete directory before loading

These options are available as the build properties "team.scm.fetchDestination" and "team.scm.deleteDestinationBeforeFetch".

com.ibm.ao.creditcheck.integration.ao_rel2 - Found 7 Builds (250 ms)

Build	Label	Progress	Estimated Completion	Start Time	Duration	Tags
com.ibm.ao.creditcheck.in...		Pending				
✓ com.ibm.ao.creditcheck.in...	20080320-1057	Completed		March 20, 2008 10:5...	23 seconds	
✗ com.ibm.ao.creditcheck.in...	20080320-1054	Completed		March 20, 2008 10:5...	547 milliseconds	
✓ com.ibm.ao.creditcheck.in...	20080320-1047	Completed		March 20, 2008 10:4...	19 seconds	
✓ com.ibm.ao.creditcheck.in...	20080320-1043	Completed		March 20, 2008 10:4...	20 seconds	
✗ com.ibm.ao.creditcheck.in...	20080320-1007	Completed		March 20, 2008 10:0...	11 seconds	
✗ com.ibm.ao.creditcheck.in...	20080320-0946	Completed		March 20, 2008 9:46:...	10 seconds	

Figure 7-7 Build definition used by the Credit Check team for continuous component integration and validation

The team build capability in Rational Team Concert can be integrated with Enterprise Build solutions by using Rational Build Forge.

Process awareness

The team process capabilities help teams to consistently deploy and execute the processes that they have agreed upon. The team process does not define the process. It is process neutral. However, it maintains a constant presence of the process in Rational Team Concert and encourages or enforces the processes that the team has adopted.

The team process is chosen when a new project is defined. Rational Team Concert includes both sample and production processes, such as the Open Unified Process (OpenUp) and Eclipse Way processes, that can be deployed as is or customized to suit the needs of a project.

The team process governs all activities, artifacts, artifact relationships, and operations that are pursued inside the associated project area. A process is defined by a process specification and an iteration structure. Both are stored in the project area. The process specification describes the roles that team members can play and the process rules that apply to each role within the several iterations.

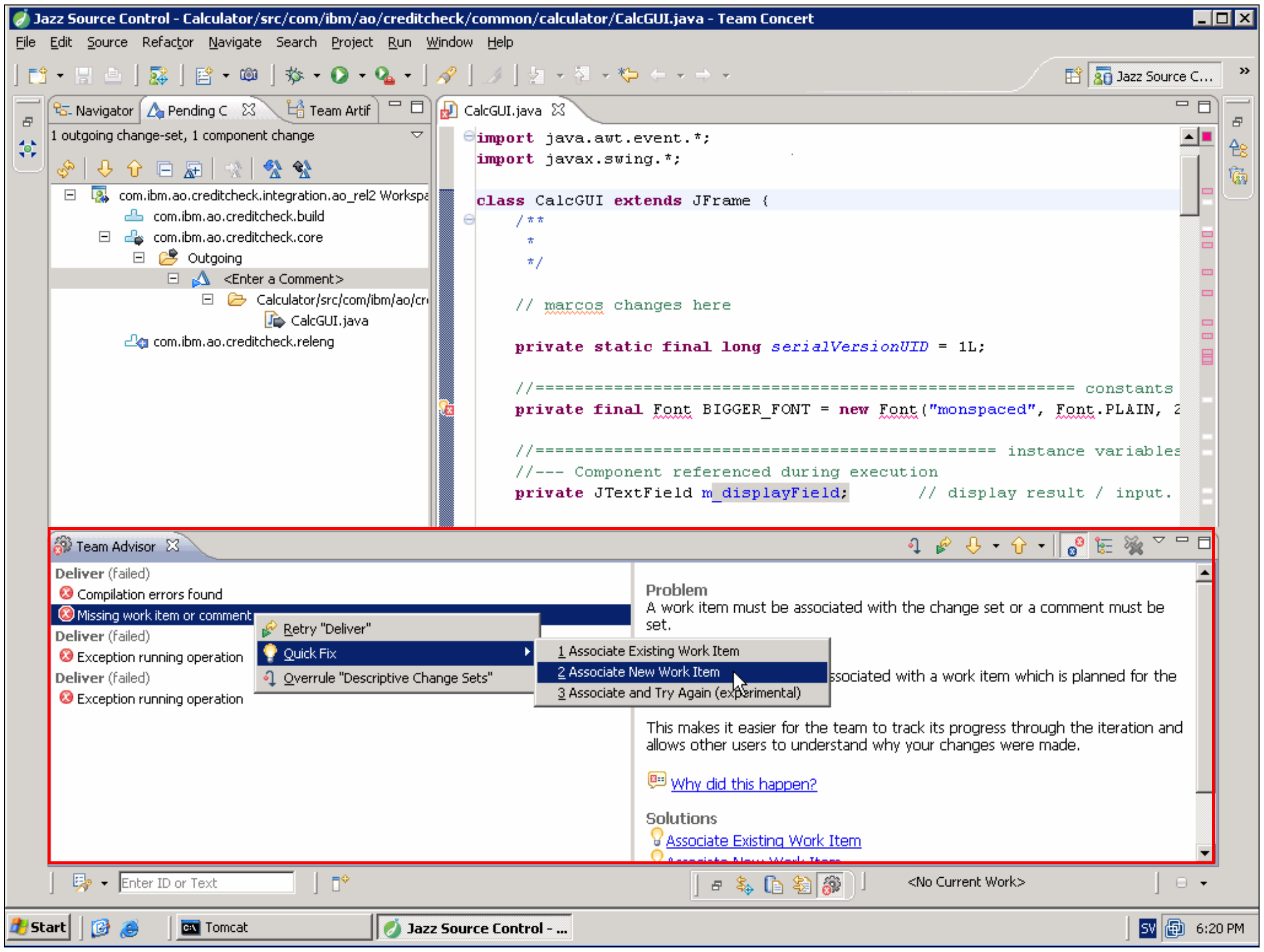
Rational Team Concert components and capabilities are process enabled. A *process-enabled tool component* is one that is designed to play well in a world where a process known to the system governs the work that is being done.

For example, the Eclipse Way process enforcement does not allow the following actions to occur:

- Delivery of code with compilation errors
- Delivery of a source file without a copyright notice
- Work item creation without an assigned category (“filed against”)
- Delivery of a change set without an associated work item or comment

The process specification also defines which of the core process rules on the Jazz platform can further be customized.

Figure 7-8 on page 243 shows an example of process awareness in Rational Team Concert. The highlighted section exemplifies several process breakages, such as an attempt to deliver code with compilation errors or an attempt to deliver a change set without a work item associated. This figure also exemplifies how Rational Team Concert provides guidance on the root cause of the process violation, as well as Quick Fix options to proceed.


Jazz Source Control - Calculator/src/com/ibm/ao/creditcheck/common/calculator/CalcGUI.java - Team Concert
File Edit Source Refactor Navigate Search Project Run Window Help
Navigator
Pending C
Team Artif
1 outgoing change-set, 1 component change
com.ibm.ao.creditcheck.integration.ao_rel2 Workspa
com.ibm.ao.creditcheck.build
com.ibm.ao.creditcheck.core
Outgoing
<Enter a Comment>
Calculator/src/com/ibm/ao/cr
CalcGUI.java
com.ibm.ao.creditcheck.releng
CalcGUI.java
Team Advisor
Retry "Deliver"
Quick Fix
Overrule "Descriptive Change Sets"
1 Associate Existing Work Item
2 Associate New Work Item
3 Associate and Try Again (experimental)
Enter ID or Text
<No Current Work>
Start
Tomcat
Jazz Source Control - ...
6:20 PM

```
import java.awt.event.*;
import javax.swing.*;

class CalcGUI extends JFrame {
    /**
     *
     */

    // marcos changes here

    private static final long serialVersionUID = 1L;

    //========================================================= constants
    private final Font BIGGER_FONT = new Font("monspaced", Font.PLAIN, 2

    //============================================== instance variables
    //--- Component referenced during execution
    private JTextField m_displayField;       // display result / input.
```

Deliver (failed)
Compilation errors found
Missing work item or comment
Deliver (failed)
Exception running operation
Deliver (failed)
Exception running operation

Problem
A work item must be associated with the change set or a comment must be set.

ssociated with a work item which is planned for the

This makes it easier for the team to track its progress through the iteration and allows other users to understand why your changes were made.

Why did this happen?

Solutions
Associate Existing Work Item

Figure 7-8 Example of process awareness in Rational Team Concert

Team collaboration

Rational Team Concert provides several core team collaboration capabilities that are built into the Jazz platform to support the agile principles of collaborative development.

Team collaboration includes the following capabilities among others:

- Transparency and team awareness

  Rational Team Concert brings teams together and encourages team responsibility and personal accountability. The collective effort and workload of the entire team is an overall theme of most tool views and allows awareness the team's workload and the personal workload of each team member. The Team Central view gives transparency to the status and recent events from the team progress. The My Work views give awareness of the personal schedule and current priorities.

- Dashboards and other real-time views

  The Web Dashboard and Team Central view are two examples of a customizable real-time view of what is happening in the project. It provides a transparent view into the work of the entire project team. It also enables transparency and collaboration for external team contributors.

- Notifications

  The tool components in Rational Team Concert support event notification where team members can subscribe to changes in specific items, but also to more general event types. Notification enables the team to more quickly react and, therefore, streamlines the development workflows. The events also let team members make constant updates without interrupting their ongoing work.

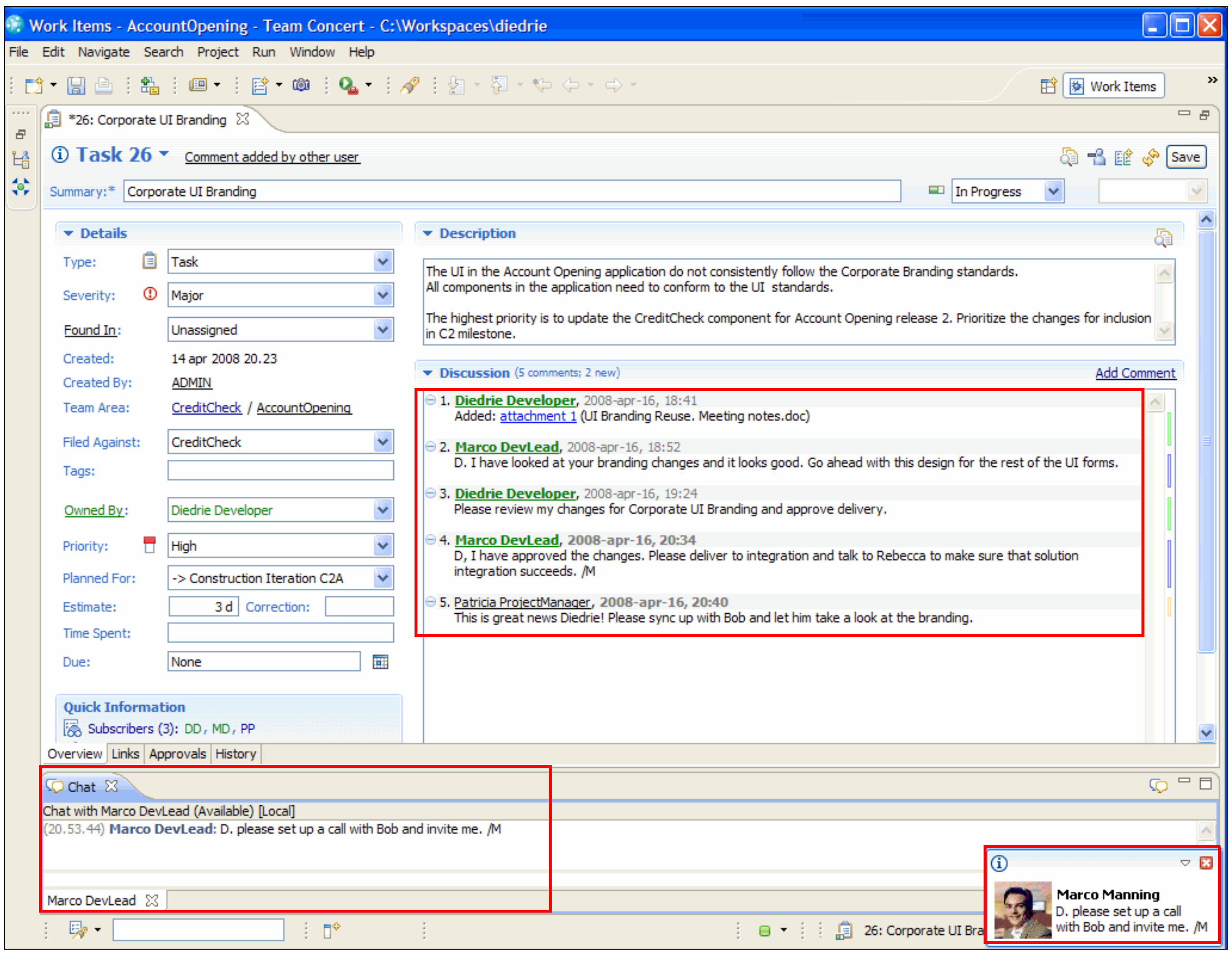
- Contextual collaboration

  Collaboration with team members, regardless of their geographic location, is a key success factor for distributed development. In addition, the ability to persist a discussion in context of the topic of the discussion is key for teams to record discussions for later reference by others on the team. This capability is highly important for agile teams to re-balance workload and for distributed teams to share work cross geographic boundaries.

- Presence awareness

  Rational Team Concert supports integration with instant messaging and provides access to other team members at their fingertips. The seamless integration allows references to artifacts, such as work items or change set references, to be easily included in a chat as clickable links.

Figure 7-9 shows an example of the team collaboration capabilities in Rational Team Concert. The highlighted section exemplifies the in-context discussion threads of a work item, the integrated chat view, and a fly-in notification that contains team events, such as Marco's new instant message.


Work Items - AccountOpening - Team Concert - C:\Workspaces\diedrie
File Edit Navigate Search Project Run Window Help
Work Items
*26: Corporate UI Branding
Save
Overview Links Approvals History
Chat
Marco DevLead
26: Corporate UI Bra

Task 26 Comment added by other user

Summary:* Corporate UI Branding — In Progress

Details

Type:	Task
Severity:	Major
Found In:	Unassigned
Created:	14 apr 2008 20.23
Created By:	ADMIN
Team Area:	CreditCheck / AccountOpening
Filed Against:	CreditCheck
Tags:	
Owned By:	Diedrie Developer
Priority:	High
Planned For:	-> Construction Iteration C2A
Estimate:	3 d Correction:
Time Spent:	
Due:	None

Quick Information

Subscribers (3): DD, MD, PP

Description

The UI in the Account Opening application do not consistently follow the Corporate Branding standards.
All components in the application need to conform to the UI standards.

The highest priority is to update the CreditCheck component for Account Opening release 2. Prioritize the changes for inclusion in C2 milestone.

Discussion (5 comments; 2 new) Add Comment

1. Diedrie Developer, 2008-apr-16, 18:41
   Added: attachment 1 (UI Branding Reuse. Meeting notes.doc)
2. Marco DevLead, 2008-apr-16, 18:52
   D. I have looked at your branding changes and it looks good. Go ahead with this design for the rest of the UI forms.
3. Diedrie Developer, 2008-apr-16, 19:24
   Please review my changes for Corporate UI Branding and approve delivery.
4. Marco DevLead, 2008-apr-16, 20:34
   D, I have approved the changes. Please deliver to integration and talk to Rebecca to make sure that solution integration succeeds. /M
5. Patricia ProjectManager, 2008-apr-16, 20:40
   This is great news Diedrie! Please sync up with Bob and let him take a look at the branding.

Chat with Marco DevLead (Available) [Local]
(20.53.44) Marco DevLead: D. please set up a call with Bob and invite me. /M

Marco Manning
D. please set up a call with Bob and invite me. /M

Figure 7-9 Team collaboration capabilities in Rational Team Concert

Artifact traceability

In this section, we introduce the artifact traceability capabilities in Rational Team Concert. Rational Team Concert provides a seamless experience where traceability across the application lifecycle is provided and managed by the Jazz platform. Because this is a core capability of the platform, we can only take a few examples of how this surfaces in the Rational Team Concert user interface.

Work items are the entry point for multiple traceability links as in the following examples:

- To the iteration for which the work is planned
- To related artifacts
- To a change set that contain the implementation of the change
- To external repositories that hold the original source of the defect or enhancement
- To approvals that are required to deliver the change for component or solution integration

Rational Team Concert creates and manages most of the traceability in the background as the developer conducts the regular collaborative development workflows. The artifact traceability capabilities are embedded in the Rational Team Concert user interface, as shown in Figure 7-5 on page 238 and in Figure 7-10, where the traceability links surface as clickable links the UI Branding work item.

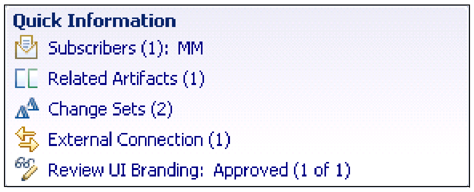

Quick Information
Subscribers (1): MM
Related Artifacts (1)
Change Sets (2)
External Connection (1)
Review UI Branding: Approved (1 of 1)

Figure 7-10 The traceability links presented on a work item

7.3 Rational Team Concert for agile development

In this section, we provide detailed information about how the agile component team uses Rational Team Concert for collaborative development as illustrated in Figure 7-11.

Synopsis: Act 2 of the story demonstrates how Diedrie uses Rational Team Concert to design, develop, test, integrate, and deliver the changes required by the UI Branding work item that she has the responsibility to deliver in the current iteration. This act also demonstrates how Diedrie collaborates with Al to find a reusable asset and how Marco reviews and approves her design and code changes. The final scene in the act shows how Diedrie delivers her changes to integration and monitors the build process.

The story in Act 2 is described in further detail in 6.2, “A reference scenario for collaborative development” on page 221.

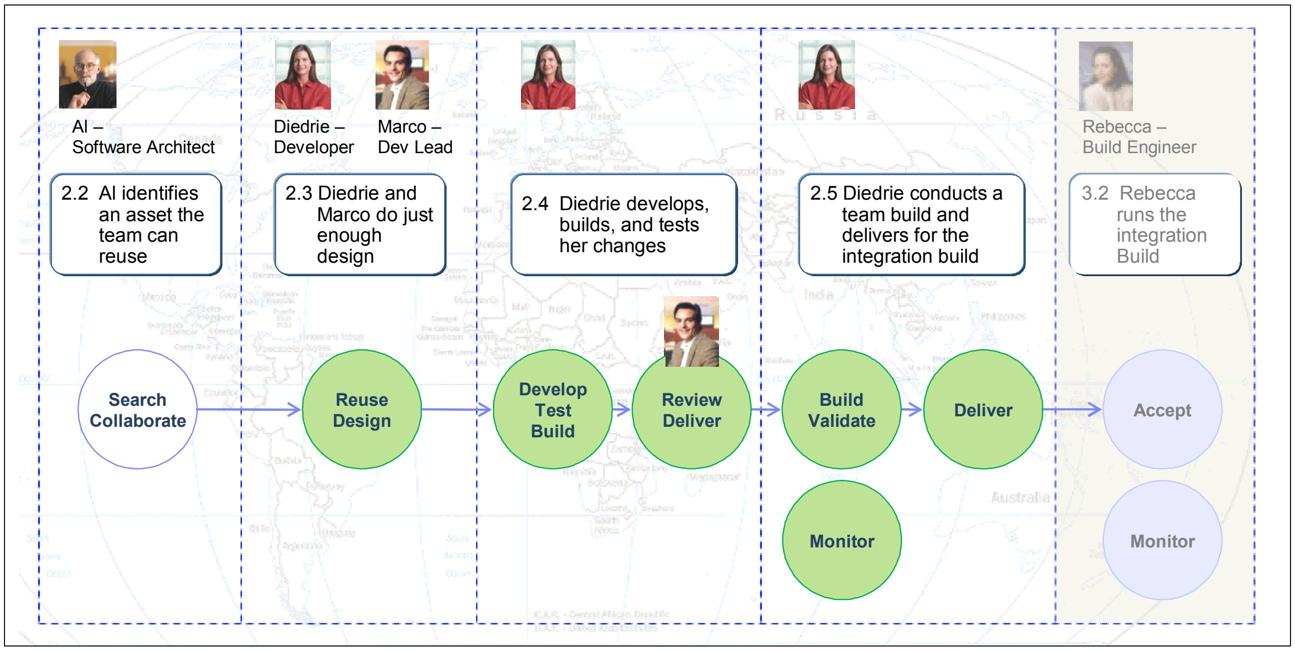

Al – Software Architect
Diedrie – Developer
Marco – Dev Lead
Rebecca – Build Engineer
2.2 Al identifies an asset the team can reuse
2.3 Diedrie and Marco do just enough design
2.4 Diedrie develops, builds, and tests her changes
2.5 Diedrie conducts a team build and delivers for the integration build
3.2 Rebecca runs the integration Build
Search Collaborate
Reuse Design
Develop Test Build
Review Deliver
Build Validate
Deliver
Accept
Monitor
Monitor

Figure 7-11 The collaborative development workflow in Act 2

The usage of Rational Team Concert in Act 2 creates several new artifacts and establishes traceability to support the life-cycle collaboration. Figure 7-12 shows the life-cycle collaboration artifacts that are established at the end of this act. See 7.4, “Life-cycle collaboration” on page 286, which describe the assets and relationships in Figure 7-12 in greater detail.

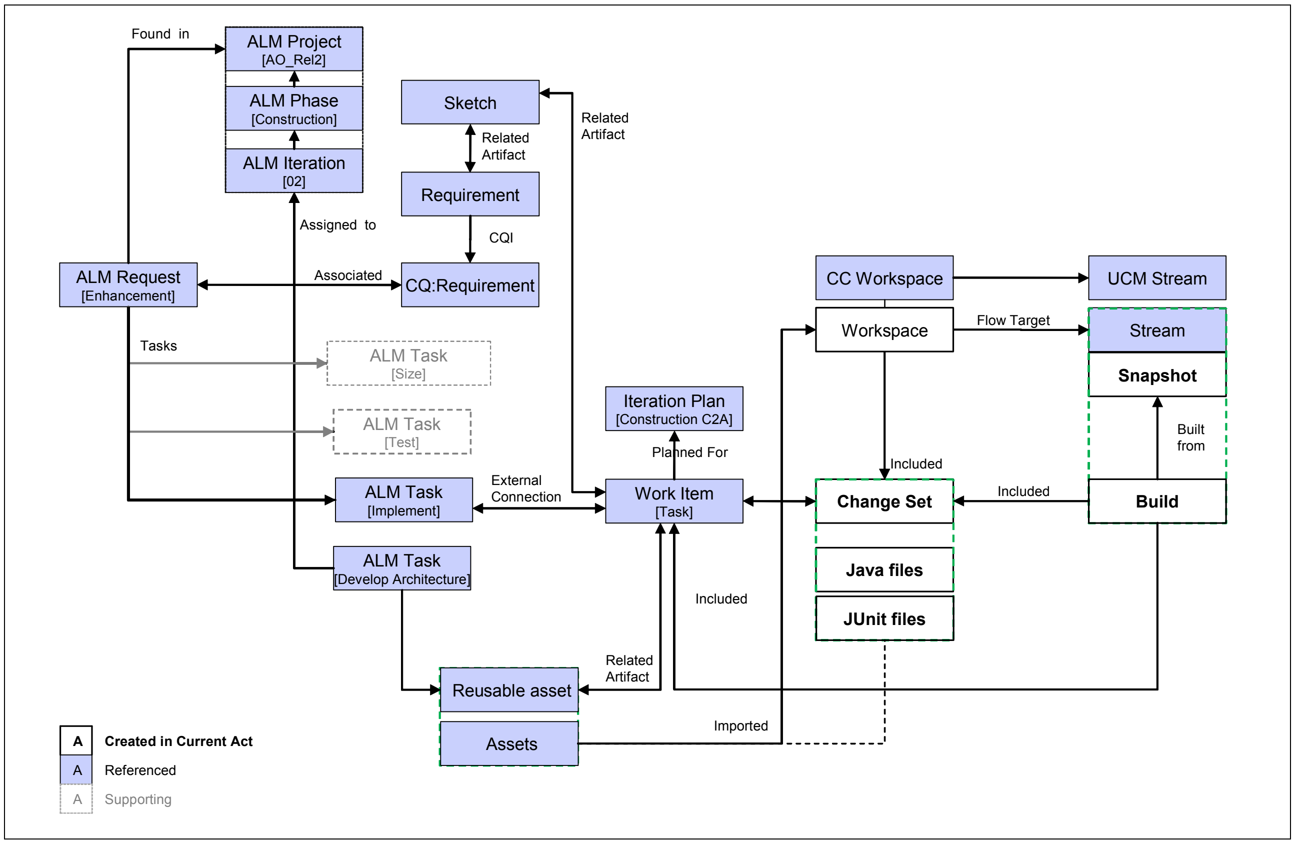


Figure 7-12 Life-cycle collaboration artifacts established in collaborative development

7.3.1 Marco monitors component health

Synopsis: Marco conducts daily stand-up meetings with his component team to align the work for the day. The team talks about their work and deliveries in context of their live iteration plan. Marco lets each team member have the opportunity to collaborate with the rest of the team on work progress, change dependencies, and blocking issues.

During the daily stand-up meetings, the team uses the live iteration plan to make tactical changes to improve project health. New work items are triaged or added. Ownership and priorities or work is clarified. Blocking issues are reprioritized.

Diedrie has been assigned the work to re-brand the Credit Check user interface and make other changes as well. She estimates and schedules the work assigned to her for the current iteration.

The step in this scene entails holding daily stand-up meetings.

Holding daily stand-up meetings

Goal: The goal is to hold daily stand-ups to let the team self-configure their plans based on component health.

Marco and his team have been successful in using an agile way of working for many projects. Recently, when the smaller company was acquired by the enterprise, changes were made to the demands on their development processes. However, the component team seeks to maintain their agile development style while working in the context of the larger solution team. In fact, the previous successes from the smaller company have resulted in the adoption of a more Agility-at-Scale approach by Patricia and the larger solution project team.

For the most part, Marco and his component team are co-located, but from time to time, team members work from remote locations or from home. Also, the component is integrated and tested in the larger solution. The team frequently interacts with other teams in the solution project team, for example the functional leads in the project or the solution test team.

The component team has agreed to meet every morning to align on the work of the day. They use a meeting room with a projector to display information in Rational Team Concert on the wall. Remote users access information in Rational Team Concert via the Web interface.

Marco tries to keep the stand-up meeting to a short 15-minute meeting. He knows that the team will benefit from four key topic areas:

- Project and component health
- Announcement of major work starting or to be delivered
- Resolution of any blocking items or dependencies
- Triage of selected work items

The team uses the Dashboard and Iteration Plan views in Rational Team Concert to support stand-up meeting discussions.

Rational Team Concert Dashboard

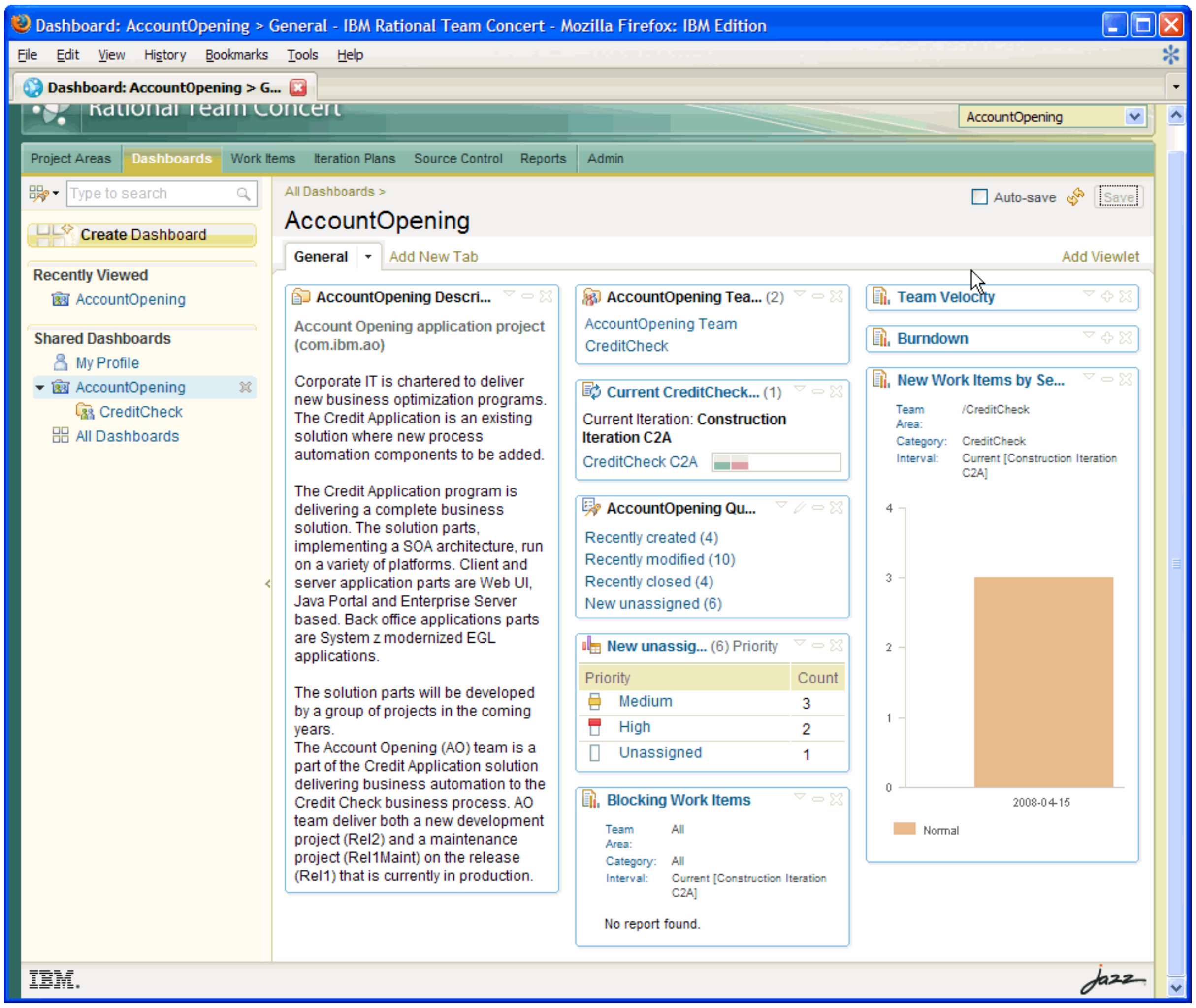
Goal: The goal is to make the dashboard a real-time view of the team health.

Marco and the team use the Web dashboard to access a live overview of the component health indicators. The dashboard helps the team identify significant changes in trends.

Marco discusses anomalies among the following trends with the team:

- The change rate; recently created, modified, and closed work items
- The incoming rate; new unassigned work items
- The project velocity and burndown
- Trends in build health

Figure 7-13 shows an example of the Rational Team Concert Dashboard. We describe how the team uses these metrics in 7.5, “Planning and measuring success in collaborative development” on page 288.


Dashboard: AccountOpening > General - IBM Rational Team Concert - Mozilla Firefox: IBM Edition
File Edit View History Bookmarks Tools Help
Dashboard: AccountOpening > G...
Rational Team Concert
AccountOpening
Project Areas Dashboards Work Items Iteration Plans Source Control Reports Admin
Type to search
Create Dashboard
Recently Viewed
AccountOpening
Shared Dashboards
My Profile
AccountOpening
CreditCheck
All Dashboards
All Dashboards >
Auto-save Save
General Add New Tab Add Viewlet
Team Velocity
Burndown
New Work Items by Se...
4
3
2
1
0
2008-04-15
Normal
IBM.
Jazz

AccountOpening

AccountOpening Descri...

Account Opening application project (com.ibm.ao)

Corporate IT is chartered to deliver new business optimization programs. The Credit Application is an existing solution where new process automation components to be added.

The Credit Application program is delivering a complete business solution. The solution parts, implementing a SOA architecture, run on a variety of platforms. Client and server application parts are Web UI, Java Portal and Enterprise Server based. Back office applications parts are System z modernized EGL applications.

The solution parts will be developed by a group of projects in the coming years.
The Account Opening (AO) team is a part of the Credit Application solution delivering business automation to the Credit Check business process. AO team deliver both a new development project (Rel2) and a maintenance project (Rel1Maint) on the release (Rel1) that is currently in production.

AccountOpening Tea... (2)

AccountOpening Team
CreditCheck

Current CreditCheck... (1)

Current Iteration: **Construction Iteration C2A**
CreditCheck C2A

AccountOpening Qu...

Recently created (4)
Recently modified (10)
Recently closed (4)
New unassigned (6)

New unassig... (6) Priority

Priority	Count
Medium	3
High	2
Unassigned	1

Blocking Work Items

Team Area:	All
Category:	All
Interval:	Current [Construction Iteration C2A]

No report found.

New Work Items by Se...

Team Area:	/CreditCheck
Category:	CreditCheck
Interval:	Current [Construction Iteration C2A]

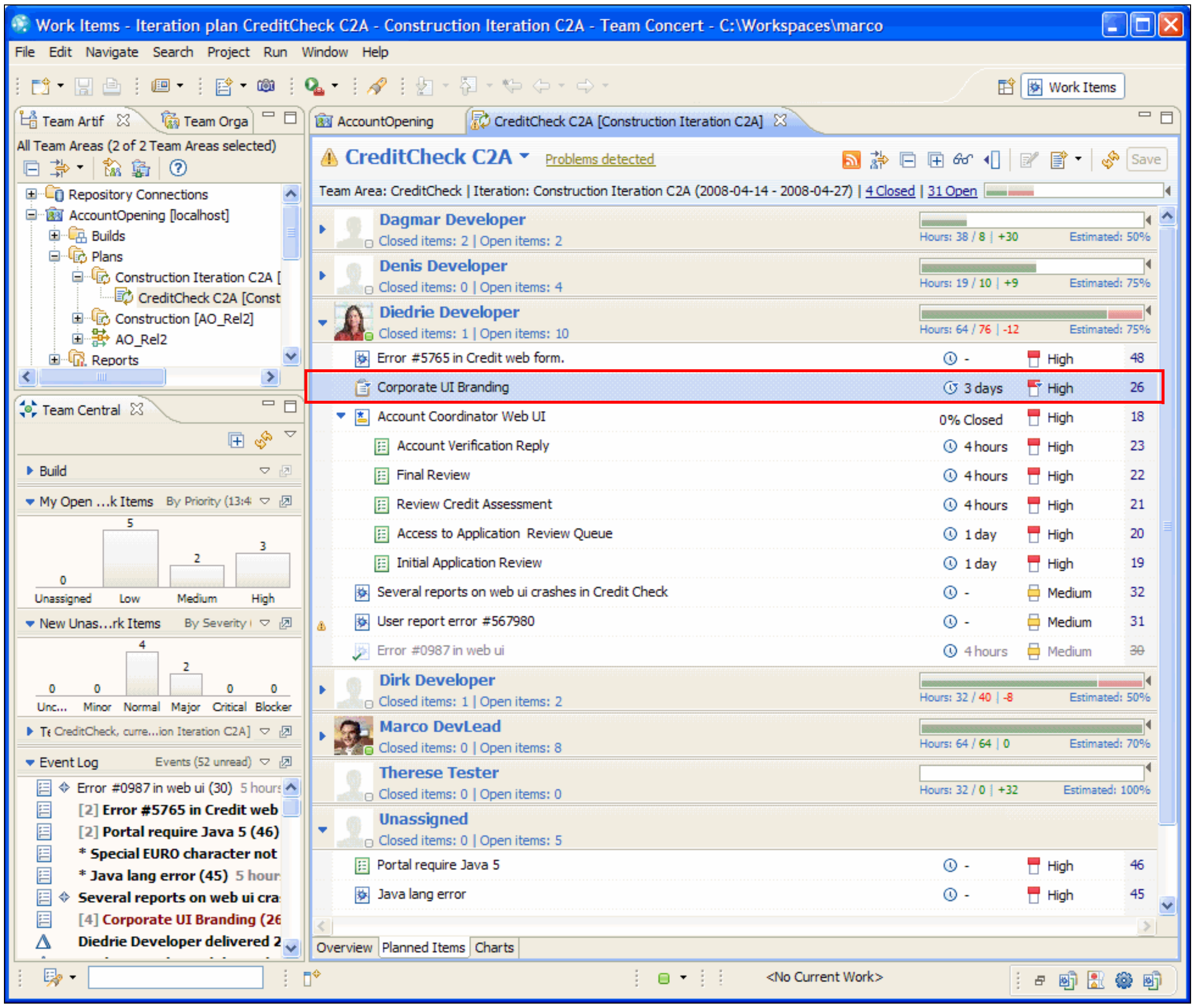
Figure 7-13 Rational Team Concert Dashboard for daily stand-up meetings to access project health indicators

Rational Team Concert iteration plan

Goal: The goal is to make the iteration plan a live document for the team to use for self-configuring their team and individual schedules.

Marco and the team use the iteration plan to display the details of the currently planned work for the team. The iteration plan is available in Rational Team Concert or on the Web. The seamless integration between the iteration plan and the work items, and the transparency to the actual work scheduling made by the team members, presents the team with accurate health indicators for the stand-up meeting. The team knows that the health indicators are real and not an outdated view from yesterday or even last week.

Figure 7-14 shows an example of the AO_Rel2 Construction C2A iteration plan that is used by the team.


Work Items - Iteration plan CreditCheck C2A - Construction Iteration C2A - Team Concert - C:\Workspaces\marco

File Edit Navigate Search Project Run Window Help

All Team Areas (2 of 2 Team Areas selected)

- Repository Connections
- AccountOpening [localhost]
  - Builds
  - Plans
    - Construction Iteration C2A [
      - CreditCheck C2A [Const
    - Construction [AO_Rel2]
    - AO_Rel2
  - Reports

Team Central

- Build
- My Open …k Items — By Priority (13:4

Unassigned	Low	Medium	High
0	5	2	3

- New Unas…rk Items — By Severity

Unc…	Minor	Normal	Major	Critical	Blocker
0	0	4	2	0	0

- Te CreditCheck, curre…ion Iteration C2A]
- Event Log — Events (52 unread)
  - Error #0987 in web ui (30) 5 hours
  - [2] Error #5765 in Credit web
  - [2] Portal require Java 5 (46)
  - * Special EURO character not
  - * Java lang error (45) 5 hour
  - Several reports on web ui cra
  - [4] Corporate UI Branding (26
  - Diedrie Developer delivered 2

CreditCheck C2A — Problems detected

Team Area: CreditCheck | Iteration: Construction Iteration C2A (2008-04-14 - 2008-04-27) | 4 Closed | 31 Open

Dagmar Developer — Closed items: 2 | Open items: 2 — Hours: 38 / 8 | +30 — Estimated: 50%

Denis Developer — Closed items: 0 | Open items: 4 — Hours: 19 / 10 | +9 — Estimated: 75%

Diedrie Developer — Closed items: 1 | Open items: 10 — Hours: 64 / 76 | -12 — Estimated: 75%

Error #5765 in Credit web form.	-	High	48
Corporate UI Branding	3 days	High	26
Account Coordinator Web UI	0% Closed	High	18
Account Verification Reply	4 hours	High	23
Final Review	4 hours	High	22
Review Credit Assessment	4 hours	High	21
Access to Application Review Queue	1 day	High	20
Initial Application Review	1 day	High	19
Several reports on web ui crashes in Credit Check	-	Medium	32
User report error #567980	-	Medium	31
Error #0987 in web ui	4 hours	Medium	~~30~~

Dirk Developer — Closed items: 1 | Open items: 2 — Hours: 32 / 40 | -8 — Estimated: 50%

Marco DevLead — Closed items: 0 | Open items: 8 — Hours: 64 / 64 | 0 — Estimated: 70%

Therese Tester — Closed items: 0 | Open items: 0 — Hours: 32 / 0 | +32 — Estimated: 100%

Unassigned — Closed items: 0 | Open items: 5

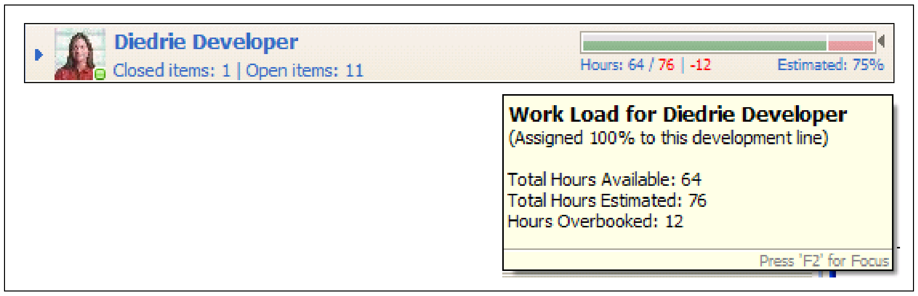
Portal require Java 5	-	High	46
Java lang error	-	High	45

Overview | Planned Items | Charts

<No Current Work>

Figure 7-14 Rational Team Concert iteration plan for daily stand-up meetings to access planned work items

Iteration plans in Rational Team Concert consolidate the plan information into plan health indicators for the entire team and for each individual team member. Figure 7-15 shows the details of such a plan. The health indicator shows how much of the available time for the iteration has been scheduled (horizontal bar) and the percentage of the work that has been estimated (vertical bar). These two health metrics are good indicators for work balancing, overcommit, and plan quality. Over time, the comparison of estimated and actual work can be used in retrospective to improve the planning capabilities of the team.


Diedrie Developer
Closed items: 1 | Open items: 11
Hours: 64 / 76 | -12
Estimated: 75%

Work Load for Diedrie Developer
(Assigned 100% to this development line)

Total Hours Available: 64
Total Hours Estimated: 76
Hours Overbooked: 12

Press 'F2' for Focus

Figure 7-15 Diedrie's plan health indicator for iteration C2A

As shown between Figure 7-14 on page 250 and Figure 7-15, Diedrie and the team still have some iteration planning to complete. For example, Diedrie has estimated 75% of her work, and the remaining 25% is adding risk and lowering the quality of the plan. She is also overcommitted by 12 hours of work. At the stand-up meeting, Marco requests that everyone in the team must complete their plans by the end of the day.

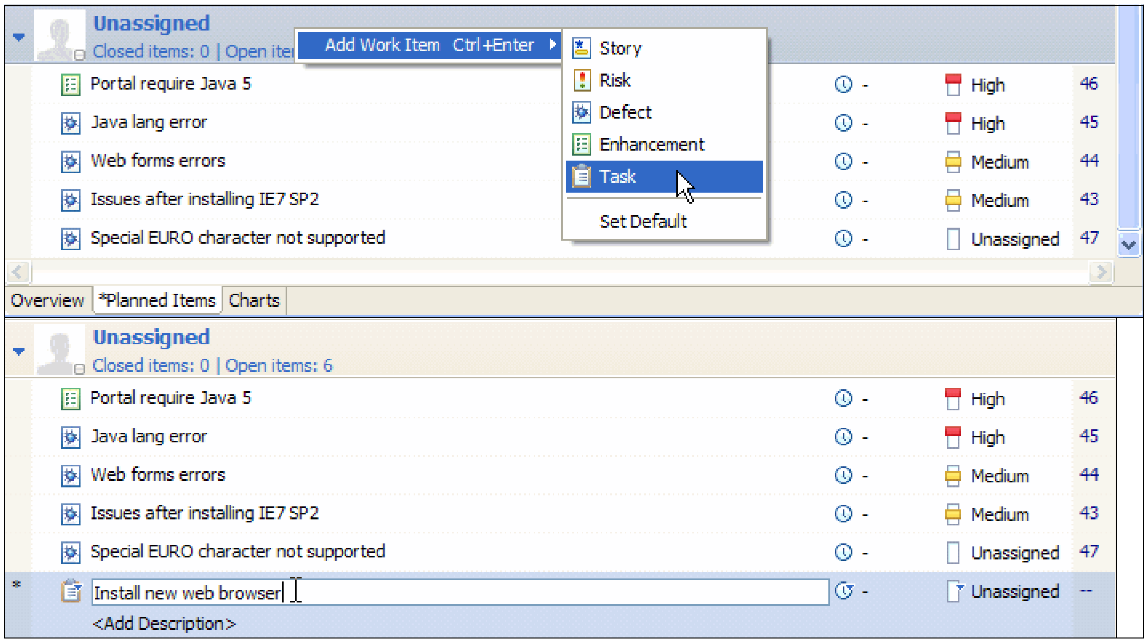
At the stand-up meeting, Diedrie announces that she will start working on the UI Branding work item of all CreditCheck Web forms, which is selected in Figure 7-14 on page 250. She intends to spend the day prototyping the changes and report the status tomorrow with a better understanding on any new design patterns to be used by the team. She intends to sync up with Al, the solution architect, regarding possible asset reuse. Diedrie also lets the team know that she is overbooked and asks that others on the team see which work items can be rebalanced.

The team triages the new incoming defects and decides who should take ownership. One of the team members commits to look at work item 45. Marco assigns ownership by dragging the work item in the plan to the new owner. Marco can also assign work items by right-clicking the work item and selecting the Assign To Owner option.

At the end of the stand-up meeting, the team starts to discuss the new Web browser service release that was just made available. The team must update the build and test machines, and Marco adds a new task in the plan for someone to pick up.

To add a new work item task to the plan, Marco performs the following actions:

1. Marco right-clicks the unassigned role in the plan. Alternatively, he can choose a team member to assign the new work item to. Then he selects **Add Work Item** → **Task** as shown in Figure 7-16.
2. He uses in-place editing to enter a work item summary and description.
3. He clicks the **clock** icon and selects **1 hour** from the menu.
4. He click **Save** on the iteration plan document to save the changes to the iteration plan.



Unassigned
Closed items: 0 | Open items: 6

Portal require Java 5	-	High	46
Java lang error	-	High	45
Web forms errors	-	Medium	44
Issues after installing IE7 SP2	-	Medium	43
Special EURO character not supported	-	Unassigned	47
Install new web browser	-	Unassigned	--

<Add Description>

Figure 7-16 Marco adding new work items to the iteration plan

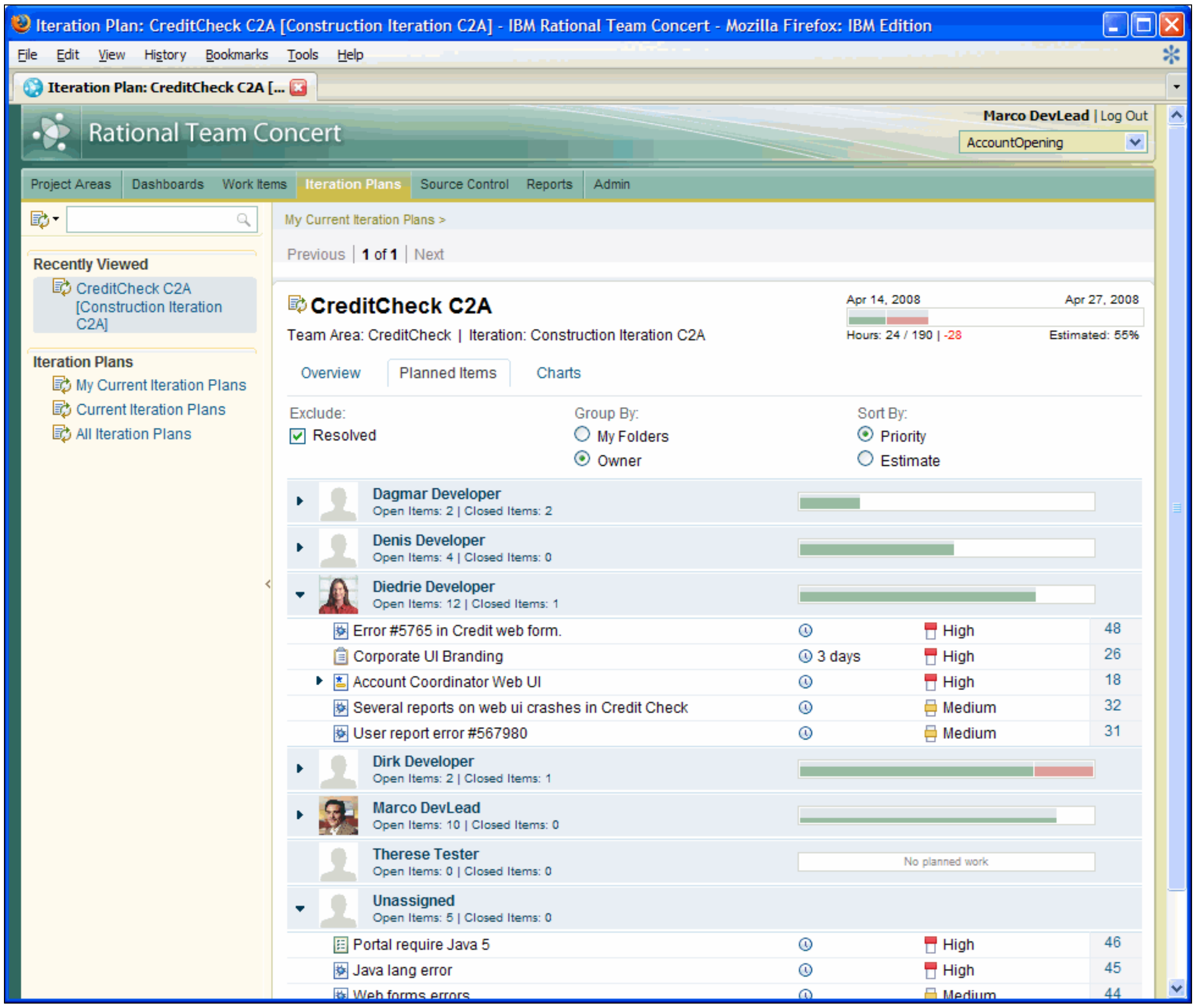
Using the Rational Team Concert Web

Goal: The goal is to enable access to team collaboration for contributors.

The Rational Team Concert Web UI provides functionality for external contributors who require easy access to view, for example, the Dashboard view or the Iteration Plan view, as shown in Figure 7-17 on page 253.

Bob, the product owner, is attending the team stand-up meetings on a frequent basis. He only uses the Web client to access team health from the Dashboard view or the Iteration Plan view. During stand-up meetings, he helps the team by clarifying stakeholder requests and release requirements. Because Bob is a registered user in the Jazz repository, the team can use the presence awareness capabilities in Rational Team Concert to quickly connect with Bob and resolve questions.

At the meeting today, the team discusses the UI Branding requirements and brings Bob into the discussion by using the Chat view. Bob shares the link to the corporate branding home page.


Iteration Plan: CreditCheck C2A [Construction Iteration C2A] - IBM Rational Team Concert - Mozilla Firefox: IBM Edition

File Edit View History Bookmarks Tools Help

Iteration Plan: CreditCheck C2A [...

Rational Team Concert

Marco DevLead | Log Out

AccountOpening

Project Areas | Dashboards | Work Items | Iteration Plans | Source Control | Reports | Admin

Recently Viewed
- CreditCheck C2A [Construction Iteration C2A]

Iteration Plans
- My Current Iteration Plans
- Current Iteration Plans
- All Iteration Plans

My Current Iteration Plans >

Previous | 1 of 1 | Next

CreditCheck C2A

Team Area: CreditCheck | Iteration: Construction Iteration C2A

Apr 14, 2008 — Apr 27, 2008

Hours: 24 / 190 | -28 Estimated: 55%

Overview | Planned Items | Charts

Exclude: [✓] Resolved

Group By: ○ My Folders [● Owner]

Sort By: [● Priority] ○ Estimate

Dagmar Developer
Open Items: 2 | Closed Items: 2

Denis Developer
Open Items: 4 | Closed Items: 0

Diedrie Developer
Open Items: 12 | Closed Items: 1

Error #5765 in Credit web form.		High	48
Corporate UI Branding	3 days	High	26
Account Coordinator Web UI		High	18
Several reports on web ui crashes in Credit Check		Medium	32
User report error #567980		Medium	31

Dirk Developer
Open Items: 2 | Closed Items: 1

Marco DevLead
Open Items: 10 | Closed Items: 0

Therese Tester
Open Items: 0 | Closed Items: 0
No planned work

Unassigned
Open Items: 5 | Closed Items: 0

Portal require Java 5		High	46
Java lang error		High	45
Web forms errors		Medium	44

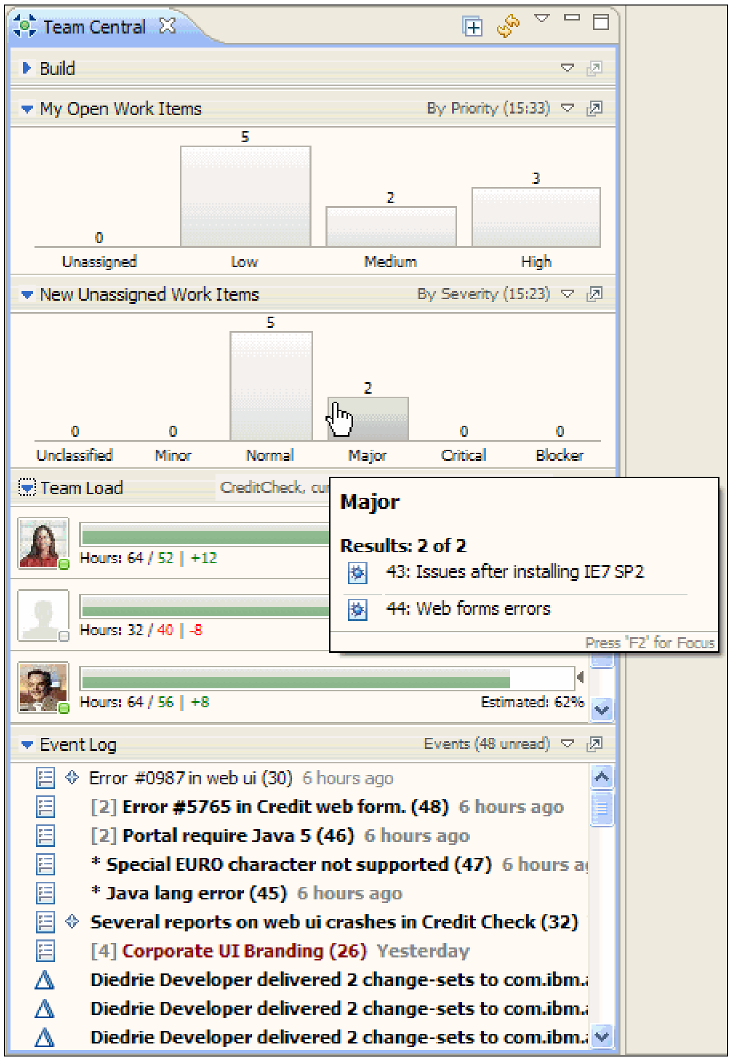
Figure 7-17 The iteration plan shown in the Rational Team Concert Web UI

Team Central to monitor component health

Goal: The goal is to provide transparency to team health and event information at the fingertips of each team member.

Marco and the team, by using an agile way of working, have a shared responsibility for monitoring and reacting on component health. The Team Central view in Rational Team Concert helps each individual team member to keep updated on team health metrics, their individual schedule, and any events that can impact their work plans.

Figure 7-18 on page 254 show how Marco has configured Team Central to keep the component health indicators that he is keeping as his fingertips. In Team Central, he can view new incoming work items, the iteration plan health for his team, build stability, and major events that he is subscribing to from his team. By hovering over the items, he can quickly access drill-down information or click items to open their item editor views.



Error #0987 in web ui (30) 6 hours ago
[2] Error #5765 in Credit web form. (48) 6 hours ago
[2] Portal require Java 5 (46) 6 hours ago
* Special EURO character not supported (47) 6 hours a
* Java lang error (45) 6 hours ago
Several reports on web ui crashes in Credit Check (32)
[4] Corporate UI Branding (26) Yesterday
Diedrie Developer delivered 2 change-sets to com.ibm.
Diedrie Developer delivered 2 change-sets to com.ibm.
Diedrie Developer delivered 2 change-sets to com.ibm.

Figure 7-18 Team Central view used by Marco to keep component health indicators at his fingertips

My Work view for work scheduling

Goal: The goal is to provide individual work scheduling integrated with the team iteration plan.

At the daily stand-up meeting, Marco requested that all team members must complete their iteration schedule and work estimation before the end of the day. Diedrie must complete this work and open the C2A iteration plan and her My Work view.

The My Work view provides Diedrie a compilation of all work that is assigned to her for this iteration. With this view, Diedrie can capture her schedule, in an agile way, and specify the order in which she intends to address the stack of work. The view also keeps her updated as new work is assigned to her. She has also used My Work customization to colorize the items, for example, red for defects and blue for new development work. This color coding helps her gain an additional perspective of planned work.

Figure 7-19 shows Diedrie's current My Work view, which shows the following concepts:

- Diedrie has new work assigned to her that she must schedule into her plan for work.
- Her current plan is to start working on the Corporate UI Branding today.
- She also has a few items for the iteration that she still must estimate.

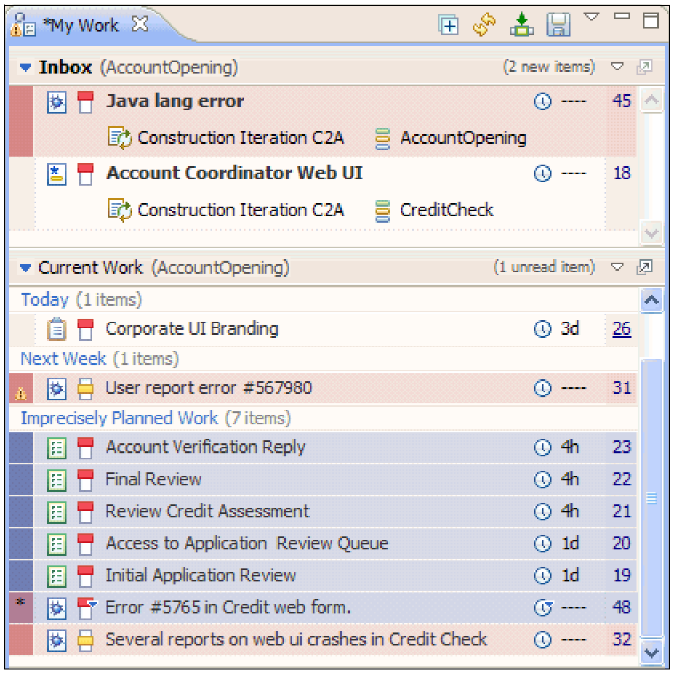
To record her estimations on a Work Item, Diedrie performs the following actions:

1. Diedrie selects the work item.
2. She clicks the clock icon on the work item and selects an assessment of hours, days, or weeks.

To rearrange her work schedule, Diedrie drags the item to a new position in the work item stack. The higher up in the stack she places the item, the sooner she intends to address the work. Diedrie can also plan work for today, tomorrow, or later. To make such a change, she drops the work item in the desired section. For scheduling new items, from the Inbox, she drags the item to the proper stack location.

Diedrie now intends to finalize her iteration planning:

1. Diedrie drags the two new items from the Inbox to the section for next week.
2. She provides estimates for her remaining items.
3. She decides to pick up the new work item on installing the new Web browser version. She drags the work item from the iteration plan to her My Work view section for today. In doing so, she assigns the work to herself and plans to do it today.
4. To complete her planning, she clicks the **Save** icon on the view toolbar.



*My Work

Inbox (AccountOpening) (2 new items)

Java lang error ---- 45
Construction Iteration C2A AccountOpening
Account Coordinator Web UI ---- 18
Construction Iteration C2A CreditCheck

Current Work (AccountOpening) (1 unread item)

Today (1 items)
Corporate UI Branding 3d 26
Next Week (1 items)
User report error #567980 ---- 31
Imprecisely Planned Work (7 items)
Account Verification Reply 4h 23
Final Review 4h 22
Review Credit Assessment 4h 21
Access to Application Review Queue 1d 20
Initial Application Review 1d 19
Error #5765 in Credit web form. ---- 48
Several reports on web ui crashes in Credit Check ---- 32

Figure 7-19 Diedrie's My Work view for viewing, planning, and scheduling her work items

7.3.2 Al identifies an asset that the team can reuse

Synopsis: Diedrie is considering her options on the Corporate UI Branding work that is assigned to her. She needs advice from Al regarding a reusable component. She sends Al an instant message, attaches a link to her work item, and asks for his guidance. Al knows that there is a reusable component available because he has been helping other teams on the same topic. Sharing this component with Diedrie saves her and the rest of the team development and testing time.

Al logs into the asset repository and starts his search in the Credit Management community. This community is set up to support this business line, and the Credit Check team has approvals and permission to participate in this community.

Al searches the entries in the Credit Management community and gets several hits on branding. However, he focuses his attention to the “Re-brand UI Component” asset that has received high scores from other users. After viewing the details of the asset, he decides to share his discovery with Diedrie and Marco. He collaborates by sending an e-mail that includes a link to the found asset.

Diedrie and Marco receive the e-mail and invite Al to a Web discussion about the topic. Al shares his desktop and helps the team consume the documentation, the design, and the test cases. Diedrie makes notes about the discussion and saves them as an attachment to the her UI branding work item. She also saves the link to the reuse component as a related artifact.

The following tasks occur in this scene as illustrated in Figure 7-20:

- Searching for a reusable asset
- Collaborating on reusable assets

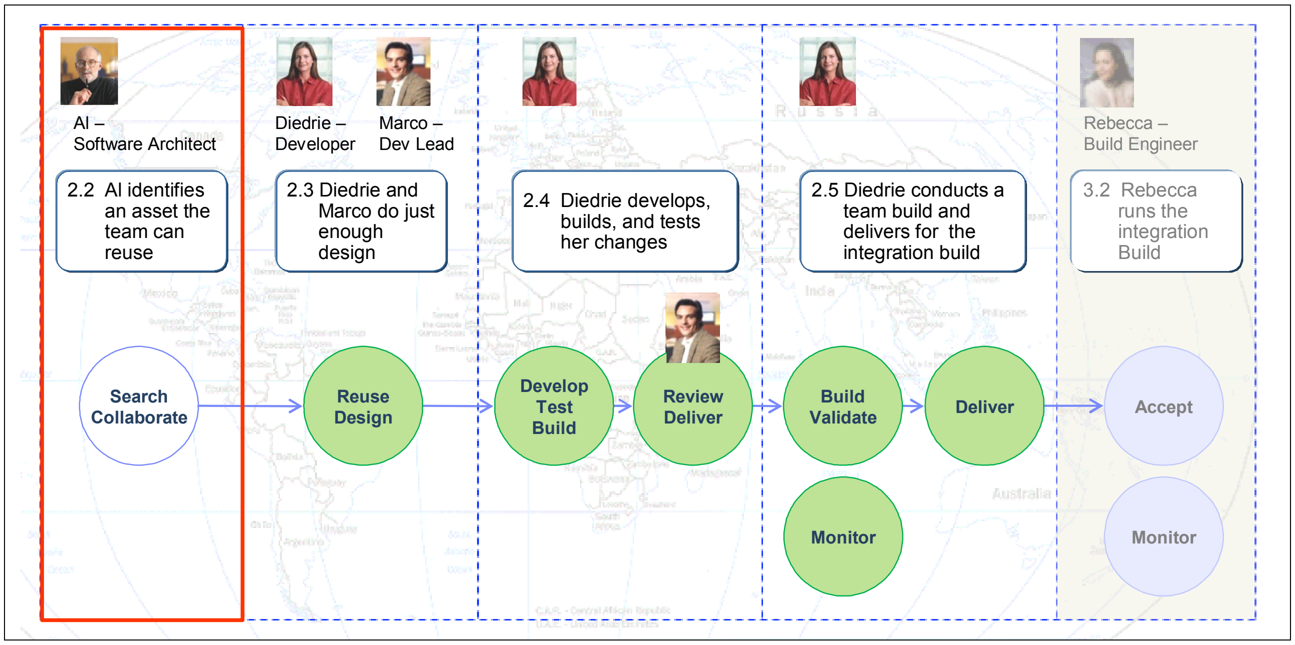

Al – Software Architect
Diedrie – Developer
Marco – Dev Lead
Rebecca – Build Engineer
2.2 Al identifies an asset the team can reuse
2.3 Diedrie and Marco do just enough design
2.4 Diedrie develops, builds, and tests her changes
2.5 Diedrie conducts a team build and delivers for the integration build
3.2 Rebecca runs the integration Build
Search Collaborate
Reuse Design
Develop Test Build
Review Deliver
Build Validate
Deliver
Accept
Monitor
Monitor

Figure 7-20 Al identifying an asset for the team to reuse

Figure 7-21 show the life-cycle asset and relationships that are impacted in this scene.

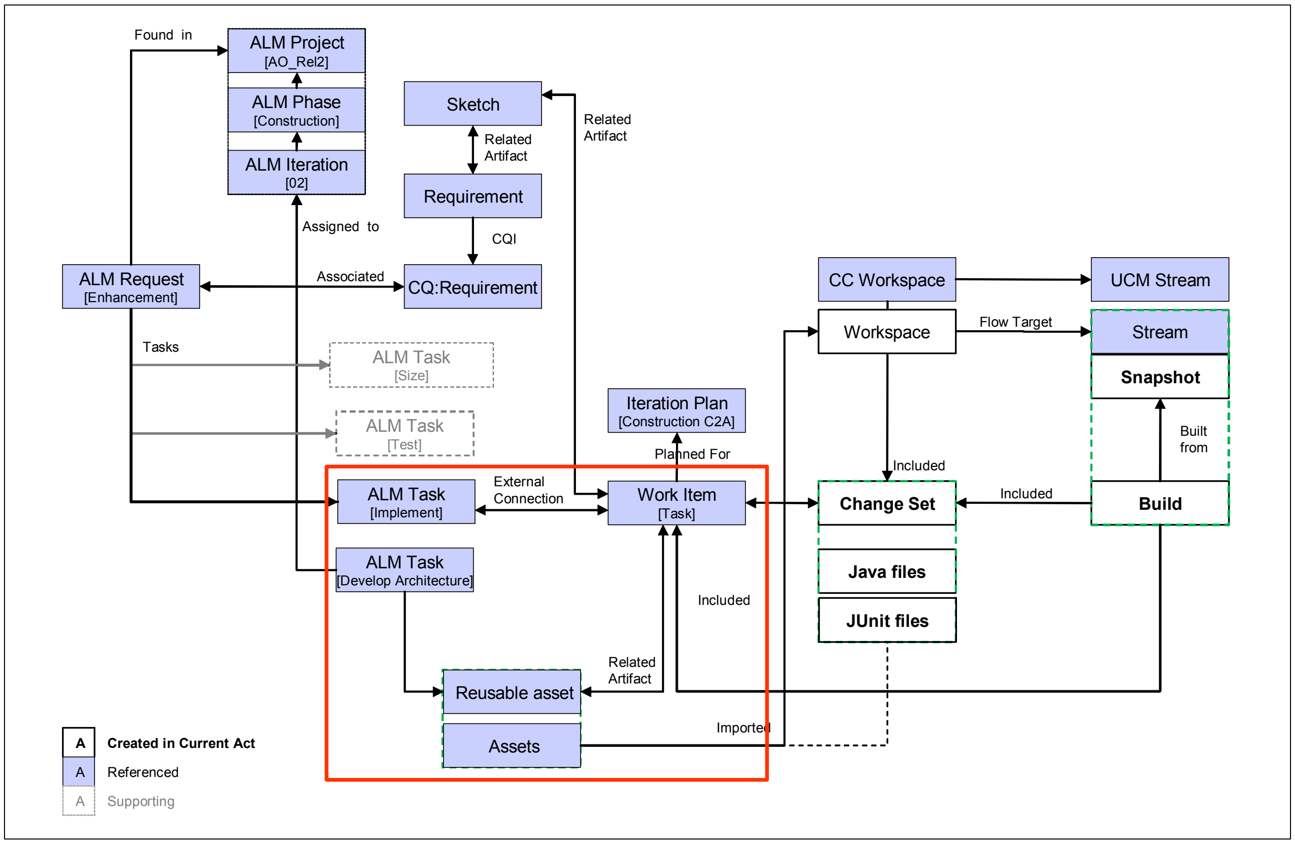


Figure 7-21 Diedrie establishing traceability from the work item to the reusable component in Rational Asset Manager

Rational Asset Manager

Rational Asset Manager reduces software development costs and improves quality by facilitating the reuse of all types of software development-related assets. Rational Asset Manager provides the following key capabilities:

- Asset organization

  Categorize assets based on type, attributes, and relationships between assets. Package related artifacts of all types, such as documents, source code, executables, and test cases into assets.

- Search

  Search assets by using keywords or content search in files, documents, or archives.

- Security and access

  Protect assets with fine-grained permissions based on groups, roles, users, or asset types.

- Tracking

  Track asset usage, quality, approvals, feedback, and ranking.

- Collaboration

  Collaborate with other project members through discussions, e-mail subscriptions, and RSS feeds for notification of asset changes.

- User interface

  Use Eclipse client integration or a Web UI.

In our scenario, Al uses the Rational Asset Manager Web UI, while Diedrie uses Eclipse integration in her Rational Team Concert development environment.

Searching for a reusable asset

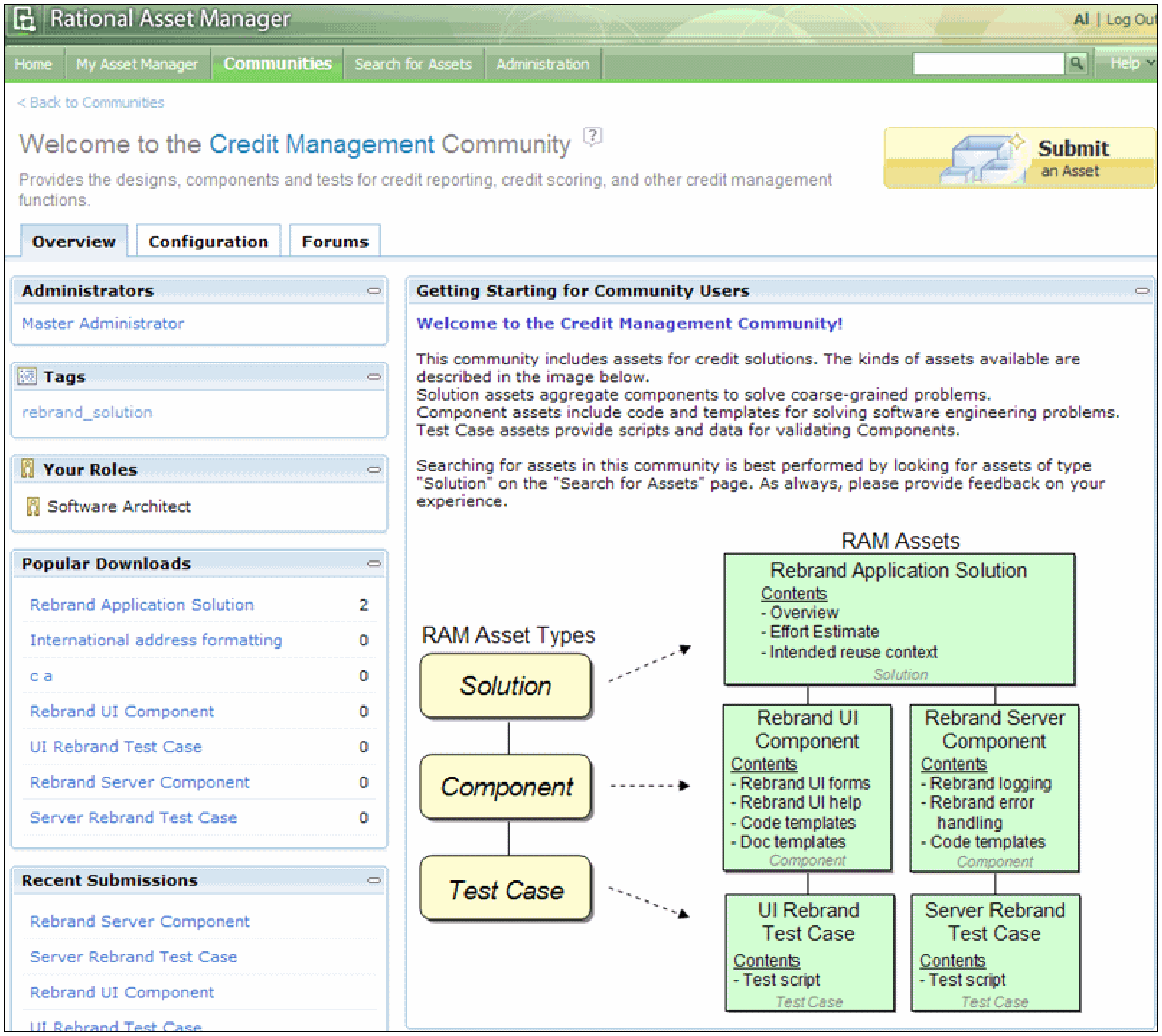
Goal: The goal is to use the corporate reuse repository in Rational Asset Manager to identify a proven component for reuse that saves time and effort for the development team.

Al has been asked by Diedrie for advice on a reusable component for Corporate UI Branding. Al knows that such a component is available because he has been helping other teams on the same topic in other projects. By using the reuse component, Al achieves architectural consistency across the projects that he is tracking, and it saves Diedrie and the rest of the team development and testing time.

By using the Rational Asset Manager Web UI, Al logs into the repository and starts his search effort. Rational Asset Manager is organized with one or more communities. A *community* includes a collection of users, assets, and other relevant items that together share a common interest for a project. Al's projects are all part of the Credit Management community. Al, Marco, Diedrie, and others are given permission to participate in this community.

To search the community Al performs the following actions:

1. Al selects the **Communities** tab and browses to the Credit Management community.
2. On the Credit Management home page, community home page (Figure 7-22 on page 259), Al sees the popular downloads in the community and can obtain a summary of the kinds of assets and other items that are relevant to the community.


Rational Asset Manager

Home | My Asset Manager | Communities | Search for Assets | Administration

< Back to Communities

Welcome to the Credit Management Community

Provides the designs, components and tests for credit reporting, credit scoring, and other credit management functions.

Submit an Asset

Overview | Configuration | Forums

Administrators

Master Administrator

Tags

rebrand_solution

Your Roles

Software Architect

Popular Downloads

Rebrand Application Solution	2
International address formatting	0
c a	0
Rebrand UI Component	0
UI Rebrand Test Case	0
Rebrand Server Component	0
Server Rebrand Test Case	0

Recent Submissions

Rebrand Server Component
Server Rebrand Test Case
Rebrand UI Component
UI Rebrand Test Case

Getting Starting for Community Users

Welcome to the Credit Management Community!

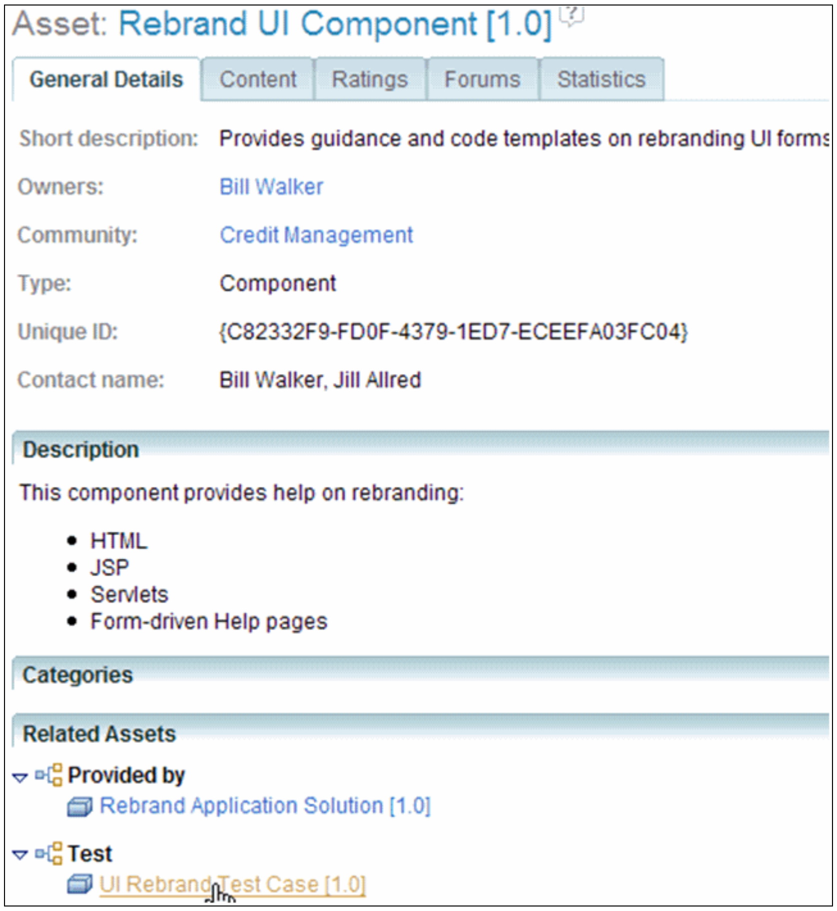
This community includes assets for credit solutions. The kinds of assets available are described in the image below.
Solution assets aggregate components to solve coarse-grained problems.
Component assets include code and templates for solving software engineering problems.
Test Case assets provide scripts and data for validating Components.

Searching for assets in this community is best performed by looking for assets of type "Solution" on the "Search for Assets" page. As always, please provide feedback on your experience.

Figure 7-22 The Credit Management Community in Rational Asset Manager

3. He clicks the **Search for Assets** tab along the top and enters a keyword search for “`Rebrand`.”

4. From the list of assets that is returned on the search, Al opens the **Rebrand Application Solution** asset and finds that a related asset, the “Rebrand UI Component” asset, seems applicable to Diedrie’s request. Al clicks its link and browses this related asset to see its details (Figure 7-23).


Asset: Rebrand UI Component [1.0]

Short description: Provides guidance and code templates on rebranding UI forms

Owners: Bill Walker

Community: Credit Management

Type: Component

Unique ID: {C82332F9-FD0F-4379-1ED7-ECEEFA03FC04}

Contact name: Bill Walker, Jill Allred

Description

This component provides help on rebranding:

- HTML
- JSP
- Servlets
- Form-driven Help pages

Categories

Related Assets

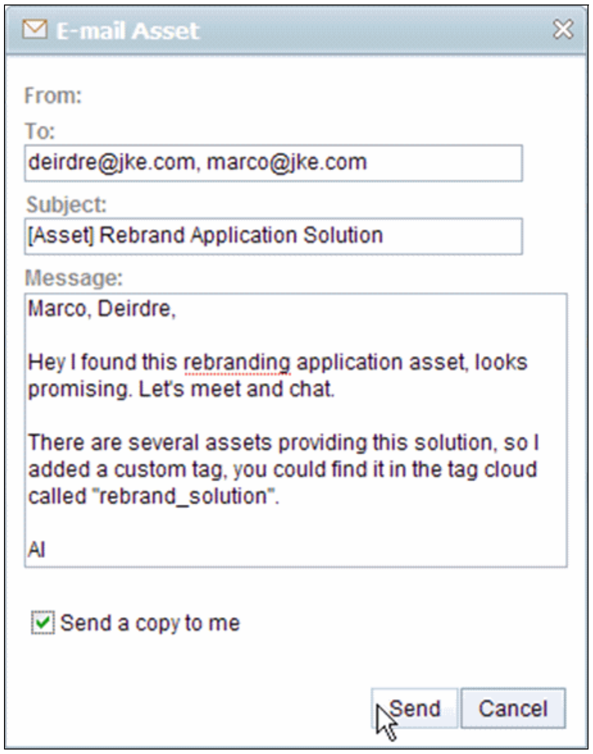
Provided by
Rebrand Application Solution [1.0]

Test
UI Rebrand Test Case [1.0]

Figure 7-23 The Rebrand UI Component asset found in the Rational Asset Manager repository

5. Because Al wants to tag the assets that he has found for later reference, he opens the Tags window and adds a `rebrand_solution` tag keyword.

6. Al decides that its time to share his findings with Diedrie and Marco and to discuss the feasibility of the reuse component. He opens the **Asset Detail Tools** view and chooses **e-mail** to send a notification (Figure 7-24).



deirdre@jke.com, marco@jke.com

[Asset] Rebrand Application Solution

Marco, Deirdre,

Hey I found this rebranding application asset, looks promising. Let's meet and chat.

There are several assets providing this solution, so I added a custom tag, you could find it in the tag cloud called "rebrand_solution".

Al

Figure 7-24 Al sharing his findings with Diedrie and Marco

Al puts this activity on hold pending a response from Diedrie.

Collaborating on reusable assets

Goal: The goal is to collaborate within the development team to provide guidance on assets for reuse.

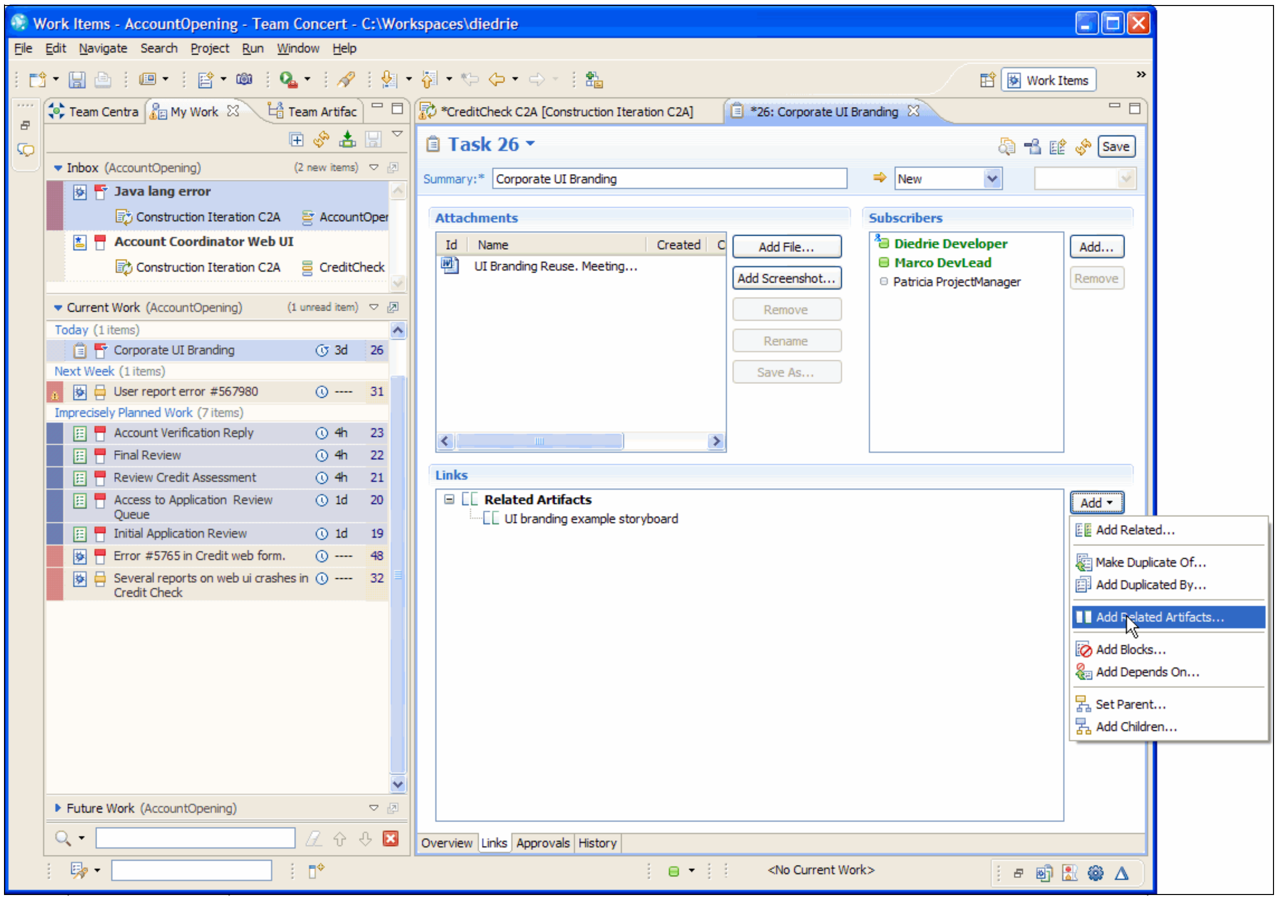
Back on the component team, Diedrie is pleased that Al has found a component for the team to reuse. She schedules a Web meeting with Al and Marco for later in the day to discuss the topic.

At the meeting, Al shares his screen so that everyone can look closely at the branding assets. Al opens the “Rebrand Application Solution” asset. The three look at the available documentation for solution branding. After they walk through it, Al, Diedrie and Marco have determined that they should proceed with the prototype design by using the asset suggested by Al.

Diedrie documents their decisions on reuse by using the following actions:

1. Diedrie opens Rational Team Concert and starts the UI Branding work item.
2. She opens the UI Branding work item and selects the **Link** tab.
3. She attaches her meeting notes by dragging the document file to the Attachments section of the **Link** tab.

4. She adds a traceability link to the reusable asset by clicking the **Add** button and selecting **Add Related Artifact** (Figure 7-25). She names the link "`Rebrand Application Solution`" and pastes the URL to the asset in Rational Asset Manager.
5. She clicks the Save button to save the changes to the work item.



Inbox (AccountOpening) (2 new items)
- Java lang error — Construction Iteration C2A — AccountOper
- Account Coordinator Web UI — Construction Iteration C2A — CreditCheck

Current Work (AccountOpening) (1 unread item)

Today (1 items)

Corporate UI Branding	3d	26

Next Week (1 items)

User report error #567980	----	31

Imprecisely Planned Work (7 items)

Account Verification Reply	4h	23
Final Review	4h	22
Review Credit Assessment	4h	21
Access to Application Review Queue	1d	20
Initial Application Review	1d	19
Error #5765 in Credit web form.	----	48
Several reports on web ui crashes in Credit Check	----	32

Future Work (AccountOpening)

Task 26

Summary:* Corporate UI Branding — New

Attachments

Id	Name	Created	C
	UI Branding Reuse. Meeting...		

Subscribers
- Diedrie Developer
- Marco DevLead
- Patricia ProjectManager

Links
- Related Artifacts
  - UI branding example storyboard

Figure 7-25 Diedrie capturing traceability links to the team collaboration on reuse

7.3.3 Diedrie, Marco, and Al do 'just enough' design

Synopsis: Diedrie is ready to start working on Credit Check UI branding. She opens her work item and changes its state to indicate that work has begun. She creates a new, clean, and updated workspace from the latest component baseline to integrate her changes.

Diedrie then logs into the reuse repository, from her development environment, by using the link that Al provided. She reviews the online asset documentation and her notes from her discussion with Al. She then starts importing all related assets into her workspace. Some assets have Eclipse projects in them, and others must point to a target project in order to be imported.

Diedrie then proceeds and starts prototyping the re-design in one of the CreditCheck UI forms. She saves her design changes, and a new change set is created and associated with her work assignment.

To confirm her design strategy, she invites Marco and Al to a design review of the changes that are captured in the change set. Marco and Al accept her changes into their sandbox workspaces and start browsing the changes. Both Marco and Al confirm that the design of the changes looks acceptable. To get validation from the stakeholder, Diedrie brings Bob into the discussion and runs a demonstration of the updated UI form. Bob confirms that the corporate brand design and manner of use are now correct.

This scene involves the following tasks as shown in Figure 7-26:

- Beginning the work
- Reusing the asset
- Designing
- Reviewing the design

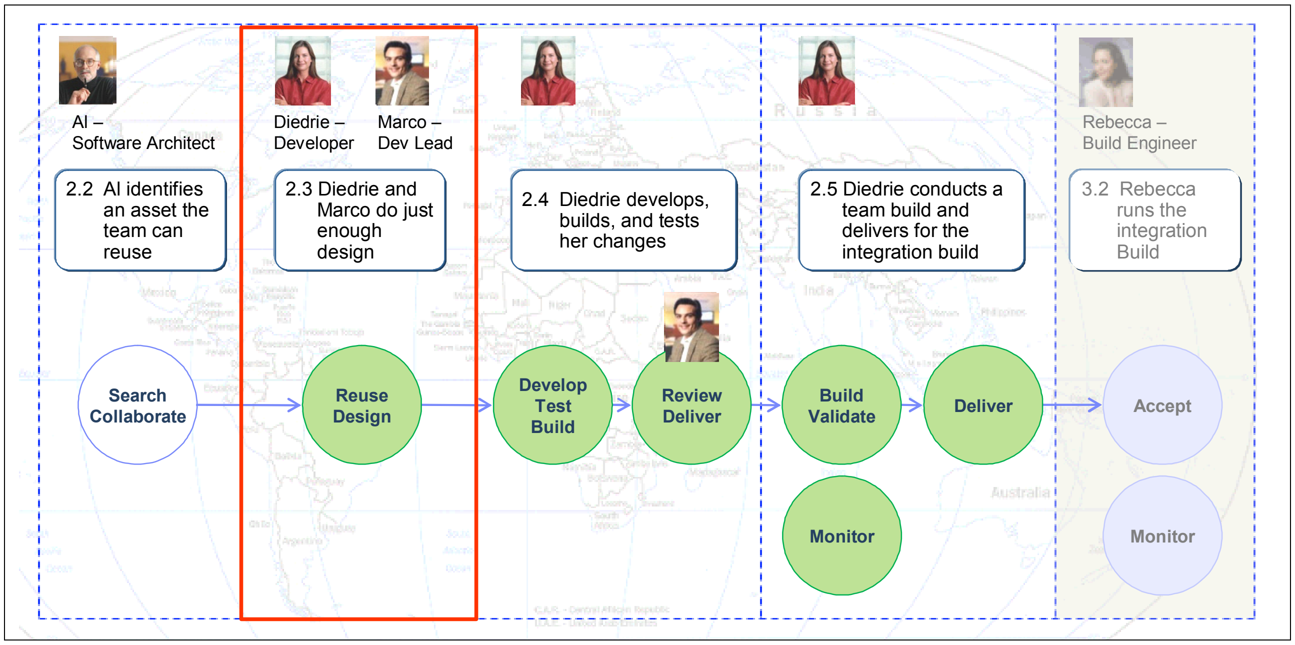

Al – Software Architect
Diedrie – Developer
Marco – Dev Lead
Rebecca – Build Engineer
2.2 Al identifies an asset the team can reuse
2.3 Diedrie and Marco do just enough design
2.4 Diedrie develops, builds, and tests her changes
2.5 Diedrie conducts a team build and delivers for the integration build
3.2 Rebecca runs the integration Build
Search Collaborate
Reuse Design
Develop Test Build
Review Deliver
Build Validate
Deliver
Accept
Monitor
Monitor

Figure 7-26 Diedrie, Marco and Al collaborating on a design

Figure 7-27 show the lifecycle asset and relationships that are impacted in this scene.

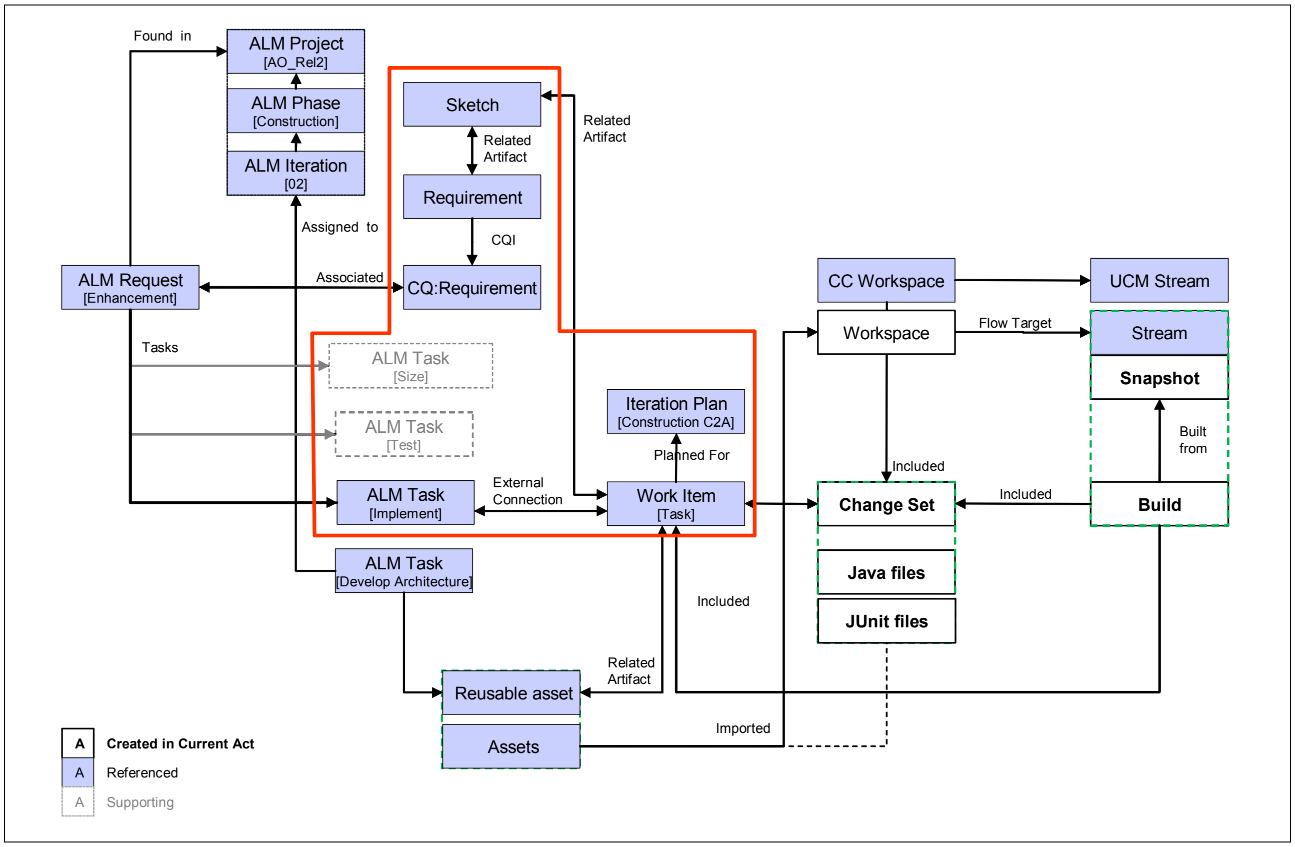


Figure 7-27 Diedrie using traceability from the Work Item to the artifacts containing supplementary information

Diedrie begins her work

Diedrie begins her work by taking fundamental steps in collaborative development:

- She notifies the rest of her team on what work she is doing.
- She creates a new sandbox workspace to host her changes.

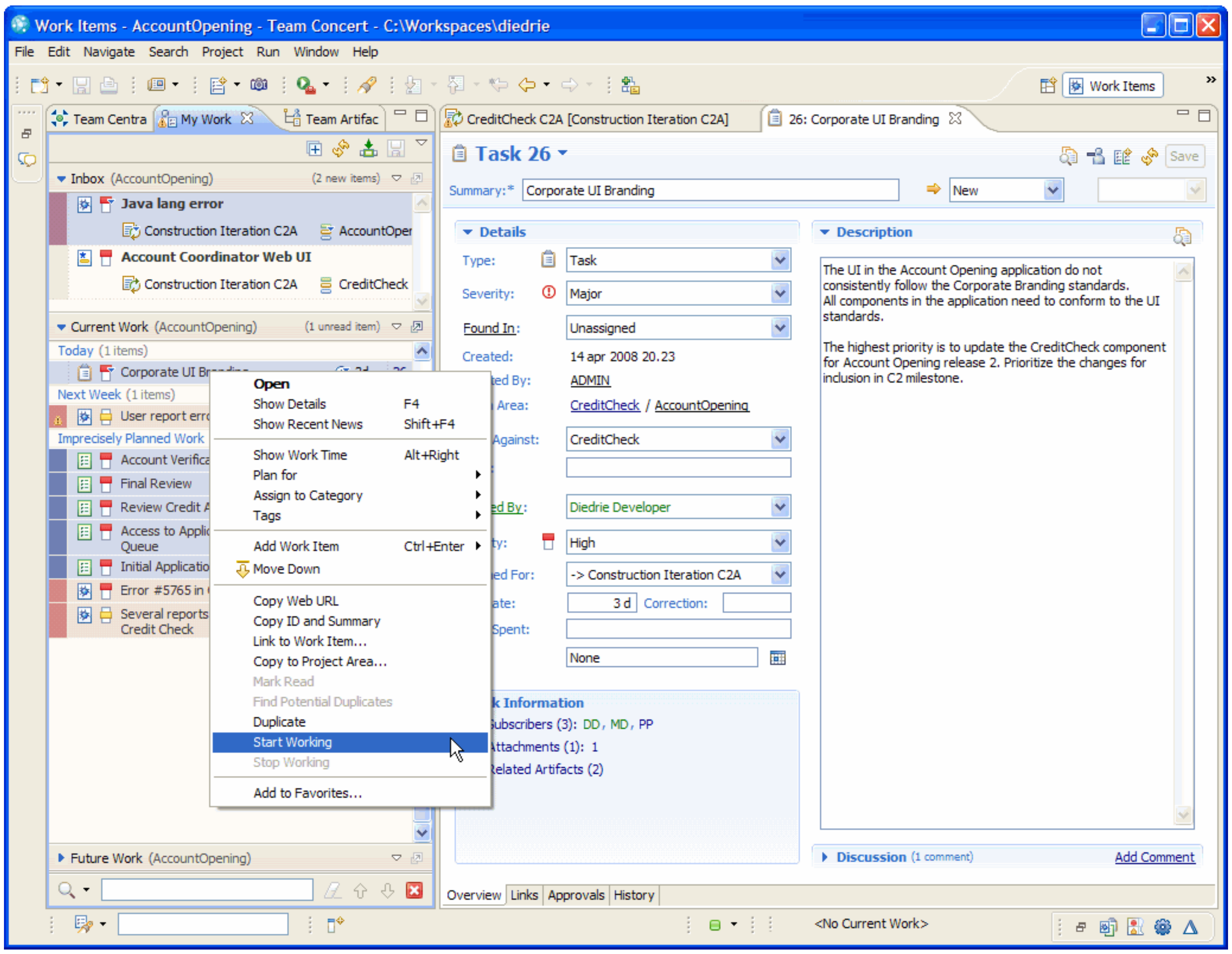
Making the work item in progress

Goal: The goal is to notify the team that the work is starting and begin associating changes to the work item.

Rational Team Concert supervises the creation of new artifacts and the bookkeeping of traceability. As Diedrie starts working on the UI Branding work item, all related changes and dependencies are transparently associated with the work item. Rational Team Concert maintains traceability as changes are made, delivered, captured in baselines, built, and later tested. By actively updating her work items state, Diedrie's development activities become a trackable and integral part of the team's change process and iteration plan tracking. It enables Rational Team Concert to automatically maintain traceability and for Diedrie's work to contribute to the live health indicators for the whole team.

Diedrie starts working on the UI Branding work item:

1. Diedrie opens the UI Branding work item from her My Work view.
2. She changes the state of the work item from Triage to **Start Working** as shown in Figure 7-28.



Summary:* Corporate UI Branding
New

Type: Task
Severity: Major
Found In: Unassigned
Created: 14 apr 2008 20.23
ed By: ADMIN
Area: CreditCheck / AccountOpening
Against: CreditCheck
ed By: Diedrie Developer
ty: High
ed For: -> Construction Iteration C2A
ate: 3 d Correction:
Spent:
None

k Information
ubscribers (3): DD, MD, PP
ttachments (1): 1
elated Artifacts (2)

The UI in the Account Opening application do not consistently follow the Corporate Branding standards.
All components in the Application need to conform to the UI standards.

The highest priority is to update the CreditCheck component for Account Opening release 2. Prioritize the changes for inclusion in C2 milestone.

Figure 7-28 In the My Work view, Diedrie choosing the UI Branding work item and Start Working

Creating a new workspace

Goal: The goal is to use an integrated sandbox to design and develop the software changes.

Diedrie needs a workspace area where she can view, modify, and test her changes to the software components before delivering them to the integration build.

Rational Team Concert provides server-side workspace management that is seamlessly integrated into the Eclipse environment. A workspace contains a set of components, each of which is a distinct version from a stream or another workspace. You can load (or unload) your local workspace with the files and folders, or a subset of them, from your repository workspace.

With Rational Team Concert, individual users can keep multiple server-side workspaces from various streams and baselines. This is a key capability for an agile developer to easily switch the context of work from the development of a new enhancement in a stream that contains work for an upcoming iteration, to a defect fix in a stream that contains work for an urgent hot-fix release.

Diedrie needs a workspace that is derived from the latest stabile baseline. By getting a recent baseline, she minimizes the need to merge changes from the rest of the team when she finally delivers the resolved work to integration.

To create a new workspace, Diedrie performs the following actions:

1. Diedrie opens the Team Artifacts view.
2. She expands the **Streams** node and selects the stream that she wants a workspace created from. She right-clicks the **com.ibm.ao.creditcheck.integration.ao_rel2** stream and chooses **New Repository Workspace**.
3. Because Diedrie wants the latest baseline and all components in the stream, she clicks **Finish** and has the workspace created and loaded into her local workspace.
4. On the are rare occasions when she might want a subset of the components or another history version, she clicks **Next** and makes additional choices.

After the new workspace is loaded into Eclipse, all projects start to rebuild.

Diedrie reuses the asset

To integrate the reusable asset, Diedrie must import it into her workspace by performing the following tasks:

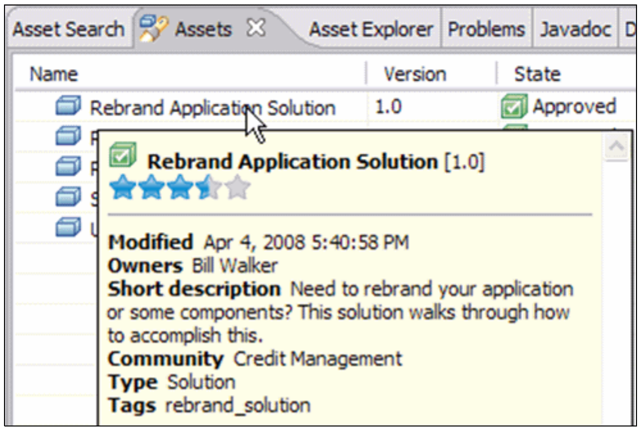
- Browsing the asset repository in Rational Team Concert
- Importing the asset into her workspace

Browsing assets from Rational Team Concert

Rational Asset Manager provides integration of Eclipse with Rational Team Concert so that Diedrie can browse the reuse assets. With the integration, Diedrie can import assets into her workspace.

To browse the Rational Asset Manager repository, Diedrie performs the following actions:

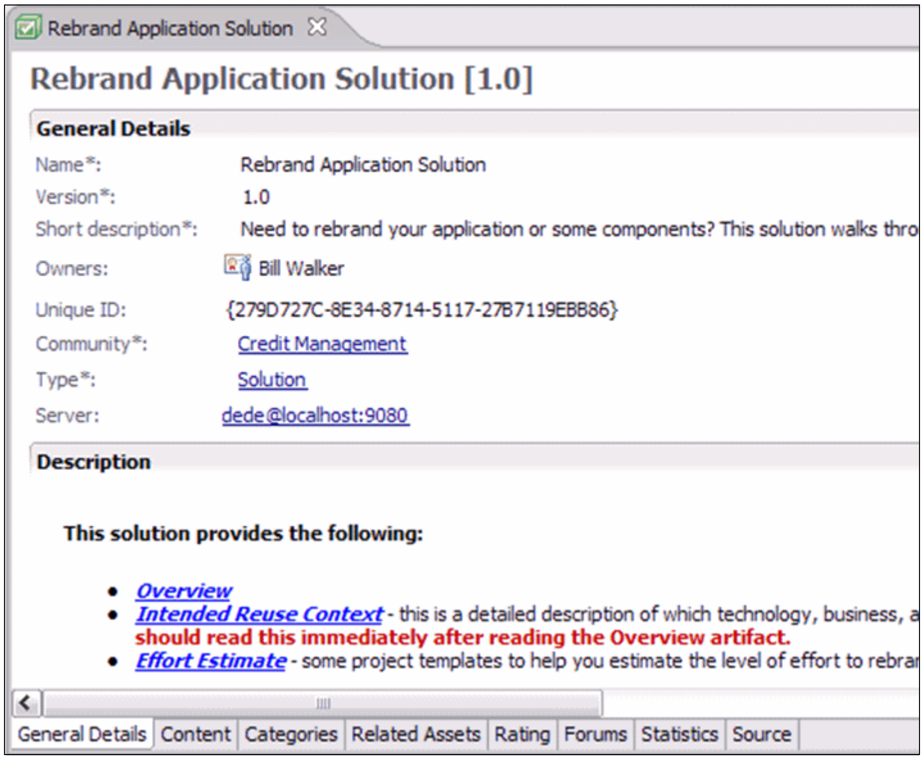
1. Diedrie opens Rational Team Concert and the **Asset Search** view.
2. She clicks the **Tag Clouds** panel and selects the **rebrand_solution** task, which Al updated during the walk through.
3. She browses the assets that Al associated with the tag. By hovering over an asset, she is presented with a content summary like the example in Figure 7-29.



Modified Apr 4, 2008 5:40:58 PM
Owners Bill Walker
Short description Need to rebrand your application or some components? This solution walks through how to accomplish this.
Community Credit Management
Type Solution
Tags rebrand_solution

Figure 7-29 Diedrie browsing the reusable assets tagged by Al

4. She double-clicks the **Rebrand Application Solution asset** to open the Details page (Figure 7-30).


Rebrand Application Solution [1.0]

General Details

Name*: Rebrand Application Solution

Version*: 1.0

Short description*: Need to rebrand your application or some components? This solution walks thro

Owners: Bill Walker

Unique ID: {279D727C-8E34-8714-5117-27B7119EBB86}

Community*: Credit Management

Type*: Solution

Server: dede@localhost:9080

Description

This solution provides the following:

- Overview
- Intended Reuse Context - this is a detailed description of which technology, business, a should read this immediately after reading the Overview artifact.
- Effort Estimate - some project templates to help you estimate the level of effort to rebra

Figure 7-30 Diedrie browsing the details of the Rebrand Application Solution asset

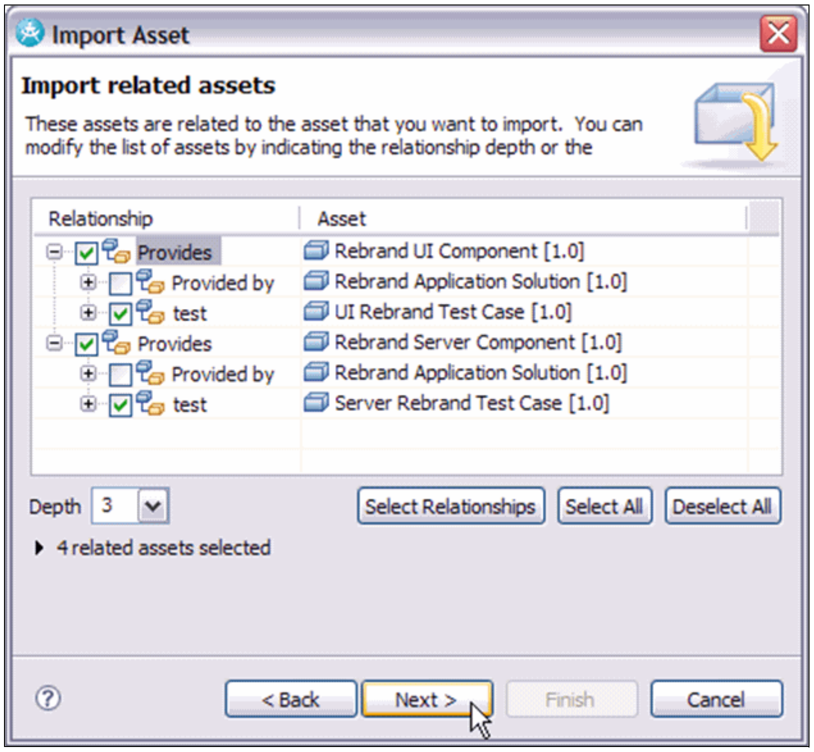
Importing assets into Rational Team Concert

Goal: The goal is to import the reusable assets into the workspace.

Diedrie is now ready to import all related assets for UI Branding into her workspace. Rational Asset Manager tracks the asset relationships and presents an option to select and import related assets.

To import all required assets, Diedrie performs the following steps:

1. Diedrie opens the Asset view and clicks the **Import** button.
2. In the Import Asset wizard (Figure 7-31 on page 268), she selects the related assets that she wants to import. Some of the assets have Eclipse projects in them, while others must point to a target project to be imported.
3. She opens her workspace and the existing and new Eclipse projects and validates that files were imported to the expected locations.
4. She saves any changes.


Import Asset

Import related assets

These assets are related to the asset that you want to import. You can modify the list of assets by indicating the relationship depth or the

Relationship	Asset
☑ Provides	Rebrand UI Component [1.0]
☐ Provided by	Rebrand Application Solution [1.0]
☑ test	UI Rebrand Test Case [1.0]
☑ Provides	Rebrand Server Component [1.0]
☐ Provided by	Rebrand Application Solution [1.0]
☑ test	Server Rebrand Test Case [1.0]

Depth 3

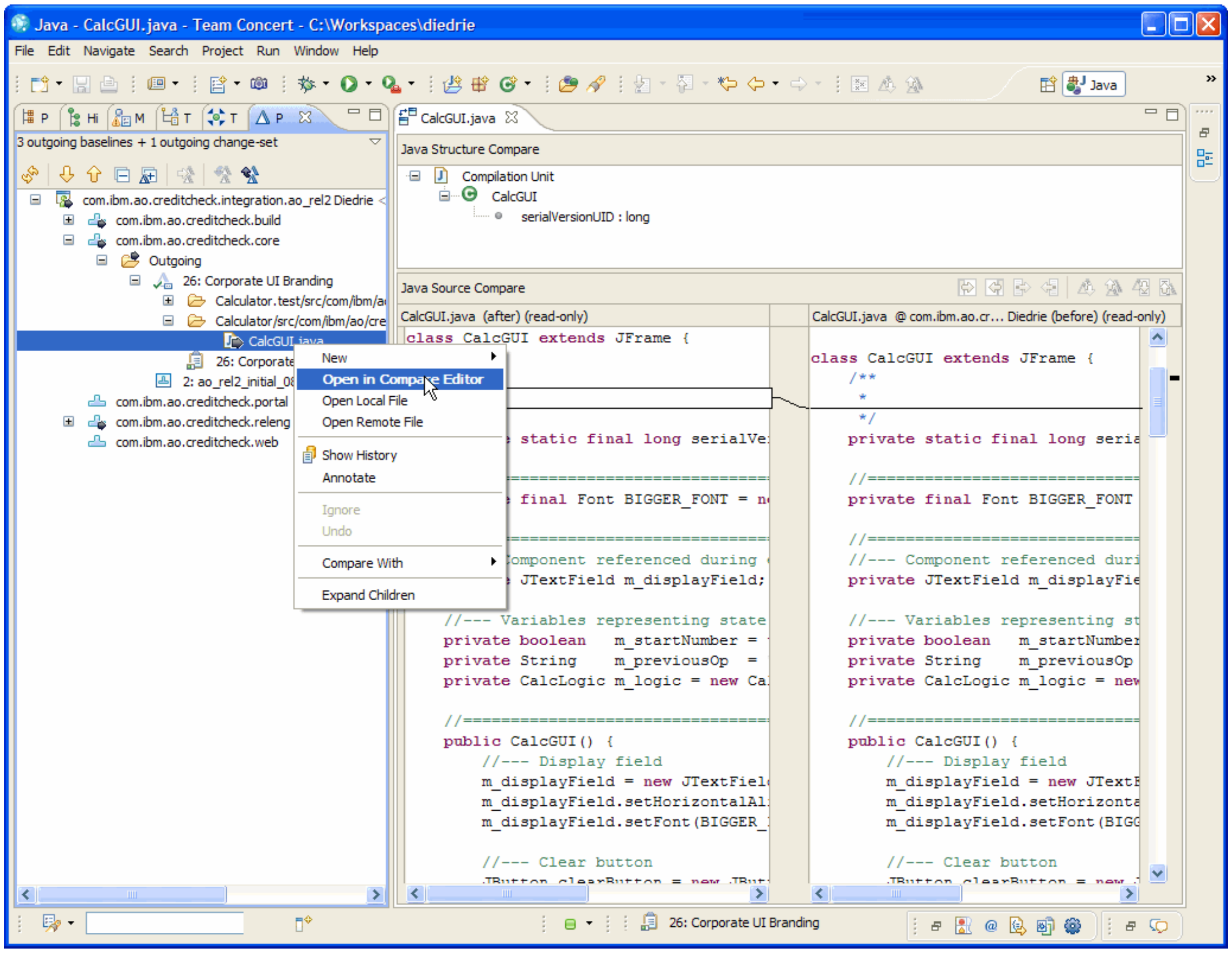
4 related assets selected

Figure 7-31 Diedrie selecting the related artifacts to be imported

Diedrie prototypes the design

With all related assets imported in her workspace, Diedrie is now ready to start prototyping the re-design in one of the CreditCheck UI forms. She chooses the form that Bob sketched in Rational RequisitePro Composer.

Because Diedrie is applying the UI Branding design patterns from the reusable asset, her source code files change. The Rational Team Concert SCM component transparently tracks all file changes, which are added to the current change set that is associated with the UI Branding work item that Diedrie is working on. Diedrie can view her change sets by using the Change Explorer view or the Pending Changes view that is available, for example, in the Java Perspective as shown in Figure 7-32 on page 269.


Java - CalcGUI.java - Team Concert - C:\Workspaces\diedrie
File Edit Navigate Search Project Run Window Help
Java
3 outgoing baselines + 1 outgoing change-set
com.ibm.ao.creditcheck.integration.ao_rel2 Diedrie
com.ibm.ao.creditcheck.build
com.ibm.ao.creditcheck.core
Outgoing
26: Corporate UI Branding
Calculator.test/src/com/ibm/a
Calculator/src/com/ibm/ao/cre
CalcGUI.java
26: Corporate
2: ao_rel2_initial_0
com.ibm.ao.creditcheck.portal
com.ibm.ao.creditcheck.releng
com.ibm.ao.creditcheck.web
New
Open in Compare Editor
Open Local File
Open Remote File
Show History
Annotate
Ignore
Undo
Compare With
Expand Children
CalcGUI.java
Java Structure Compare
Compilation Unit
CalcGUI
serialVersionUID : long
Java Source Compare
CalcGUI.java (after) (read-only)
CalcGUI.java @ com.ibm.ao.cr… Diedrie (before) (read-only)
26: Corporate UI Branding

```
class CalcGUI extends JFrame {

  static final long serialVe

 =============================
  final Font BIGGER_FONT = n

 =============================
 omponent referenced during
  JTextField m_displayField;

 //--- Variables representing state
 private boolean    m_startNumber =
 private String     m_previousOp    =
 private CalcLogic m_logic = new Ca

 //=================================
 public CalcGUI() {
     //--- Display field
     m_displayField = new JTextFiel
     m_displayField.setHorizontalAl
     m_displayField.setFont(BIGGER_

     //--- Clear button
     JButton clearButton = new JBut
```

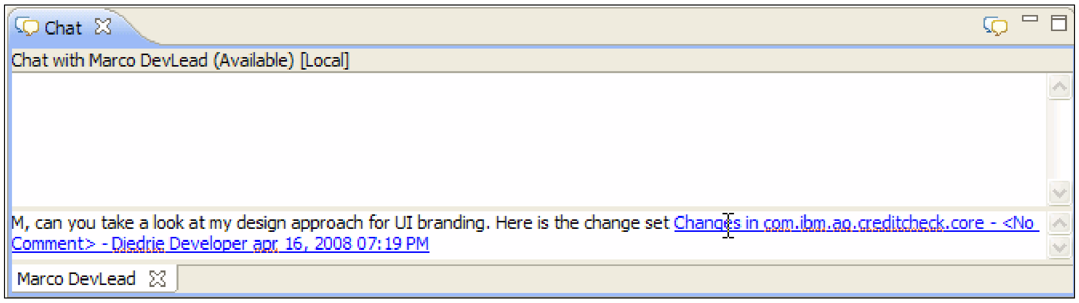
```
class CalcGUI extends JFrame {
    /**
     *
     */
    private static final long seria

    //=============================
    private final Font BIGGER_FONT

    //=============================
    //--- Component referenced duri
    private JTextField m_displayFie

    //--- Variables representing st
    private boolean    m_startNumber
    private String     m_previousOp
    private CalcLogic m_logic = new

    //=============================
    public CalcGUI() {
        //--- Display field
        m_displayField = new JTextF
        m_displayField.setHorizonta
        m_displayField.setFont(BIGG

        //--- Clear button
        JButton clearButton = new J
```

Figure 7-32 Diedrie using the Pending Changes view and Compare Editor to view file changes in her change set

Diedrie collaborates and validates her design

To confirm her design strategy, Diedrie wants Marco and Al to look at the changes that are captured in the change set. To collaborate with Marco, Diedrie does the following actions:

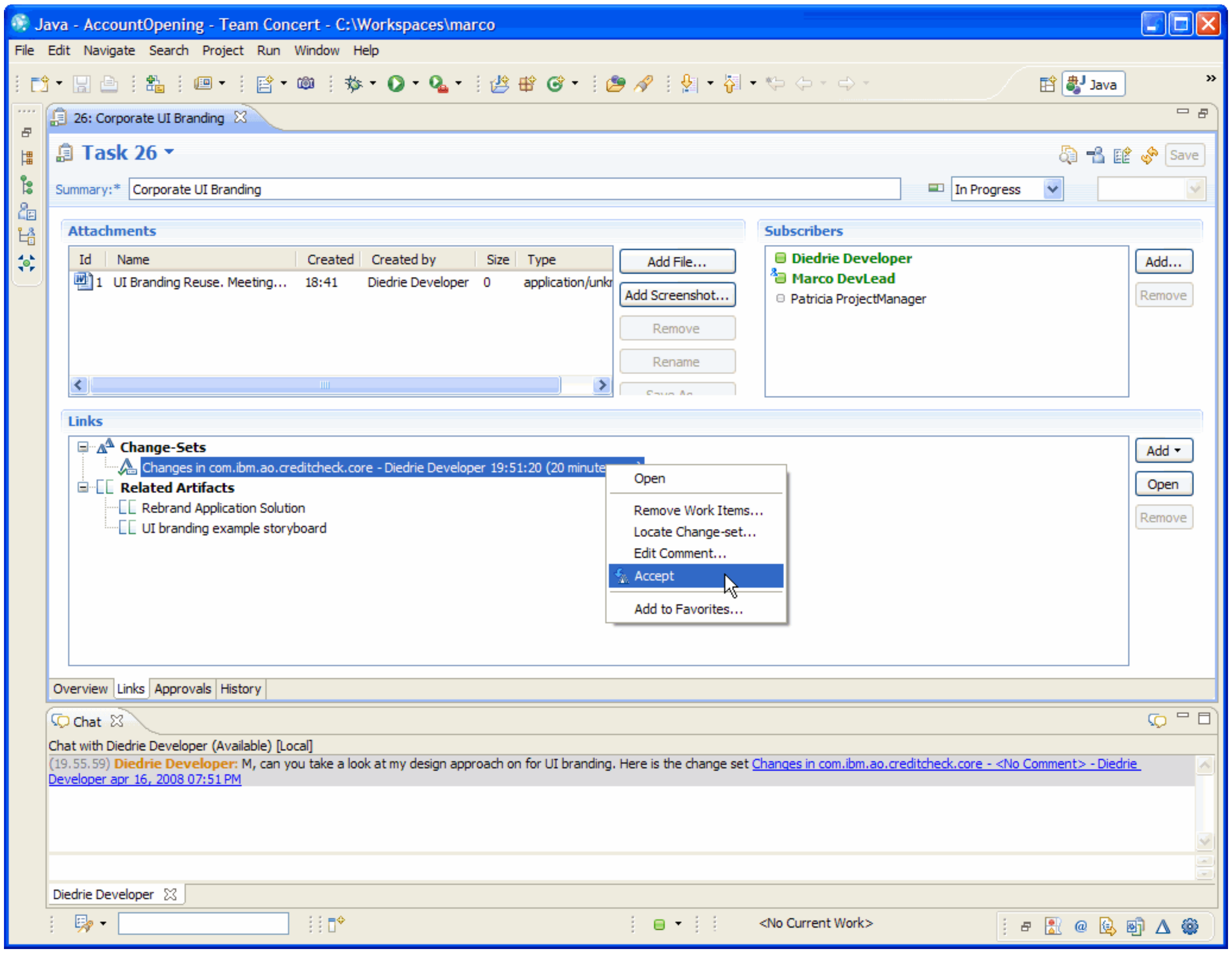
1. Diedrie opens the Chat view and creates a new instant message for Marco. She asks him to review her design approach for the UI Branding work item.
2. She drags the change set from the work item to the Chat view. The change set is added as a link as shown in Figure 7-33.
3. She presses the Enter key to send the message.



M, can you take a look at my design approach for UI branding. Here is the change set Changes in com.ibm.ao.creditcheck.core - <No Comment> - Diedrie Developer apr 16, 2008 07:19 PM

Figure 7-33 Diedrie sharing her design prototype (contained as a change set) with Marco in a chat

Marco is working on an urgent change in the application when he receives Diedrie's instant message. He can choose two alternative approaches to review Diedrie's changes. The brief review approach involves browsing the changes in the Change Set Explorer and reviewing the code changes. The in-depth review approach (Figure 7-34) involves accepting Diedrie's changes into Marco's workspace so that he can view and run the code.



Summary:* Corporate UI Branding

In Progress

Attachments

Id	Name	Created	Created by	Size	Type
1	UI Branding Reuse. Meeting...	18:41	Diedrie Developer	0	application/unkr

Subscribers

- Diedrie Developer
- Marco DevLead
- Patricia ProjectManager

Links

- Change-Sets
  - Changes in com.ibm.ao.creditcheck.core - Diedrie Developer 19:51:20 (20 minute
- Related Artifacts
  - Rebrand Application Solution
  - UI branding example storyboard

Chat with Diedrie Developer (Available) [Local]

(19.55.59) Diedrie Developer: M, can you take a look at my design approach on for UI branding. Here is the change set Changes in com.ibm.ao.creditcheck.core - <No Comment> - Diedrie Developer apr 16, 2008 07:51 PM

Figure 7-34 Marco performing an in-depth review of Diedrie's changes and accepts her change set into his workspace

To use the Change Explorer, Marco does the following actions:

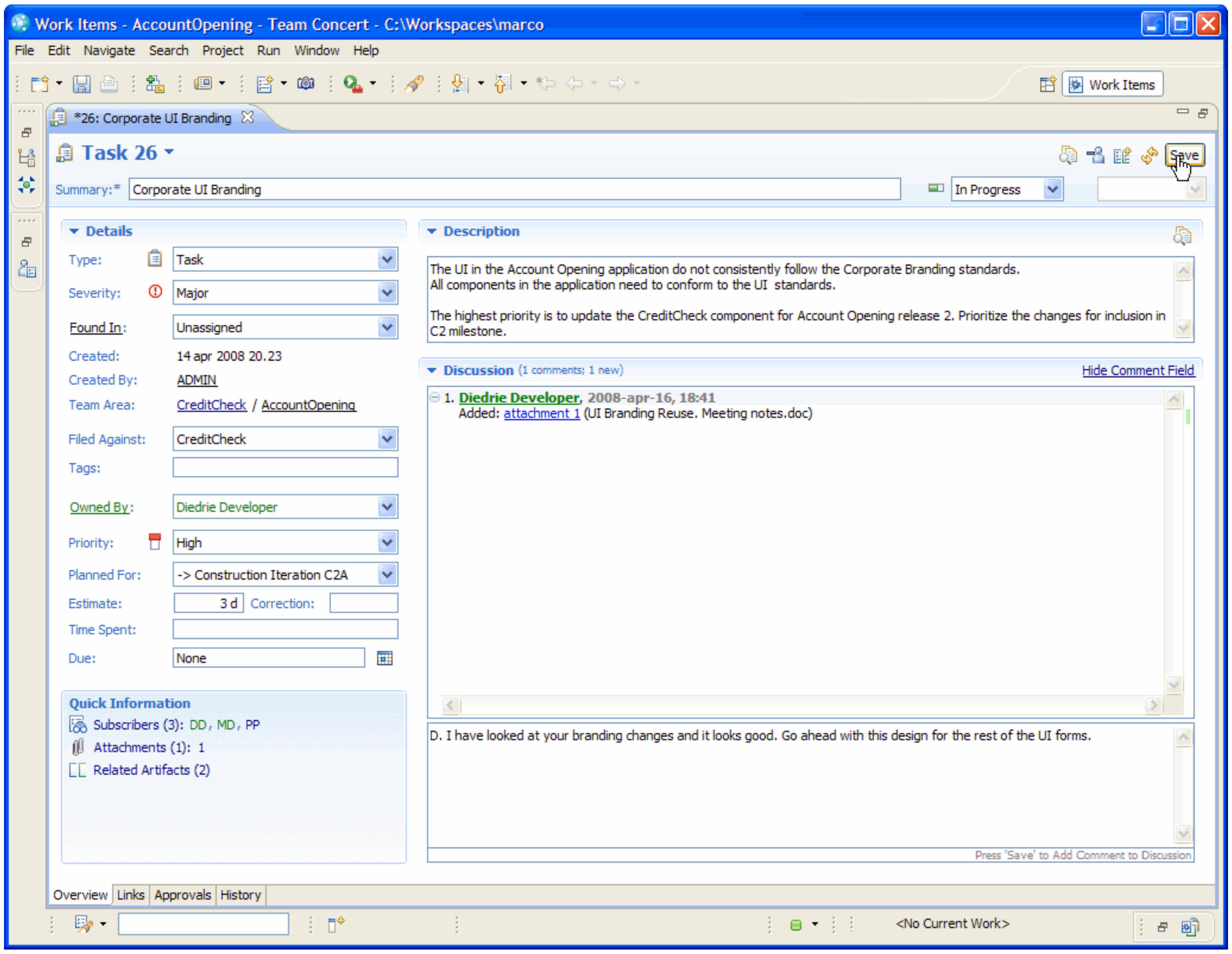
1. Marco right-clicks the change set or change sets and selects **Open**.
2. In the Change Explorer, he browses the code changes.
3. He opens the Compare Editor to see the code-level changes side-by-side with the original code.

To accept changes into his workspace, Marco does the following actions:

1. Marco opens the Pending Changes view and browses for outgoing changes.
2. He right-clicks the change sets and chooses **Suspend**.
3. He opens Diedrie's change set, right-clicks, and chooses **Accept**, which applies Diedrie's changes to his workspace.
4. He views and runs Diedrie's changes.
5. When Marco is done with validating Diedrie's changes, he resumes his previous work by right-clicking his suspended change set and selecting **Resume**.

Both Marco and Al confirm that the design of the changes looks acceptable. To add an in-context discussion entry in the work items, Marco performs the following steps:

1. He opens the Chat view with the instant message from Diedrie and clicks the link to the UI Branding work item.
2. In the work item, Marco clicks the **Add Comment** button and enters a discussion entry for Diedrie. He save the changes to the work item.
3. Diedrie and all other subscribers to the work item receive a notification that the work item was updated. Diedrie clicks the notification and opens the work item to read Marco's entry (Figure 7-35).



Work Items - AccountOpening - Team Concert - C:\Workspaces\marco
File Edit Navigate Search Project Run Window Help
Work Items
*26: Corporate UI Branding
Task 26
Save

Summary:* Corporate UI Branding

In Progress

Details

Type: Task
Severity: Major
Found In: Unassigned
Created: 14 apr 2008 20.23
Created By: ADMIN
Team Area: CreditCheck / AccountOpening
Filed Against: CreditCheck
Tags:
Owned By: Diedrie Developer
Priority: High
Planned For: -> Construction Iteration C2A
Estimate: 3 d Correction:
Time Spent:
Due: None

Quick Information
Subscribers (3): DD, MD, PP
Attachments (1): 1
Related Artifacts (2)

Description

The UI in the Account Opening application do not consistently follow the Corporate Branding standards.
All components in the application need to conform to the UI standards.

The highest priority is to update the CreditCheck component for Account Opening release 2. Prioritize the changes for inclusion in C2 milestone.

Discussion (1 comments; 1 new) Hide Comment Field

1. Diedrie Developer, 2008-apr-16, 18:41
Added: attachment 1 (UI Branding Reuse. Meeting notes.doc)

D. I have looked at your branding changes and it looks good. Go ahead with this design for the rest of the UI forms.

Press 'Save' to Add Comment to Discussion

Figure 7-35 Marco discussing his review of the changes in context of the work item

For further validation from the stakeholder, Diedrie brings Bob into the discussion and runs a demonstration of the updated UI form. Bob confirms that the corporate brand design and manner of use are now correct.

7.3.4 Diedrie develops, builds, and tests her changes

Synopsis: Diedrie is now ready to proceed and complete the development of re-designing all UI forms.

The component team is doing test-driven development (TDD), and Diedrie focuses first on the JUnit tests that validate her changes. She updates and runs her tests to confirm that the tests fail. She then applies the design pattern for the UI changes, builds her changes, and re-runs the tests. She proceeds until all CreditCheck forms are updated and all tests pass.

Diedrie is now ready to merge any incoming changes from her teammates. This is a prerequisite to deliver her changes to the integration. She accepts all incoming changes and resolves change conflicts. In some cases, she must collaborate with other developers on her team to decide on the best approaches to merge their changes.

She also must run a personal component build to validate that nothing is broken in her workspace. As new dependencies are added in the code project, she must update the component build script to reflect these dependencies. After completing the changes to the build script, she requests a private build of her workspace. The build script integrates compilation, build validations tests, and code analysis. She confirms that the build and validation results were successful.

Diedrie is now done with her changes and needs only to complete a code review before delivering her changes to the component integration stream. This review is a practice that the team enforces in their development process. She creates a new review request, attaches the change sets, and submits the request to Marco.

This scene entails the following tasks as illustrated in Figure 7-36 on page 273:

- Developing and testing
- Running a personal build
- Reviewing
- Delivering

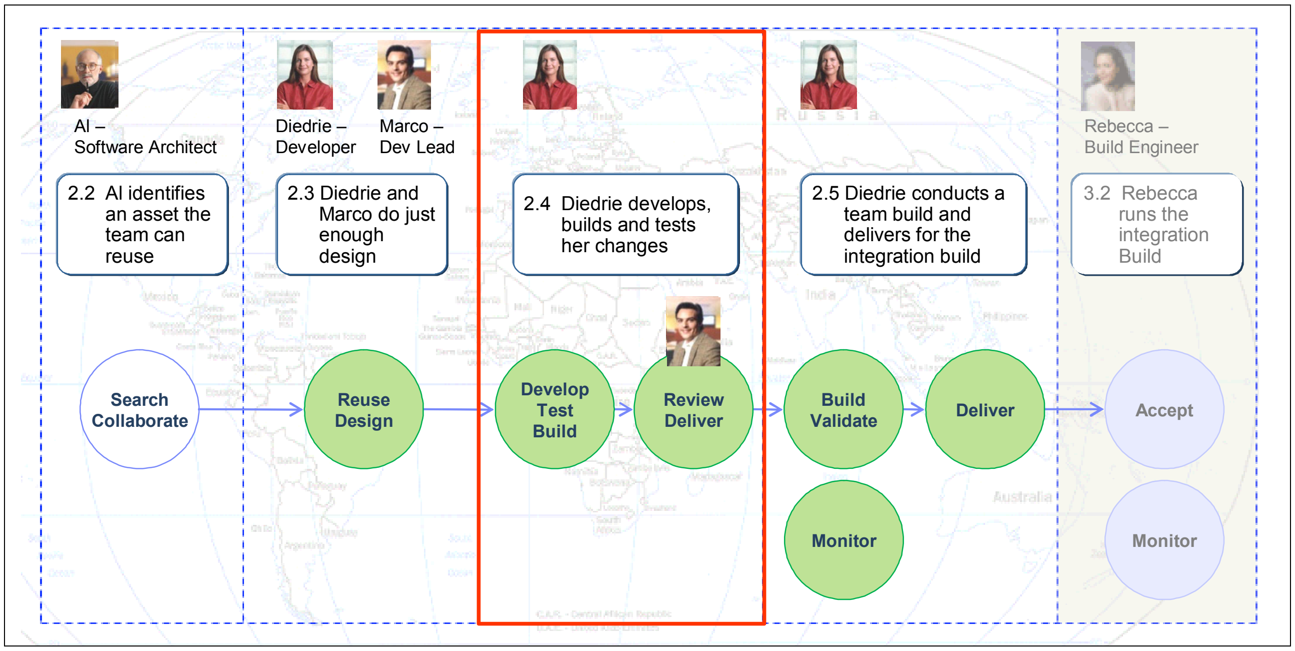

Al – Software Architect
Diedrie – Developer
Marco – Dev Lead
Rebecca – Build Engineer
2.2 Al identifies an asset the team can reuse
2.3 Diedrie and Marco do just enough design
2.4 Diedrie develops, builds and tests her changes
2.5 Diedrie conducts a team build and delivers for the integration build
3.2 Rebecca runs the integration Build
Search Collaborate
Reuse Design
Develop Test Build
Review Deliver
Build Validate
Deliver
Accept
Monitor
Monitor

Figure 7-36 Diedrie developing, building, and testing her changes

Figure 7-37 shows the life-cycle asset and relationships that are impacted in this scene.

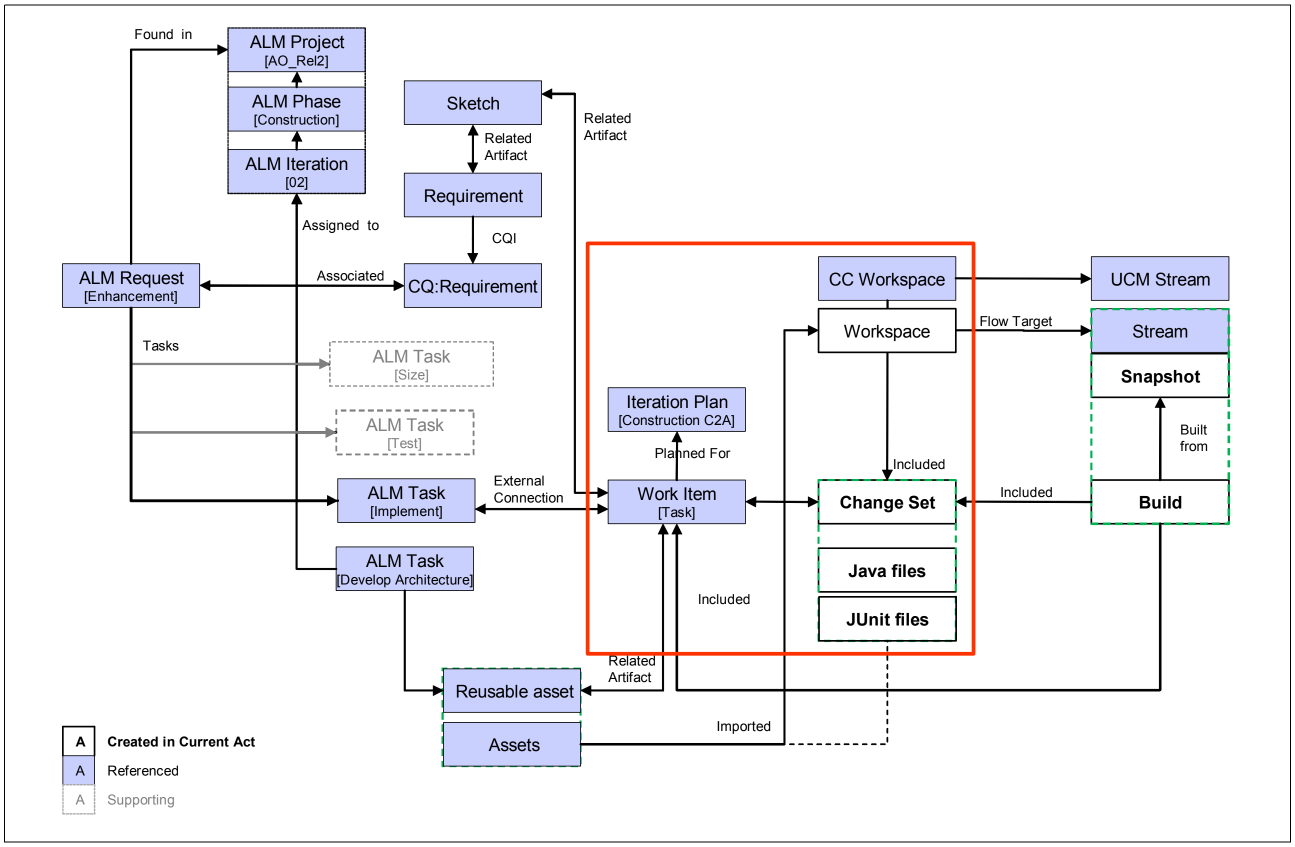


Figure 7-37 Traceability established from source code changes in the workspace to the change set and work item that Diedrie is working on

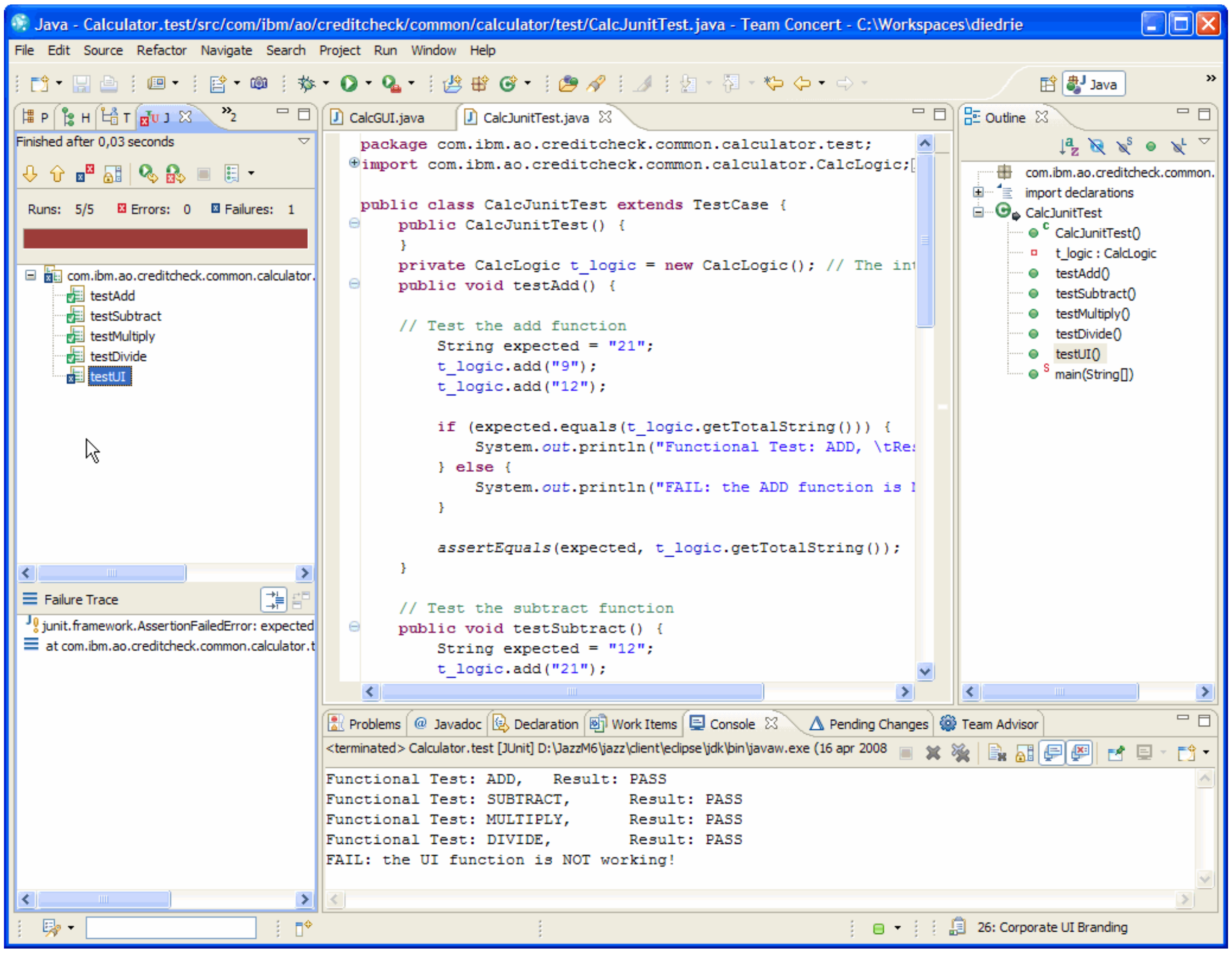
Diedrie uses test-driven development

Goal: The goal is to use tests as specifications for software changes.

Diedrie and the component team use TDD. She focuses first on the JUnit test that validate her changes. Diedrie uses the built-in JUnit capabilities in the Eclipse Java perspective. For each Java project, the team uses the practice to keep a JUnit project with the unit tests for the code project. Both code and test projects are contained and managed in the same SCM component, which allows the team to consistently develop and deliver changes to code and unit tests.

The Corporate UI Branding reusable asset provides Diedrie with a standard set of test cases to validate that a form conforms to the standards and uses the standard controls. See Figure 7-31 on page 268. For each form, she iteratively adds or updates the JUnit test cases to include the added test statements. She runs her updated test case and confirms that the test fails, as exemplified in Figure 7-38.

After ensuring that her test case is catching the UI Branding issue, Diedrie proceeds by updating the Java code. She opens the project in the Java perspective and applies the design pattern for the UI changes, builds her changes, and re-runs her test. She proceeds iteratively until all forms are updated and all unit tests pass.



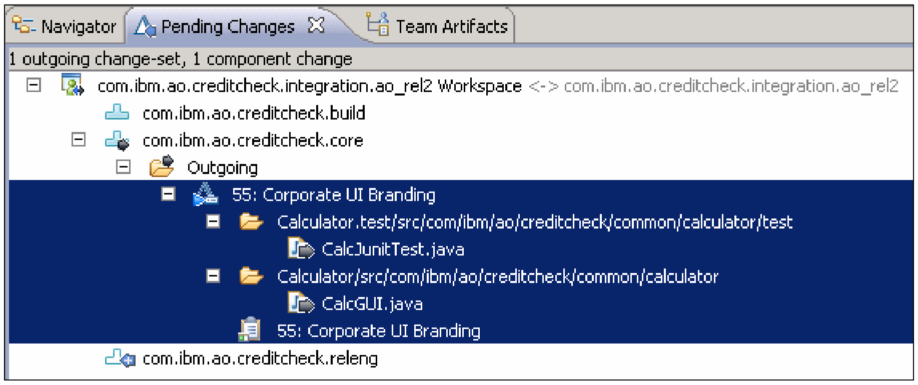
```
package com.ibm.ao.creditcheck.common.calculator.test;
import com.ibm.ao.creditcheck.common.calculator.CalcLogic;[

public class CalcJunitTest extends TestCase {
    public CalcJunitTest() {
    }
    private CalcLogic t_logic = new CalcLogic(); // The int
    public void testAdd() {

    // Test the add function
        String expected = "21";
        t_logic.add("9");
        t_logic.add("12");

        if (expected.equals(t_logic.getTotalString())) {
            System.out.println("Functional Test: ADD, \tRe
        } else {
            System.out.println("FAIL: the ADD function is 
        }

        assertEquals(expected, t_logic.getTotalString());
    }

    // Test the subtract function
    public void testSubtract() {
        String expected = "12";
        t_logic.add("21");
```

```
junit.framework.AssertionFailedError: expected
at com.ibm.ao.creditcheck.common.calculator.t
```

```
<terminated> Calculator.test [JUnit] D:\JazzM6\jazz\client\eclipse\jdk\bin\javaw.exe (16 apr 2008
Functional Test: ADD,   Result: PASS
Functional Test: SUBTRACT,      Result: PASS
Functional Test: MULTIPLY,      Result: PASS
Functional Test: DIVIDE,        Result: PASS
FAIL: the UI function is NOT working!
```

Figure 7-38 Diedrie practicing TDD and running her JUnit test cases to validate her changes

Because Diedrie has updated her Java code and JUnit test projects, Eclipse manages her files in her local workspace. As previously discussed, Rational Team Concert seamlessly performs the housekeeping of changes to the workspaces and establishes traceability between work items and the induced changes. The changes made by Diedrie to the Java code and JUnit test are visible in the Pending Changes view. Figure 7-39 shows the assembly of change set, associated work items, and drill-down into source code changes for individual files.



1 outgoing change-set, 1 component change

```
com.ibm.ao.creditcheck.integration.ao_rel2 Workspace <-> com.ibm.ao.creditcheck.integration.ao_rel2
    com.ibm.ao.creditcheck.build
    com.ibm.ao.creditcheck.core
        Outgoing
            55: Corporate UI Branding
                Calculator.test/src/com/ibm/ao/creditcheck/common/calculator/test
                    CalcJunitTest.java
                Calculator/src/com/ibm/ao/creditcheck/common/calculator
                    CalcGUI.java
                55: Corporate UI Branding
    com.ibm.ao.creditcheck.releng
```

Figure 7-39 The changes made by Diedrie associated and prepared to be delivered and resolved with the Corporate UI Branding work item

Diedrie runs a personal build

Diedrie must minimize the risk of breaking the team integration when delivering her changes.

- She merge all recent code changes from the team.
- She test a full integration build in her local sandbox.

Keeping up with team changes

Goal: The goal is to develop as a collaborative team, not in a silo.

In steady state development, the team continuously delivers changes to the integration stream. On a regular basis, Diedrie must keep up with incoming changes to avoid major updates and merging, or at worse rework, when delivering her changes.

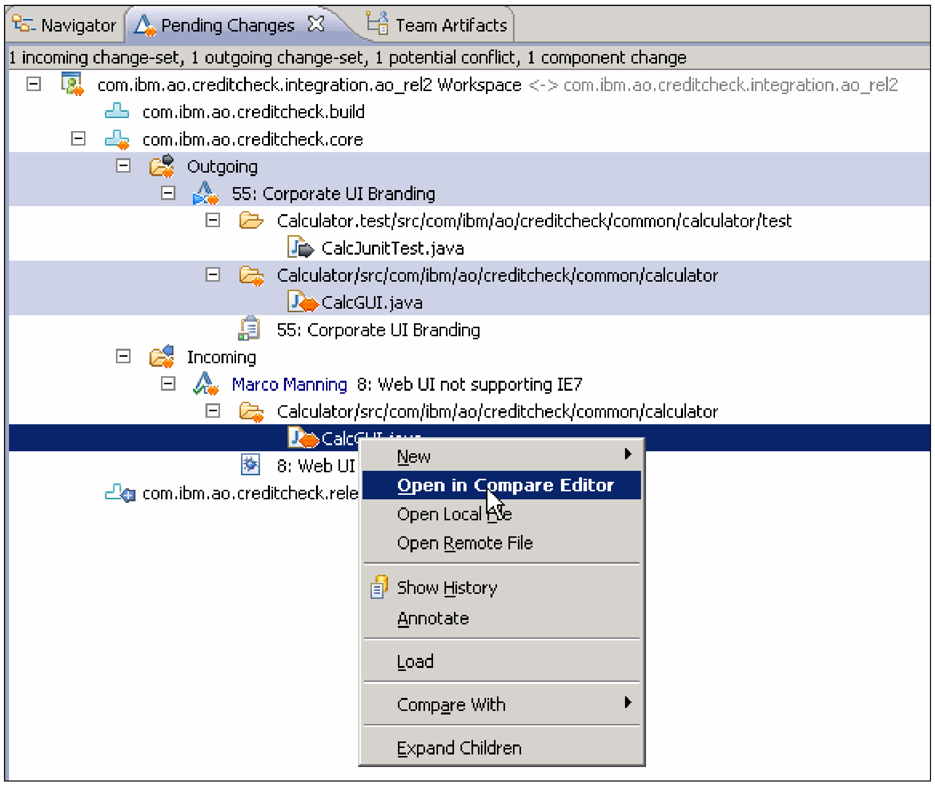
The Pending Changes view in Rational Team Concert helps Diedrie to view the combination of her outgoing changes and the incoming changes that are delivered by her team members. Any conflicts between the incoming and outgoing changes are also highlighted in the Pending Changes view. Rational Team Concert assists by either automatically merging nonconflicting changes or manually resolving more complex conflicting cases by using the Compare Editor.

Diedrie opens the Pending Changes view to accept any delivered changes from the team. She finds that Marco has delivered changes that are in conflict with her code updates (Figure 7-40 on page 276).

To keep up with changes delivered by Diedrie's team, she must perform the following steps:

1. Diedrie opens the Pending Changes view.
2. She expands the components and identifies any incoming and outgoing change sets.
3. For all incoming change sets, she right-clicks the change set and selects **Accept**. Accepting the incoming change sets can create conflicts in a workspace if the same files that have incoming changes also were changed locally.

4. To resolve such conflicts, she chooses **Open in Compare Editor** to view and merge the changes manually.



1 incoming change-set, 1 outgoing change-set, 1 potential conflict, 1 component change

```
com.ibm.ao.creditcheck.integration.ao_rel2 Workspace <-> com.ibm.ao.creditcheck.integration.ao_rel2
  com.ibm.ao.creditcheck.build
  com.ibm.ao.creditcheck.core
    Outgoing
      55: Corporate UI Branding
        Calculator.test/src/com/ibm/ao/creditcheck/common/calculator/test
          CalcJunitTest.java
        Calculator/src/com/ibm/ao/creditcheck/common/calculator
          CalcGUI.java
      55: Corporate UI Branding
    Incoming
      Marco Manning 8: Web UI not supporting IE7
        Calculator/src/com/ibm/ao/creditcheck/common/calculator
          CalcGUI.java
      8: Web UI
  com.ibm.ao.creditcheck.rele
```

Figure 7-40 Diedrie identifying and resolving incoming changes to her workspace

Running a personal build

Goal: The goal is to test before delivery without breaking the component integration build.

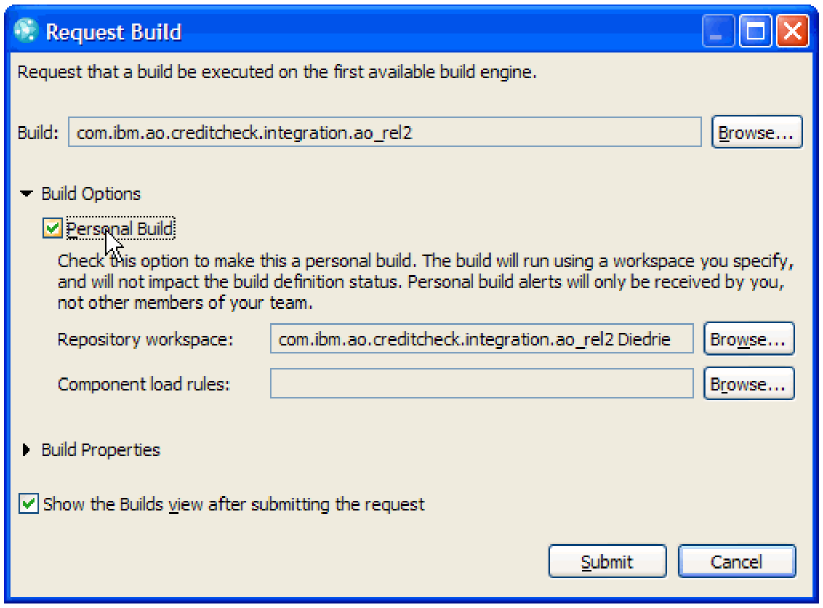
Diedrie is now getting ready to deliver her changes to integration. To minimize the risk of destabilizing the integration stream and the continuous integration builds, she wants to do a test run of a full integration build, before she delivers her changes to the Credit Check integration stream.

Rational Team Concert and the integrated Jazz Build Engine provide the option to run a private build that uses the files in any specified Jazz workspace, instead of the usual build workspace for a given definition. That is, Diedrie requests a build to be run by using her private undelivered code, instead of the code in the team's stream. To take advantage of personal builds, the Jazz Build Engine must be used along with Jazz SCM. In addition, build scripts must be resilient to building from any Jazz workspace.

To run a personal build on her workspace, Diedrie takes the following steps:

1. Diedrie opens the Team Artifacts view and browses for the Builds section.
2. She right-clicks the **com.ibm.ao.creditcheck.integration.ao_rel2** build and chooses **Request Build**.

3. In the Request Build window:
   a. Diedrie expands the **Build Options** and selects the **Personal Build** option.
   b. Optional: She selects the workspace to build from as shown in Figure 7-41.
   c. She clicks **Submit** to run the build.



Request that a build be executed on the first available build engine.

Build: com.ibm.ao.creditcheck.integration.ao_rel2

[✓] Personal Build

Check this option to make this a personal build. The build will run using a workspace you specify, and will not impact the build definition status. Personal build alerts will only be received by you, not other members of your team.

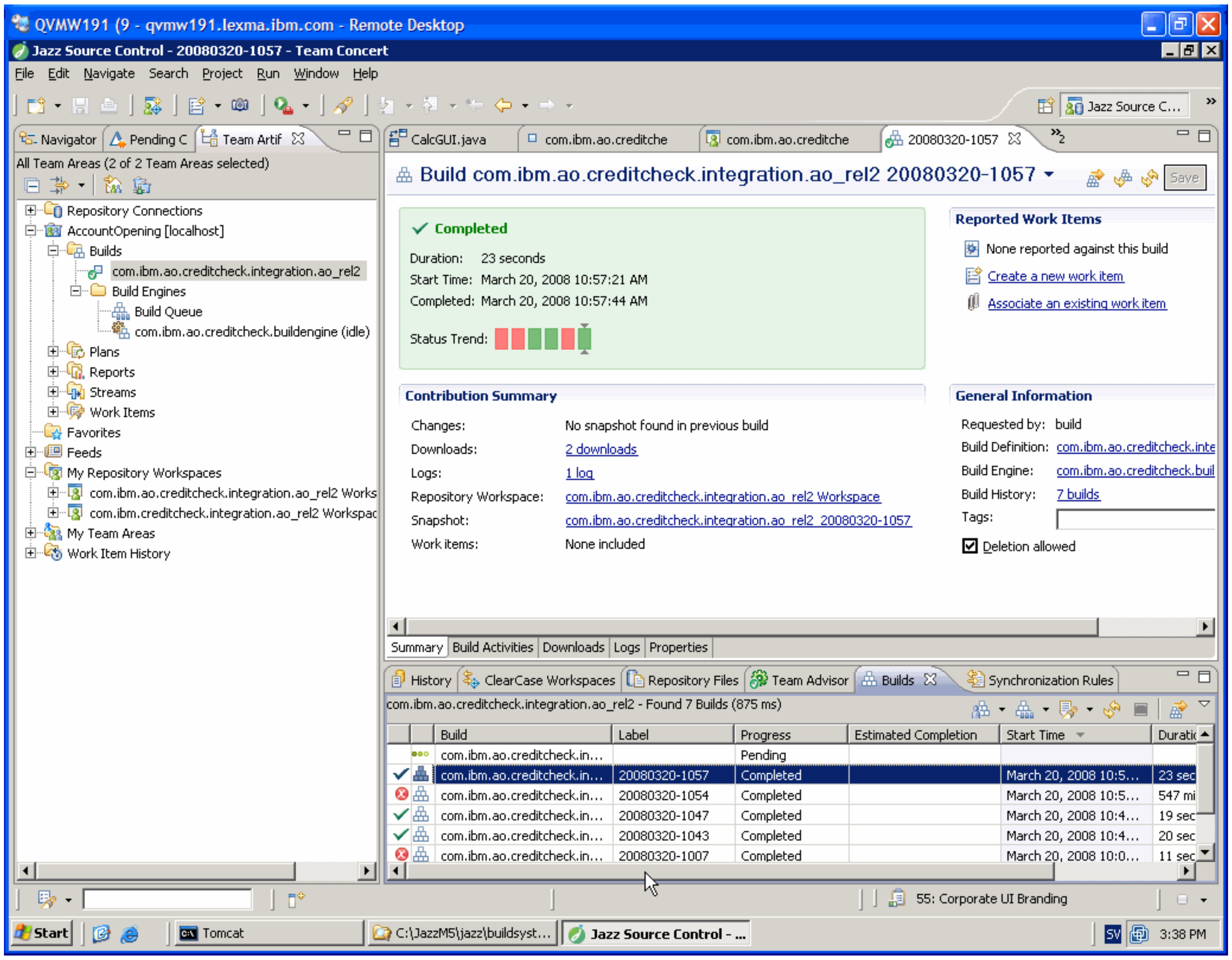
Repository workspace: com.ibm.ao.creditcheck.integration.ao_rel2 Diedrie

Component load rules:

[✓] Show the Builds view after submitting the request

Figure 7-41 Diedrie directing a personal build to be run on her local workspace

Diedrie waits for the build to complete. She monitors the status in the Builds view. After the build is completed, she opens the Build result to access the build log (Figure 7-42).


QVMW191 (9 - qvmw191.lexma.ibm.com - Remote Desktop
Jazz Source Control - 20080320-1057 - Team Concert
File Edit Navigate Search Project Run Window Help
Jazz Source C...
Navigator
Pending C
Team Artif
CalcGUI.java
com.ibm.ao.creditche
com.ibm.ao.creditche
20080320-1057
All Team Areas (2 of 2 Team Areas selected)
Repository Connections
AccountOpening [localhost]
Builds
com.ibm.ao.creditcheck.integration.ao_rel2
Build Engines
Build Queue
com.ibm.ao.creditcheck.buildengine (idle)
Plans
Reports
Streams
Work Items
Favorites
Feeds
My Repository Workspaces
com.ibm.ao.creditcheck.integration.ao_rel2 Works
com.ibm.creditcheck.integration.ao_rel2 Workspac
My Team Areas
Work Item History
Save
Summary
Build Activities
Downloads
Logs
Properties
History
ClearCase Workspaces
Repository Files
Team Advisor
Builds
Synchronization Rules
55: Corporate UI Branding
Start
Tomcat
C:\JazzM5\jazz\buildsyst...
Jazz Source Control - ...
3:38 PM

Build com.ibm.ao.creditcheck.integration.ao_rel2 20080320-1057

✓ Completed

Duration: 23 seconds
Start Time: March 20, 2008 10:57:21 AM
Completed: March 20, 2008 10:57:44 AM

Status Trend: 🟥🟥🟩🟩🟥🟩

Reported Work Items

None reported against this build
Create a new work item
Associate an existing work item

Contribution Summary

Changes:	No snapshot found in previous build
Downloads:	2 downloads
Logs:	1 log
Repository Workspace:	com.ibm.ao.creditcheck.integration.ao_rel2 Workspace
Snapshot:	com.ibm.ao.creditcheck.integration.ao_rel2_20080320-1057
Work items:	None included

General Information

Requested by:	build
Build Definition:	com.ibm.ao.creditcheck.inte
Build Engine:	com.ibm.ao.creditcheck.buil
Build History:	7 builds
Tags:	

☑ Deletion allowed

com.ibm.ao.creditcheck.integration.ao_rel2 - Found 7 Builds (875 ms)

	Build	Label	Progress	Estimated Completion	Start Time	Duratio
	com.ibm.ao.creditcheck.in...		Pending			
✓	com.ibm.ao.creditcheck.in...	20080320-1057	Completed		March 20, 2008 10:5...	23 sec
✗	com.ibm.ao.creditcheck.in...	20080320-1054	Completed		March 20, 2008 10:5...	547 mi
✓	com.ibm.ao.creditcheck.in...	20080320-1047	Completed		March 20, 2008 10:4...	19 sec
✓	com.ibm.ao.creditcheck.in...	20080320-1043	Completed		March 20, 2008 10:4...	20 sec
✗	com.ibm.ao.creditcheck.in...	20080320-1007	Completed		March 20, 2008 10:0...	11 sec

Figure 7-42 Diedrie viewing the result log of her personal build

Diedrie requests a review

Goal: The goal is to collaborate with the team on the code changes.

In this section, we explain how Diedrie requests a review of her code changes prior to delivering the change set to integration.

Requesting a code review

Marco and his team have instituted a development process where a senior team member reviews code changes prior to delivery into the integration stream.

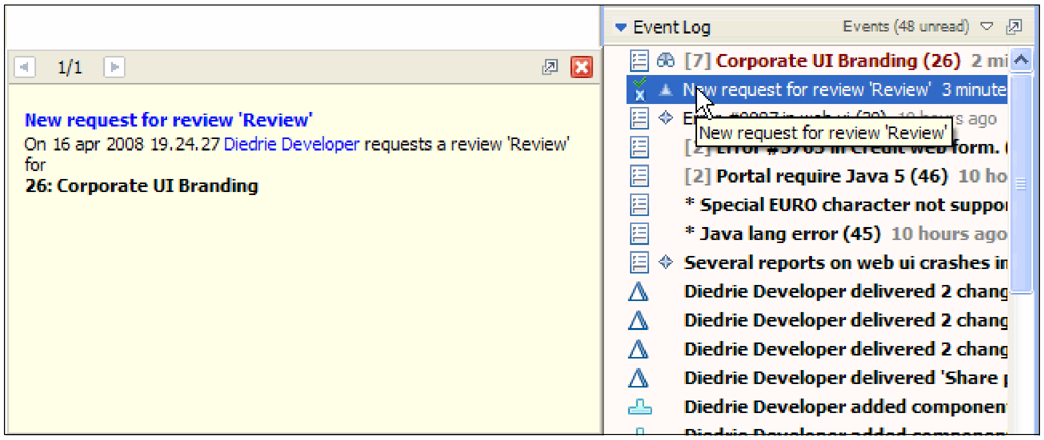
Diedrie has completed her development activities of the predelivery by successfully running her unit tests and a personal build. The personal build has passed both code analysis and test coverage as well as build validation tests.

Rational Team Concert streamlines team processes by adding review items to the work item. To submit a Change Set that is associated with a work item for review, Diedrie performs the following steps:

1. Diedrie opens the Pending Changes view and browses for the outgoing change set.
2. Before sharing the change set with others, Diedrie right-clicks the change set and chooses **Complete**.
3. She right-clicks the change set and chooses **Submit for review…**.
4. Optional: Diedrie selects the **Suspend change sets** option to put further work on hold until the review is completed. This depends on her plans to start work on the next scheduled work item while the review is ongoing.
5. She clicks **Add** to add a reviewer or reviewers. For example, Diedrie adds Marco as her reviewer.
6. She adds a short summary as a comment to the review.
7. She clicks **Finish** to submit the review.

Performing a code review

After the review is submitted, all reviews receive a notification that a review is due as shown in Figure 7-43.



New request for review 'Review'
On 16 apr 2008 19.24.27 Diedrie Developer requests a review 'Review' for
26: Corporate UI Branding

[7] Corporate UI Branding (26) 2 mi
New request for review 'Review' 3 minute
New request for review 'Review'
[2] Portal require Java 5 (46) 10 ho
* Special EURO character not suppo
* Java lang error (45) 10 hours ago
Several reports on web ui crashes in
Diedrie Developer delivered 2 chang
Diedrie Developer delivered 2 chang
Diedrie Developer delivered 2 chang
Diedrie Developer delivered 'Share
Diedrie Developer added componen

Figure 7-43 Marco receiving a notification to review Diedrie's UI Branding changes

Marco receives his review notification and completes the following steps:

1. He opens the Pending Changes view and browses for current change sets in the workspace.
2. He opens the review request. The request is hosted in the UI Branding work item that contains the change set or change sets to be reviewed.

Reviewing change sets

A reviewer can take a variety of approaches to access the work item and the associated change set or change sets. Two approaches are described in "Diedrie collaborates and validates her design" on page 269. Here, it might be sufficient for Marco to do a brief review of the code changes and look for key validation points. In his review, Marco typically looks for the following items:

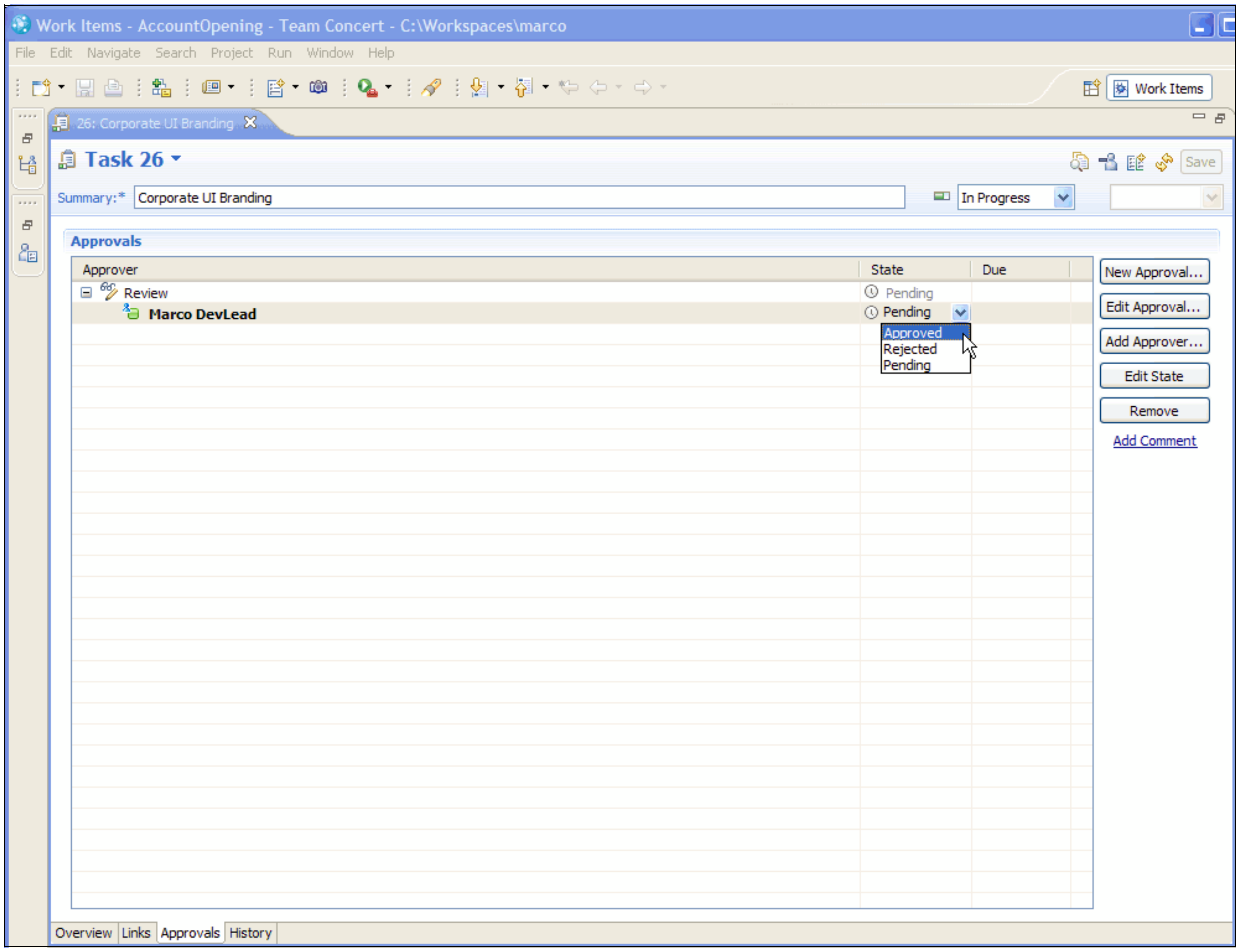
- Whether the code is obscure or hard to understand
- Whether the code is commented sufficiently, especially for Javadoc™ for APIs, but also any obscure implementation code

- Whether the code duplicates any code that already exists and if there are opportunities to share more code
- Whether there are any especially long or confusing methods that should be broken up
- Whether there are any “red flags” from a performance perspective
- Whether there is a cleaner, simpler, or faster way to do the implementation
- Whether the use of exceptions is appropriate
- Whether the code follows established project conventions
- Whether sufficient unit tests were delivered along with the new code

Approving the work item changes

When Marco is confident in the accuracy and stability of the changes, he can approve the work item and the associated change set or change sets. To approve the review, Marco completes the following steps:

1. Marco opens the reviewed work item and switches to the **Approval** tab.
2. Under Review UI Branding, he selects his approval entry.
3. He sets the state to **Approved** as shown in Figure 7-44.
4. He clicks **Save** to persist his changes.


Work Items - AccountOpening - Team Concert - C:\Workspaces\marco
File Edit Navigate Search Project Run Window Help
Work Items
26: Corporate UI Branding
Task 26
Save
Approvals
New Approval...
Edit Approval...
Add Approver...
Edit State
Remove
Add Comment
Approved
Rejected
Pending
Overview Links Approvals History

Summary:* Corporate UI Branding   In Progress

Approver	State	Due
Review	Pending	
Marco DevLead	Pending	

Figure 7-44 Marco approving the review of the work item

After the review is completed, Diedrie receives a notification that the changes have been approved. She can now resume her suspended change set or change sets and proceed with the delivery to the integration stream.

7.3.5 Diedrie delivers her changes and builds the component

Synopsis: Diedrie is now ready to deliver her changes to the integration stream and resolve her work assignment. She makes sure that her workspace points to the integration stream and then delivers her changes. The delivery includes her changes to the application code, the unit tests, and changes to the build and validation scripts.

To catch any issues with her delivery as soon as possible, she requests a new integration build to be run. She awaits the build result and validates that it completed successfully.

To complete her work Diedrie opens her work assignment. She adds summary comments on the resolution, adds an entry for the milestone release notes, changes the state of her work to resolved, and saves her changes.

This scene involves the following tasks as shown in Figure 7-45:

- Delivering changes for integration
- Monitoring continuous builds
- Delivering daily changes for solution integration

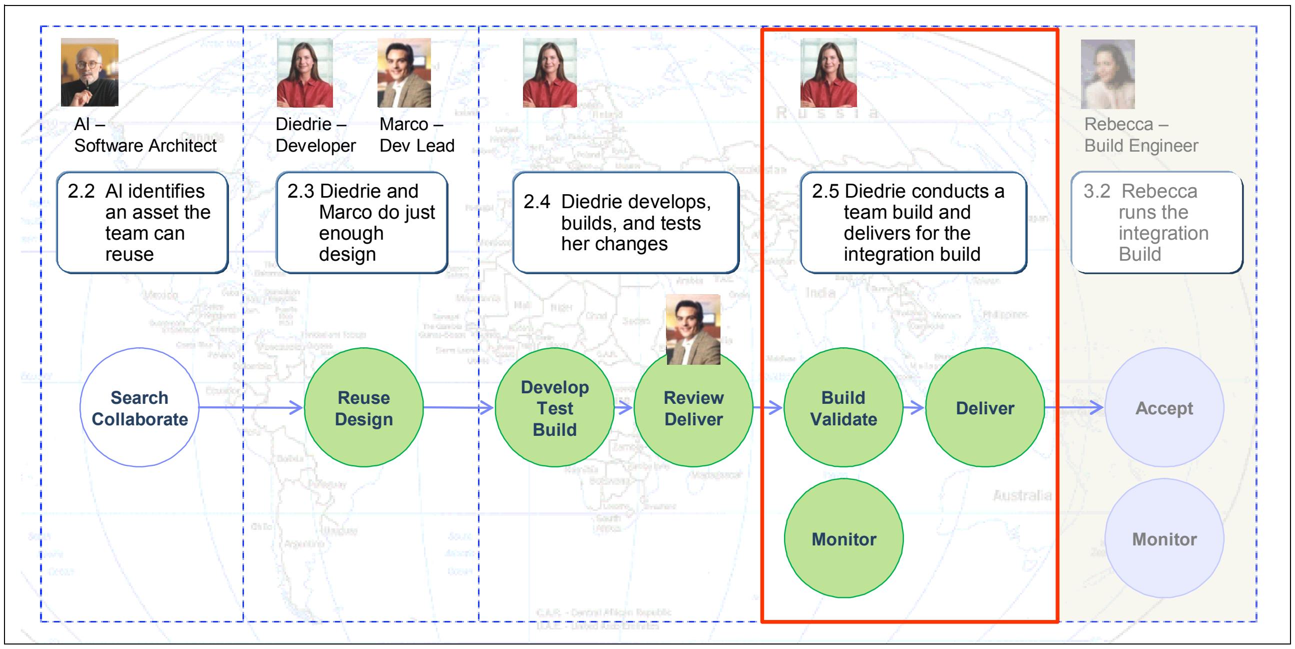

Al – Software Architect
Diedrie – Developer
Marco – Dev Lead
Rebecca – Build Engineer
2.2 Al identifies an asset the team can reuse
2.3 Diedrie and Marco do just enough design
2.4 Diedrie develops, builds, and tests her changes
2.5 Diedrie conducts a team build and delivers for the integration build
3.2 Rebecca runs the integration Build
Search Collaborate
Reuse Design
Develop Test Build
Review Deliver
Build Validate
Deliver
Monitor
Accept
Monitor

Figure 7-45 Diedrie delivering her changes to the integration stream and monitoring the build

Figure 7-46 show the life-cycle asset and relationships that are impacted in this scene.

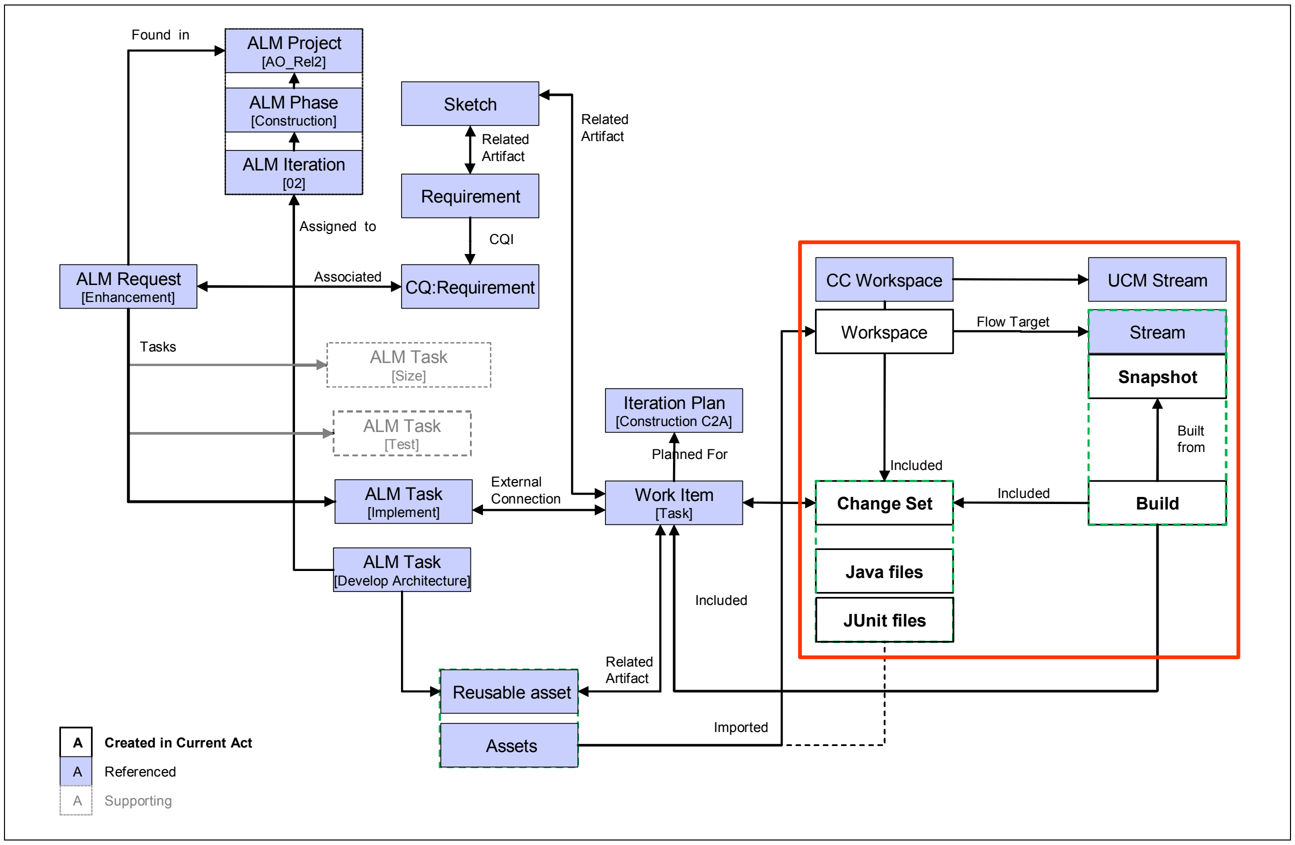


Figure 7-46 Traceability established from the build to the change set or change sets and work item included in the snapshot created by the build

Diedrie delivers her changes for integration

Goal: The goal is to make code changes available to the team and to the integration builds.

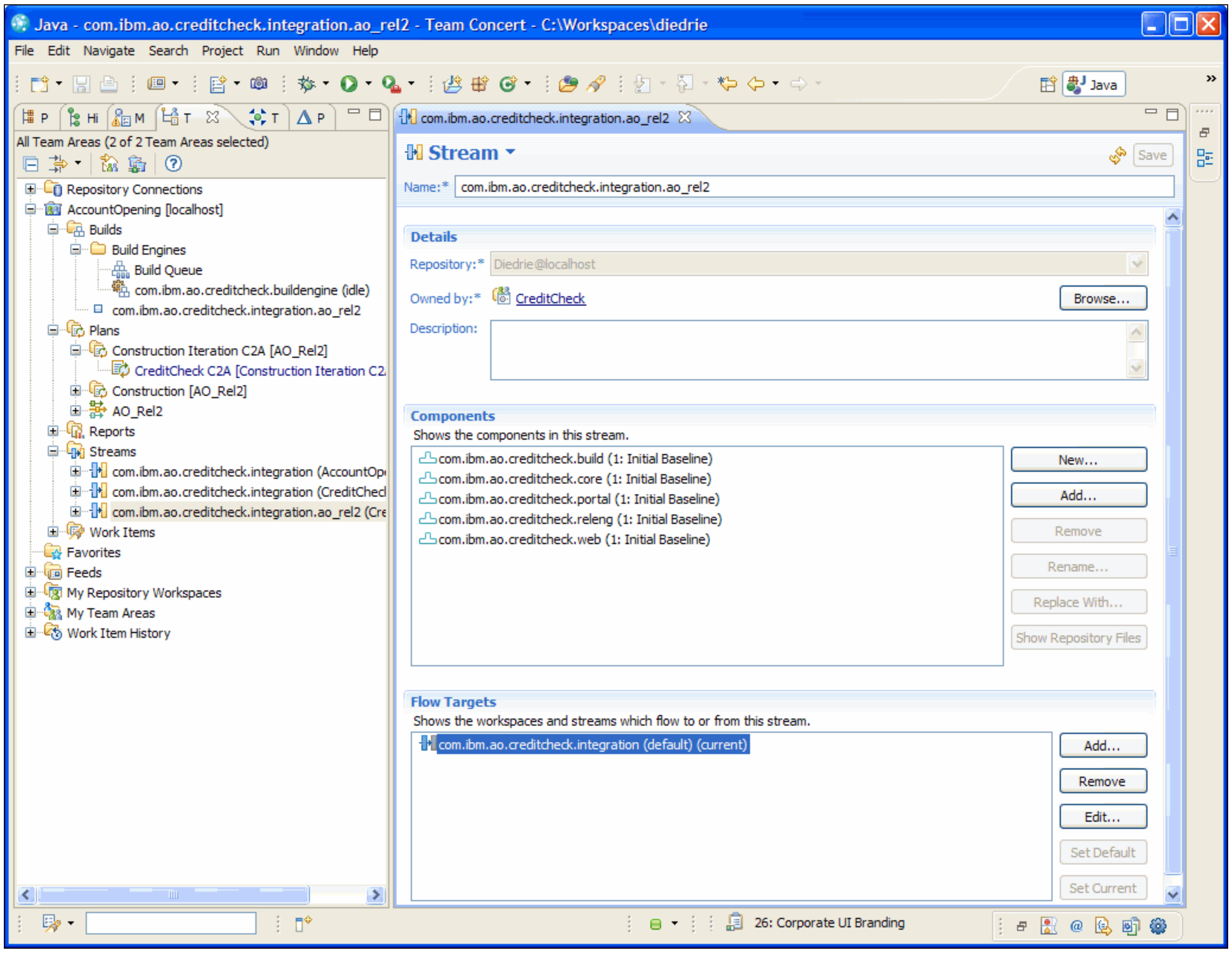
Diedrie is now ready to deliver the UI Branding work item and the changes developed for this enhancement.

Managing the stream with flow targets

Rational Team Concert uses a concept of *flow targets* to manage the stream from which changes are accepted and delivered. In this chapter, we have not discussed the use of flow targets but simply flowing to and from the CreditCheck component integration stream. By default, the flow target is set when creating a workspace.

The flow target setting is easily viewed in Rational Team Concert. In the Pending Changes view, the name of the workspace is preceded with a hyphen enclosed in angle brackets (**<->**) and the current flow target. Figure 7-40 on page 276 shows that Diedrie's workspace is flowing with the com.ibm.ao.creditcheck.integration.ao_rel2 integration stream. The Workspace Editor view also has a section for viewing and managing the flow target or flow targets of a workspace as shown in Figure 7-47 on page 283.

Diedrie intends to deliver her UI Branding work item to the integration stream so that no flow target changes are required. However, later when she delivers to the Solution Integration stream, she must use flow targets for activity.


Java - com.ibm.ao.creditcheck.integration.ao_rel2 - Team Concert - C:\Workspaces\diedrie
File Edit Navigate Search Project Run Window Help
Java
P
Hi
M
T
T
P
All Team Areas (2 of 2 Team Areas selected)
Repository Connections
AccountOpening [localhost]
Builds
Build Engines
Build Queue
com.ibm.ao.creditcheck.buildengine (idle)
com.ibm.ao.creditcheck.integration.ao_rel2
Plans
Construction Iteration C2A [AO_Rel2]
CreditCheck C2A [Construction Iteration C2
Construction [AO_Rel2]
AO_Rel2
Reports
Streams
com.ibm.ao.creditcheck.integration (AccountOp
com.ibm.ao.creditcheck.integration (CreditChec
com.ibm.ao.creditcheck.integration.ao_rel2 (Cre
Work Items
Favorites
Feeds
My Repository Workspaces
My Team Areas
Work Item History
com.ibm.ao.creditcheck.integration.ao_rel2
Stream
Save
Details
Browse...
Components
New...
Add...
Remove
Rename...
Replace With...
Show Repository Files
Flow Targets
Add...
Remove
Edit...
Set Default
Set Current
26: Corporate UI Branding

Name:* com.ibm.ao.creditcheck.integration.ao_rel2

Repository:* Diedrie@localhost

Owned by:* CreditCheck

Description:

Shows the components in this stream.

com.ibm.ao.creditcheck.build (1: Initial Baseline)
com.ibm.ao.creditcheck.core (1: Initial Baseline)
com.ibm.ao.creditcheck.portal (1: Initial Baseline)
com.ibm.ao.creditcheck.releng (1: Initial Baseline)
com.ibm.ao.creditcheck.web (1: Initial Baseline)

Shows the workspaces and streams which flow to or from this stream.

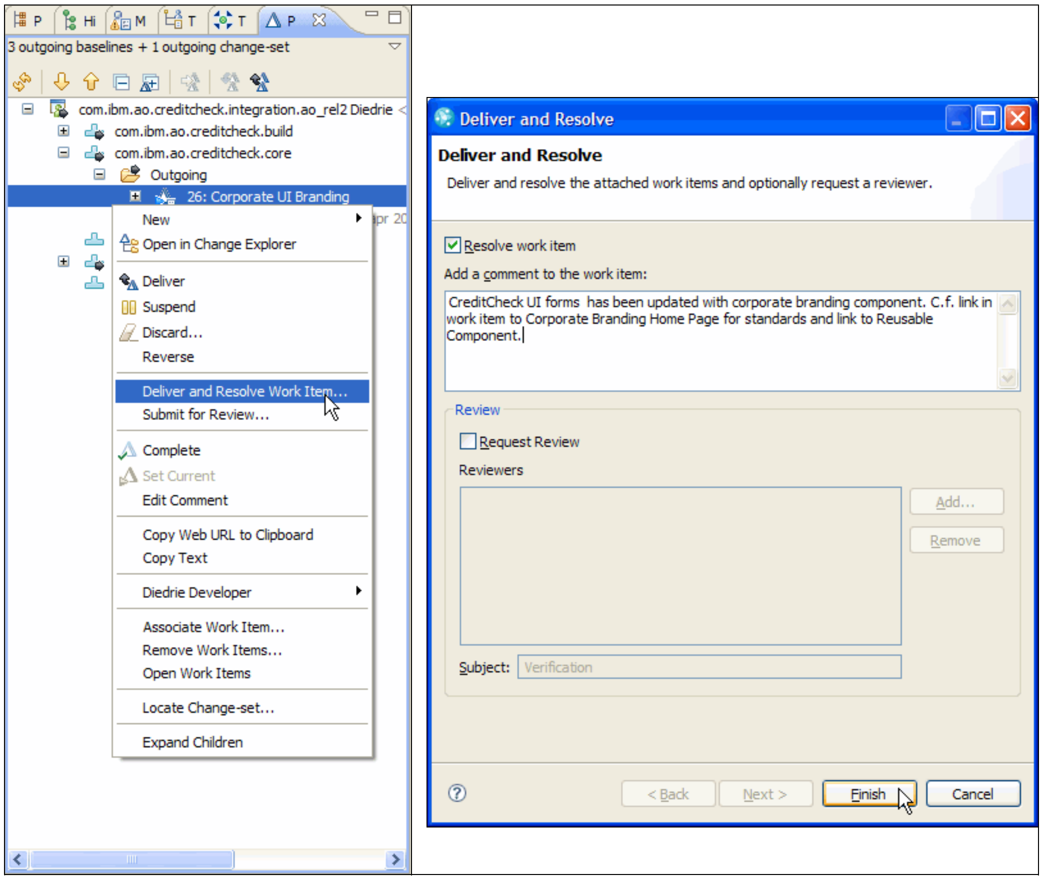
com.ibm.ao.creditcheck.integration (default) (current)

Figure 7-47 Using the Workspace Editor to manage the flow target or flow targets

Delivering and resolving the work item

To deliver the changes, Diedrie completes the following steps:

1. Diedrie opens the Pending Changes view and browses the change set or change sets in the tree.
2. She right-clicks each change set and chooses **Deliver and Resolve Work Item** as shown in Figure 7-48 on page 284.
3. She selects the **Resolve the associated Work Item after delivery** option and enters a resolution comment to the work item.
4. She clicks **Finish** to complete the delivery and mark the work item resolved.



Deliver and Resolve

Deliver and Resolve

Deliver and resolve the attached work items and optionally request a reviewer.

[✓] Resolve work item

Add a comment to the work item:

CreditCheck UI forms has been updated with corporate branding component. C.f. link in work item to Corporate Branding Home Page for standards and link to Reusable Component.

Review

[ ] Request Review

Reviewers

Subject: Verification

Figure 7-48 Diedrie delivering a change set

Diedrie monitors the continuous builds

Goal: The goal is to act on build and integration instabilities.

Diedrie has the responsibility in the component team to supervise builds, but it is a shared responsibility across the entire team to act on build failures. Build notifications are available to all team members and give transparency to build stability for the entire team.

The build definition is set to continuously build with 30-minute intervals, but only if changes have been delivered to the CreditCheck component integration stream.

Diedrie has multiple options to monitor the build health:

- See fly-in build notifications.
- See build events in the Team Central view.
- Monitor the build queue in the Builds view.

Diedrie uses the fly-in notifications to monitor the build stability. There is a continuous stream of notifications, of which some succeed and some fail. As she sees the builds repeatedly fail, she turns her attention to the Build view and starts to analyze the root cause of the build failures by using individual build items and build logs. Figure 7-42 on page 278 shows how Diedrie can view the build status, trend, and access the build log.

Diedrie delivers daily changes for solution integration

Diedrie has the role in team to deliver the changes from Rational Team Concert to the application integration by using Rational ClearCase. She uses the ClearCase Connector and ClearCase Synchronized Streams in Rational Team Concert to perform the delivery.

Using the ClearCase Connector

Goal: The goal is to establish a software supply chain that delivers component changes for solution integration.

ClearCase Synchronized Streams provide seamless interoperation between Rational Team Concert and Rational ClearCase. The ClearCase Synchronized Stream enables a team working in Rational Team Concert to access Eclipse projects that are under Rational ClearCase source control, or deliver changes made in Rational Team Concert to Rational ClearCase users of UCM components or versioned object base (VOB) folders.

Workspaces for ClearCase Synchronized Streams are used as ordinary repository workspace. However, they are created with an additional flow target that simplifies the resolution of conflicts that are introduced by incoming change sets that imported from Rational ClearCase.

ClearCase Synchronized Streams are not designed to import every version of an artifact from Rational ClearCase to a Jazz source control. Rather, they provide support for importing the versions that are selected by a Rational ClearCase view configuration (a UCM stream or a branch and label pair) to a Jazz stream. They also provide support for exporting change sets from that stream to the Rational ClearCase view, where they are checked in as new versions. This stream-based approach takes advantage of similarities between Rational ClearCase and Jazz source control to facilitate ongoing work in both environments.

For the Credit Check team, the project is using ClearCase Synchronized Stream for each major release stream, for example com.ibm.ao.creditcheck.integration.ao_rel2, to deliver changes to the AO_Rel2 UCM project. The configuration of ClearCase Synchronized Streams is further described in “Configuring ClearCase Connectors” on page 308.

Delivering to solution integration

The Credit Check team delivers their components to solution integration at regular intervals. The solution team has agreed on the following principles that guide these deliveries:

- Deliveries are owned by the delivering team.
- Deliveries are made into the solution integration stream in Rational ClearCase.
- Only stable baselines that are successfully built should be delivered.
- Deliveries should be made at best daily or at least weekly.

To deliver the CreditCheck component to the solution integration stream, Diedrie completes the following steps:

1. Diedrie collaborates with other team members and ensures that all changes have been delivered.
2. She runs a build to validate the stability of the component integration stream.
3. She creates a new repository workspace, if needed, from the snapshot created by the build.
4. She creates a new baseline for all CreditCheck components in the integration stream.
5. She changes the flow target of the workspace to the ClearCase Synchronized Integration Stream.

6. She delivers the new baseline to the ClearCase Synchronized Integration Stream.
7. She removes the repository workspace.

Accepting changes from solution integration

The Credit Check team depends on other components in the Account Opening solution and must accept changes from ClearCase into Rational Team Concert. The configuration of the components and Eclipse projects in the ClearCase Synchronized Stream enable the team to accept changes by using the synchronized stream.

To accept the changes, Diedrie performs the following steps:

1. Diedrie receives a notification from Rebecca that the daily or weekly integration is complete.
2. She creates a new repository workspace, if needed, from the ClearCase Synchronized Stream.
3. She accepts the new changes from ClearCase into the workspace.

7.4 Life-cycle collaboration

The collaborative development discipline touches on many of the core development assets in ALM. Figure 7-5 on page 238 shows a summary of the assets and their part in the workflow.

Figure 7-49 shows the life-cycle assets in collaborative development and their dependencies.

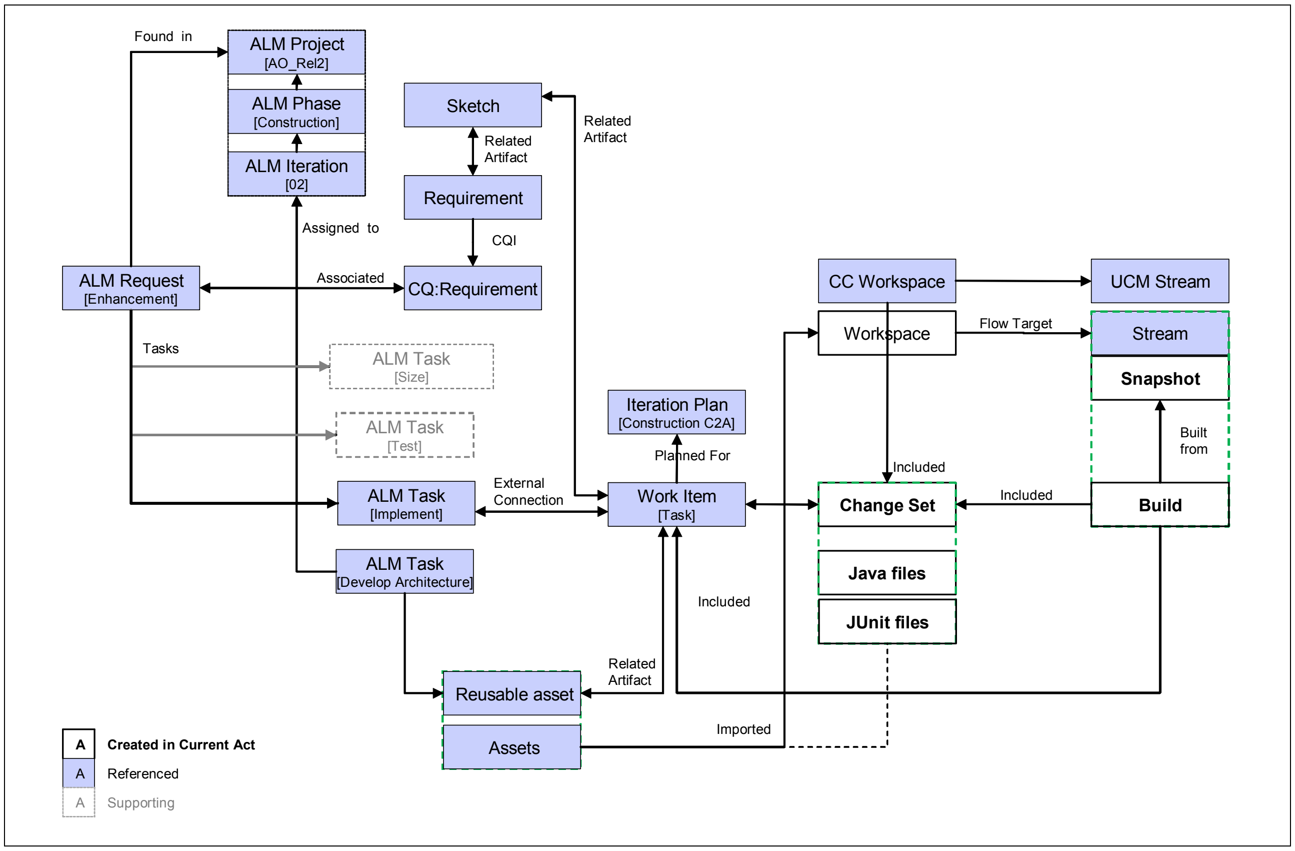


Figure 7-49 Life-cycle assets in collaborative development

Work items

The work items are key drivers for the scenario. The work items carry the following ALM information:

- Provide traceability to the iteration plan for the team and hence to the responsibilities and schedule for the assigned team member
- Provide traceability to the work assignment committed by the component team and to the ALMActivity and ALMTask that govern the solution iteration plan
- Provide traceability to ALMRequest for stakeholder needs
- Provide traceability to the change sets that contain the code changes resolving the work item
- Contain the approvals that govern delivery and quality of change in the software supply chain
- Become traceable from stream, baseline, and snapshot histories
- Become traceable from build results that enumerate the change sets included in the build

By collecting metrics on work items, significant insight can be achieved on project and iteration health. See 7.5.1, “Measuring success with Rational Team Concert” on page 288, which elaborates on the success metrics based on work items.

Change sets

Change sets form containers for source code changes that are delivered as an atomic unit. Change sets can live in isolation, but from an ALM perspective they have the following characteristics:

- Provide traceability to the changes resolving a work item
- Become the unit of delivery between workspaces and streams, for example integration streams
- Become traceable from snapshots and builds to the change sets that are included in a build

The capability in Rational Team Concert to exchange and collaborate on change sets is key to the collaborative development capabilities. Examples of such collaboration is demonstrated in “Diedrie collaborates and validates her design” on page 269.

Source code (Java and JUnit)

Source code changes saved to a workspace become traceable as part of a change set. There are multiple views in Rational Team Concert to browse and compare code changes. In “Diedrie prototypes the design” on page 268, we show one scenario for browsing change sets and using the Compare Editor for viewing code level changes.

Workspaces

The workspace is not a persistable ALM asset but plays an important role for hosting and collaboration on change sets. The ability to manage change sets by applying and discarding them from workspaces is key to the collaborative development capabilities.

Streams

Streams are key to manage change for a version of a solution, such as a release. Streams provide traceability to the history of the following items:

- Delivered changes sets
- Baselines and snapshots

Snapshot

The snapshot item provides traceability to the version of files in a set of components at a given time, such as at the time of a build.

Builds

The build items are key drivers for the scenario. They provide traceability to the following ALM information:

- The snapshot taken at the time of the build
- The new work items and changes sets that are delivered into the build
- The artifacts constructed by the build
- The build result and build logs

ClearCase Synchronized Streams

ClearCase Synchronized Streams provide cross-platform traceability from Rational Team Concert to Rational ClearCase.

7.5 Planning and measuring success in collaborative development

In this section, we discuss how to measure success in collaborative development by using Rational Team Concert.

7.5.1 Measuring success with Rational Team Concert

In this section, we discuss the capabilities in Rational Team Concert to monitor health and measure success.

Agile planning

Agile planning principles dictate that just enough planning is conducted in order to deliver working software to stakeholders. Teams should not be burdened with the production of expansive project plans or documentation that does not contribute to project delivery. Planning must be directly tied to project execution.

In agile planning, the team is given the requirements or work stack to be delivered in the iteration by the product owner. The agile plan is captured in Rational Team Concert as an iteration plan. Given this release scope, the team self-configures the following items:

- Capture of the stories that present the customer value view of the requirements
- Elaboration of the work that is involved in supporting the stories
- Work assignments and ownership
- Enumeration of perceived risks

Agile planning also empowers the entire project team to actively estimate the time they need to complete assigned tasks. This improves the accuracy and reliability in the iteration plan. The churn that frequently occurs in planning, replanning, and re-estimating that causes many projects to fail is frequently caused by incorrect or unrealistic estimates that were dictated taking a top-down approach with project deadlines.

The estimates are provided in Rational Team Concert as detailed attributes on work items. The estimates are rolled up to parent items, such as the total for each team member, to estimate work for a composite task, or to the entire team iteration. Estimations provide a measurement of the quality of planning for an iteration based on how much of the work assigned to the iteration has been estimated. This gives the team a basis for understanding whether the work allocated for the iteration can realistically be completed.

Team transparency

Transparency is a fundamental quality on a truly agile team that is empowered with self-direction in order to respond effectively and rapidly to project challenges and changing project needs. Transparency enables teams to "pull" tasks as needed rather than to only have tasks that are "pushed" to them by their manager. The triage and direction provided by the development lead still remain essential to ensure that the project remains focused on delivery commitments, but with transparency and self-direction.

Workload balancing is a key example of team transparency benefits. In a traditional model, the project manager regularly assesses the workload of the project team and assigns work based on his perception of available team capacity. This approach has drawbacks as it shifts focus of work to individuals rather than the productivity and success of the team. That is, team members define their individual work rather than the team working against a particular milestone.

Further, a lack of project transparency and whole team focus encourages project teams and work owners to hide challenges and issues they are facing, working on them in isolation in order to avoid criticism or negative perception. This limits the overall productivity and potential of the collective project team. Project rebalancing can also become bottlenecked by the project manager. Without transparency and a team success culture, project challenges and changes in project work are not addressed until the project manager can reorchestrate the project team.

Complete workload transparency creates trust among team members because it allows the truth to be known that everyone is working hard toward project delivery by way of team workload visibility. It also encourages team members to take responsibility for their role in the team and hold each other accountable for the collective success of the team. Project transparency eliminates excuses for unproductive behavior. Project transparency also improves the overall quality of project planning, scheduling, and estimation. This is especially true for geographically and organizationally distributed teams where work estimates might be far more subjective or prone to distortion.

Rational Team Concert provides a completely transparent team environment that encourages team success. As workloads change, it makes transparent the areas where help is needed and provides team members with the ability quickly jump in and pair to complete work items. Rational Team Concert brings together team responsibility with personal accountability enabling teams to be responsive to the changing needs of the project. Rational Team Concert reinforces the agile manifesto principle: "Responding to change over following a plan."

Examples of supporting team transparency are shown in the following figures:

- Figure 7-13 on page 249, which shows the team Web Dashboard
- Figure 7-17 on page 253, which shows the team iteration plan in the Web view
- Figure 7-18 on page 254, which shows the Team Central view

The Iteration Plan view

Many organizations draft a project plan in one tool and then translate that plan to tasks and work items in the environment that their teams use. This strategy has a number of weaknesses. It burdens the project team and the project manager with the overhead of constantly synchronizing their plans with a change management system and reflecting project progress.

In Rational Team Concert, the project plan and work items are fully integrated. Hence the tasks in the project plan are the same assets as the work items that the team is working against. The tasks on the project plan are the same as the actual work items that the team is working against. This alleviates the burden of project plan synchronization for progress reporting for the entire team.

The following metrics are examples of what is collected from the iteration plan:

- Iteration and team workload and balance
- Risk level from unestimated work
- Work closure rate

Examples of iteration plans are shown in Figure 7-14 on page 250 and Figure 7-15 on page 251.

The Team Central view

The Team Central view is a user customizable real-time view of what is happening in the project. It provides a transparent view into the work of the entire project team. This view empowers team members to take personal responsibility for the success of the team. It shows the collective effort and workload of the entire team and allows you to link the team's workload to your own personal workload.

Team Central provides a section based view, which can be individually configured to show measurements of the key success indicators for each team role. Some general sections apply to all team members and roles, and Rational Team Concert provides, by default, configuration for Team Central.

The following sections are provided in Team Central:

- The *Builds section* shows build events from one or more build definitions. By using this section, team members can watch for build failures.
- The *Open* and *New Work Items sections* show work that is assigned to the individual team member and new work for the team that is not yet assigned. By using these sections, team members can see an overview of the work stack and new assigned work. Note that changes are indicated on the bars, for example 5 (+2).
- The *Team Load section* provides a graphical summary of the total number of work hours that remain before the iteration ends and the amount of estimated work they have assigned for the iteration. By using this section, team members and team leads can see an overview of work balance and overcommit.
- The *Event Log section* shows event notifications for the items or feeds subscribed to. By using this section, team members can stay updated on changes to the project and selected key items.
- The additional sections to the default set (the previous sections in this list) can be configured to report on additional success indicators for the team. This might entail additional work item queries or event notification sources. By using additional queries, team members can track specific items over the project or iteration life cycle.

The My Work view can be configured by adding new sections on News, Events, or Queries. By adding a new query section and dragging a favorite query to the section, the team gains continuous tracking capabilities to important work metrics. Additional News and Events sections give continuous tracking capabilities to external feeds providers, such as external tools that generate events that must be monitored.

Figure 7-18 on page 254 shows an example of the Team Central view.

The My Work view

Rational Team Concert ties the Team Central view to a similar view that is focused on personal project responsibilities called “My Work.” The My Work view shows work items that have been assigned specifically to you. It allows you to create a personal work schedule based on your assigned work items and the effort required to complete them.

The tight integration of work assignment, effort estimation, and planning improves the overall quality of the project plan. These tasks are normally centralized as the primary responsibility of the project manager with selective input from the project team. The constant focus on planning and input of the work owners makes the quality of planning much better. Planning, estimation, and scheduling become an implicit part of project delivery with Rational Team Concert. With Rational Team Concert active planning, estimation, and scheduling, your team can keep up with the changing needs of the project.

The team-centric view of Team Concert enables teams to self-direct and self-adjust in step with the needs of the project.

Figure 7-19 on page 255 shows an example of the My Work view.

The Web Dashboard

Rational Team Concert provides a Web portal for both development team members and external team contributors that demand easy access to project health information and assets by using Web 2.0 usage patterns. The Jazz Web portal is accessed by browsing the Jazz server, for example by using the following URL:

```
https://localhost:9443/jazz/web
```

The Web portal provides access to project areas, server administration, and dashboards. In this section, we focus the discussion on the Web Dashboard.

The Jazz Web Dashboard provides capabilities to configure shared dashboards to publish common project health information or where individuals create and configure dashboards based on individual monitoring needs. By providing guest access, and integrating with enterprise user authentication services, teams can establish a balance between information security, stakeholder access, and information management.

The Jazz Web Dashboard provides multiple tabs where viewlets are added and configured from a range of context areas, for example:

- Personal, team member, and project information
- Preconfigured work item queries, health metrics, and trends
- Preconfigured build health metrics and trends
- General tools to publish notifications, bookmarks, and HTML formatted information

Figure 7-56 on page 300 shows an example of the Jazz Web Dashboard.

7.5.2 Reporting team health with Rational Team Concert

In collaborative development, key reports should be visible and understood by everyone on the project. This knowledge allows everyone on the team to take responsibility for the projects progress and make contributions to correct unhealthy project trends. For example, team members who view a report that shows open work for the current iteration can see that scope creep for the iteration might be a problem if the open enhancements keep rising as the project iteration progresses. They can also sense a quality problem if the open defects keep rising as the iteration progresses. These indicators give everyone on the project a team sense for potential project challenges and project health.

Reporting provides projects with the ability to quickly address project challenges collectively as a team rather than rely solely on the prowess and actions of project managers and stakeholders to correct unhealthy trends. Agile teams have the ability to take team responsibility for challenges and “do their part” to correct unhealthy trends. For example, individuals who look at an overcommitted iteration can see their own individual commitments to understand their contribution to the trend. They might look for a rise in defects and enhancement requests. They might also validate their work estimates for open work. In doing so, they can ask the project manager to help them address such issues as replanning work to other iterations. They can trade or share work with peers by exchanging work items that they can accomplish faster in their area of expertise.

Report data should be collected automatically from direct measurement, and reports should tie real-time status to historical trend information. In this section, we exemplify the reporting in Rational Team Concert with some key success indicators.

For additional information and examples about how to use reports to measure team success, see the Reports section on the Jazz.net Web site (sign-on required):

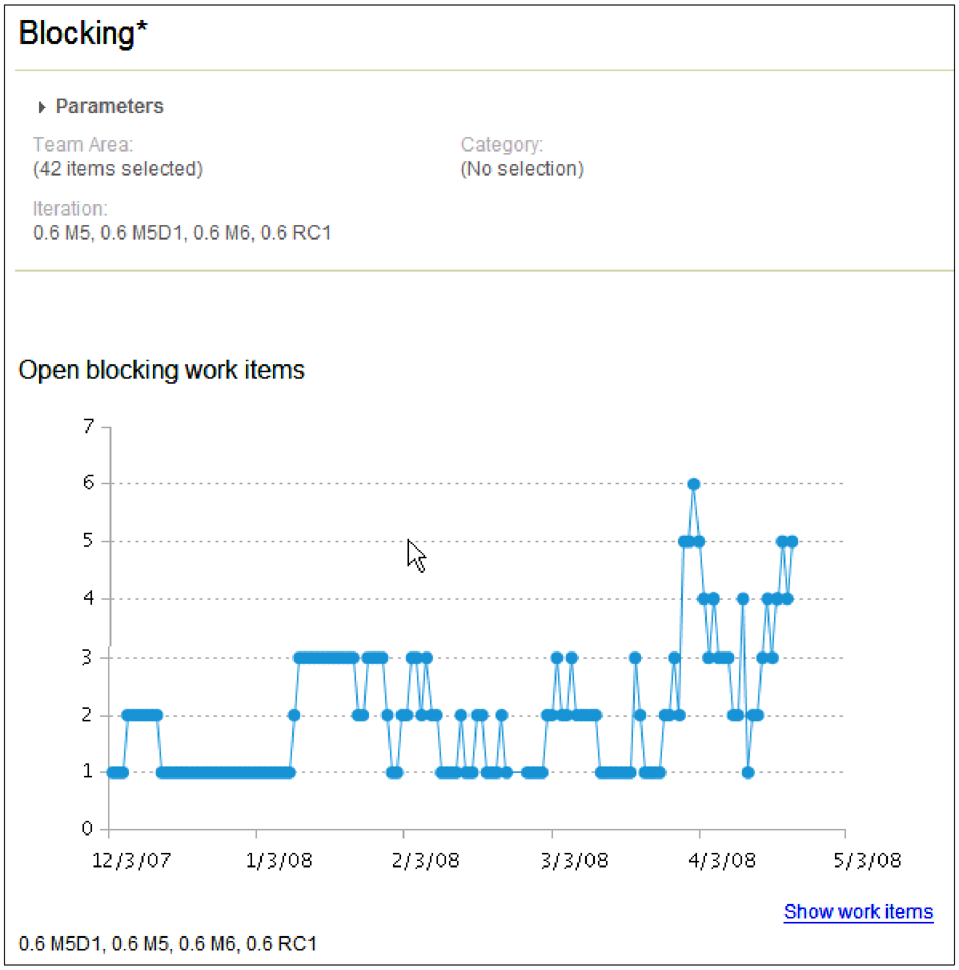
https://jazz.net/jazz/web/projects/Jazz%20Project#action=jazz.viewPage&id=com.ibm.team.reports

Blocking tasks: Keeping an eye on the critical path

Work items that must be completed and resolved before other planned work can be completed are called *blockers*. These work items must be addressed with greater urgency because they prevent the work of other work items from progressing and threaten the on-time delivery of the iteration. Teams must be especially cognizant of work items that are blocked and on the critical path.

Additionally, a rise trend in blocking work items late in the iteration is cause for greater concern. Late stage blocking work items increase the risk of on-time delivery of the iteration. Ideally, you want to keep the number of blocking work items at zero. Blocking work items should be a focus area for status meetings. Since all contributors can get a real-time list of all blocking work items, there should be a culture of collaborating on the resolution of blocking work items to fast track their completion.

In Rational Team Concert, the Blocking report (Figure 7-50) plots all open work items with the blocker severity over time. If an iteration is specified, only those work items planned for that iteration are shown. By using the Blocking report, teams can watch for a high or increasing number of blocking work items close to the end of an iteration, which can indicate that the iteration end date is in danger.


Blocking*

Parameters

Team Area:
(42 items selected)

Category:
(No selection)

Iteration:
0.6 M5, 0.6 M5D1, 0.6 M6, 0.6 RC1

Open blocking work items

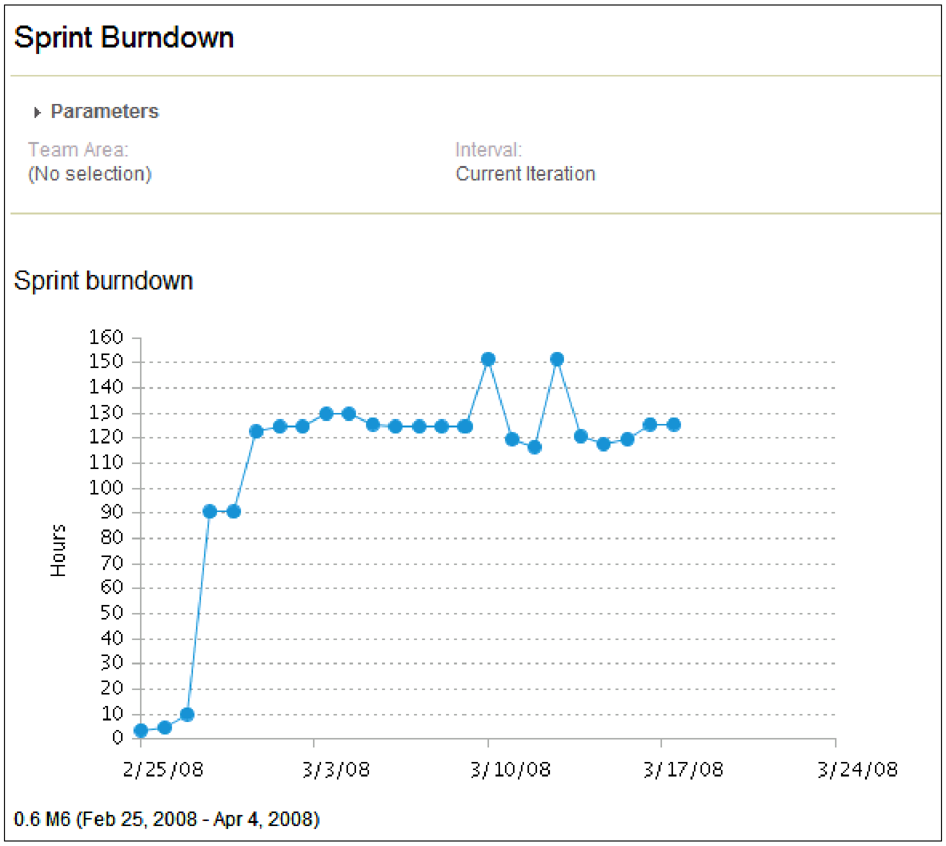
0.6 M5D1, 0.6 M5, 0.6 M6, 0.6 RC1

Figure 7-50 Report on open blocking work items from the Jazz project

Iteration or Sprint Burndown: Timeboxing scope of work

The Sprint Burndown report (Figure 7-51) plots the remaining backlog of work in terms of the time estimated to complete it. Agile development methodologies, such as Scrum, use a burndown to plot the daily progress toward the end of a sprint. Only work items that are open and in progress, and that have an estimate specified, are included in the calculation. Ideally, the chart shows a trend toward zero hours of remaining work as the sprint comes to a close.

The Sprint Burndown report help teams watch for burndown trends that do not approach zero, which can indicate unrealistic planning estimates.



Sprint Burndown

Parameters

Team Area:
(No selection)

Interval:
Current Iteration

Sprint burndown

0.6 M6 (Feb 25, 2008 - Apr 4, 2008)

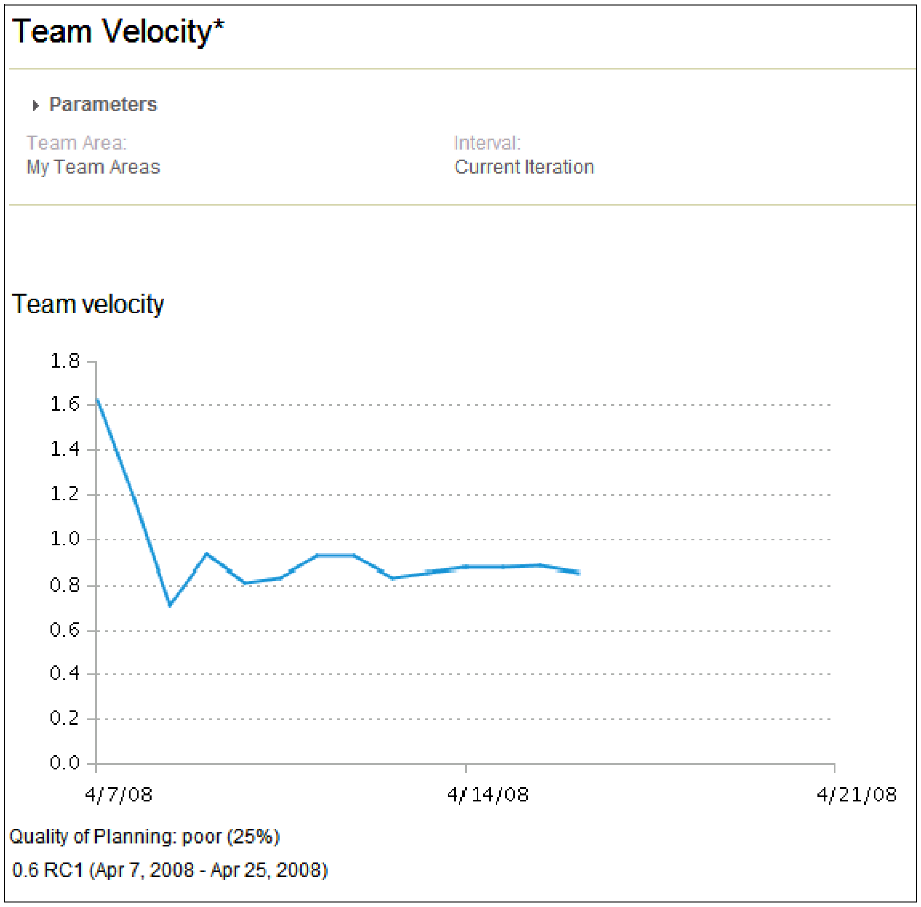
Figure 7-51 Sprint Burndown report from the Jazz project

Team Velocity: Checking the reality of your planning and estimating

The Team Velocity report (Figure 7-52) plots the velocity of a team over time and measures how effective a team is at estimating and planning for the current iteration. In Rational Team Concert, each work item can specify a time estimate and, after the fact, the actual time spent. The velocity is defined as the estimated time divided by the actual time for all closed work items.

Ideally, a velocity trend over time of close to 1.0 is considered good. This means that the estimates are realistic. For this report, only work items that are closed and that have both an estimate value and a time spent value specified are included in the calculation.

The Team Velocity report helps teams watch for trend velocities that are considerably greater than 1.0. This can indicate that the time estimates are becoming too high and some modifications must be made when making such estimates in the future. Likewise, velocities considerably less than 1.0 indicate that actual time spent on work is much greater than estimated. Also, project velocity can help the team understand the credibility of future work estimates and as a result, the integrity of the project schedule.


Team Velocity*

Parameters

Team Area: My Team Areas

Interval: Current Iteration

Team velocity

Quality of Planning: poor (25%)

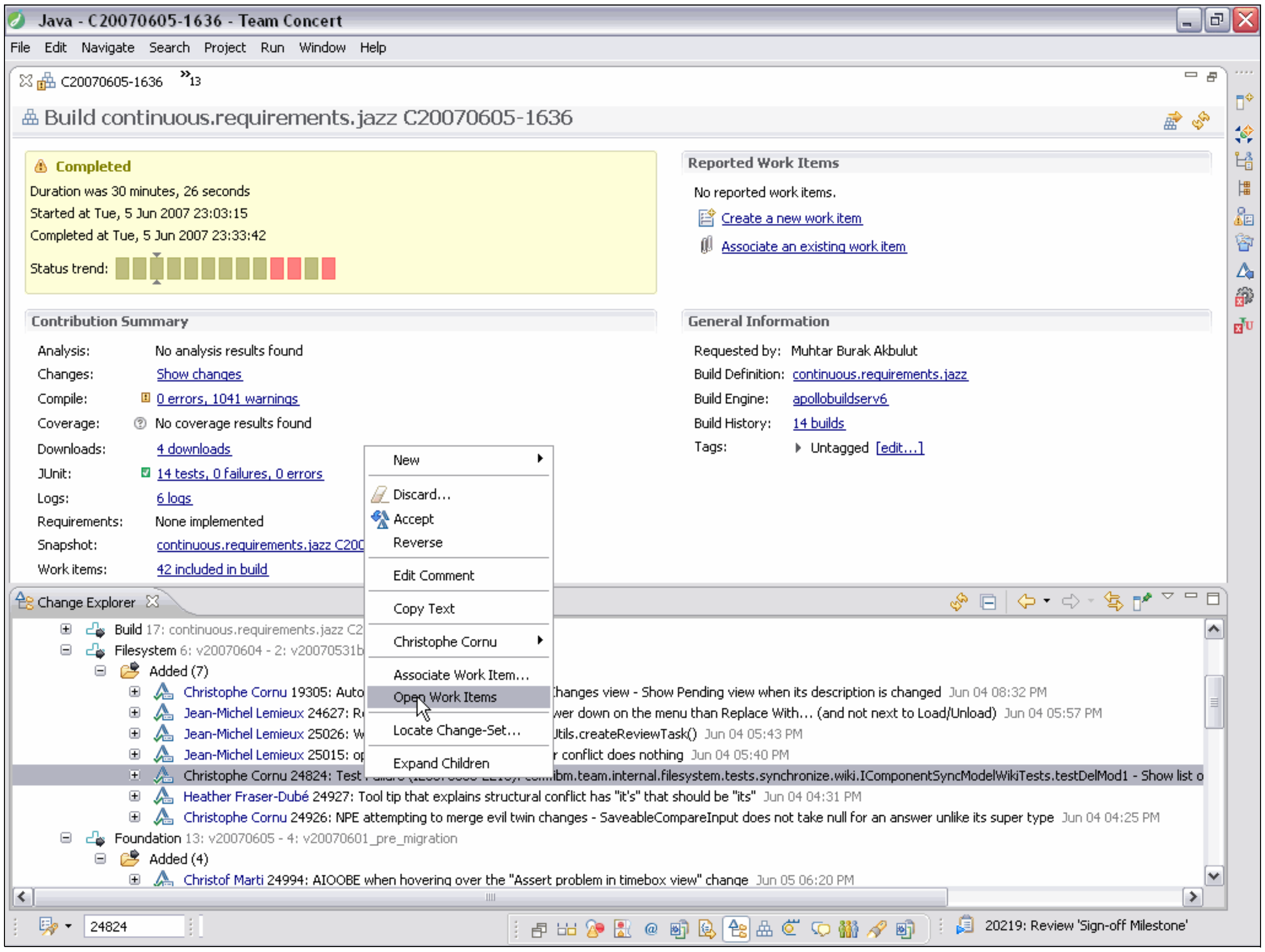
0.6 RC1 (Apr 7, 2008 - Apr 25, 2008)

Figure 7-52 Team Velocity report from the Jazz project

Build results and trend: Working software for the team

The *build trend* is a team metric that provides value for everyone on the project. It is the heartbeat of every project and should be highly visible to everyone on the team. Ideally, the build trend measure is automatically generated as a result of a continuous integration system.

The build result, as shown in Figure 7-53, orients the entire team on the most important measure for success in agile development, working software. In many organizations, the build and build results are left in the developer domain as their concern because they are not important to other team members. The primary reason is because, in the past, build systems have been solely focused on integrating the code into working software but not at integrating valuable project and change information with the build. Now, collaborative development environments are not only delivering integrated project results with builds, but they enable the project teams to share information and work with each other directly in the context of project results.



Build continuous.requirements.jazz C20070605-1636

Completed
Duration was 30 minutes, 26 seconds
Started at Tue, 5 Jun 2007 23:03:15
Completed at Tue, 5 Jun 2007 23:33:42
Status trend:

Reported Work Items
No reported work items.
Create a new work item
Associate an existing work item

Contribution Summary

Analysis:	No analysis results found
Changes:	Show changes
Compile:	0 errors, 1041 warnings
Coverage:	No coverage results found
Downloads:	4 downloads
JUnit:	14 tests, 0 failures, 0 errors
Logs:	6 logs
Requirements:	None implemented
Snapshot:	continuous.requirements.jazz C200
Work items:	42 included in build

General Information

Requested by:	Muhtar Burak Akbulut
Build Definition:	continuous.requirements.jazz
Build Engine:	apollobuildserv6
Build History:	14 builds
Tags:	Untagged [edit...]

```
Build 17: continuous.requirements.jazz C2
Filesystem 6: v20070604 - 2: v20070531b
    Added (7)
        Christophe Cornu 19305: Auto                 Changes view - Show Pending view when its description is changed  Jun 04 08:32 PM
        Jean-Michel Lemieux 24627: R                 wer down on the menu than Replace With... (and not next to Load/Unload)  Jun 04 05:57 PM
        Jean-Michel Lemieux 25026: W                 Utils.createReviewTask()  Jun 04 05:43 PM
        Jean-Michel Lemieux 25015: op                r conflict does nothing  Jun 04 05:40 PM
        Christophe Cornu 24824: Test                 ibm.team.internal.filesystem.tests.synchronize.wiki.IComponentSyncModelWikiTests.testDelMod1 - Show list o
        Heather Fraser-Dubé 24927: Tool tip that explains structural conflict has "it's" that should be "its"  Jun 04 04:31 PM
        Christophe Cornu 24926: NPE attempting to merge evil twin changes - SaveableCompareInput does not take null for an answer unlike its super type  Jun 04 04:25 PM
Foundation 13: v20070605 - 4: v20070601_pre_migration
    Added (4)
        Christof Marti 24994: AIOOBE when hovering over the "Assert problem in timebox view" change  Jun 05 06:20 PM
```

24824   20219: Review 'Sign-off Milestone'

Figure 7-53 Build Result view from the Jazz project

The build trend, as shown in Figure 7-54, can help answer important questions about the team. For example, it lets the development teams know how well their parallel development and integration efforts are going. It gives the testers an early sense for the stability of the application. It gives the project manager a read on whether the project rhythm is effective.

With Rational Team Concert, the development team and testing team are always in close contact on the assets that are most important to both of them, especially working and *not* working software. For the testing team, the build result provides an immediate view into the working and *not* working components of the application. It orients them on the new changes that were added into the latest build, giving them the vital information they need for their testing. It also enables them to collaborate and communicate directly with the owners who implemented specific changes and orients them on the delivery team that is responsible for various parts of the application.

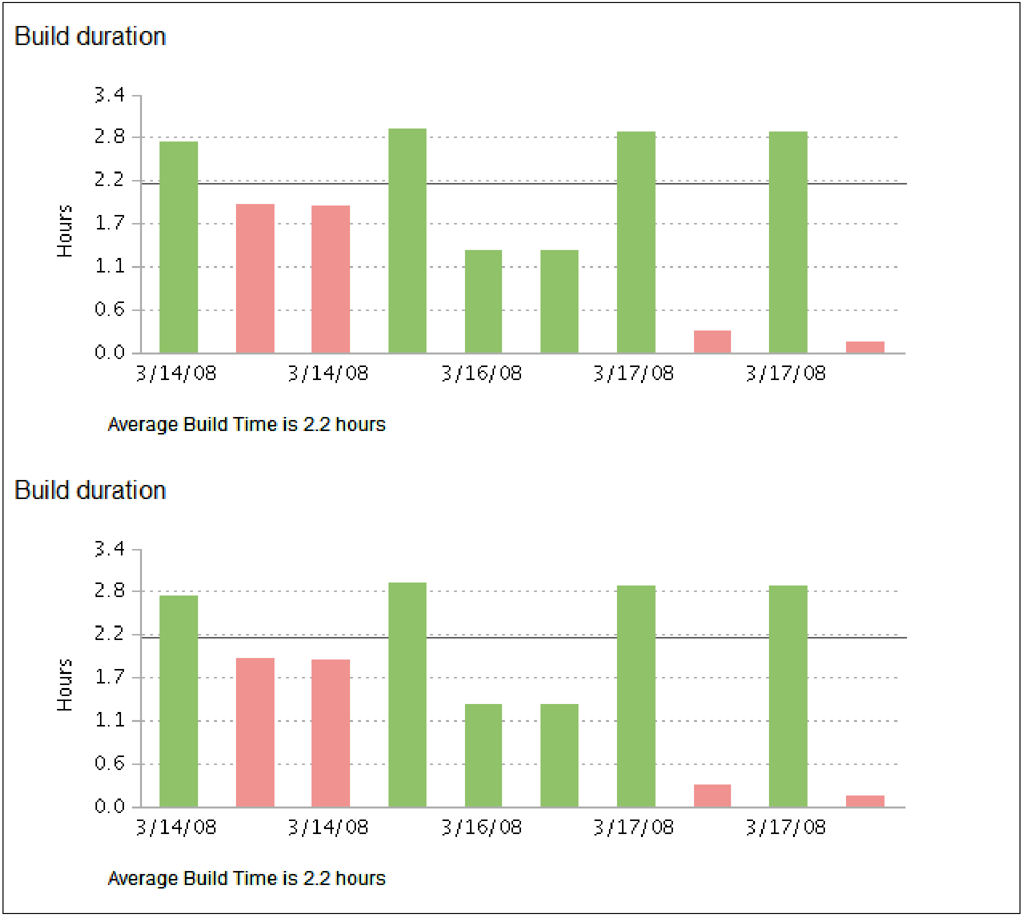


Figure 7-54 Build health report from the Jazz project

7.5.3 Measuring success by role

In this section, we describe the practices of measuring success in collaborative development by using Rational Team Concert from a *role* perspective. The following roles are covered:

- Developer role
- Team lead role
- Project lead role

Diedrie: Measuring success as a developer

Diedrie has two roles, the developer role and the component builder role, on which she must ensure success. She spends most of her development time in the Rational Team Concert Java perspective.

In her developer role, Diedrie focuses primarily on her current work assignments and the schedule to deliver the work in the current iteration. The My Work view helps her monitor the assigned work and the order list that forms the schedule of completing the work. The schedule in the My Work view is kept updated automatically by Rational Team Concert, but its quality is only as good as the information that is provided. As a practice, Diedrie updates the ownership, priority, and estimated and delivered work so that Rational Team Concert presents accurate health information to her and her team. To see the bigger picture, that is the health of the component team, she uses the iteration plan for the current iteration. This view shows load balance and overcommits, as well as indicates new unassigned work. These are all good indicators for her if others need her contributions.

For her builder role, Diedrie monitors the build notifications in the Team Central Build and Events sections. When a build fails, she browses the Build Results and looks for the root cause of the breakage. With the Build Results editor, she can create and assign new work items to resolve build issues.

To make her daily work easier, Diedrie has customized the Java perspective to include the Team Central and My Work views. By docking multiple views, or minimizing views as shown in Figure 7-55, she has quick and easy access to the Team Central and My Work views when needed. She has also added a few additional work item queries to the default options. The following queries are her favorites:

- The *Fixed to be verified query* looks up all defects that she has submitted and that have been fixed in the iteration. Diedrie wants to ensure that the fix resolves the defect that she found. If not, she reopens the defect.
- The *Monitor query* looks up a set of work item by IDs that Diedrie wants to monitor. There are a variety of reasons for her interest, such as blocking, or depending on her work, resolving build errors. She uses this query as a complement to subscribing to work item events.

By adding a new My Queries section to Team Central, and dropping the query into the new section, Diedrie has quick and easy access to her favorite queries as shown in Figure 7-55.

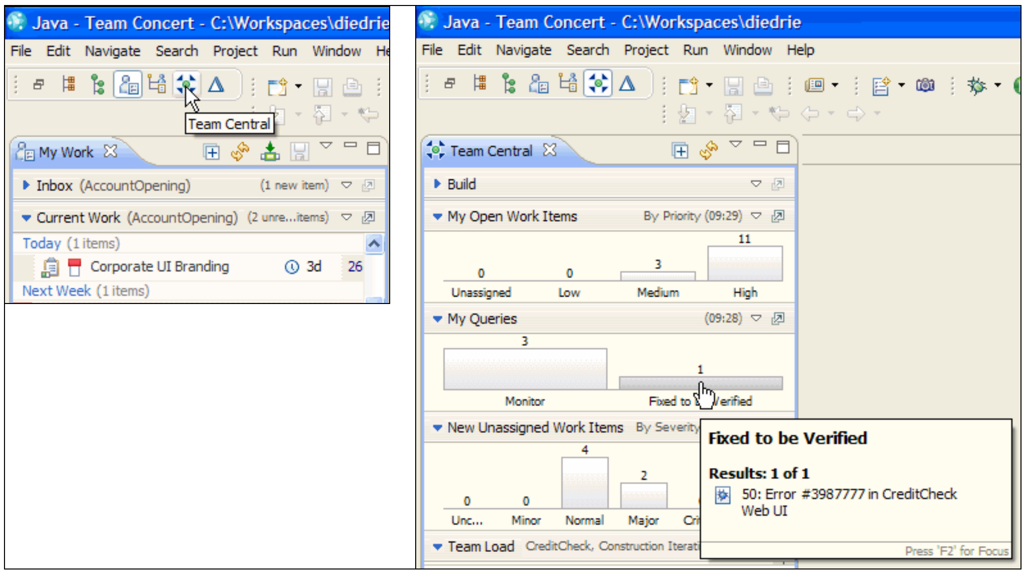

Java - Team Concert - C:\Workspaces\diedrie
File Edit Navigate Search Project Run Window Help
Team Central
My Work
Inbox (AccountOpening) (1 new item)
Current Work (AccountOpening) (2 unre...items)
Today (1 items)
Corporate UI Branding 3d 26
Next Week (1 items)
Team Central
Build
My Open Work Items By Priority (09:29)
Unassigned Low Medium High
My Queries (09:28)
Monitor Fixed to be Verified
New Unassigned Work Items By Severity
Unc... Minor Normal Major Cri
Team Load CreditCheck, Construction Iterat
Fixed to be Verified
Results: 1 of 1
50: Error #3987777 in CreditCheck Web UI
Press 'F2' for Focus

Figure 7-55 Quick and easy access to minimized views and favorite queries for monitoring Diedrie's success

Marco: Measuring success as a team lead

Marco must ensure that his team keeps their delivery commitments, with requested functionality and sufficient quality. He has two primary focuses in his role as team lead when measuring success:

- Track changes to work items in the current iteration
- Track the quality of his components in integration builds

To track changes, Marco uses Team Central and its Event Log section as explained in "The Team Central view" on page 290. By using this view, Marco sees a list of all changes made to work items or other events types to which he subscribes.

To monitor the build status, Marco uses the notification feeds from both the local component integration builds and the centralized solution integration builds. Any notifications from the builds that state a failure might draw his attention. In such situations, he browses to the build log and seeks the source of the failures. Because the build logs are organized by component, he can quickly determine if the failure is within his area of responsibility. For solution integration builds, Rebecca, the release engineer, also tracks build failures and notifies Marco if his component is breaking several subsequent builds. For component integration builds, the team has a shared responsibility to react on build failures, and Marco expects the team to resolve any build issues without his direction.

Through the iteration, Marco focuses on measuring success shifts. Initially in the development phase, he focuses on tracking or planning new work. In the later stabilization phase, his attention shifts to defects and blocking items. For these needs, he generally configures work item queries that pick up the items to help with tracking and scoping decisions.

Marco's team has a dashboard configured with the key health metrics that the team is using at the stand-up meetings. This dashboard is shared within the team and serves as a common statement of health. At some meetings, the dashboard is used, where at other meetings, the team uses public work item queries or queries created on the fly.

For additional information and examples about measuring team success, see the Development section of the Jazz.net Web site (sign-on required):

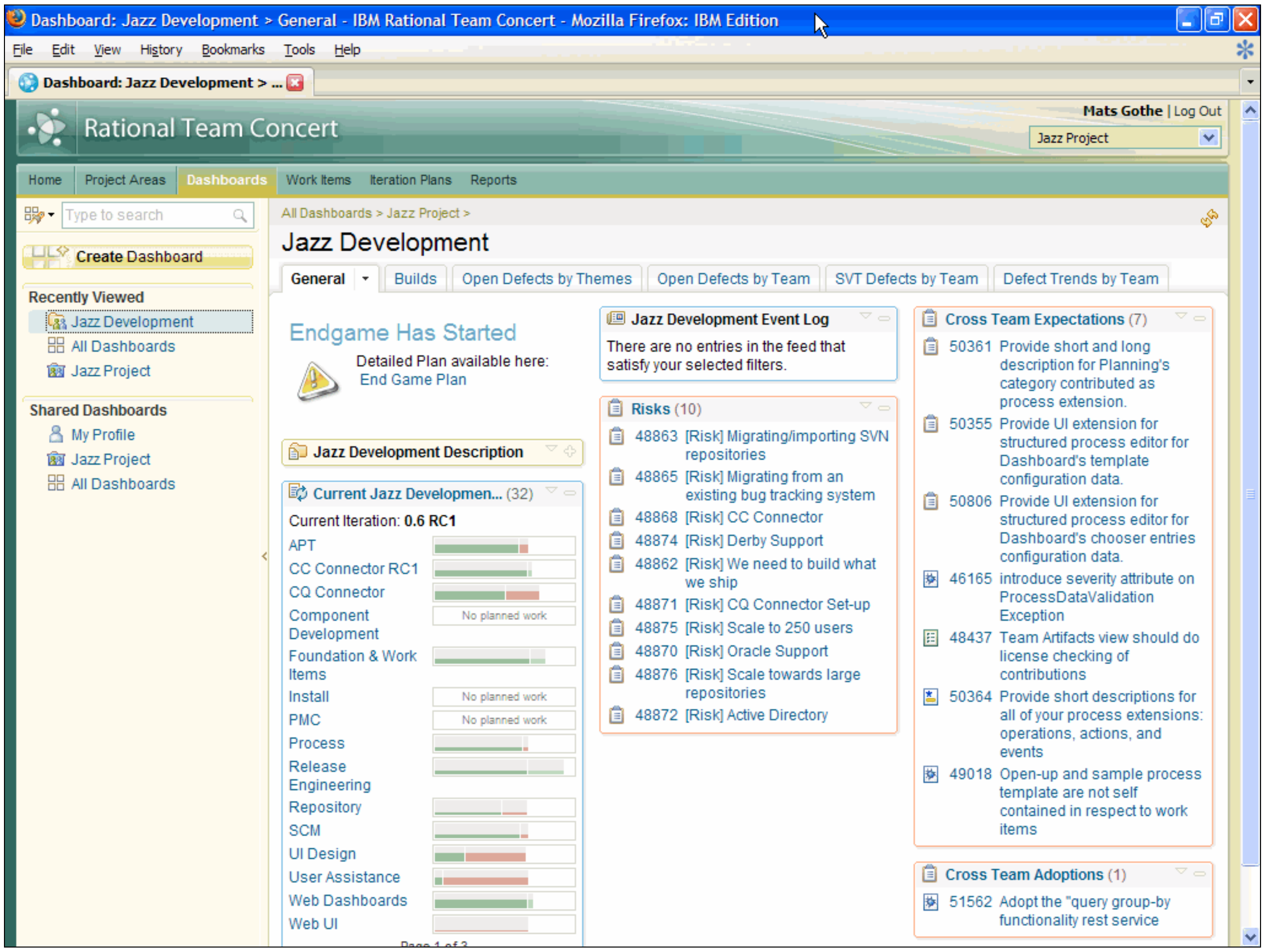
https://jazz.net/jazz/web/projects/Jazz%20Project#action=com.ibm.team.dashboard.viewDashboard

Measuring success as a project lead

In this section, we discuss how the project lead measures project success when using Rational Team Concert. In the Account Opening project scenario used in this book, we use a tool configuration in which Patricia uses Rational ClearQuest to plan and monitor the project. Hence, basing our discussion on measuring project success with Patricia in this section seems to conflict with the scenario story. Instead we use the practices and experiences from the Rational Jazz development team as a basis for this discussion.

Generally, the project lead role, together with the project leadership team, must balance the tracking and management of project plans, events, and risks with the trust and delegation of ownership and responsibility to the individual component teams. Some tracking success metrics are key in providing health status information that can serve as team reports and indicators for any required management action.

The Jazz team uses the Jazz Development Dashboard to measure and report project success as shown in Figure 7-56.


Dashboard: Jazz Development > General - IBM Rational Team Concert - Mozilla Firefox: IBM Edition

Mats Gothe | Log Out
Jazz Project

All Dashboards > Jazz Project >

Jazz Development

Endgame Has Started
Detailed Plan available here:
End Game Plan

Jazz Development Description

Current Jazz Developmen... (32)
Current Iteration: 0.6 RC1

APT
CC Connector RC1
CQ Connector
Component Development — No planned work
Foundation & Work Items
Install — No planned work
PMC — No planned work
Process
Release Engineering
Repository
SCM
UI Design
User Assistance
Web Dashboards
Web UI

Jazz Development Event Log
There are no entries in the feed that satisfy your selected filters.

Risks (10)
48863 [Risk] Migrating/importing SVN repositories
48865 [Risk] Migrating from an existing bug tracking system
48868 [Risk] CC Connector
48874 [Risk] Derby Support
48862 [Risk] We need to build what we ship
48871 [Risk] CQ Connector Set-up
48875 [Risk] Scale to 250 users
48870 [Risk] Oracle Support
48876 [Risk] Scale towards large repositories
48872 [Risk] Active Directory

Cross Team Expectations (7)
50361 Provide short and long description for Planning's category contributed as process extension.
50355 Provide UI extension for structured process editor for Dashboard's template configuration data.
50806 Provide UI extension for structured process editor for Dashboard's chooser entries configuration data.
46165 introduce severity attribute on ProcessDataValidation Exception
48437 Team Artifacts view should do license checking of contributions
50364 Provide short descriptions for all of your process extensions: operations, actions, and events
49018 Open-up and sample process template are not self contained in respect to work items

Cross Team Adoptions (1)
51562 Adopt the "query group-by functionality rest service

Figure 7-56 The Jazz team development dashboard

The Jazz team measures their success based on the following factors:

- Jazz development plan health
- Jazz development event logs
- Risks
- Cross-team expectations
- Cross-team adoptions

The *Jazz development plans* are the collection of individual component iteration plans that form the composite project development plan. The Jazz Development Dashboard is configured to show project status for the current iteration, drill-down overview pages for each iteration, and drill-down into the individual iteration plan documents.

The *Jazz development event logs* are the events notifications from the life-cycle assets, filtered by the relevance of the project health.

The *Jazz risks* are used to enumerate and track perceived project risks. The risks are used to draw the team's attention to areas of concern. The list of risks is reviewed daily with the team leads and the leadership team. The risks are captured as work items tagged with the keyword "pmc_risk."

The *cross-team expectations* and *cross-team adoptions* are common cross-team directions to which each component team is expected to conform. The dashboard surfaces these directions, draws the attention to any new expectations, and unifies the team vision and execution.

The Jazz team also tracks the following finer granular metrics:

- The *build health* is tracked by trending information about build success versus failure and the number of test cases run on each build. Build health is also tracked by ranking the most frequent JUnit test case failures. These metrics indicate the health of the code base, the coverage of the tests, and any areas of frequent instability.
- *Open defects* are tracked by several views including themes, team, testing, and trends. Each view provides health information about, for example, the stability of a new tool capability, the delivered quality of each component, or the success of the project quality team.

For information and examples about measuring project success, see the Development section of Jazz.net Web site (sign-on required) at the following address:

`https://jazz.net/jazz/web/projects/Jazz%20Project#action=com.ibm.team.dashboard.viewDashboard`

7.6 Reference architecture and configuration

In this section, we explain how Rational Team Concert fits into the overall solution architecture and how the tools have been configured for this act of the storyboard.

7.6.1 Fitting into the enterprise ALM solution

Rational Team Concert as used in this act illustrates part of an enterprise ALM solution with an agile component team that is integrated into a larger enterprise project. In this chapter, we have presented the workflows and tool usage in Rational Team Concert for an integrated ALM solution that supports the team in alignment of work, iteration planning, reuse, delivery of change, and build integration. Figure 7-57 highlights the part with the enterprise ALM solution that is discussed in this chapter.

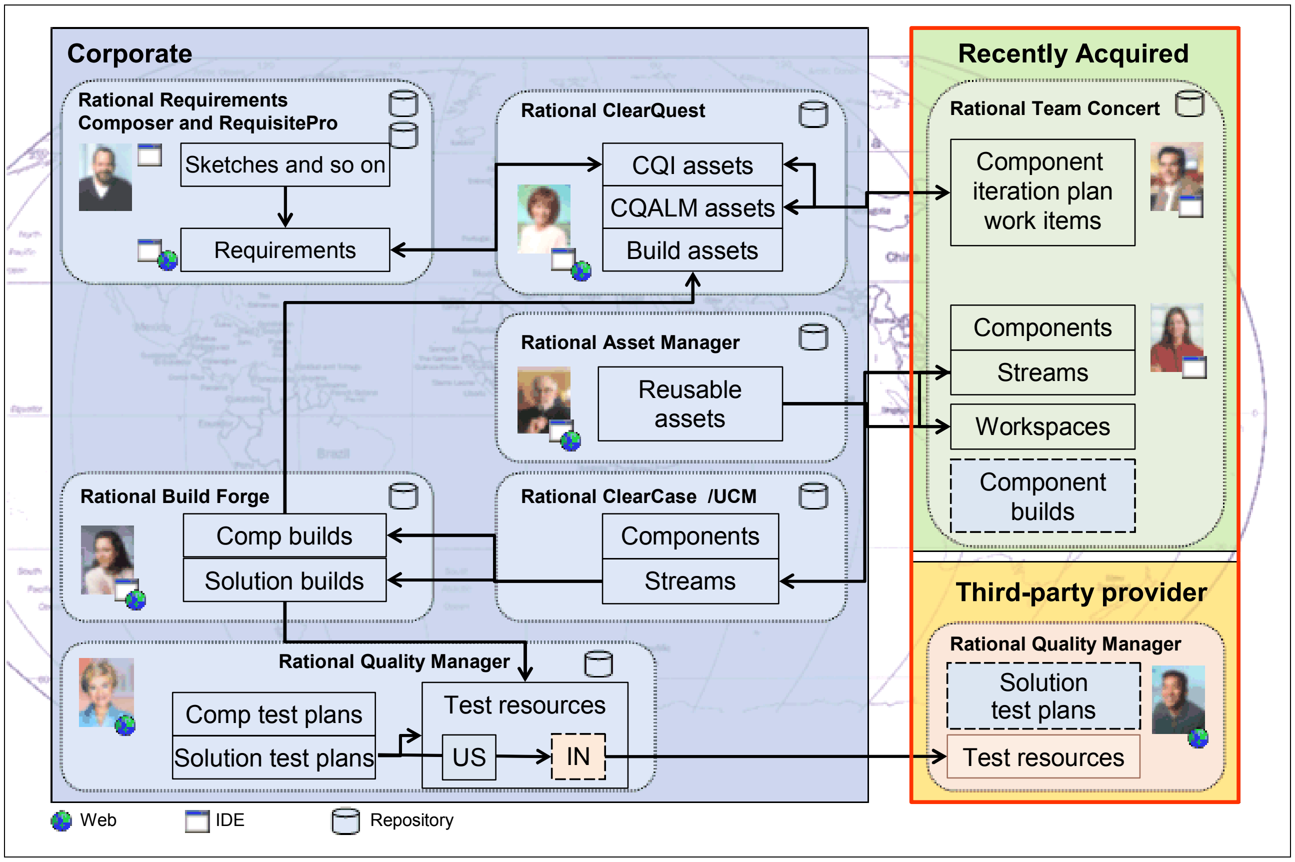


Figure 7-57 Rational Team Concert as one part of the enterprise ALM solution

Deploying Rational Team Concert

The Jazz platform is based on client-server Java 2 platform, Enterprise Edition (J2EE™) architectures. The Jazz server normally runs on a secured server-class machine and hosts services, a repository, and team artifacts. Remote clients communicate with the Jazz server over a local area network (LAN) or wide area network (WAN) by using HTTP. Remote clients come in many forms, such as seamless integration in the Eclipse integrated development environment (IDE) or Web browser clients and portals. In addition, Jazz-specific command line tools or Ant scripts operate in headless mode. Web browser access benefits casual access because there is no need to install Jazz-specific software on the machine. The Jazz platform client-server architectures support deployments of a central server servicing globally distributed teams, or team members, over WAN connections.

The Jazz platform is based on standard middleware as illustrated in Figure 7-58 on page 303. Smaller teams that self-administer can use the Jazz platform on open source middleware, such as Jabber, Tomcat, or Derby. If the team has requirements for more robust enterprise-sized middleware, it can use the Jazz platform with IBM Lotus, WebSphere, and DB2.

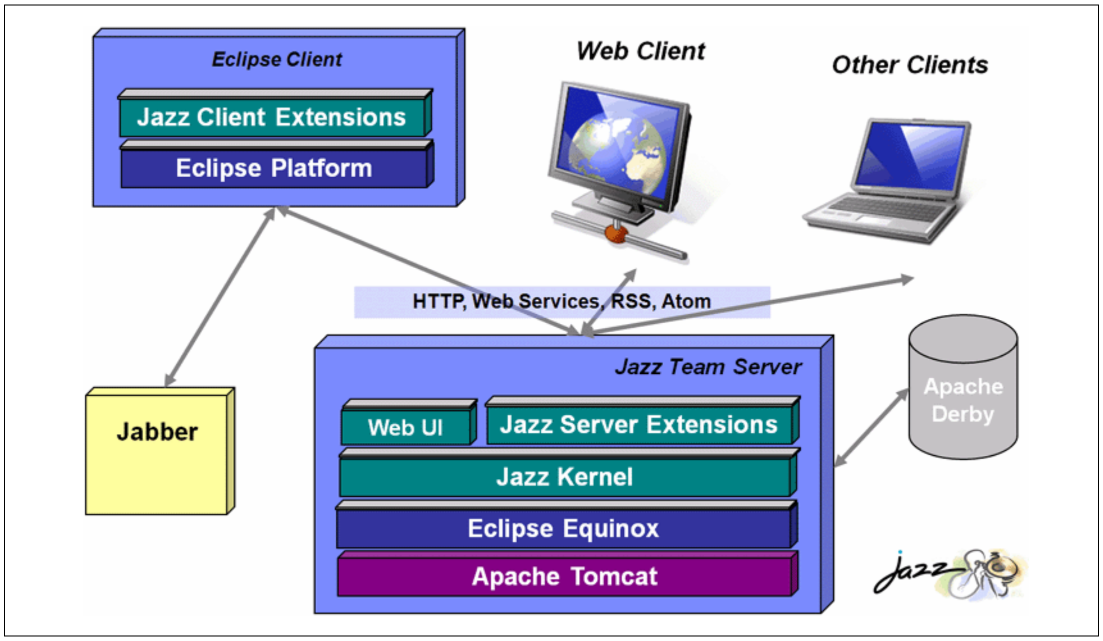


Figure 7-58 Jazz platform architecture with open source middleware

In the scenario described in this book, open source middleware was used. Rational Team Concert was deployed to a single Windows 2003 server, which runs the Jazz platform, Jabber, Tomcat, and Derby servers. The server also runs the build engine as described in "Configuring team builds" on page 309.

A deeper description of the Jazz architecture is provided in the *Jazz Platform Technical Overview* on the Jazz.net Web site at the following address (sign-on required):

`https://jazz.net/learn/LearnItem.jsp?href=content/docs/platform-overview/index.html`

Integrating Rational Team Concert and Rational ClearQuest

Tool and platform interoperability is a key enabler in the enterprise ALM solution that is described in this book. The integration between Rational Team Concert and Rational ClearQuest is used to enable the development team to access the following information:

- Cross-team development tasks related to a request
- Detailed requirements definitions and managed requirements
- Other supplementary artifacts

By using the ClearQuest Connector for Rational Team Concert, you can synchronize information between work items for Rational Team Concert and ClearQuest records. Through synchronization operations, the ClearQuest Connector maps ClearQuest records, such as ALM Activities, to Rational Team Concert work items. When a user creates or modifies a Rational ClearQuest record, the ClearQuest Connector creates or modifies a corresponding Rational Team Concert work item. The creation and modification changes also flow from work items to Rational ClearQuest records.

The ClearQuest Connector has multiple parts that must be deployed, including the ClearQuest Gateway, ClearQuest Packages, and synchronization rules files. For information about how to configure the ClearQuest Connector parts, see "Configuring ClearCase Connectors" on page 308.

Figure 7-59 shows the reference architecture for deploying the ClearQuest Connector with Rational Team Concert and Rational ClearQuest. We recommend that you co-locate the deployment of the ClearQuest Gateway and the Rational ClearQuest repository server because a LAN connection is required to achieve production quality performance. It is possible to co-deploy the ClearQuest Gateway and the Rational ClearQuest repository to a single server, but there might be performance implications. Rational Team Concert server can be deployed remotely from the ClearQuest Gateway by using a WAN connection.

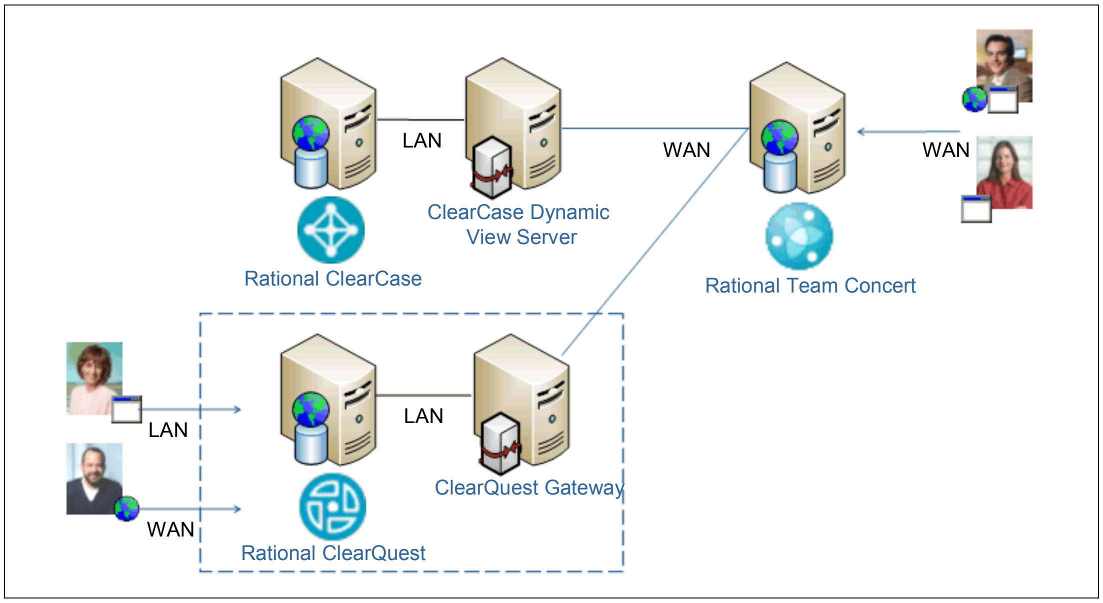


Figure 7-59 Deployment reference architecture for Rational Team Concert interoperability

Integrating Rational Team Concert and Rational ClearCase

The software supply chain that is described in this chapter exemplifies how a team develops and delivers components in a larger solution. The supply chain is implemented by using the integration between Rational Team Concert and Rational ClearCase. By using this integration, the Credit Check team can perform the following functions:

- Develop, build, and test software components by using the Rational Team Concert SCM capabilities
- Deliver component baselines to the ClearCase Synchronized Streams for solution integration
- Connect UCM activities with deliveries

With ClearCase Synchronized Streams, you can use the capabilities of Rational Team Concert to work on Eclipse projects that are under Rational ClearCase source control. You can also make projects that are under Jazz source control available to Rational ClearCase users of UCM components or VOB folders.

To use ClearCase Synchronized Streams, you configure one or more host computers to support synchronization between the Jazz source control and Rational ClearCase. Then you import an initial set of projects from Rational ClearCase into a Jazz source control workspace (known as a *ClearCase Workspace*). After this step is complete, users can work on these projects in both Rational ClearCase and Rational Team Concert and periodically synchronize work between the two environments. Figure 7-59 shows the reference architecture for deploying the ClearCase Synchronized Streams with Rational Team Concert and Rational ClearCase.

To use ClearCase Synchronized Streams, you must deploy one host with both a Rational Team Concert client and a Rational ClearCase client that supports dynamic views. This host supports the initial import and subsequent synchronization operations. Co-locate this view server host with the Rational ClearCase repository to achieve the required Rational ClearCase dynamic view performance over a LAN connection. Also, schedule synchronization to occur at regular intervals, such as nightly, during periods of low Rational ClearCase activity. The ClearCase Workspace owner monitors synchronization results in case merges are required or errors are reported, as described in “Diedrie delivers daily changes for solution integration” on page 285.

7.6.2 How the products are configured for this scenario

In this section, we describe the key configurations in Rational Team Concert that are used for this act of the scenario.

Configuring project and team areas

The deployment of Rational Team Concert for the scenario in this book is captured in Figure 7-59 on page 304. The Rational Team Concert repository is deployed co-located with the component team. Even though this is not a technical constraint, it is beneficial to the agile team to have local access to the server infrastructure. In practice, such a decision as this can depend on the availability of local administrative staff and skills.

Rational Team Concert was configured with a repository that contains a single AccountOpening project area. This project area currently only supports the Credit Check team, but in practice, it supports other teams in the Account Opening application that decided to deploy Rational Team Concert. A team area, named AccountOpening, was created for the project leader roles. Another team area, named CreditCheck, was created for the Credit Check team.

When enabling access for a team, it is important to create the user account and assign the appropriate roles for the various team areas (Figure 7-60). In the scenario in this book, the team was created and configured with the role that is available in the OpenUP. It is also important to personalize the user accounts. Use pictures of the team members, assign e-mail addresses, provide instant messaging server information, and update the work environment information. Team member properties improve team collaboration and accuracy of iteration planning and health.

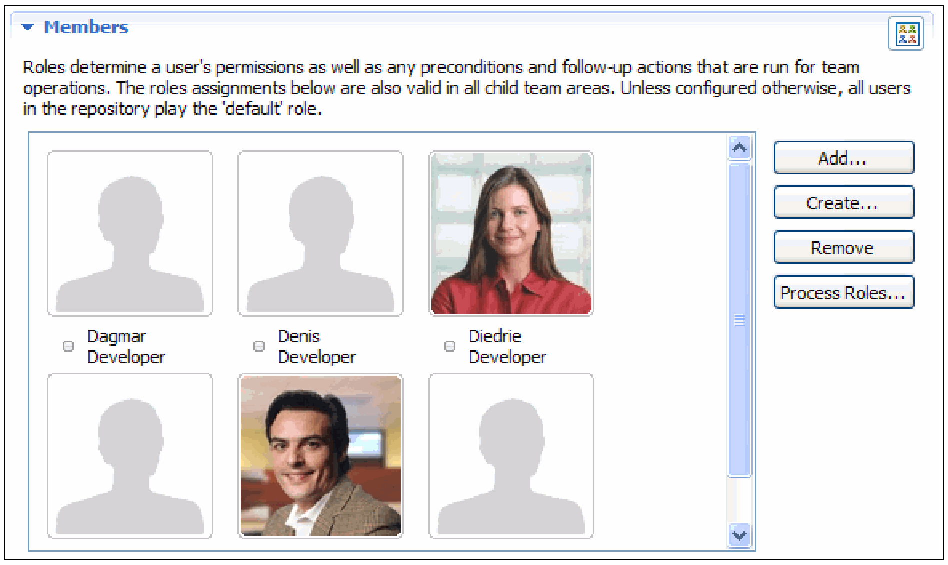
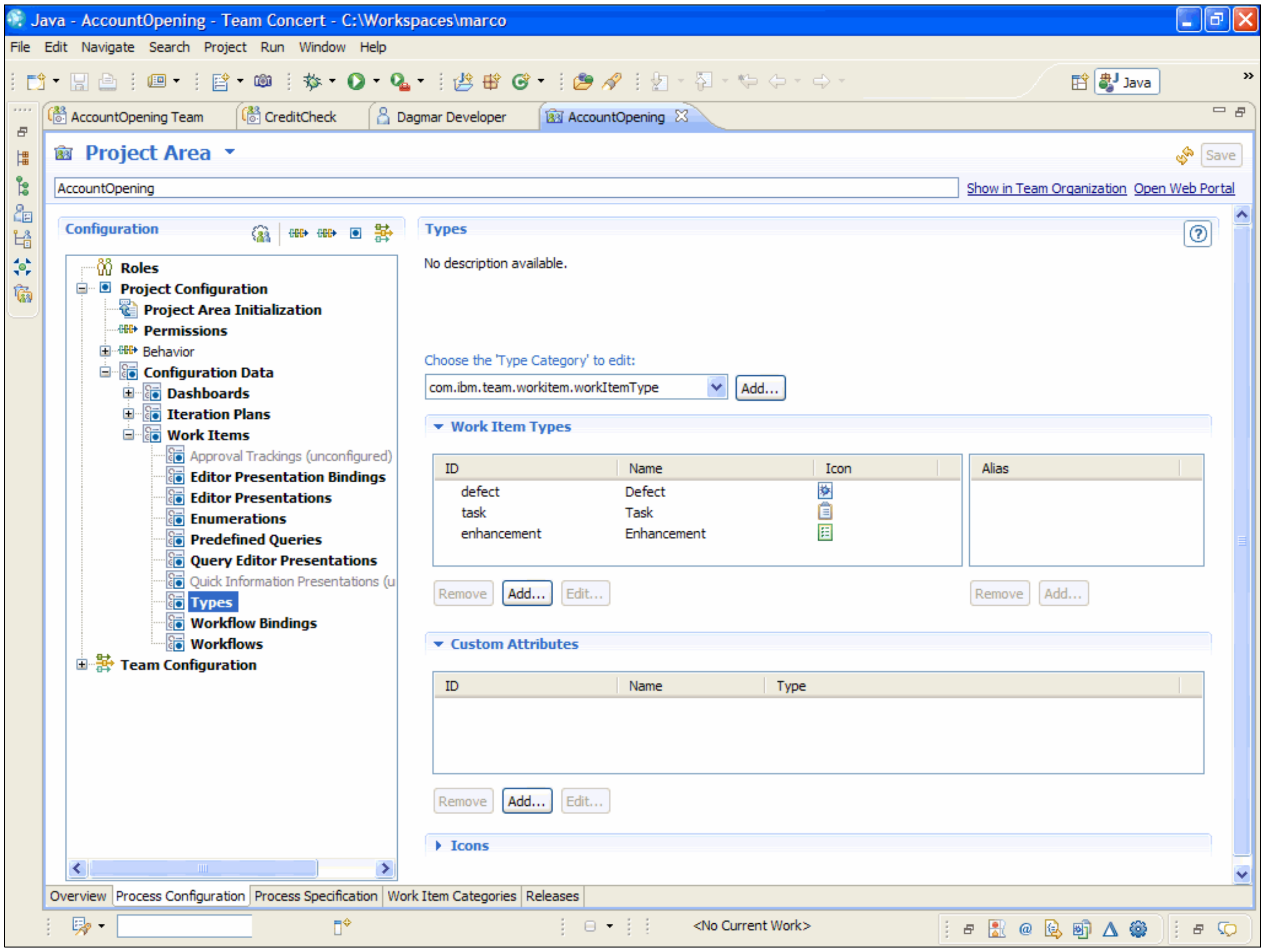

Roles determine a user's permissions as well as any preconditions and follow-up actions that are run for team operations. The roles assignments below are also valid in all child team areas. Unless configured otherwise, all users in the repository play the 'default' role.

Figure 7-60 Configuring team member information

The scenario configuration did not require any configuration of additional work item types or changes to the default work item state workflows. Such configurations are made in the Project Area Process Configuration. To modify the process, open the project area, select the **Process Configuration** tab, and browse the Project Configuration section as shown in Figure 7-61.



AccountOpening

Configuration

- Roles
- Project Configuration
  - Project Area Initialization
  - Permissions
  - Behavior
  - Configuration Data
    - Dashboards
    - Iteration Plans
    - Work Items
      - Approval Trackings (unconfigured)
      - Editor Presentation Bindings
      - Editor Presentations
      - Enumerations
      - Predefined Queries
      - Query Editor Presentations
      - Quick Information Presentations (u
      - Types
      - Workflow Bindings
      - Workflows
- Team Configuration

Types

No description available.

Choose the 'Type Category' to edit:

com.ibm.team.workitem.workItemType Add...

Work Item Types

ID	Name	Icon
defect	Defect	
task	Task	
enhancement	Enhancement	

Alias

Remove Add... Edit... Remove Add...

Custom Attributes

ID	Name	Type

Remove Add... Edit...

Icons

Figure 7-61 Configuring the properties of the project process

Configuring the iteration plans

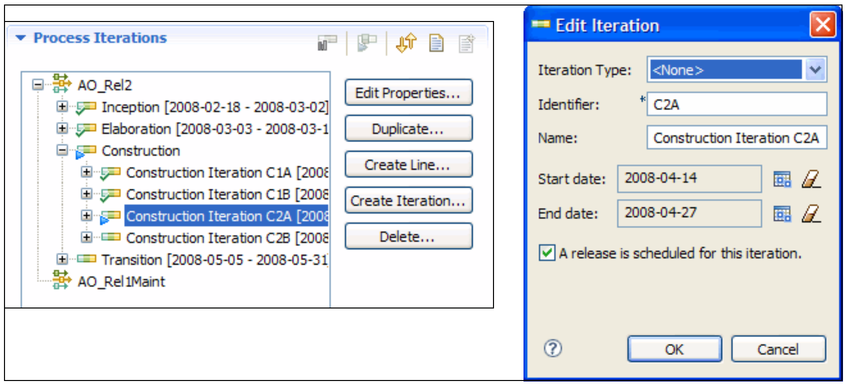
A project area is instantiated with a selected development process. In the scenario presented in this book, the project uses the OpenUP. This process is preconfigured and available in Rational Team Concert. OpenUP takes the Rational Unified Process (RUP) names for phases and iterations.

The Credit Check team in this scenario uses two-week iterations, while the project uses four-week iterations. To configure these differences, additional iterations were added to the process iterations in the AccountOpening project area.

To add additional iterations, expand the **Construction** section in the plan, select the **Construction Iteration**, and click **Duplicate**. Open the new iteration and click **Edit Properties**. In the Edit Iteration window, update the name, ID, and dates.

In this scenario, four iterations, C1A, C1B, C2A, and C2B, were created as shown in Figure 7-62. When creating a new iteration, make sure to assign unique identifiers for each iteration.

Rational Team Concert can also be configured to manage multiple development lines. This scenario mainly focuses on the AO_Rel2 release. However, Figure 7-62 also shows the AO_Rel1Maint development line that the Credit Check team participates in for maintaining the currently deployed application.


Process Iterations
AO_Rel2
Inception [2008-02-18 - 2008-03-02]
Elaboration [2008-03-03 - 2008-03-1
Construction
Construction Iteration C1A [2008
Construction Iteration C1B [2008
Construction Iteration C2A [2008
Construction Iteration C2B [2008
Transition [2008-05-05 - 2008-05-31
AO_Rel1Maint
Edit Properties...
Duplicate...
Create Line...
Create Iteration...
Delete...
Edit Iteration
OK
Cancel

Iteration Type: <None>
Identifier: * C2A
Name: Construction Iteration C2A
Start date: 2008-04-14
End date: 2008-04-27
☑ A release is scheduled for this iteration.

Figure 7-62 Configuring the iteration properties

For each iteration, a new iteration plan must be created. To create a new plan:

1. Expand the **Plans** section in the Team Area or in the Team Artifacts view.
2. Expand the iteration, right-click, and select **New** → **Iteration Plan** (Figure 7-63).

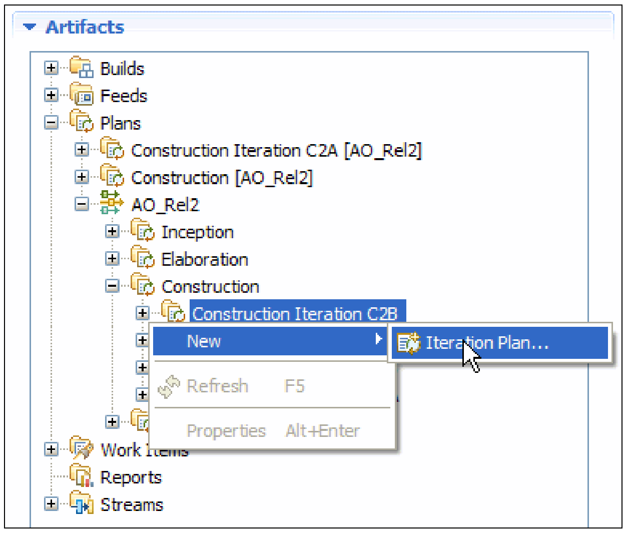


Figure 7-63 Creating a new iteration plan

Configuring source configuration management

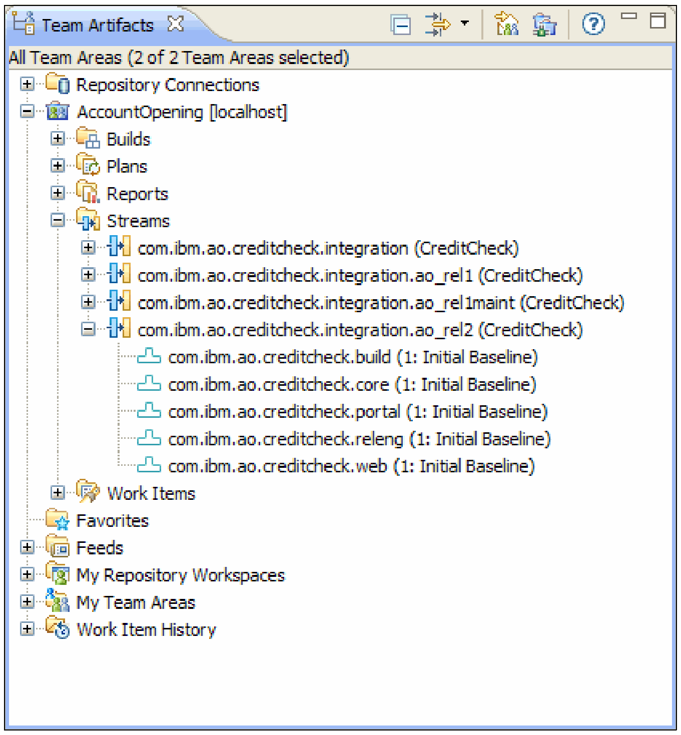
A simplified stream strategy was deployed for the scenario in this book:

- A main integration stream with the release history
- Integration streams for each release
- Integration streams for each maintenance of a release

Each stream contains a set of Credit Check SCM components (Figure 7-64). These components can be architectural parts of the Credit Check component, versioned artifacts, such as build scripts, or staging areas for release engineering.

Rational Team Concert is an Eclipse Team provider, and Eclipse projects are, when shared, added to an SCM component. To add an Eclipse project to an SCM component:

1. Select the **Eclipse project**, right-click, and select **Team** → **Share Project**.
2. In the Share Project wizard, choose the **Jazz Source Control** repository type, select an **SCM Component** in a repository workspace, and click **Finish** to add the project to the component.


Team Artifacts

All Team Areas (2 of 2 Team Areas selected)

```
Repository Connections
AccountOpening [localhost]
    Builds
    Plans
    Reports
    Streams
        com.ibm.ao.creditcheck.integration (CreditCheck)
        com.ibm.ao.creditcheck.integration.ao_rel1 (CreditCheck)
        com.ibm.ao.creditcheck.integration.ao_rel1maint (CreditCheck)
        com.ibm.ao.creditcheck.integration.ao_rel2 (CreditCheck)
            com.ibm.ao.creditcheck.build (1: Initial Baseline)
            com.ibm.ao.creditcheck.core (1: Initial Baseline)
            com.ibm.ao.creditcheck.portal (1: Initial Baseline)
            com.ibm.ao.creditcheck.releng (1: Initial Baseline)
            com.ibm.ao.creditcheck.web (1: Initial Baseline)
    Work Items
Favorites
Feeds
My Repository Workspaces
My Team Areas
Work Item History
```

Figure 7-64 Account Opening and Credit Check streams in Rational Team Concert

Configuring ClearCase Connectors

In this section, we describe the key configurations in Rational Team Concert that are used for the ClearCase Connectors.

Configuring ClearCase Synchronized Streams

In the scenario in this book, the Credit Check team delivers frequent baselines to solution integration in Rational ClearCase. A ClearCase Synchronized Stream was configured in Rational Team Concert to synchronize Rational Team Concert and Rational ClearCase streams.

To create a new ClearCase workspace:

1. In Team Concert, select **File** → **New** → **ClearCase Synchronized Stream**.
2. Enter the ClearCase stream selector, for example, `ao_rel2_integration@\com.ibm.ao_pvob`.
3. Enter a storage location for the dynamic view, for example, `\\localhost\VIEWSTORAGE`.

4. Select the Rational Team Concert project area to connect to, which is the location of the new repository workspace, and the team area to own the stream.
5. Type a new ClearCase Synchronized Stream name.
6. Click **Finish**.

Configuring ClearCase Synchronization

Synchronization between Rational Team Concert and Rational ClearCase is run as a separate process by using the Jazz Team Build component. A Jazz account must be created under the person whose credentials the synchronization process can run. The synchronization process account must be a member of the team area that contains the synchronized stream and the account's name and password on the Jazz Build Engine command line when starting a synchronization process. The synchronization process must be started to enable ongoing synchronization.

To start the synchronization process, run the **`syncengine.startup.bat`** script by using the credentials of the synchronization workspace owner.

For further details about configuring ClearCase Synchronization, go to the following Web address:

`http://jazz.net`

Configuring ClearQuest Connectors

We explain how to configure ClearQuest Connectors in Appendix B, "Configuring interoperability" on page 565.

Configuring team builds

In this section, we describe the key configurations in Rational Team Concert that are used for the team builds.

Configuring the build engine for Rational Team Concert

The team build component of the Jazz platform provides support for the automation, monitoring, and awareness notification of the team's builds. The team build requires configuration that fits the build process that is required by the team.

The agile team requires the team builds to provide continuous build support on the baselines that the team is working on. The team expects full transparency to any issues that are uncovered by the builds, so that team members can correct failing deliveries without further delay.

To configure a team build:

1. Install the Jazz Build Engine and Ant toolkit on the build machine. The Jazz repository server, or a dedicated build machine, can be selected as the build machine.
2. Configure a build engine item in the Jazz repository.
3. Configure a build definition item in the Jazz repository.
4. Request builds by using the build definition.

Configuring the build engine

Install the Jazz Build Engine and Ant toolkit on the Jazz server or on the build machine by using the instructions provided with the software distribution or at the following Web address:

`http://jazz.net`

To configure the build engine:

1. In Rational Team Concert, create a new build engine item:
   a. Open the **Team Artifacts** view.
   b. Browse to the **Builds** section.
   c. Right-click the **Builds** item and choose **New Build Engine**.
   d. Give the build engine a name, for example `com.ibm.ao.creditcheck.buildengine`.
   e. Click **Save** to persist the changes to the new build engine.
2. On the build machine, create a new command file, for example `buildengine.startup.bat`.
3. Edit the command file and add the command to start the build engine:

```
jbe -repository http://localhost:9080/jazz -engineId
com.ibm.creditcheck.build.engine -userId build -pass "" -sleeptime 1
```

Note: The command example assumes that the build engine is running on the Jazz server, called `com.ibm.creditcheck.build.engine`, and is run by the user "build" that has a blank password.

4. Save and run the file to start the build engine.

Configuring the build definitions

The Jazz Team Build can manage multiple definitions of builds in parallel. Each definition specifies a particular build, such as a continuous component build or a weekly project-wide integration build. If builds are run from various streams, such as a maintenance stream and a new release stream, each of the builds should have their individual definitions.

The Account Opening project maintains two parallel builds:

- The builds for the AO_Rel2 new development stream
- The builds for the AO_Rel1Maint maintenance stream

To create a build definition, Diedrie completes the following steps:

1. Diedrie opens the **Team Artifacts** view and browses to the **Builds** section.
2. She right-clicks the **Builds** item and chooses **New Build Definition**.
3. On the **General** tab, she enters the following information:
   a. She types a name for the build, for example `com.ibm.ao.creditcheck.integration.ao_rel2`.
   b. She associates the build with a team area, such as **CreditCheck**.
   c. She selects the build engine or engines to run the builds.
4. She clicks the **Jazz SCM** tab and enters the following information:
   a. She clicks the **Select** button and chooses an existing integration workspace, or creates a new workspace from the integration stream, for example `com.ibm.ao.creditcheck.integration.ao_rel2 Workspace`.
   b. She enters a directory where the build workspace is to be loaded, for example `C:\buildarea\creditcheck`.
   c. She choose **Accept latest changes before loading**.
5. She clicks the **Ant** tab and enters the following information:
   a. She enters the path to the build.xml file that scripts the build step.
   b. She clicks **Save** to persist the new build definition.

Extending build scripts with Jazz Build Ant tasks

The build steps are scripted in a build.xml file by using Ant. Rational Team Concert extends Ant scripting with several tasks and properties when using the Jazz Build Engine. These properties are needed to use the Jazz Ant tasks to publish information to the repository. For example, the StartBuildActivityTask task updates the build with current activities. These activities are visible in the Team Concert UI. This is a simple mechanism that provides valuable awareness of the build's activity to the team.

For more information and examples about Ant build scripts, see the following Web address:

`http://jazz.net`

Integrating with Rational Build Forge

In the scenario presented in this book, the Credit Check team uses the local Jazz Team Build component for integration builds. However, other deployments can choose to integrate with a centralized enterprise integration build service for component builds and personal builds.

Rational Team Concert provides several build templates to configure build definitions. To integrate with Rational Build Forge:

1. Select the **Build** section in the Team Artifacts view, right-click and choose **New Build Definition**.
2. Choose the **Build Forge Build Engine** build definition template.

Part D

Act 3: Enterprise integration builds

This part continues with the storyboard where the release engineer conducts an integration build of the overall solution. In Chapter 8, “The release engineer conducts the integration build” on page 315, we provide information about Enterprise Build Management as it relates to the scenario. In Chapter 9, “Rational Build Forge for enterprise integration build” on page 341, we provide detailed information about the Rational products that are used to support this act of the story.

Role-based guide: To understand how the content in this part applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

8

The release engineer conducts the integration build

In this chapter, we include the following sections:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

8.1 Introduction to enterprise build management

By definition, lifecycle management involves managing a cycle. However, it is more interesting to note that there are cycles within cycles for a single software solution. Implementation involves continuous cycles of coding, unit testing, delivering, building, deploying, and testing against the requirements. The test results indicate the level of quality, which then drives additional development cycles within iterations. A defect has a cycle. To produce a final result, each of these team members must be aware of each other's activities and work in a coordinated and synchronized fashion. Each discipline involved in contributing to a build (development, configuration management, testing, and release management) uses its preferred tools for its own activities and often has a mature, well-structured process in place. What is lacking is the quality of integration across those disciplines in producing a build.

We must be clear in that an integration build is far more than a compilation of the application. In fact, an entire process exists to build a solution. The source code must be gathered into a build area, the application is compiled for one or more operating systems, build verification tests are run, the build results are staged, and the entire process must leave an audit trail. Not only that, but reports of build performance must be prepared and submitted for review.

As illustrated in Figure 8-1, build management is the bridge between development and test. Being sandwiched between the development and test disciplines, integration build activities can fall through the cracks of existing toolsets. This is problematic because proficient integration build management is crucial to overall team efficiency. If you cannot efficiently produce software builds, both development and testing suffer.

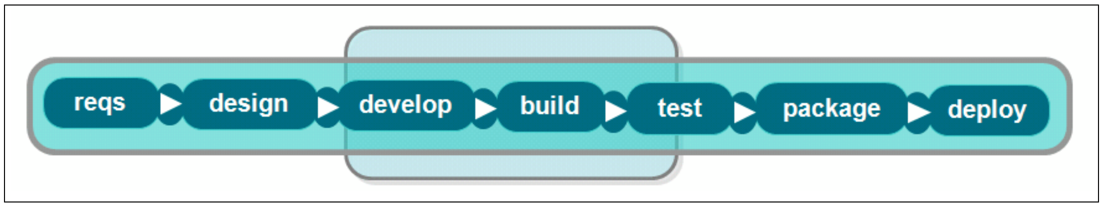


Figure 8-1 The build bridges of development and test

8.1.1 The changing enterprise build management market

The creation of a solution build for testing and deployment into production has evolved from compiling the application into a sophisticated synchronization and coordination of manual and automated tasks that use multiple computing languages and operating systems. Solutions have quickly changed from monolithic client applications running on a user's computer. They have become highly sophisticated composite applications that use service-oriented architectures (SOAs) to integrate with disparate systems running in data centers spread around the world.

For many organizations, the time spent to develop and deploy an application has been reduced from 18-24 months to what some now call a "state of perpetual beta." In this state, individual aspects of an enterprise solution are constantly being updated so that the solution is never "done." This places even higher demands on Collaborative Lifecycle Management (CALM). No longer can manual management of files and directories ensure that a consistent application is deployed. This state of perpetual change requires a streamlined and efficient enterprise integration build to keep the software deliveries cranking.

In this new world, building a solution entails many dynamics and dimensions:

- Application complexity is increasing, and with it, the processes and scripts for building that solution also increase. Solutions are made of many components that come from many sources. Some of these components are reused across multiple solutions and might be compiled for different operating systems. Some integration builds are highly complex with the need to spread the load across multiple worker machines in a server pool.
- With this added complexity comes a heightened need for build clarity and quality. *Build clarity* defines the features and defects that are implemented in a build. Understanding what is in the build helps the testers determine what to test. *Build quality* provides an indicator of the build stability. Release engineers are under increasing pressure to run build verification tests against the build before staging to the test team.
- The frequency of builds impacts the team's approach to integration build management, from continuous, nightly, scheduled or on-demand builds. Frequency also impacts the development and testing teams if the builds cannot be produced fast enough.
- The reason for conducting the build influences the approach. The reason might simply be a nightly build. Another reason might be to re-create a solution that was released three years ago. The build scripts and build environment must be recreated along with locating the source code and test cases. Knowing why the build occurred and where the results were staged are crucial to knowing how that build can be used. The life cycle of build artifacts requires "promotion levels," which are determined within the project domain and nature and specifies its maturity in this context.
- Compliance requirements require that you have a process in place and can prove that you use it. It also requires that you can recreate a build from any given point in time. Keeping build scripts on one person's personal computer is no longer sufficient. Instead the management of the build information must be centralized and accessible by many members of the team.
- Geographical distribution is the norm in large enterprises, and the build management solution must support that distribution. The build servers might be in a completely different geography than the source code. The staging servers might also be in another geographic location. The build solution must be able to reach across these boundaries without suffering in performance.
- Organization distribution, such as outsourcing and partnerships with third-party providers, requires the organization to have control over what these trusted partners can or cannot do on their networks. For example, an organization might want to allow their trusted third-party users to run a build, but not edit the scripts that produce the build.
- The degree of governance required impacts whether the release engineering team needs to generate reports and metrics regarding the builds. Some organizations require reports regarding build success and failure, build server performance and so forth.

As you can see, the build is not simply an act of compiling. Enterprise build management requires the coordination of disciplines such as source code management, change management, and quality management. Often there is a large margin for error when one function hands off a deliverable to another. Team members within each discipline generally operate somewhat autonomously relative to the other disciplines, in the sense that their focus is on their core competency, not on the linkage between their work and the overall process of building the product. As a result, sometimes one team might not know what another team is doing when the time comes to build the product. Against these challenges, realizing and implementing an enterprise build management system and fulfilling the needs for audit and compliance are almost impossible without using an enterprise build framework.

Yet amidst this rapid change of pace, many build management systems are essentially homegrown and maintained by resident "experts." These systems have evolved over time to the point where few knowledgeable experts know how to use them. Teams often automate their integration build processes through scripting. However, these script-driven systems rarely scale across the increasingly large, complex, and distributed environments that characterize modern software development. Often each task operates independently requiring wait times to be built in and properly coordinated. Multiple approaches to building these wait times are taken, and most often these are manually built in and estimated.

In this kind of scenario, if one step takes longer than anticipated, the entire build can break. The lack of a consistent, repeatable process can lead to difficulties in reproducing builds. This is particularly true in cases where processes are not adequately documented, so that knowledge about them resides with just one or a few people.

Sorting everything out in the context of broken builds can thus be inefficient. Build results are difficult to interpret, and much time is spent chasing down errors. A broken nightly build can easily mean that the test team will sit idle the next day, losing precious testing time, while the development and build teams troubleshoot the errors. Likewise, script-driven build management systems cannot make optimal use of build server resources, because often activities are hard coded to run on specific systems. In the same way, hard-coded scripts make it more difficult for processes to shift, so that work can be shared or reallocated across distributed teams. Project risk increases inordinately if a key team member who knows how to work the build system leaves the project.

It is not hard to see the general need for improvement in the integration build process. Simply put, home-grown build management systems are not fit for the challenges that are involved in enterprise integration builds.

8.1.2 Enterprise build management blueprint

Enterprise build management provides the bridge between development and test. As such, it serves as a software assembly line to keep the team moving toward a final, quality release that is suitable for production. The approach to solving this complex set of challenges is to automate the noncreative repetitive tasks that are involved in the process by choosing an enterprise build management system. To choose wisely, you must first decompose the problem into more solvable pieces:

- Capture, retain, and refine the knowledge that is currently known by a few valuable release engineers in a way that can be shared and used by other, less knowledgeable team members.

  The process for gathering the latest code, running all build scripts, and staging the build can be highly complex and time consuming. In many cases, the knowledge for building the application is in the memories of a handful talented build engineers. Finding a way to retain this knowledge is paramount.

- Automate the process to ensure that the same set of tasks is performed in every build, with a complete bill of materials that details the contents of the build.

  Rather than manually attempting to capture this information, an automated build process ensures that the same steps are performed the same way every time the build runs. Each change to the source code in an application triggers the need to produce a new build. Testers must know which changes went into each build. An enterprise integration build process must capture the changes in the bill of materials.

- Streamline and coordinate the process by defining processes that must run sequentially (in order that one does not start until the other has successfully completed) and those that can run in parallel across multiple worker machines.

  In homegrown systems, the build scripts are often interdependent and run with cron jobs. However, if one script in the chain takes longer than expected, the entire build can fail.

Debugging the failure can be a long and arduous process. Worse, the entire build must start again from the beginning before a good build is available. It is time to move away from manual processes or hard-coded scripts and begin to automate the processes that commonly exist within in the assembly line of software development.

- Decouple the build process from the hardware that runs it.

  The ability to abstract and define the environments independent of the process frees the team to focus on enhancing and streamlining the process. In this approach, only one process is maintained rather than one process for each hardware and operating system combination.

- Drive quality management into the build.

  With a centralized build system, developers and release engineers share the same build process and worker machines for producing the builds. We call this a *preflight build*. By using the same build process and work machines as the centralized build team, developers are protected from introducing a build-breaking problem due to compiler or build-script mismatches. Additionally developers can catch build problems before delivering their changes. By automating quality management activities, such as static analysis and build verification, tests can take place at every build, thus capturing quality issues early in the development cycle.

- Build, deploy, and test.

  Each new build must be deployed to the test servers before the testers can begin. Managing this cycle of builds and deployments through the test environments can be highly complex and error prone. The ability to automate the build process and track each build and deployment through the test cycle accelerates the software delivery cycle. It also provides much needed process and compliance traceability and audit trails.

- Provide traceability and audit trails to comply with regulatory audits.

  Capture a full audit trail that can be used to explain what occurred in any build or to recreate a previously completed build. The results of the build must be managed to ensure that the test team is testing from the proper build. Often file systems are full of build results, and it is left to “tribal knowledge” as to which is the appropriate build to use for testing.

Software development is a process that involves many people, and the business must absorb the costs of when these resources are idle due to an error or failed build. To understand the requirements of a software assembly line, IBM Rational has produced the *enterprise build management blueprint* as shown in Figure 8-2 on page 320.

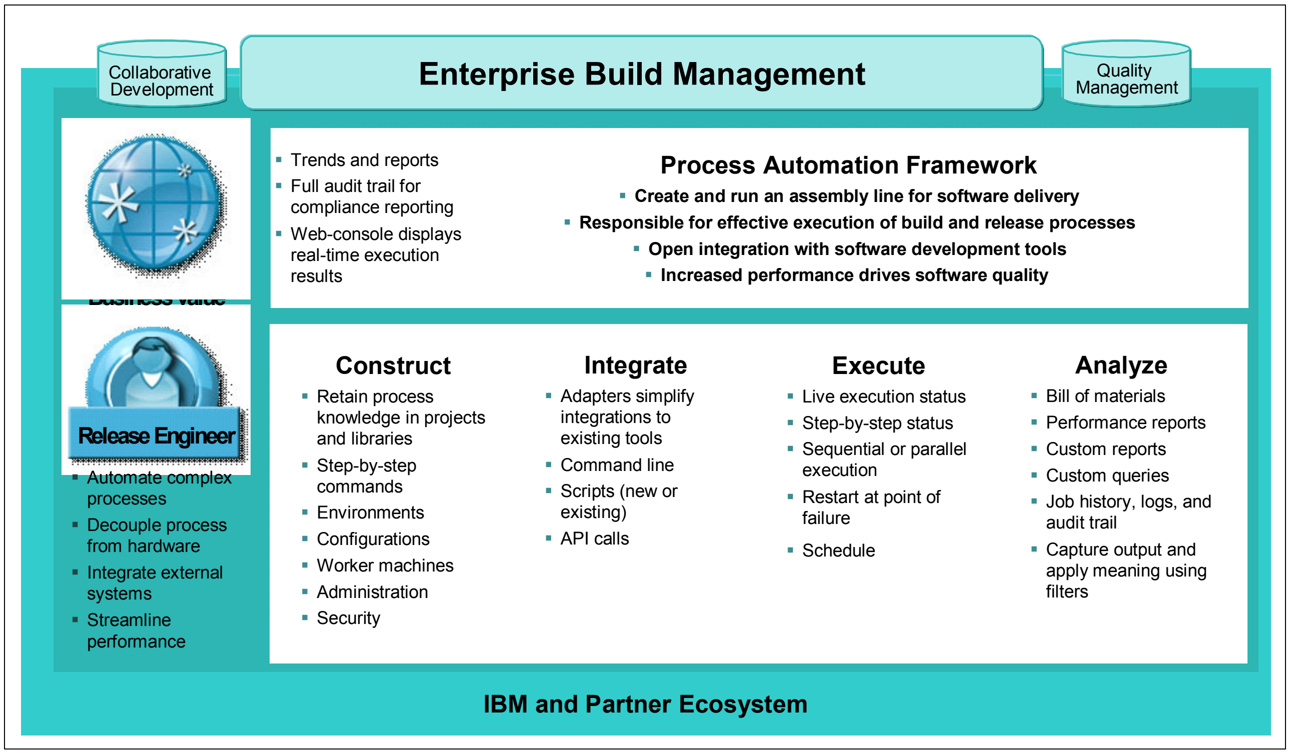

Collaborative Development
Enterprise Build Management
Quality Management
Trends and reports
Full audit trail for compliance reporting
Web-console displays real-time execution results
Process Automation Framework
Create and run an assembly line for software delivery
Responsible for effective execution of build and release processes
Open integration with software development tools
Increased performance drives software quality
Business Value
Release Engineer
Automate complex processes
Decouple process from hardware
Integrate external systems
Streamline performance
Construct
Retain process knowledge in projects and libraries
Step-by-step commands
Environments
Configurations
Worker machines
Administration
Security
Integrate
Adapters simplify integrations to existing tools
Command line
Scripts (new or existing)
API calls
Execute
Live execution status
Step-by-step status
Sequential or parallel execution
Restart at point of failure
Schedule
Analyze
Bill of materials
Performance reports
Custom reports
Custom queries
Job history, logs, and audit trail
Capture output and apply meaning using filters
IBM and Partner Ecosystem

Figure 8-2 Enterprise build management blueprint

Process automation framework

Effective enterprise build management requires an adaptive process execution framework that automates, orchestrates, manages, tracks, and logs all the processes. A process automation framework offers the following benefits:

- Frees the team to focus on process optimization
- Increases team productivity with on-demand access to run preflight builds and immediate feedback on build status

  Teams can pinpoint and troubleshoot errors quickly, which improves quality and prevents schedule slips.
- Enables real-time management visibility into development progress

Rather than require extensive rewrites to the existing process, the framework provides a layer above the development team management systems by integrating with existing systems.

The release engineers are not the only team members who are interested in the status of the build. Today, a Web-based user interface and project dashboard are required for visibility and clarity into the build process.

A process automation framework transforms the release engineering team into an automated software factory by empowering the team to create an assembly line of software development as illustrated in Figure 8-3 on page 321.

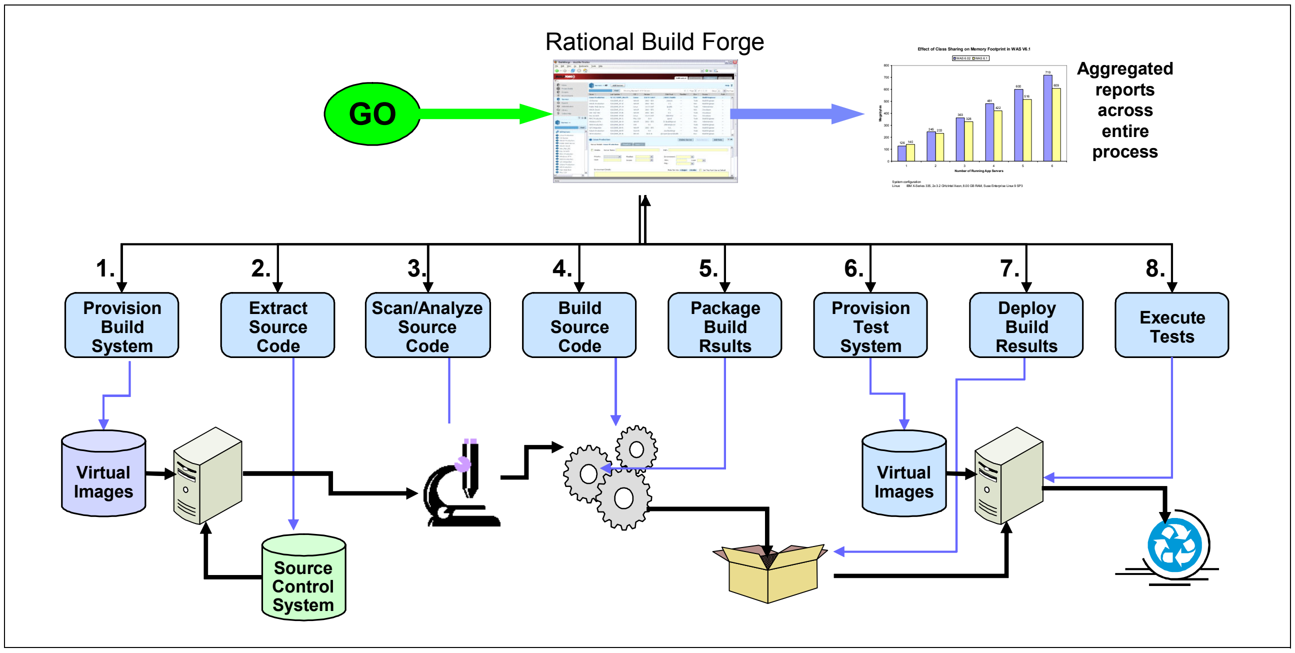


Figure 8-3 Automating the 'assembly line' of software development

Construct

Does your organization have a "build master," and is that person the only one, or one of a few, who know how to conduct the build? If that person is not available, is your team capable of running the build? The knowledge of how products and applications are built and delivered to your customers is a precious corporate asset. Typically, this knowledge resides in the minds of a few individuals who can leave your organization. Should this occur, this loss will complicate their replacements' ability to take over and render your business vulnerable. On this level, seeking reproducibility involves knowledge retention with the following goals:

- Protect important development knowledge.
- Reflect the entire application assembly process from source code through to deployment.
- Retain comprehensive process data in a secure centralized knowledge base.

The most crucial part of enterprise build management is constructing the process that will reflect the needs of the development, release engineering, and test teams. This information must captured in a centralized knowledge base that can be shared, viewed, and refined by the key stakeholders. Each build is represented by a project, and each project has a set of steps that must succeed for the build to be considered complete.

The process breakdown structure can be scripted into steps. Most teams already have scripts in place such as Make or Ant. An enterprise build management solution must allow a team to reuse existing scripts. Large and unwieldy scripts can be broken into discrete steps that become more manageable and easier to diagnose. The needs and requirements for the whole process should be determined precisely to deduce the environment variables, configurations, and server needs. When this knowledge is captured in a meaningful and constructive manner, inconsistencies and contradictions can be eliminated. The result leads to a standard, reproducible, and consistent process flow and knowledge for the build, release, and deployment.

Environments

You have one build machine and one script. The only way to build an application is to use that script and that build machine, which can create bottlenecks, risk, and a single point of failure. Depending on a single build machine is fraught with risk. Instead, the process must be designed to abstract out environment specific variables, thus enabling a single build process that can be used in many environments. In doing so, you ensure that the same process is used and that a change in the process is applied to all environments. By decoupling your build process from the hardware that runs it, you can future proof your build process.

Some organizations must support applications that run on a wide variety of hardware and operating system combinations. When build scripts are written for every environment, the problem can be solved. However, the risk of conducting the build in a consistent manner is lost. If the build script changes, it is easy to lose track of which systems have the updated script. The key is to separate the process for conducting a build from the environment on which it will run. Over time, the hardware will change, but the build process itself will not.

Additionally any change to the process is completed once, and that process is used across multiple environments. Having a system that helps you to create and manage multiple environments for a single application frees your build process from its hardware. As new operating systems are released, new environments are created and managed. The effort spent on reusability by using environment variables and virtual servers and configurations provides a great advantage to compact the project and mitigate the maintenance work as well as increase understandability.

Worker machines

The use of pools of servers can decrease the amount of time required to complete a build. To speed the build process, Rational Build Forge manages a pool of servers. Build and release management requires scalable runs. When your frequency of build increases, you can add new build servers just by defining them in the build and release tool with flexible selection parameters. The optimization of the selection should be left to the tool itself. Worker machines and their configurations are maintained together within the tool without having any inconsistency.

For example, you must patch the system that you deployed three years ago, but do you know how to build it? The auditors are coming in next week, and you must quickly start documenting the build process. Reproducibility comes into play on several dimensions. The first dimension involves time. Can your team reproduce a problem in a system that was built a year ago or perhaps six months ago? If a high profile customer requires a patch to a system that has since been revised several times, are you able to deliver?

Administration and security

A point to consider is the products' ability to allow multiple project teams to share a common infrastructure while enforcing a consistent process. When many project teams share a repository, security considerations must be taken into account:

- Authenticate against various identity management systems.
- Decide who can do what, when, and where.
- Track who does what, when, and where and customize the actions that can be taken, such as define who can view, run, or change a project.
- Provide team members with controlled access to predefined processes, either by project or sub-project, or by role.
- Hide the underlying infrastructure by eliminating direct access to servers.

These considerations allow better standardization and quality. Fine-grained security capabilities are essential for a successful and reliable enterprise build management solution.

Integrate

A perennial challenge in creating higher quality software faster is the need to balance speed and efficiency with adherence to a consistent, reliable, and structured approach. Whether the mission is development, testing, or code control, teams do not want constraints on how they operate, particularly in their choice of tools. Efficiency within each discipline is paramount. An enterprise build management solution must ensure the following objectives:

- Preserve development’s freedom of choice. Allow for process standardization without the political warfare of which tool to use for a step in the process.
- Use readily available integrations with source control, testing, defect tracking, and packaging contribute to rapid implementation.
- Customize integrations to fit your processes and adapt to changing needs and tools over time.
- Provide a self-documenting system with accurate, ongoing documentation of release contents, and a bill of materials for communication and compliance.

When seen from the wider perspective of the organization as a whole, optimizing efficiency might entail additional structure. Many software artifacts that are managed by different servers and tools, take part in build and release management. For this reason, the tool that you are using for build and release management should be able to support every kind of integration:

- Command line
- Scripting
- API calls
- Adapters or connectors

Integration with existing tools, such as IBM Rational ClearCase and ClearQuest, for change management and the reuse of existing scripts, such as ANT, is a fundamental aspect in the enterprise build management blueprint.

Execute

In a software development environment where agile development, continuous builds, and frequent testing are mantras, it becomes perfectly clear that the build process must be automated and efficient. By building early and often, teams can reduce the amount of time in the cycle of finding, fixing, and verifying a defect. Using a process engine that allows for sequential and parallel tasks adds additional efficiency to the overall process. For example, with Rational, builds that once took over 24 hours were reduced to approximately three hours. This massive reduction in build time resulted in an immediate improvement in quality. The ability to find, fix, and verify defects dramatically improved as the time required to build the solution was reduced.

When the build projects are constructed and integrated with the existing systems, the automated process can be run. Executing the process should be as simple as clicking a Run button.

During execution, live execution status and step execution details provide real-time feedback on the status of the build. The progress of the build is tracked on a centralized and Web-based dashboard. Should the build fail, the exact step is highlighted on the dashboard. Builds can take many hours to complete. A failure in the middle or toward the end of the build can be costly. The ability to repair and restart a build from the point of failure saves the team countless hours in waiting for the build to complete. The problem can be diagnosed, and the build can be started from the point of failure.

During execution, the efficiency and performance of the process can be examined. It is here where opportunities to create threads for parallel execution, instead of sequential execution, become more evident.

There are times when only a portion of the build process needs running. The solution should provide a way to disable (or enable) specified steps with each run. This makes testing and debugging easier by allowing the release engineer to focus on a specific section of the process without waiting for the entire process to run.

Last but not least teams, the build must run like clock work. Therefore, flexible scheduling for running the build is critical to creating the software delivery assembly line.

Analyze

What good is the build if you do not know what is in it? Having a detailed bill of materials that lists the changes that went into the build helps bring clarity into the contents of the build. This clarity helps the test team to target their test efforts on areas that are ready for testing. It also reduces the possibility of not testing a function due to lack of visibility. The bill of materials also provides a key piece of information to a compliance auditor when the auditor asks what changes went into the build. This bill of materials must be automatically produced each time the build is run.

Within the project steps, filters and patterns must be used to interpret the output of each step for reporting or decision making purposes. For example, there are times when the build integrates with a third-party application to automate a step in the build. The results that are returned from the application might be vague or come in a vocabulary that makes sense to that application. An automation framework must provide a means of interpreting the result to indicate whether the step has passed of failed. For example, when running a static analysis tool, a threshold or policy can be set to determine the quality of the source code. While the scan itself might complete successfully, enough errors are found to indicate a quality failure. By providing the ability to filter this result, the automation framework can catch this type of quality failure and indicate that the step has failed due to a lack of quality in code.

Custom query reports provide insight into the build results over time. A team can analyze their success or failure rate in terms of the enterprise integration build and thus seek strategies to improve their success rates. For example, how often does the build break and how long does it take to fix? With this insight a team can employ strategies to improve their results. Ready-to-use performance reports provide an indication on whether the build time can be improved. Teams can seek strategies, such as identifying steps that can run in parallel or steps that can be spread across multiple worker machines to optimize the build time.

Automation is powerful, but can be dangerous if the steps are not logged. Enterprise build management solutions must capture everything that occurred as part of the automated process, from project history, a bill of materials for each job, and a full audit trail of what occurred.

Consider, for example, the issue of regulatory compliance. Many businesses in industries, such as financial services and health care, are under significant pressure to document what they do during software development projects. Documentation takes time and requires communication, potentially reducing efficiency. But the risk of sanctions can be significant to the company overall.

Can your team produce documentation that describes exactly how it built a product that was released a year ago? Can it reproduce that product during a compliance audit? Likewise, can it reproduce the build and release environment itself, which includes the operating systems, libraries, server memory configurations, and so forth?

For audit and compliancy or monitoring reasons, you must have a detailed job history and full audit trail administration. By abstracting out the build process in a repeatable automation, the steps taken to produce the build become visible to anyone who might need to inspect it.

By automating the procedure, you can prove to auditors that you have a process and that it is enforced. By managing build projects in a centralized build system, your organization can retain information about prior builds that allows the auditor to look into the past (Figure 8-4).

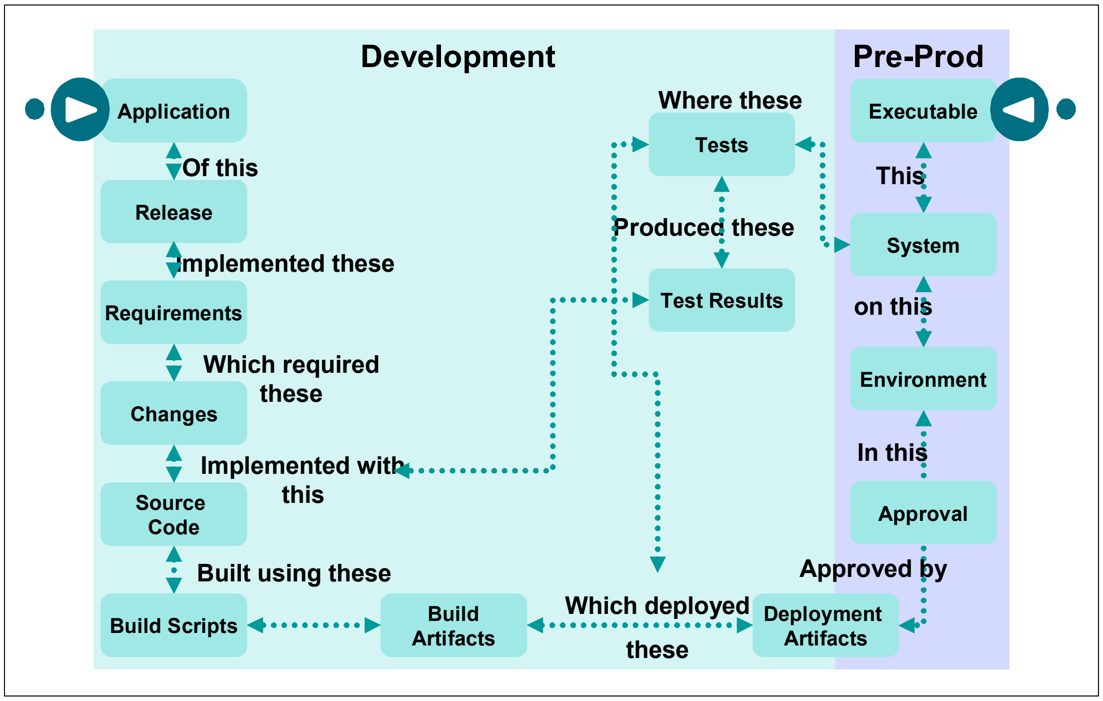


Figure 8-4 Ensuring traceability among the artifacts

8.2 A reference scenario for enterprise build management

In this section, we provide an overview of the steps taken by Rebecca to the complete the weekly integration build.

In Part C, “Act 2: Collaborative development” on page 211, the development team implemented the change. This included a developer running unit tests prior to delivery and a team build that conducted build verification tests. Diedrie confirmed that the team build was of sound enough quality to deliver the team’s changes to the integration stream.

The primary actor in this scenario is Rebecca, the release engineer. As shown in Figure 8-5 on page 326, Rebecca is responsible for integrating the individual team builds into an integrated solution for the testing team. Each team is expected to conduct builds for their component, and when sufficient quality is met, the team delivers their changes to the integration stream for Rebecca to build.

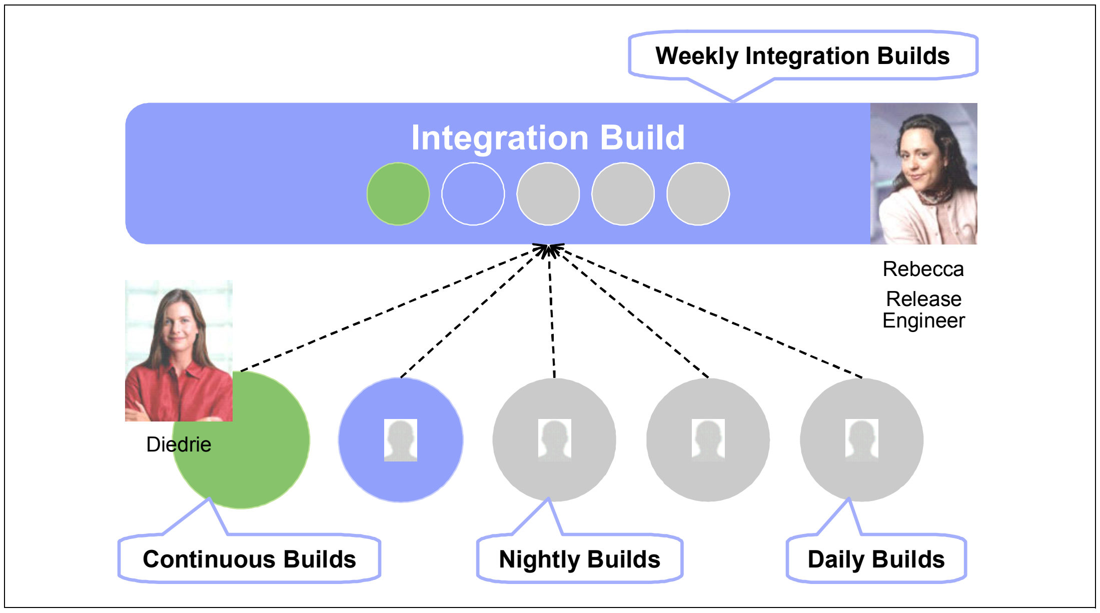


Figure 8-5 Rebecca integrating the team builds into a solution

In this part, we describe Act 3 of the reference scenario, which is illustrated in Figure 8-6. This act contains the following scenes:

- Rebecca responds to a failed build.
- Rebecca runs the integration build.

This act might appear simple with only two scenes. However, Rebecca has a sophisticated build process that she has automated. While her scenes in the storyboard might be limited, the build automation process is expansive.

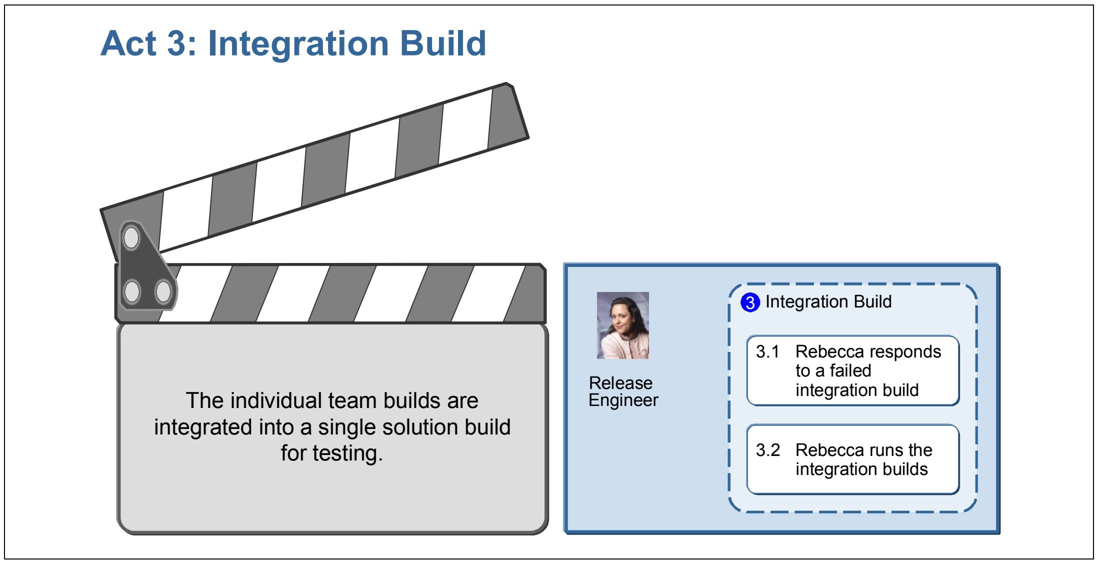

Act 3: Integration Build
The individual team builds are integrated into a single solution build for testing.
Release Engineer
3 Integration Build
3.1 Rebecca responds to a failed integration build
3.2 Rebecca runs the integration builds

Figure 8-6 Act 3 involves creating a successful integration build

8.2.1 The actors

Rebecca is the release engineer who oversees the solution integration builds, providing global build support to the team. She sets the delivery, enterprise integration policies within the project. While each component team is responsible for successfully building their component, it is Rebecca's job to bring all of the components together into a solution build. To bring agility into this enterprise solution, Rebecca produces a weekly solution build. This enables the team to diagnose solution build problems as soon as they occur and enables Tammy's team to test the solution build more often.

8.2.2 The workflow

At a glance, the workflow for this scenario is deceptively simple. An in-depth view into the automation script that Rebecca uses, as provided in Figure 8-7, reveals the sophistication of her automation.

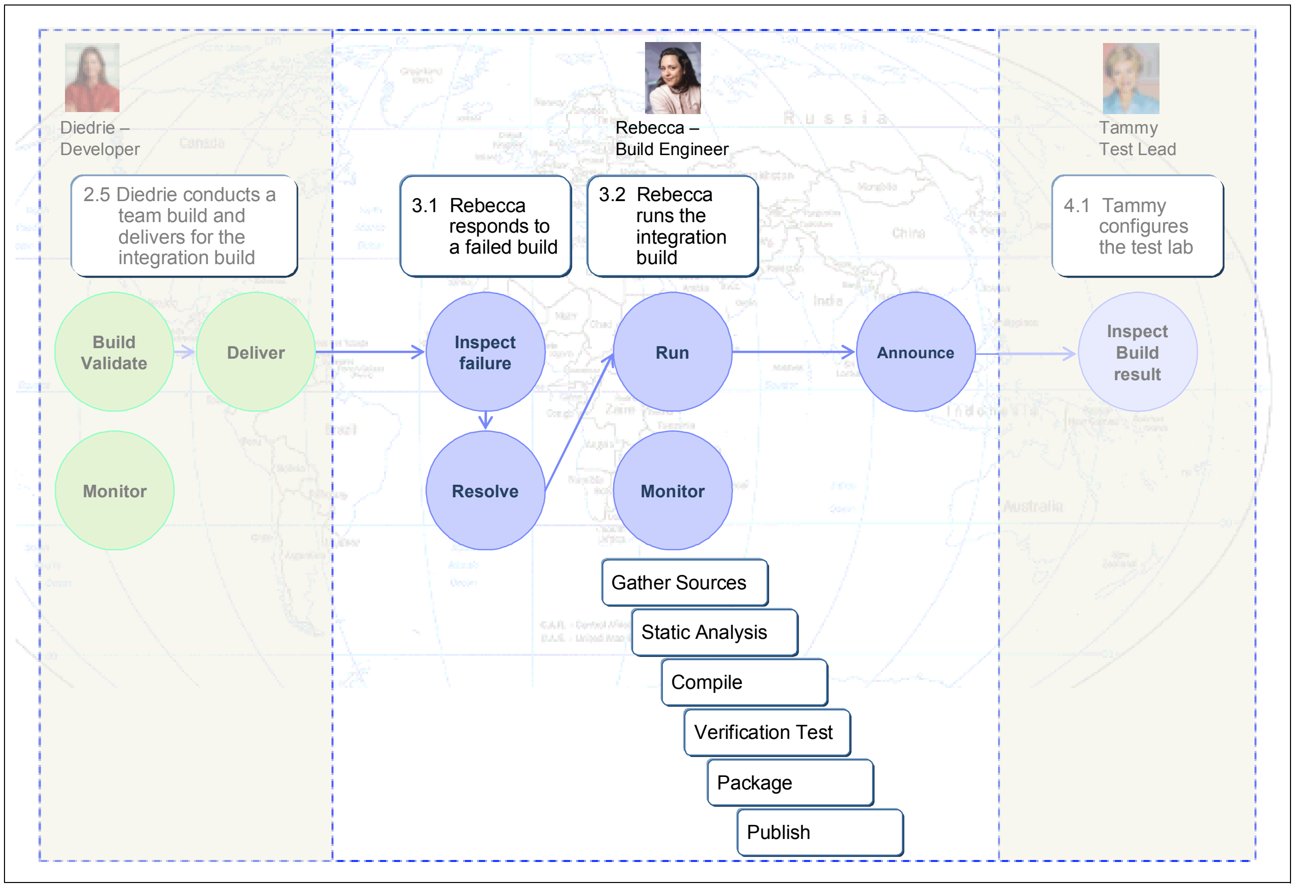


Figure 8-7 Integration build workflow

The act starts with Rebecca inspecting the build and resolving a build failure. Then, she starts the build and monitors the result. While the build is running, several important tasks are automated:

- The script gathers the sources and prepares the build environment.
- Static analysis is run against the source code base.
- The solution is compiled.
- Verification tests are run.

- The solution is packaged into a distribution archive.
- The solution is published.

  Publishing the solution involves staging the distribution archive, archiving and baselining the build project, and creating records in Rational ClearQuest to publish the build result and baseline information.

The act ends with Rebecca notifying the team of the build result.

8.2.3 Rebecca inspects the integration build

Synopsis: Rebecca has received notification that the integration build failed. She logs into IBM Rational Build Forge Enterprise and inspects the build log. The Rational Build Forge dashboard indicates that the build failed. Rebecca identifies the problem and implements a solution.

The workflow in this scene captures how Rebecca accomplishes the following tasks:

- Receives notification of the build failure
- Inspects and identifies the problem
- Resolves the problem

8.2.4 Rebecca runs the integration build

Synopsis: Rebecca starts the build again. While her part in the storyboard is small, the Rational Build Forge project that she uses is sophisticated. The build project that Rebecca has created automates far more than simply compiling the solution. In fact, the compile is just one step in a series of automated steps that take place each time she runs the integration build.

Her project integrates with Rational ClearCase for source code, Rational ClearQuest for Application Lifecycle Management (ALM), and Rational Software Analyzer. She also uses Ant and JUnit. In Chapter 9, “Rational Build Forge for enterprise integration build” on page 341, we explain the details of how she has created this automated build process.

The workflow in this scene captures how Rebecca completes the following tasks:

- Runs an automated build process that performs the following actions:
  - Gathers the sources and prepares the build environment
  - Runs static analysis against the source code base
  - Compiles the solution
  - Runs a set of verification tests
  - Packages the solution into a distribution archive
  - Publishes the solution
- Announces the build status to the team

8.3 Considerations in enterprise build management

As discussed, Rational Build Forge is a highly flexible and powerful tool for creating automation jobs. The scenario provided for this book provides a view into an automation strategy, but does not cover every aspect. In this section, we discuss additional considerations for enterprise build management.

8.3.1 Managing the build artifacts

Consider an example where your colleague Joe is out today, and you need to find his build scripts. Where does he keep them?

Every time the source code is modified, the application must be built and verified. When verified, the build is deployed to the test servers for testing. This pattern of delivering source code changes, building the application, deploying the application to test servers, and testing occurs regardless of the scope or magnitude of a release, whether it is a green-field application, a major revision of an existing application, a patch, or a hot fix. When errors are found, defects are logged, and source code is modified to fix the defect. Again, the application must be built and deployed back to the test servers for testing. The question is, how do you manage all of this change?

Throughout this cycle, the scripts that are used to conduct the build are updated and maintained to ensure a successful build. These scripts are just as important as the source code they are compiling. Therefore, they too should be managed by a software configuration management system and versioned with each change. Just as source code is labeled (or baselined), the scripts that are used to successfully build the application can be labeled. This ensures an alignment of the source code and scripts that are used to successfully build the application.

What do you do with the results of the build, such as the .ear, .war, or .exe files? After the build is complete, the results are typically staged in a location for the rest of the team to access. Many organizations stage the build results on a file system. This makes sense because the file system is easily accessed by the other members of the team. However, from a compliance perspective, this leaves the build results open to tampering by a disgruntled employee.

Staging the results of the build in a secure repository, such as Rational Asset Manager or Rational ClearCase, ensures that all changes have an audit trail. Not only does this prevent unwanted tampering, but the latest version is always accessible. Also, if the team must go back to a previous version, those versions are available with the milestone builds that are being identified with labels (or baselines). With this in mind, the source code, the build scripts, and the build results can all be given the same label in the source code management (SCM) system. This enables a team to easily identify the exact version of source that was built by using this version of the build scripts that produced this result.

However, many times builds go bad, or defects are found, and new builds are conducted. During the course of the development cycle, builds can pile up. Teams that use an agile continuous build strategy can encounter numerous builds in a single day. The key question for the testing organization is knowing which build to use for testing. Often knowing which build to test is left to tribal knowledge, where you need to know the right people to know which build to use. Rational ClearQuest 7.0.0.0 introduced a record type for identifying the state of a build. Tracking the builds in Rational ClearQuest helps the team identify which build to deploy to a test server and ensures that the build that passed testing is the same build that is deployed into production.

When using Rational ClearQuest, Unified Change Management (UCM), and Rational Build Forge, the UCM activities that are delivered with each build can be automatically captured and communicated to the team in Rational ClearQuest. This helps the test team know which defects to verify with the new build. Teams can also identify which tests have been run against the build. Additionally, when defects are found, they can reference the build record to indicate which build contains the defect.

8.3.2 Managing quality

Managing quality occurs at every step of the application lifecycle, and the integration build is no exception.

Build automation and verification testing

In Part C, “Act 2: Collaborative development” on page 211, Marco’s component team implements an automated build strategy. For Marco and his team, they ensure that their component builds without errors. They run build verification tests on their component to ensure quality in their build. When they are satisfied with the quality of their build, they deliver their changes into the integration stream. This ensures that each component has met with a predefined quality level prior to submitting it to the integration build.

Rebecca monitors the integration build. She has automated the integration builds and includes build verification testing at the integration level. This level of testing ensures that, when the components come together in the integration build, a predefined level of quality is met before deploying the build to the test servers.

By automating the builds, the team ensures that the exact same process is used for every build, which ensures consistency and reduces the chances of human error. In addition the automation system keeps an audit trail of every step in the automation. At any point, the team can inspect the audit trail and determine what occurred.

Frequent integration tests

Rebecca conducts the integration build on a weekly basis, and Tammy chooses which of the builds to deploy into the test lab. Frequent integration tests help Tammy’s team to identify defects early in the development cycle. They are also more likely to catch regressions from week to week.

Code review and analysis

In addition, Rebecca runs static analysis on the source code of every integration build. Static analysis provides a consistent set of tests across the entire source code base as it is delivered from each component team. By running source code analysis at the integration build, it is guaranteed to happen as a step in every build, and it ensures consistency across the code base.

Your build process is an ideal place for your organization to centralize code reviews and analysis. Centralizing code reviews in your build process ensures that your organization’s coding standards are being adhered to by project delivery teams. This enables you to manage the code review and quality process at the organizational level. Build-level code review analysis enables you to easily disseminate changes to your coding standards and best practices without policing individual projects or developers.

For example, if your software projects include contractors or system integrators, implementing code analysis and review ensures that everyone is following the defined coding standards. This reduces the risk of shortcuts or applications that break common architectural and coding best practices and weaken the quality of your application. Automating code review and analysis also makes the development organization more productive. Rather than concentrating on the code review and analysis, teams can focus on value-added corrective actions that progress the project toward high quality, working software.

Ideally, build-level code review and analysis should complement developer- or desktop-level code review and analysis. Developer-level code review and analysis are important because they catch weaknesses in individual contributors code before they are advanced and integrated into the composite application. Code review at the developer- or desktop-level ensures that corrections can be made early when they are easy to identify and implement.

Rational Software Analyzer provides both developer-level and desktop-level build and integration level-code review and analysis. With this capability, you benefit from organizational code quality management and developer-level code quality and best practices execution.

8.3.3 Building clarity

Leaving the build as a black box might be common practice, but when viewed from a life-cycle collaboration perspective, doing so introduces risk to the team. After all, the test team depends on the build to complete their work. Knowing what is in the build helps them to target their testing effort and ensures that everything in the build is tested. Without knowing what is implemented in the build, the test team can unknowingly miss the testing of critical functionality. Also, those who must conform with regulatory compliance must be able to state which changes where implemented in each build. Leaving the build as a black box can leave an organization exposed when it comes time for a regulatory audit.

Rational Build Forge generates a bill of materials after each project run. The bill of materials contains information about the steps in the run and the changes to files that resulted from it. The bill of materials can be provided to consumers of the project run, such as the quality assurance department, for help in understanding the contents of a new build. It can also serve as an audit solution for your integration build process.

With the bill of materials, you can easily include other relevant information as in the following examples:

- The ability to see a complete list of all files in the baseline that are used for this build
- The differences between this build and the last build (date/date, time/time, and label/label)
- The creation of a modified view on which the process execution is based
- A list of all defects that were included in the process run
- Automatic advancement of those defects to the next logical state in Rational ClearQuest
- Update of Rational ClearQuest with a build record of total compliance and traceability

These extensive tracking mechanisms make the Rational Build Forge system a valuable tool for documenting processes, providing an audit trail to assist with regulatory compliance requirements. A bill of materials offers the following benefits:

- Reduces time spent reading log data, with a less chance of errors being undetected, which increases quality
- Quickly diagnoses build output and locates items of interest
- Greatly reduces time spent on analyzing builds and troubleshooting errors
- Produces results in faster build turnarounds, greater productivity, and faster time to market
- Provides better support to monitor systems and aids in compliance

With the bill of materials, you get complete documentation of a build’s contents. It can include the build results, notes, environments, lists of build files, and code changes. This information can be used to compare and summarize the state of builds across the enterprise. The system generates a bill of materials for each build automatically.

Also log filters in Rational Build Forge have a crucial importance in build clarity. By using them, we can determine success or failure or decide a conditional behavior, such as chaining to other projects. Log filters can be used to recognize any pattern in an output and can be useful in preparing a bill-of-materials section for our builds. For more information about log filters, see "Log filters" on page 348.

8.3.4 Running static analysis during the build

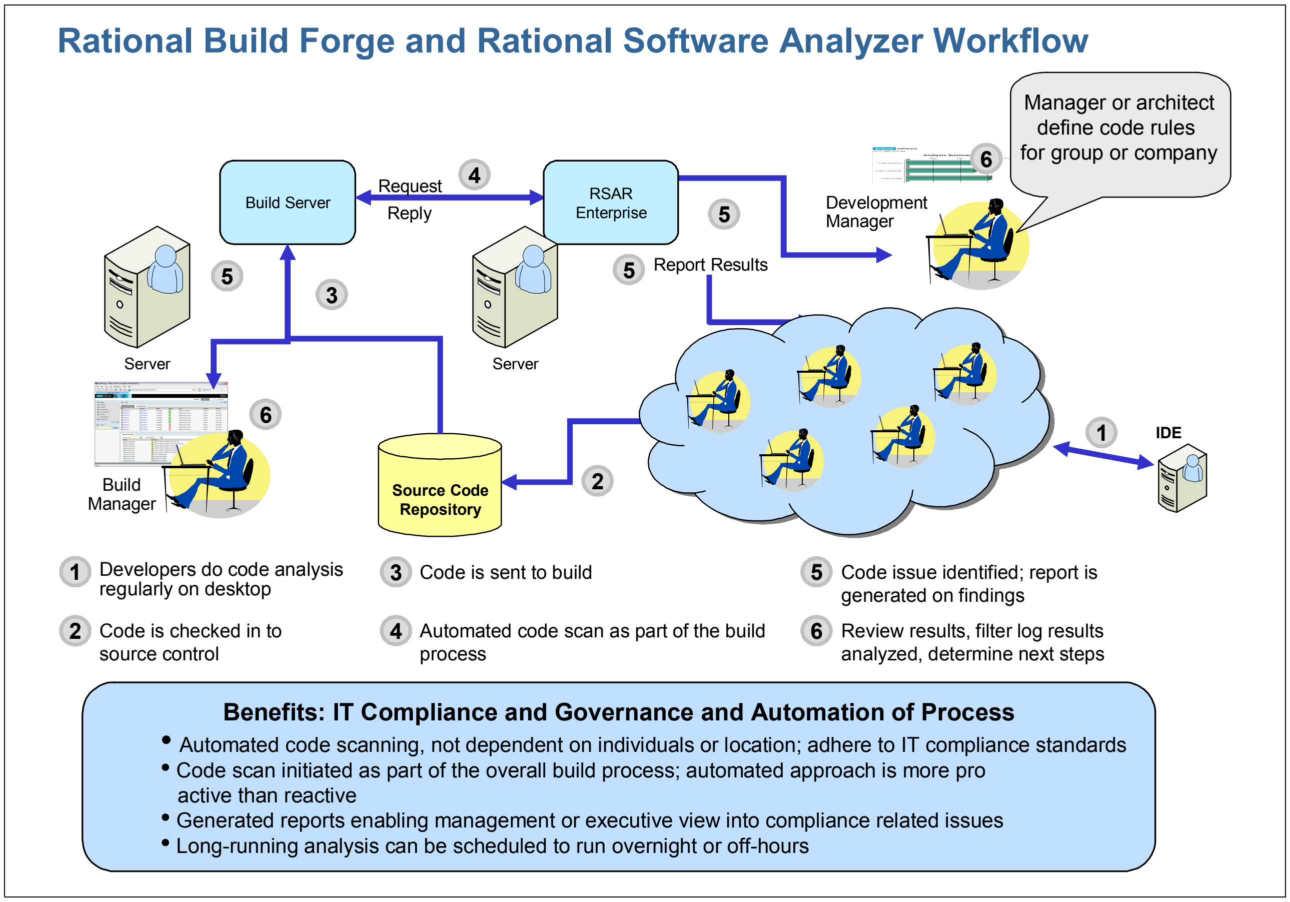
Many organizations understand the value of static analysis, but find it difficult to implement. This difficulty is due primarily to the fact that it requires a change in the team culture. This means that traditional approaches to static analysis required each developer to run the analysis on their local code base prior to delivering their change sets. This manual approach leads to inconsistent approaches and practices.

In some cases, static analysis might even be viewed as optional. When under pressure, it is easy to skip the static analysis step before delivering changes. By driving static analysis into the automated build process, you ensure that the analysis is performed regularly and that the same analysis rules are applied across the entire source code base.

In this section, we provide additional information about integration Rational Software Analyzer as part of the build process in Rational Build Forge.

Implementing automated code review analysis into the build process

Rational Software Analyzer provides flexible, automated analysis that you can configure based on your requirements (Figure 8-8).



Rational Build Forge and Rational Software Analyzer Workflow

1. Developers do code analysis regularly on desktop
2. Code is checked in to source control
3. Code is sent to build
4. Automated code scan as part of the build process
5. Code issue identified; report is generated on findings
6. Review results, filter log results analyzed, determine next steps

Benefits: IT Compliance and Governance and Automation of Process

- Automated code scanning, not dependent on individuals or location; adhere to IT compliance standards
- Code scan initiated as part of the overall build process; automated approach is more pro active than reactive
- Generated reports enabling management or executive view into compliance related issues
- Long-running analysis can be scheduled to run overnight or off-hours

Figure 8-8 Build Forge and IBM Rational Software Analyzer working together for automated code analysis

You can schedule Rational Software Analyzer to run as frequently as you need in conjunction with your automated build schedule. For example, you might decide that analysis should be executed for every release build or that analysis should take place for every build. Implementing automated code review into your build process is easy with Rational Software Analyzer and Rational Build Forge. Rational Software Analyzer includes a Rational Build Forge adapter that allows you to easily integrate code review and analysis into any Rational Build Forge project.

When an analysis project is run as a part of your build process, Rational Software Analyzer creates an analysis report and includes a link to the report in the build log of Rational Build Forge. Analysis reports are saved on the Rational Software Analyzer server and are accessible from any standard Web browser. Rational Software Analyzer also includes the results of the analysis in the build log.

When deciding how frequently you should do code analysis, ask yourself the following questions:

- Will a software architect or development lead be available to respond to discoveries at the regularly scheduled intervals?
- Is sufficient time allocated between analysis runs to respond to discovered discrepancies?
- How can we adequately address code review and analysis into our development rhythm without being counter productive?

  If your code review cycles are too frequent, you will create more frequent context switching, which can result in unproductive development cycles.

Results from Rational Software Analyzer can be reviewed at status meetings, and key stakeholders can decide on the required corrective actions to achieve acceptable quality levels.

Code analysis and review planning

When you institute code review and analysis for your application or set of applications by using Rational Software Analyzer, you create an analysis configuration. It is important to understand the following concepts and develop a plan when creating an analysis configuration.

Selecting an analysis domain and analysis type

First, you must decide on the language or languages that you will analyze and the type of analysis that is required. In Rational Software Analyzer, a language is associated with an *analysis domain*. For example, Java and C/C++ are analysis domains. For each analysis domain, analysis types have different analysis objectives. Such examples include running Java Software Metrics to measure code complexity, running Java Architectural Discovery to understand code structure, or running Java Data Flow Analysis to find memory and resource leaks. The documentation for Rational Software Analyzer provides a complete list of analysis types.

When you create your analysis configurations, it is best to create separate configurations for each analysis type since each type has a different objective. This improves your ability to interpret and segregate results making it easier to evaluate and solve them for your team.

Defining the scope of your analysis

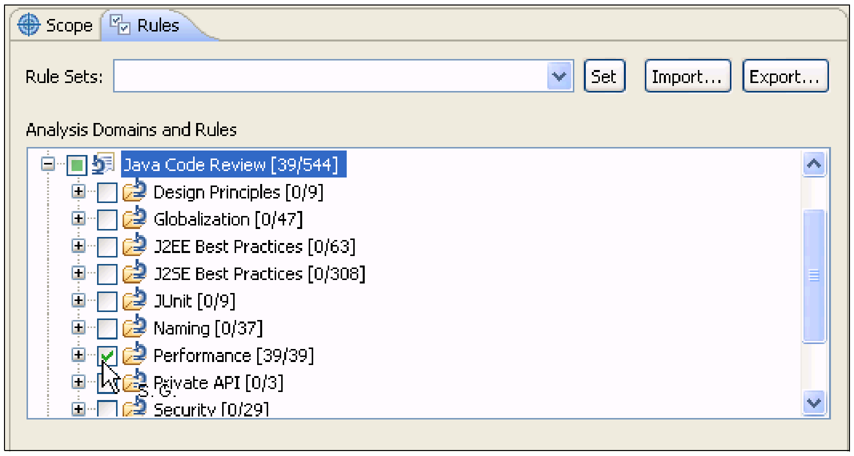
The analysis scope defines the files that you want to analyze for the rules that you have selected. You can analyze your entire application, potentially represented in multiple Eclipse projects. You can choose to create separate analysis configurations for components of your application, potentially represented in single projects. Finally, you can choose to analyze your application as a whole as well as specific configurations for your application components.

To maximize the usefulness of your code analysis, you must strike a balance between analysis depth and the scope of the analysis that you are conducting. For example, you can create an analysis configuration for your entire application with a small set of critical analysis rules across all analysis types. You can complement this by creating analysis configurations for specific components of your application if your project teams are segmented by component. In doing so, your component teams can better analyze and resolve the results. It also affords the individual component teams with flexibility in scheduling their code analysis cycle and focusing the analysis on the areas of primary importance for that component.

There is also benefit in dividing your analysis rules by type. For example, dataflow analysis rules are performed after a successful build. Therefore, you can choose to create an analysis configuration that includes your critical rules that must be resolved before an application can be delivered. Then you can place that configuration before your build step and create a separate configuration that includes your dataflow analysis rules and other noncritical rules. By dividing the analysis rules this way, you can save valuable build time by choosing not to build the application if your critical prebuild rules are not met. The value of such a configuration increases as the build time increases. For example, if you have a build that takes eight hours to complete, you can avoid building the application in instances where your prebuild code review rules are not met.

Selecting analysis rules

Rules are the smallest building blocks of a code analysis configuration in Rational Software Analyzer. Rules are grouped by domain, type, and category. For example, the rule "Avoid calling finalize() except in the finally block of the finalize method" lives in the Java Code Review domain and type under the Performance\Memory category and subcategory as shown in Figure 8-9.



- Java Code Review [39/544]
  - Design Principles [0/9]
  - Globalization [0/47]
  - J2EE Best Practices [0/63]
  - J2SE Best Practices [0/308]
  - JUnit [0/9]
  - Naming [0/37]
  - ✓ Performance [39/39]
  - Private API [0/3]
  - Security [0/29]

Figure 8-9 Code review rules organized by domain, type, and category

Rational Software Analyzer provides a large set of rules across a broad array of categories. When creating an analysis configuration, it is best to select a small set of logically related rules, which has the following benefits:

- Returns results faster for optimal performance
- Produces a smaller and more manageable set of analysis results that take less time to evaluate
- The ability to fix analysis results faster, which translates to better quality code in less time

Reference scenario for automated code review and analysis

In this reference scenario, Marco (development manager) collaborates with Rebecca (build manager), Tammy (quality manager), and Al (solution architect). Together they define the standard code review and analysis rule set for the application that will be integrated into the continuous integration process. The rule set spans code quality, code complexity, and code structure.

Project leads collaborate on code review and analysis standards

The project leads define the code review rules that they will institute and the acceptable thresholds. They select standard rules from code quality, code complexity, and code structure. For a detailed list of code analysis rules, see Appendix D, “Code review rules” on page 627. They decide on a few custom rules that are necessary.

Project leads define the code review and analysis schedule. The team decides to integrate the standard analysis rules into the nightly build and the weekly integration build. This way, developers can run the rule set on their component code throughout the week and fix problems that are found in analysis reports. Developers check in their component code daily for the nightly build.

Team leads define stakeholder reporting and communication requirements.

Rebecca configures Rational Build Forge and Rational Software Analyzer

In Rational Build Forge, Rebecca adds the general analysis scan to the Rational Build Forge projects that manage the daily build and the weekly integration build:

- She creates an analysis adapter that points to the source directories for the application and references the standard application rule set.
- She adds the adapter as a separate project step (with the `.source` command) to the Rational Build Forge projects that runs the nightly build and weekly build.

Rebecca creates user IDs for key stakeholders in Rational Build Forge and adds them to an e-mail notification group in Rational Build Forge.

Team leads establish quality baseline from initial code review

The analysis rules are run and the results are shared with the project leads. The project leads review the results and work with Al to address quality issues that are identified by the analysis. Al works with team to address issues.

Analysis rules are re-run to validate that the application meets quality standards. A quality baseline is established and agreed upon by the stakeholders.

Reports are used to communicate analysis results

Reports and the reporting server regularly provide everyone with a continuous view of code quality. Figure 8-10 shows an example report as displayed in a browser. Stakeholders receive the results by e-mail and access the report server for on-demand status. Rebecca creates a report category for the project and saves analysis reports. She turns on notification and sends the quality baseline results to stakeholders.

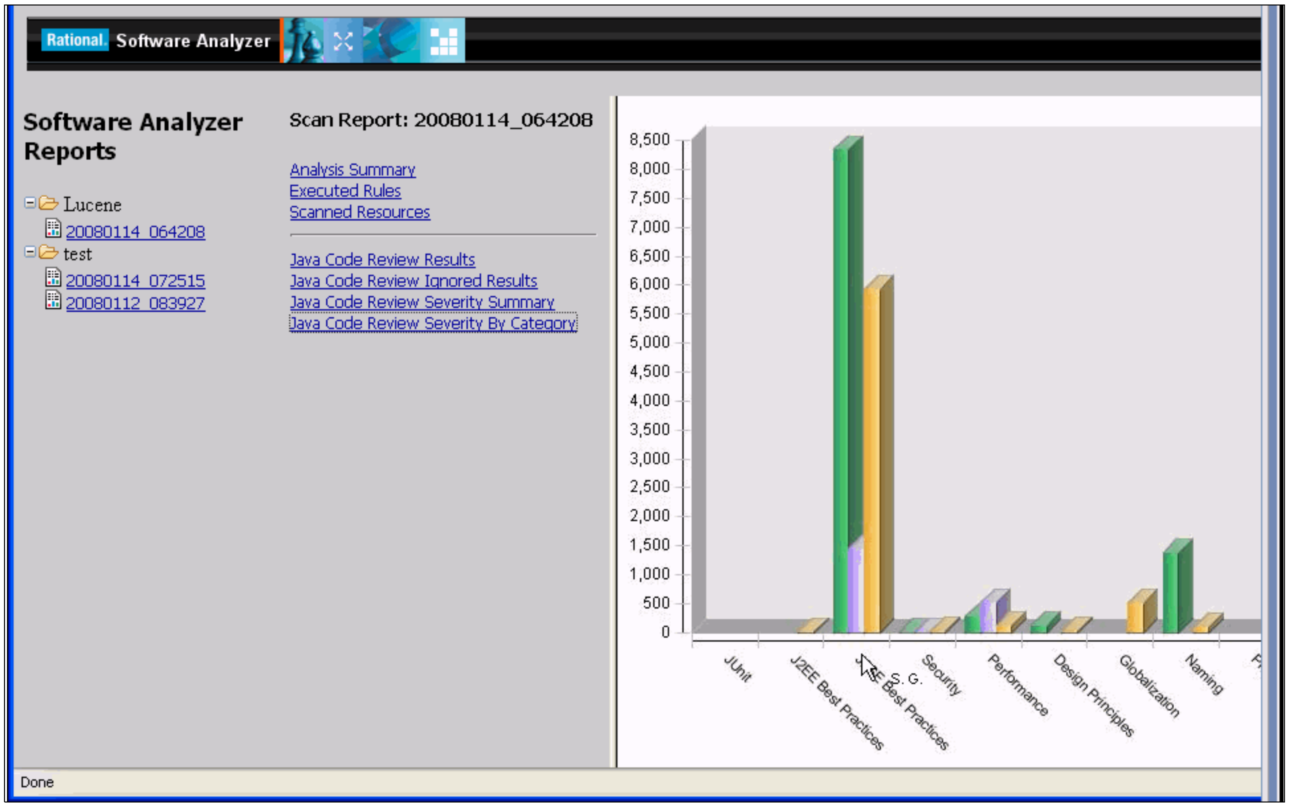


Figure 8-10 Example report from Rational Software Analyzer

8.3.5 Automating deployment

When we say "deployment," we often think of the final rollout of a software application into the production environment. In many cases, deployment is synonymous with "going live." However, deploying an application typically involves taking an application from a server (or servers) and installing and configuring it on another set of servers. This is clearly a simplified definition of the deployment challenge. However, by taking this simplified view, you can see a parallel activity taking place during the development of the application. Every time the application changes, it must be moved to the test servers for testing.

You can begin to view this as an act of deployment. Every time the application changes, it must be built and moved to the test servers for testing as illustrated in Figure 8-11 on page 338. Developers, build engineers, and deployers are each impacted by this need to deploy:

- A developer must build their changes and deploy to a sandbox for unit testing before delivering their source code changes.
- A build engineer conducts the build of all source code changes, deploys to a build verification server, and runs a basic set of tests to verify that the build is ready for the test team.

- A deployer must choose a build to deploy to the test servers, run a set of verification tests, and notify the rest of the test team that they can begin testing.

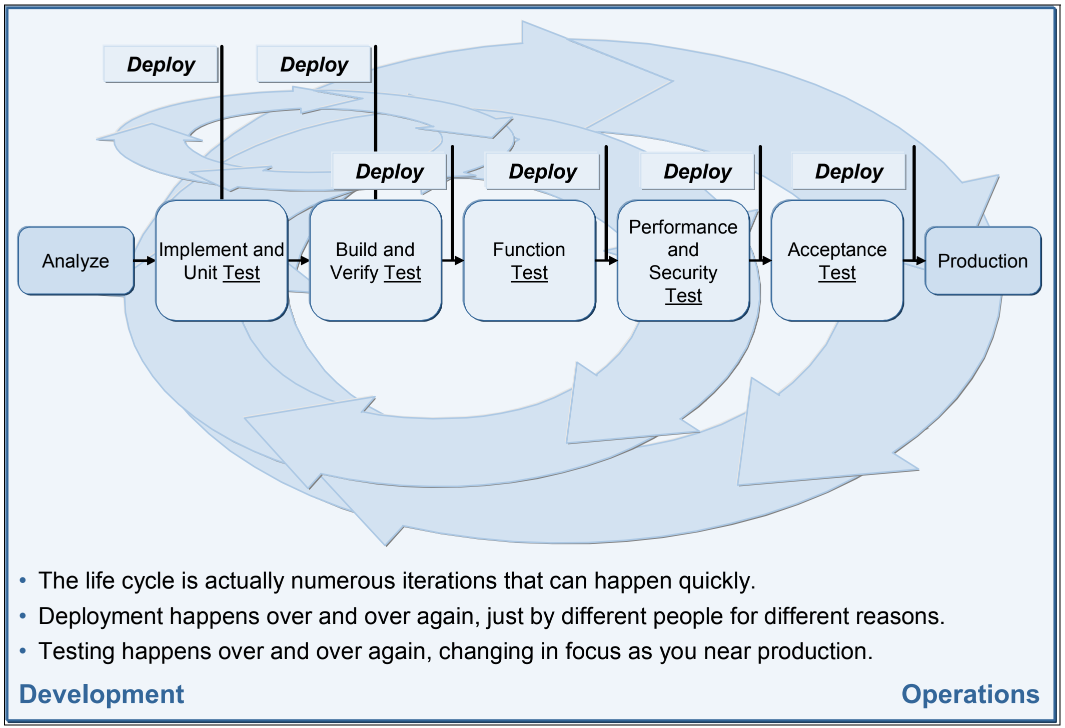

Deploy
Deploy
Deploy
Deploy
Deploy
Deploy
Analyze
Implement and Unit Test
Build and Verify Test
Function Test
Performance and Security Test
Acceptance Test
Production
• The life cycle is actually numerous iterations that can happen quickly.
• Deployment happens over and over again, just by different people for different reasons.
• Testing happens over and over again, changing in focus as you near production.
Development
Operations

Figure 8-11 Deployment occurring repeatedly until the final deployment into production

After all, the servers, middleware, and application must be configured properly for the application to run, and the results of the build process (the software application) must be moved to the test servers and configured for testing. By taking advantage of deployment to test, a development team can gain knowledge in the details of deploying the application.

Additionally, as a release candidate moves toward production, the solution is deployed through the various test environments of function, integration, system, performance, security, and user acceptance testing. In this case, the build is the same, but the underlying server configurations and the type of tests performed will change. Moving from one test environment to the next involves an act of deployment.

The challenge of deployment

A closer look at the system under test reveals a layer of configurations. First, the proper operating system and patch levels must be present. Second, the team must ensure that the middleware matches the production environment to the extent at which it is feasible. If automated test tools are used, they often require some software running on the server. Last but not least, the latest build of the application must be deployed onto the server.

Testing against an improperly configured server, middleware, or application can lead to results that differ when the application is rolled into production. Imagine having a test team spend weeks or months testing an application only to have it break when rolled into production because of a server configuration difference. Not only has your test team lost valuable testing time, they might be pulled into a long problem resolution cycle. The bottom line is that ensuring your test environment matches your production environment is the core best practice for reducing configuration errors when deploying an application.

Also the developers, build engineers, and deployers are each impacted by the need to deploy to a test server of some kind. If each of these servers is configured differently (which is most likely and common), how can you guarantee quality? A developer might claim that it works on their machine, which is a common claim. Sometimes it works because the developer's server is configured differently than the test team's server. This configuration information is highly valuable across all roles on the development team. Additionally, this same configuration information must be communicated to the operations team for the deployment into production to be successful.

How we do ensure that these complex configurations match? Many organizations rely on highly skilled IT professionals to configure the servers, middleware, and applications. Not only are these repetitive tasks, but the configurations are complex enough that a simple oversight can lead to errors in testing or production. It is an inefficient use of IT resources and an expensive choice, because some of the best employees are needed to perform this critical function. IBM Tivoli Provisioning Manager automates the provisioning of servers. Just as Rational Build Forge provides a flexible framework for automating the build process, Tivoli Provisioning Manager provides a framework for automating the provisioning of servers from bare metal to the top of the software stack.

The bottom line is that, not only is manual configuration of servers expensive, it can be error prone. Adopting strategies for automating the provisioning of servers becomes a core best practice. In doing so, the scripts that are used to build a server used in production can be reused to build the server in the test lab.

Automating deployment

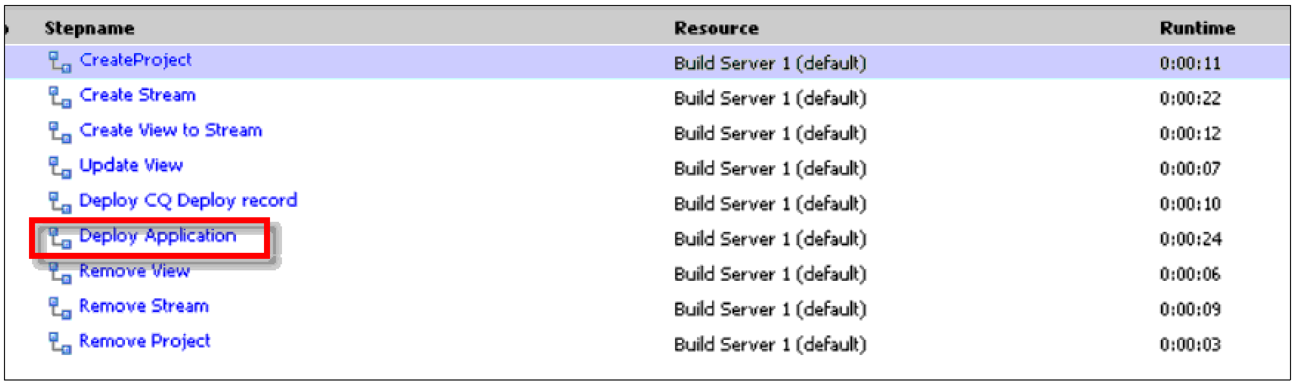
When viewed from this perspective, the repetitive nature of deployment becomes obvious, and the opportunity for automation surfaces. Rational Build Forge provides an automation framework that can address part of this need. For simple applications, you can create a Rational Build Forge project that uses File Transfer Protocol (FTP) to transport the application and a set of scripts to install it on the host machine. For more complex applications, Rational Build Forge can be used to call Tivoli Provisioning Manager to do the deployment. Tivoli Provisioning Manager provides an automation framework that targets the provisioning of servers from bare metal through the application layers and even to the network and firewall settings.

Rational Build Forge is integrated with Rational ClearCase to extract the latest of version of the source code. Rational Build Forge features build auditing capabilities that enable you to keep bills of materials for your builds and to access and reproduce previous build artifacts as needed. Additionally, use Rational Build Forge to check the build results back into Rational ClearCase, and to create build and deployment records in Rational ClearQuest.

Automating deployment as shown in Figure 8-12 on page 340 involves the following approach:

- Use Rational ClearCase to version control your source code, build scripts, and build results. When used with Rational ClearQuest, leverage UCM to manage team activities.
- By using Rational Build Forge, you can create a build process that retrieves the distribution archive from Rational ClearCase. This assumes that the archive was previously checked in by an automated build process as demonstrated in the reference scenario in this book.

- Use Rational ClearQuest to track builds and deployments through your test environments. The reference scenario demonstrates how you can automate the creation of an ALMBaseline and BTBuild record in Rational ClearQuest. You can also implement a process for managing the deployment of your release through your test environments by using a *Rational ClearQuest deployment record*. A Rational ClearQuest deployment record enables you to track the set of build artifacts that you want to deploy through a link to a Rational ClearCase deployment unit. You can use Rational Build Forge to automatically create the deployment unit and manage it in Rational ClearCase.
- The deployment unit can then be used by Tivoli Provisioning Manager to deploy the build results to the test server. This integration automates your deployment process for your specific servers. Your Rational Build Forge project includes a call to Tivoli Provisioning Manager, which then conducts the deployment. When Tivoli Provisioning Manager has completed its work, the result is reported back to the step in the Rational Build Forge project.



Stepname	Resource	Runtime
CreateProject	Build Server 1 (default)	0:00:11
Create Stream	Build Server 1 (default)	0:00:22
Create View to Stream	Build Server 1 (default)	0:00:12
Update View	Build Server 1 (default)	0:00:07
Deploy CQ Deploy record	Build Server 1 (default)	0:00:10
Deploy Application	Build Server 1 (default)	0:00:24
Remove View	Build Server 1 (default)	0:00:06
Remove Stream	Build Server 1 (default)	0:00:09
Remove Project	Build Server 1 (default)	0:00:03

Figure 8-12 Automating the deployment step by using Rational Build Forge

IBM TechWorks provides a Proof-of-Technology hands-on workshop called "Discovering Build and Deployment Automation with IBM Rational and Tivoli Solutions" that demonstrates this capability. IBM customers can contact their sales representative to arrange to enroll in the Proof-of-Technology workshop. The session is offered free of charge. IBM employees can access this Proof-of-Technology workshop from the IBM TechWorks Web site.

9

Rational Build Forge for enterprise integration build

In this chapter, we include the following sections:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

9.1 Act 3: Enterprise integration build

In this chapter, we include a step-by-step discussion about how the characters in the story complete Act 3 of the storyboard (Figure 9-1).

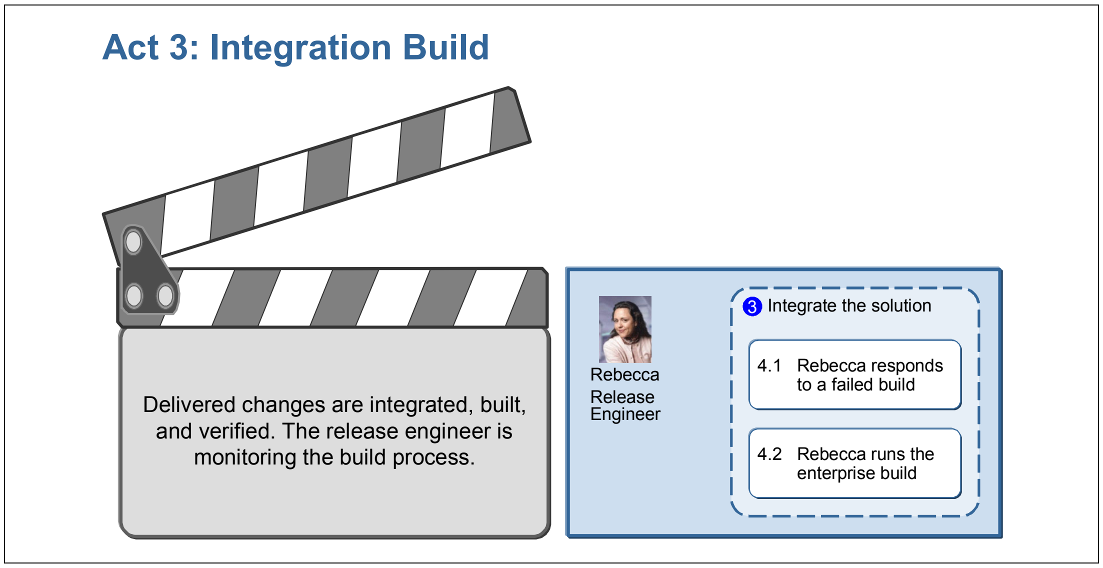

Act 3: Integration Build
Delivered changes are integrated, built, and verified. The release engineer is monitoring the build process.
Rebecca Release Engineer
3 Integrate the solution
4.1 Rebecca responds to a failed build
4.2 Rebecca runs the enterprise build

Figure 9-1 Enterprise build management

This act has the following scenes:

- Rebecca responds to a failed build.
- Rebecca runs the integration build.

The following IBM Rational products are used in this act:

- IBM Rational Build Forge Enterprise Edition
- With integrations to Rational ClearCase, Rational ClearQuest, and Rational Software Analyzer

9.2 Rational Build Forge Enterprise Edition

In this section, we provide background information about Rational Build Forge Enterprise Edition. By understanding the capabilities in this section, you will better understand the step-by-step instructions that are provided in 9.3, “Using Rational Build Forge for an enterprise integration build” on page 357.

Rational Build Forge is a centralized build and release management solution. By centralizing the build and release processes, teams can streamline and optimize their build processes, automatically produce a bill of materials for each job that is run, have a full audit trail for the build and release process, and leverage reports to pinpoint problem areas and optimize performance. Additionally all team members can use the same build process when conducting a build, thus eliminating errors related to different build scripts, environments, or compilers.

9.2.1 Process automation framework

Rational Build Forge provides an adaptive framework that allows development teams to standardize and automate repetitive tasks, share essential product information, and respond quickly to change. After the processes are implemented in Rational Build Forge, all stakeholders in the team can use them within their permissions.

Without using an adaptive framework, such as Rational Build Forge as shown in Figure 9-2, continuous build integration and automation can be a challenge for contemporary development teams that develop in different locations and use different time zones and different tools. Figure 9-2 illustrates how Rational Build Forge automates the code-build-release-test iterations. Rational Build Forge can integrate with any third-party system or tool that is accessible from the command line, API, or any script. You can integrate with your source control system, test system, and defect tracking systems to capture information that is relevant to that release.

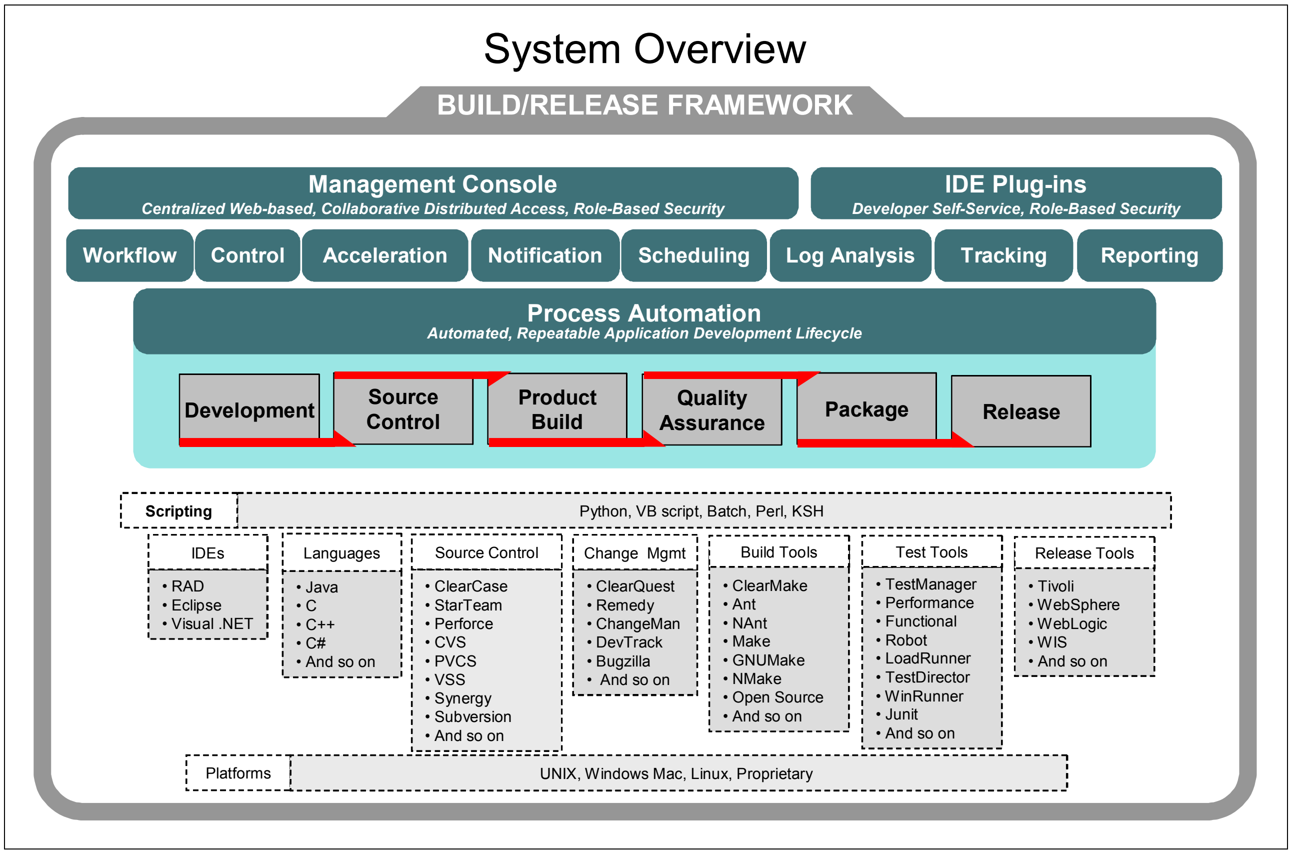


Figure 9-2 Rational Build Forge overview

All stakeholders in a team can run easily and effectively run the automation by using the Rational Build Forge integrated development environment (IDE) plug-in or the Rational Build Forge Management Console. The results of each automation job is captured in a detailed bill of materials. The Rational Build Forge framework also ensures that all processes are run consistently by the team members who have been given access to do so.

Rational Build Forge also offers the following benefits:

- Reports success or failure through e-mail, RSS feeds, and on a Web-based dashboard
- Stores attempted commands and the resulting output or error messages in logs
- Can schedule the project for repeated runs, guaranteeing that the standard process is executed the same way, every time, and occurs as often as needed

Figure 9-3 shows the Rational Build Forge ecosystem, which includes the following components:

- The Rational Build Forge core components, which consist of the engine, management console, and services layer with a quick report, are used to manage, automate, and report on the automation projects.
- A pool of worker machines is used to execute the automation jobs. The system can find available server resources at run time, rather than relying on the availability of a single machine.
- Client access is available via a Web-based console or IDE plug-in for use by developers.

Architecturally, Rational Build Forge has a three-tiered structure:

- The Management Console
- The Build Forge Engine
- Build Forge Agents

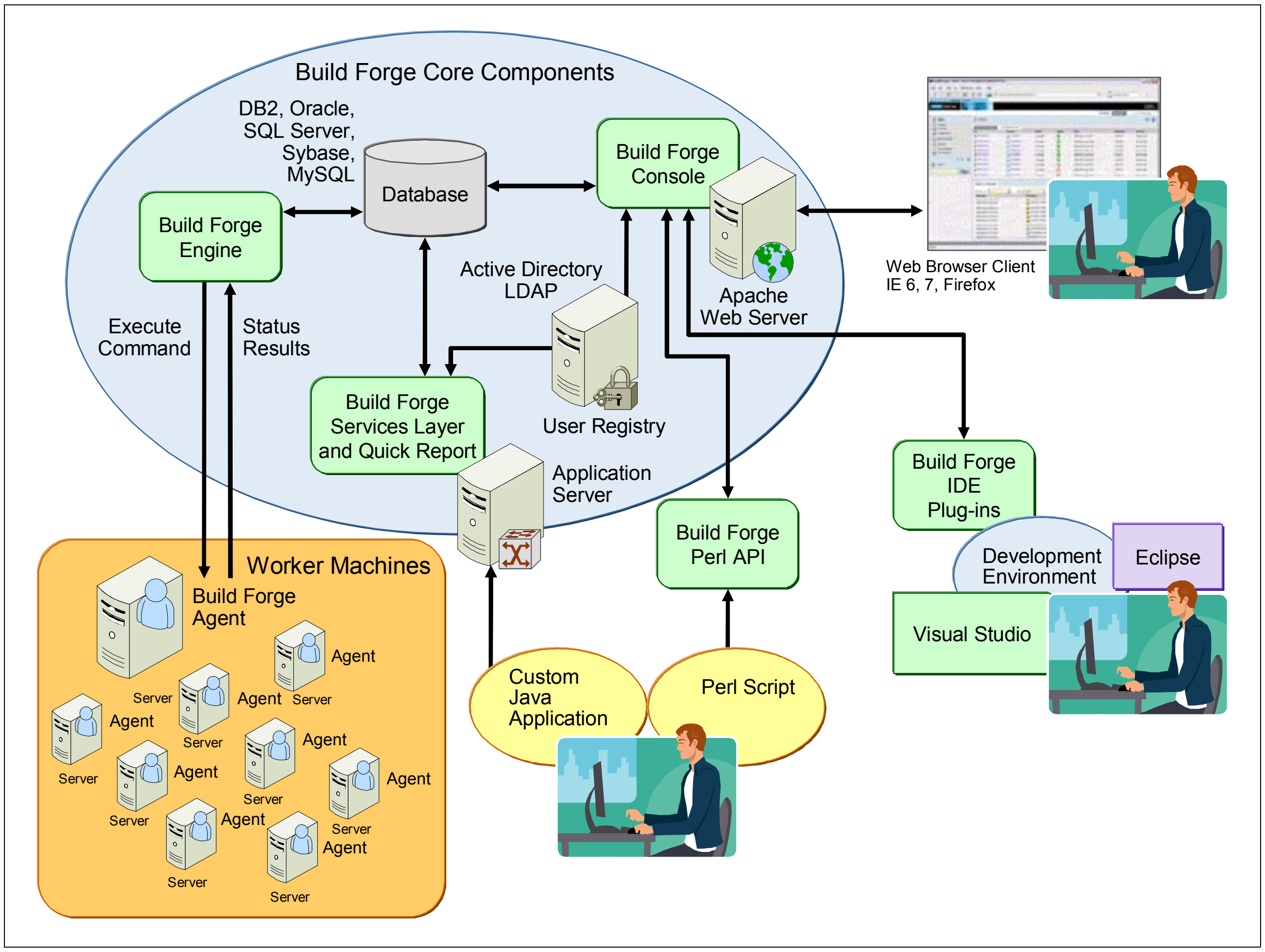


Figure 9-3 Rational Build Forge product architecture

The Management Console provides a user interface to the system. It is a Web-based PHP application that runs on an Apache HTTP server. Through it, you can organize steps into projects and manage the server resources and environment variables that those steps need.

Step: A step is a sequence of one or more commands for a specific purpose in Rational Build Forge.

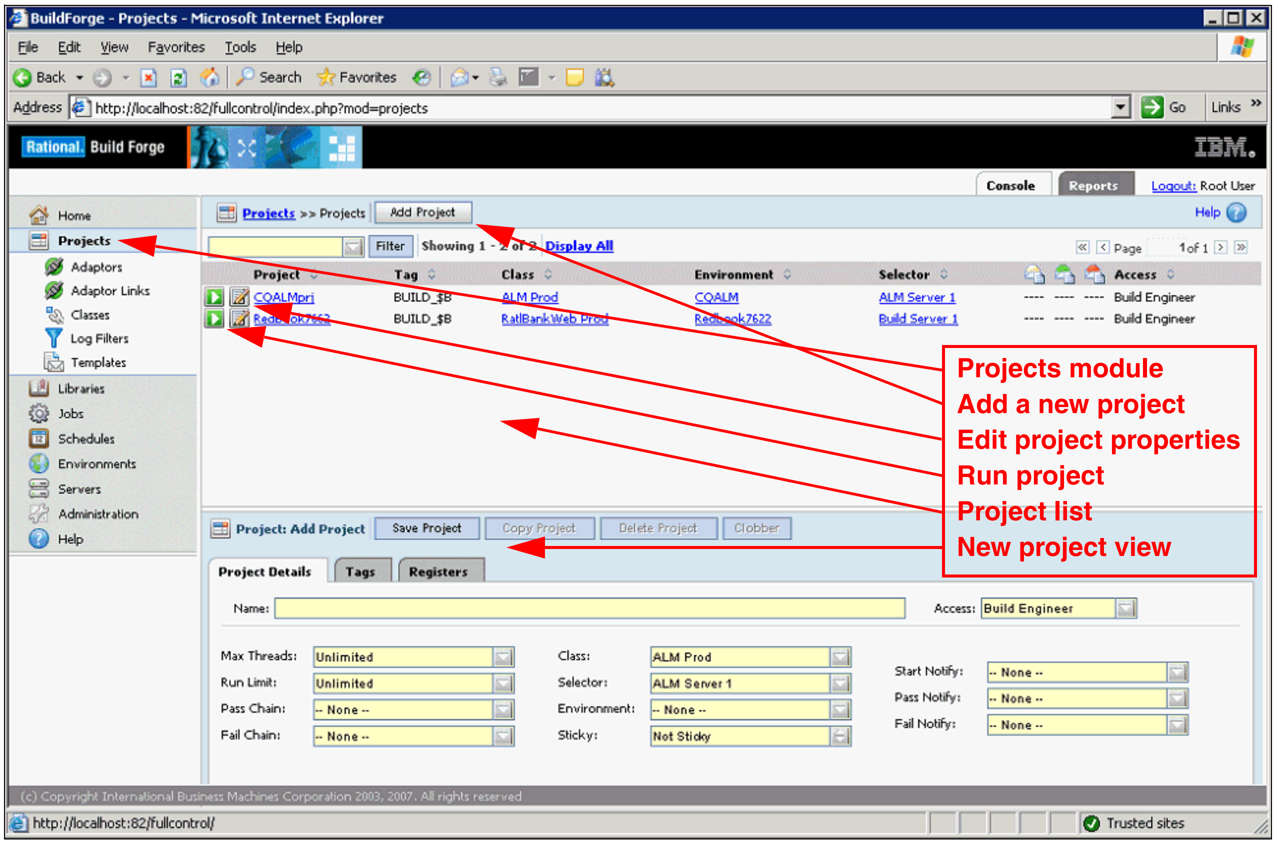
The Build Forge Engine acts on instructions that were entered from the Management Console. It uses information that is stored in the database to communicate with agents, execute project tasks, and perform notifications. It also stores comprehensive project information and tracks each run of a project. User and system actions are stored in it, so that auditing and reporting data can be extracted and analyzed.

Build Forge agents are deployed on each server where the project must run a command. The agent is a compact small binary that performs the following tasks:

- Listens for commands from the Build Forge Engine
- Executes the commands
- Sends results from the commands back to the Build Forge Engine

9.2.2 Projects

Each automation is defined and organized by a project. You use the Projects module to create new projects and edit or view existing projects (Figure 9-4). *Projects* are executable sets of steps, with their own environment group and server properties.


BuildForge - Projects - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:82/fullcontrol/index.php?mod=projects
Go Links
Rational. Build Forge
IBM.
Console Reports Logout: Root User
Home
Projects
Adaptors
Adaptor Links
Classes
Log Filters
Templates
Libraries
Jobs
Schedules
Environments
Servers
Administration
Help
Projects >> Projects Add Project
Help
Filter Showing 1 - 2 of 2 Display All
Page 1 of 1
Project: Add Project Save Project Copy Project Delete Project Clobber
Project Details Tags Registers
(c) Copyright International Business Machines Corporation 2003, 2007. All rights reserved
http://localhost:82/fullcontrol/
Trusted sites
Projects module
Add a new project
Edit project properties
Run project
Project list
New project view

Project	Tag	Class	Environment	Selector				Access
CQALMprj	BUILD_$B	ALM Prod	CQALM	ALM Server 1	----	----	----	Build Engineer
Redbook7662	BUILD_$B	RatlBankWeb Prod	Redbook7622	Build Server 1	----	----	----	Build Engineer

Name:
Access: Build Engineer

Max Threads:	Unlimited	Class:	ALM Prod	Start Notify:	-- None --
Run Limit:	Unlimited	Selector:	ALM Server 1	Pass Notify:	-- None --
Pass Chain:	-- None --	Environment:	-- None --	Fail Notify:	-- None --
Fail Chain:	-- None --	Sticky:	Not Sticky		

Figure 9-4 Projects module

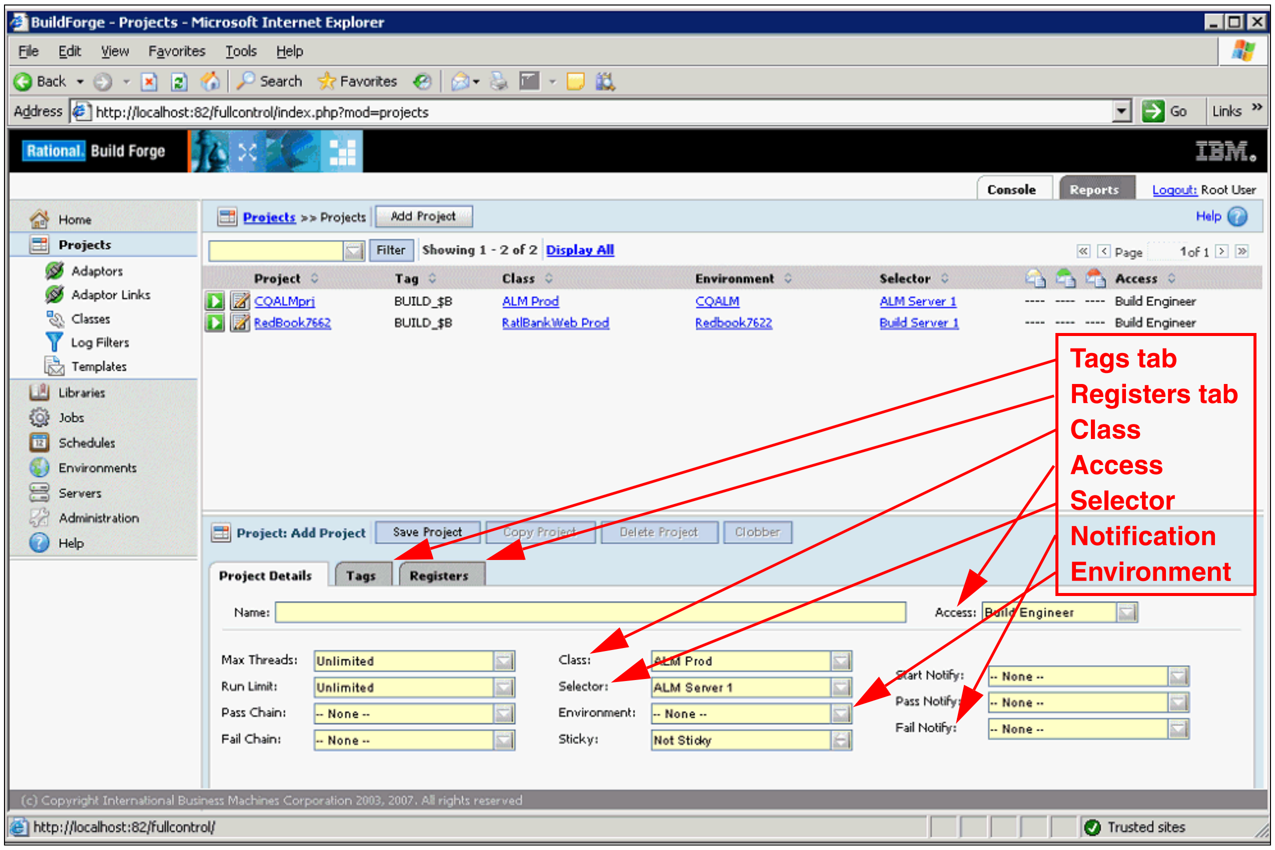
To change project-level properties (Figure 9-5), select **Projects**. Then click the icon next to the desired project's name. The name of the project is used to refer to the project in lists and in the database. The project name is used to construct the project directory when the project is executed.

Tip: With the Rational Build Forge system, you can use any characters in the project name. Since a project can contain steps that run on different operating systems, avoid using special characters and symbols that can cause problems on those operating systems.

A project contains a wealth of information. Some of the information is captured in a set of steps. Other information is captured on a set of tabs:

- Project details
- Tags
- Registers

In this section, we provide information about the Project Details and Tags tabs.


BuildForge - Projects - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:82/fullcontrol/index.php?mod=projects
Go Links
Rational. Build Forge
IBM.
Console Reports Logout: Root User
Home
Projects
Adaptors
Adaptor Links
Classes
Log Filters
Templates
Libraries
Jobs
Schedules
Environments
Servers
Administration
Help
Projects >> Projects Add Project
Help
Filter Showing 1 - 2 of 2 Display All
Page 1 of 1
Tags tab
Registers tab
Class
Access
Selector
Notification
Environment
Project: Add Project Save Project Copy Project Delete Project Clobber
Project Details Tags Registers
(c) Copyright International Business Machines Corporation 2003, 2007. All rights reserved
http://localhost:82/fullcontrol/
Trusted sites

Project	Tag	Class	Environment	Selector				Access
CQALMprj	BUILD_$B	ALM Prod	CQALM	ALM Server 1	----	----	----	Build Engineer
RedBook7662	BUILD_$B	RatlBankWeb Prod	Redbook7622	Build Server 1	----	----	----	Build Engineer

Name:
Access: Build Engineer

Max Threads:	Unlimited	Class:	ALM Prod	Start Notify:	-- None --
Run Limit:	Unlimited	Selector:	ALM Server 1	Pass Notify:	-- None --
Pass Chain:	-- None --	Environment:	-- None --	Fail Notify:	-- None --
Fail Chain:	-- None --	Sticky:	Not Sticky		

Figure 9-5 Project properties

The *Access property* is the access group that is allowed to view and use the project. The Access property is used along with permissions to determine what a user can do. For example, to launch a job, you must be a member of the access group that is specified for the project, and you must be a member of a group that has the Execute Jobs permission. Rational Build Forge has detailed permissions that enable you to differentiate the levels of control. For example, you can permit a developer to run a project but restrict the ability to edit the project.

The *Max Threads property* is the maximum number of parallel processes that the project is allowed to launch. Use this field to keep a project that uses an optimized number of system resources. Each thread-enabled step and any inline projects (which themselves might launch thread-enabled steps) can result in parallel processes, but all of those processes are counted against the maximum for the parent project. The system stops launching new parallel processes when it reaches the Max Threads value and waits until the number of parallel processes for the project drops below the Max Threads value before continuing. You modify this property to optimize performance.

The *Run Limit property* sets the maximum number of jobs of the project that are allowed at one time. If you launch a project, but the currently active jobs already equal the limit, the new job stays in the Waiting queue until one or more of the jobs completes. If a schedule attempts to launch a project when the number of running projects equals the run limit, the system does not launch a new job at all. Also, projects that are launched via an inline chain are not considered instances of the original project and do not count toward its run limit.

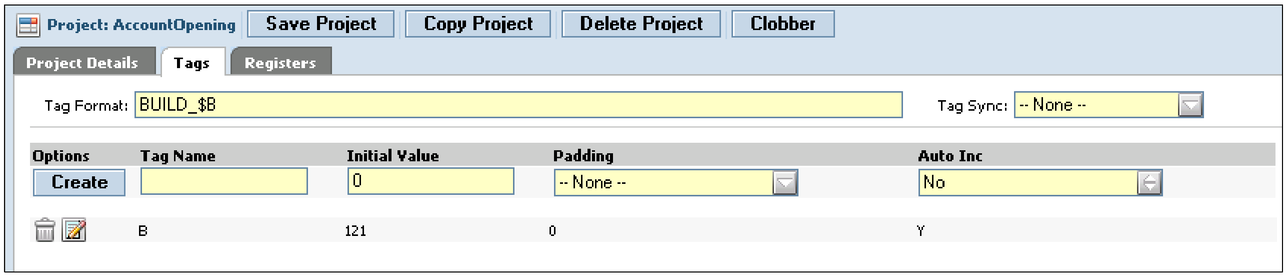
Each project must be assigned to a *class*, which assigns global properties to groups of projects. A class defines the interval at which Management Console and data are removed for a project run. For additional flexibility, a project might be executed when the purge process is executed.

The *Selector property* is the name of the selector to use when choosing a server for the project. The system uses this selector as the default for any steps within the project that do not specify their own selectors. If a selector is not specified, the project is added to the Libraries module instead of the Projects module. Library projects use the selector of the calling project.

The *Environment property* is an environment to apply after the server environment and before the step environment.

The *Start Notify, Pass Notify, Fail Notify properties* direct the system to send notification e-mails on project start, pass, or fail, by selecting an access group in one or all of these fields.

Many times a label is needed to identify the job that run. The *Tag Format property* on the Tags panel (Figure 9-6) uses plain text and tag variable references to provide such an identifier for the project.



Tag Format: BUILD_$B

Tag Sync: -- None --

Options	Tag Name	Initial Value	Padding	Auto Inc
Create		0	-- None --	No
	B	121	0	Y

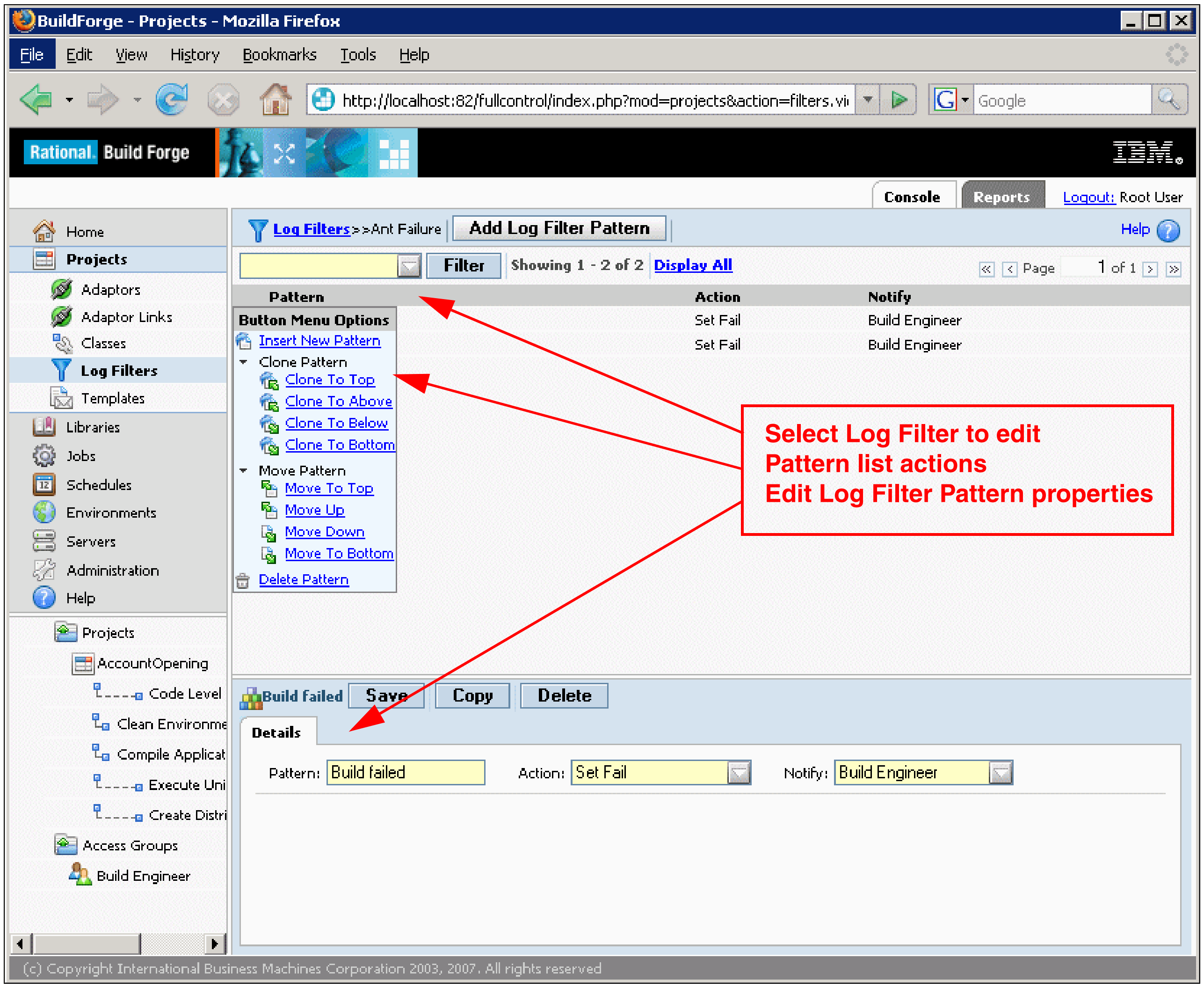
Figure 9-6 Tags in a project

You select the Sticky check box to force all the steps of the project that use the default project server to stay on the same server and to wait for it to become available if it is busy. Steps within a project can run on different servers if their selectors allow it. However, you might want all or most of the steps of a project to run on the same server, whether you specify that server in advance.

Note: In the reference scenario, Rebecca manages the AccountOpening project in Rational Build Forge.

Log filters

You use log filters (Figure 9-7) to change the success criteria for a step. By using log filters, you can evaluate step output and set step results to Fail, Pass, or Warn based on the contents of the step output. This gives you the ability to closely control the criteria that is used to determine step success or failure. If filtering is not set up, Rational Build Forge determines the success or failure of any step command by its exit status, where 0 is success and 1 is failure.



Pattern	Action	Notify
	Set Fail	Build Engineer
	Set Fail	Build Engineer

Pattern: Build failed Action: Set Fail Notify: Build Engineer

Figure 9-7 Log filters and patterns

For example, some commands always return an exit status of 0. A reporting command, such as **`net use`**, prints a list of mapped network drives. The command always succeeds, even if the list does not contain the desired drive. Using a filter set, you can parse step output to look for a specific drive and mark the step as successful if it is found.

Log filters can contain one or more filter patterns. Each filter pattern is associated with an action and optionally an access group for notification. To use the log filter, you must assign the log filter to a project step by using the step Result property.

When you assign a log filter to a step, the filter patterns in the log filter are run on the step output whenever the project runs. When you assign a log filter to a step, the step result that is set by the log filter *overrides all other criteria* for determining the success or failure of the step. This includes the exit status for the step commands or any step properties.

For example, if the step run time exceeds the time that is specified by the step Timeout property, the step stops. But its status is not considered a failure unless its associated log filter action causes it to be set to Fail.

Filter patterns

A *filter pattern* defines the character string or expression that you want to match in the step output. Each filter pattern that you create is associated with a single filter action. Both filter patterns and actions are defined in filter log sets. The ability to include multiple filter patterns in a log filter and apply it to output from a single step allows you to use multiple search criteria without constructing complex expressions (Figure 9-7 on page 348).

Note: Rebecca uses a filter pattern in the first step of the AccountOpening project. See "How Rebecca defined the project steps" on page 364 for information about her use case.

Classes

A *class* is a group of jobs. Each job must be a member of only one class. You can use classes to set up types of jobs and apply behavior to each type globally. A job gets its default class from the properties of its project, but you can manually choose a different class for a job when you launch it by selecting **Jobs** → **Start page**.

9.2.3 Jobs

The Jobs module provides a list of executed projects that you can use to view project outcomes. Use the Jobs module to launch projects, view the results of earlier runs, and get information about currently running projects.

The Jobs module has the following tabs:

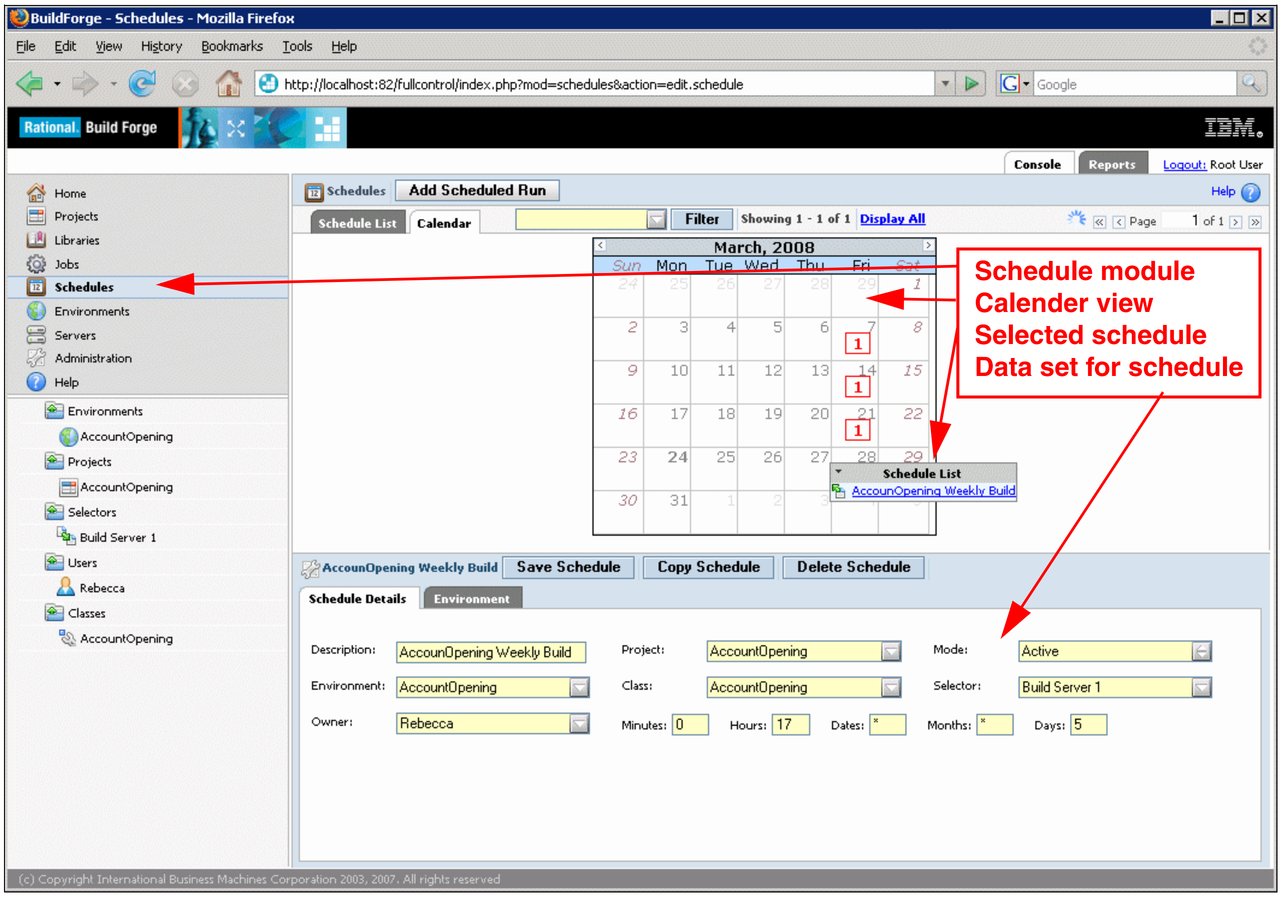
- The *All tab* lists all projects regardless of job status: completed, running, archived, or locked. Use the All tab to locate a project if its status is unknown. Projects that are listed on the All tab are displayed in the following order:
  a. Currently running jobs
  b. All other jobs in order of date and time completed, with the most recently completed job listed first

  Click a project to display information about the project.
- The *Completed tab* lists finished jobs whose logs and data you have permission to view.
- The *Running tab* lists projects that are currently running on any of the servers known to the Management Console. Use the Running tab to view projects that are in process. The system lists jobs on the Running tab until they have completed. After that, it lists them on the Completed tab.
- The *Archived tab* in lists information about project runs whose file data has been deleted, but about which the database retains console data. Use the Archived tab to view information about jobs that have been purged.

Note: In the reference scenario, Rebecca responds to a job that failed to complete. Upon resolving the error, she restarts the job.

9.2.4 Schedule

After you create a project, you can schedule it to run at a future time or at regular, repeated intervals (Figure 9-8).


BuildForge - Schedules - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:82/fullcontrol/index.php?mod=schedules&action=edit.schedule
Rational. Build Forge
IBM.
Console Reports Logout: Root User
Home
Projects
Libraries
Jobs
Schedules
Environments
Servers
Administration
Help
Schedules Add Scheduled Run
Schedule List Calendar Filter Showing 1 - 1 of 1 Display All
Page 1 of 1
March, 2008
Schedule List
AccounOpening Weekly Build
Schedule module
Calender view
Selected schedule
Data set for schedule
AccounOpening Weekly Build Save Schedule Copy Schedule Delete Schedule
Schedule Details Environment
(c) Copyright International Business Machines Corporation 2003, 2007. All rights reserved

Description:	AccounOpening Weekly Build	Project:	AccountOpening	Mode:	Active
Environment:	AccountOpening	Class:	AccountOpening	Selector:	Build Server 1
Owner:	Rebecca				

Minutes:	0	Hours:	17	Dates:	*	Months:	*	Days:	5

Figure 9-8 Schedule module

For example, you can set up a project to run every hour or every day as shown in Example 9-1. Project defaults, such as Selector, Environment, and Class, can be modified for a project's schedule. When adding a schedule, an icon appears in the Calendar view. The number displays the number of schedules that are set for that day. Hover over the number in the Calendar view to see what is scheduled, and hover over the Schedule to view the schedule configuration. When many schedules exist, use the Schedule Filter to view the projects for which you want to view schedules

Example 9-1 Scheduling Jobs

```
Run project every day except weekends, at 4:30 pm and .11:30 pm.
Minutes → 30
Hours → 16,23
Dates → *
Months → *
Days → 1-5
```

When the system has computed the next run time for the project, it displays it in the Next Run column. The system displays a dynamic calendar on the Schedules page, as well as the form displayed when modifying a schedule. The calendar shows the number of projects that are scheduled for a given day, for two months (the current and upcoming months). You can hover over individual days to see the names and schedule parameters of all the projects scheduled for a given day. If you have more than one project scheduled, the system displays a list so that you can filter the calendar display by project. You can disable a schedule temporarily or configure it to run once.

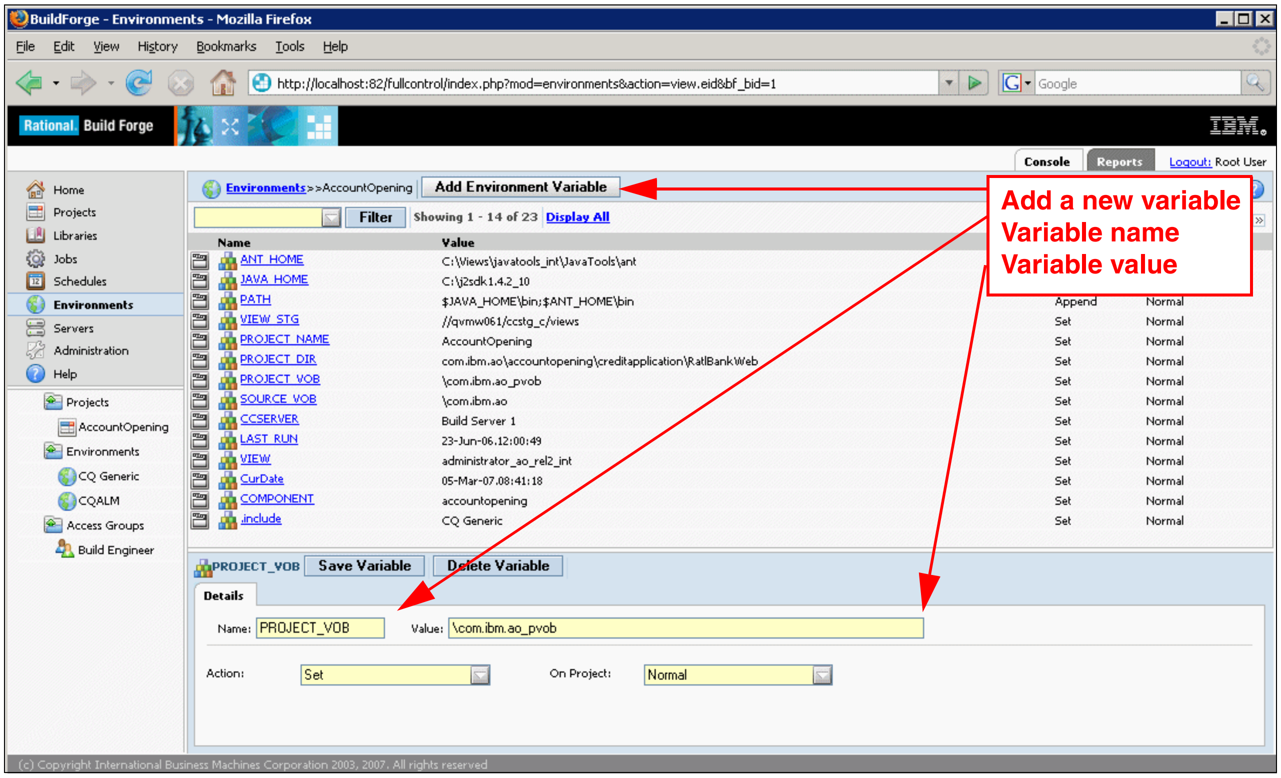
Note: In the reference scenario, Rebecca has scheduled this project to run weekly.

9.2.5 Environments

With the Rational Build Forge system, you can manage environment variables separately from the projects and servers to which they apply. You use the Environments module to create and edit environments that can be applied to servers, projects, and steps (Figure 9-9). This paradigm provides a great deal of flexibility in creating the environment for any particular command:

- You can create environments that contain environment variables.
- You can use the `.include` command to nest environments together.
- You can assign one environment to each server, one to each project, and one to each step within a project.

Environments are a powerful feature that allows projects to have broader applicability. Step behavior is not hardcoded and can be varied easily by changing the value of relevant environment variables. Also using environment variables is highly recommended for increasing easy maintainability.



Name	Value		
ANT_HOME	C:\Views\javatools_int\JavaTools\ant		
JAVA_HOME	C:\j2sdk1.4.2_10		
PATH	$JAVA_HOME\bin;$ANT_HOME\bin	Append	Normal
VIEW_STG	//qvmw061/ccstg_c/views	Set	Normal
PROJECT_NAME	AccountOpening	Set	Normal
PROJECT_DIR	com.ibm.ao\accountopening\creditapplication\RatlBankWeb	Set	Normal
PROJECT_VOB	\com.ibm.ao_pvob	Set	Normal
SOURCE_VOB	\com.ibm.ao	Set	Normal
CCSERVER	Build Server 1	Set	Normal
LAST_RUN	23-Jun-06.12:00:49	Set	Normal
VIEW	administrator_ao_rel2_int	Set	Normal
CurDate	05-Mar-07.08:41:18	Set	Normal
COMPONENT	accountopening	Set	Normal
.include	CQ Generic	Set	Normal

Name: PROJECT_VOB Value: \com.ibm.ao_pvob

Action: Set On Project: Normal

Figure 9-9 Environments module

Before the system executes a step, it applies all the relevant environments for the step to create the step environment. It evaluates the server, project, and step environments, in that order, which has the following ramifications:

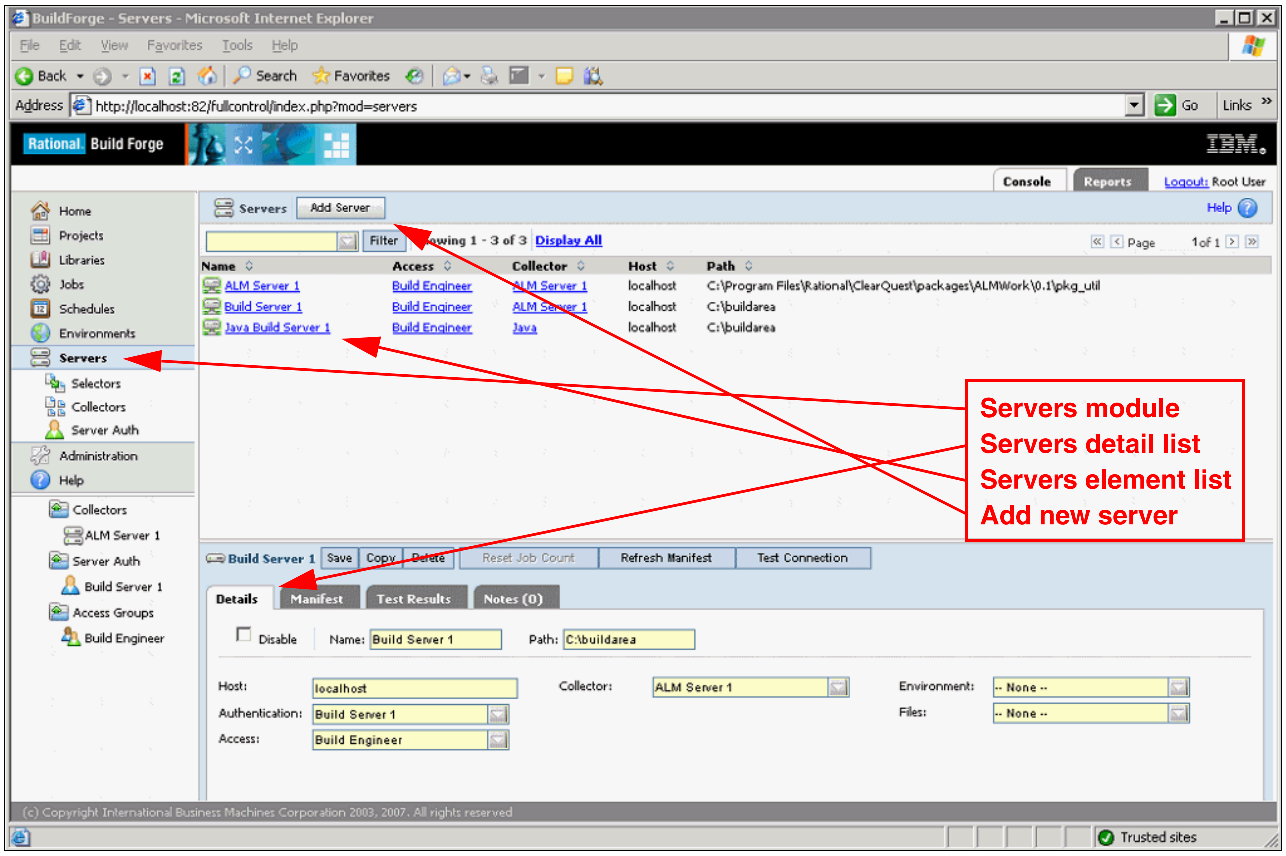
- The step gets the environment variables for the server on which it runs.
- Project variables can override server variables, and step variables can override project variables. The last value set wins.You can use variable actions to change this behavior.

When the system starts a job, it copies the project environment variables to a database record that is set aside for the job, and refers to this job environment thereafter when getting project default values. If the user modifies the starting values of any project variables when the user starts the job, those values are recorded in the job record.

Note: In the reference scenario, Rebecca has defined a number of variables by using the Environment feature. See "How Rebecca used the environment variables" on page 372.

9.2.6 Servers

A *server* is a logical reference to a physical host that the Management Console uses to execute projects or steps. Rational Build Forge provides a user interface for managing servers (Figure 9-10). The server maps to a physical machine through its definition, allowing dynamic allocation of physical resources, without requiring modification of individual projects. The ability to dynamically allocate server resources provides key benefits such as fault tolerance and optimal hardware utilization. To be used as a server host, a machine must have an agent installed, and it must be configured in the Management Console.


BuildForge - Servers - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:82/fullcontrol/index.php?mod=servers
Go Links
Rational. Build Forge
IBM.
Console Reports Logout: Root User
Home
Projects
Libraries
Jobs
Schedules
Environments
Servers
Selectors
Collectors
Server Auth
Administration
Help
Collectors
ALM Server 1
Server Auth
Build Server 1
Access Groups
Build Engineer
Servers Add Server
Help
Filter
Showing 1 - 3 of 3 Display All
Page 1of 1
Build Server 1 Save Copy Delete Reset Job Count Refresh Manifest Test Connection
Details Manifest Test Results Notes (0)
Servers module
Servers detail list
Servers element list
Add new server
(c) Copyright International Business Machines Corporation 2003, 2007. All rights reserved
Trusted sites

Name	Access	Collector	Host	Path
ALM Server 1	Build Engineer	ALM Server 1	localhost	C:\Program Files\Rational\ClearQuest\packages\ALMWork\0.1\pkg_util
Build Server 1	Build Engineer	ALM Server 1	localhost	C:\buildarea
Java Build Server 1	Build Engineer	Java	localhost	C:\buildarea

☐ Disable Name: Build Server 1 Path: C:\buildarea

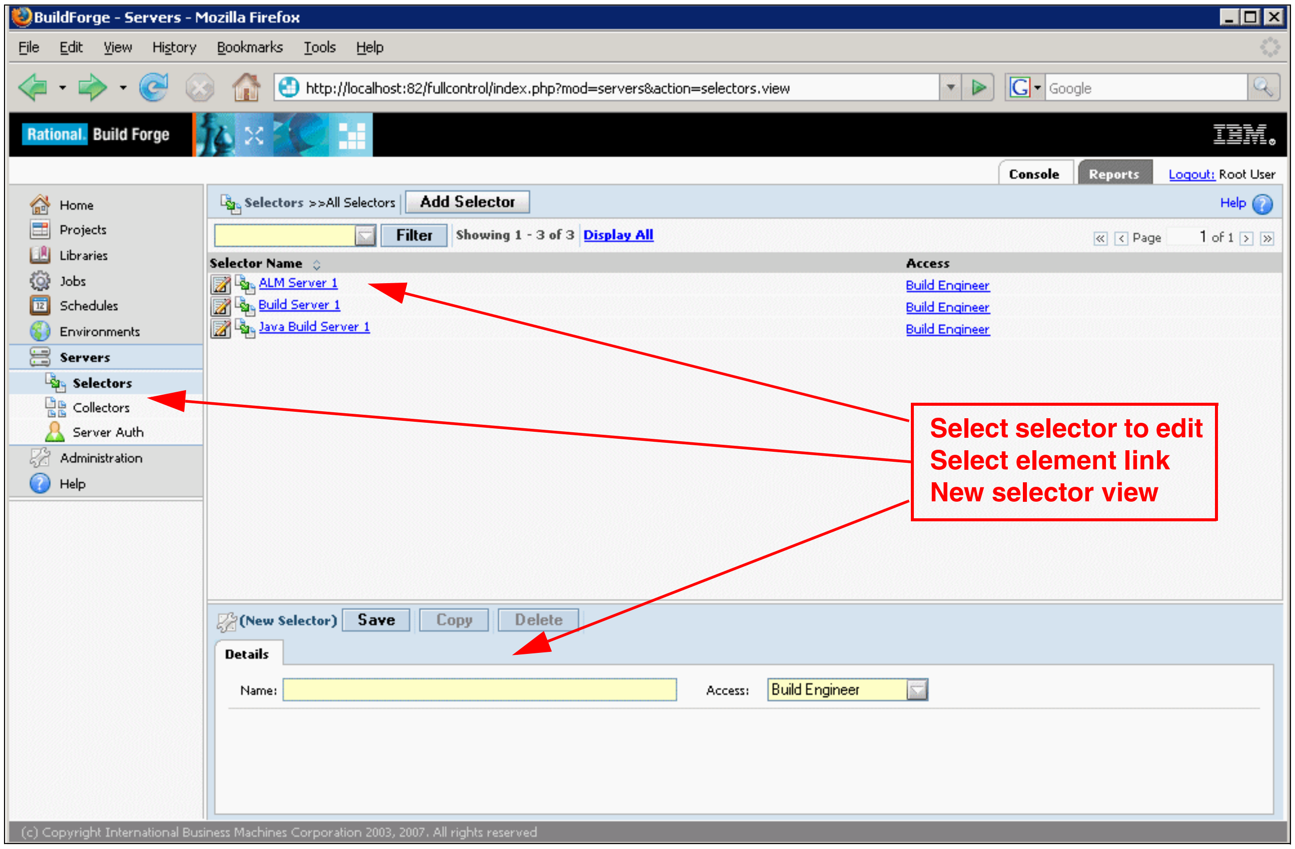
Host:	localhost	Collector:	ALM Server 1	Environment:	-- None --
Authentication:	Build Server 1			Files:	-- None --
Access:	Build Engineer				

Figure 9-10 Servers module

Note: In the reference scenario, Rebecca resolves her build problem by adding another server to her pool. See 9.3.1, “Rebecca inspects the build” on page 358, for instructions on how to add a server.

Selectors

By using server selectors, you can describe the kind of server that a project or step should use by listing desired properties and values. When you apply a selector to a project or step, the system uses the selector to determine which servers are valid choices for the task and then selects an available server from the valid ones. You can also use selectors to be specific, such as choosing a specific server by name. To manage selectors, choose **Servers** → **Selectors** to view the Selectors page (Figure 9-11).



Selector Name	Access
ALM Server 1	Build Engineer
Build Server 1	Build Engineer
Java Build Server 1	Build Engineer

Figure 9-11 Selectors module

A *selector* is a list of properties that describe the desired server. If you want to select servers based on properties that you define, first create the appropriate collector properties. See “Collectors” on page 354 for more information about collectors.

If a selector does not find a server that matches its property list, then the project or step fails and the system creates a build note.

Note: This feature is highlighted in 9.3.1, “Rebecca inspects the build” on page 358. The project fails because the system cannot locate an available server based on its selector.

Collectors

Instead of choosing servers directly, you can set up data to describe the right kind of server for a project or step. You create collectors to attach properties to servers, and those properties are stored as the server's manifest.

A *collector* is an object that defines the set of properties that the system collects from or assigns to a server. The collector assigned to a server is like a blueprint for the server's manifest. The Collectors section of the Servers module, shown in Figure 9-12, lists the available collectors. By using this section, you can create new collectors. A collector consists of a series of properties that are assigned to any server that uses the collector. However, the specific values of the properties can vary from server to server, because a collector is a set of instructions for collecting data.

Server manifests allow the system to choose the right server for a project or step dynamically, based on the current situation.

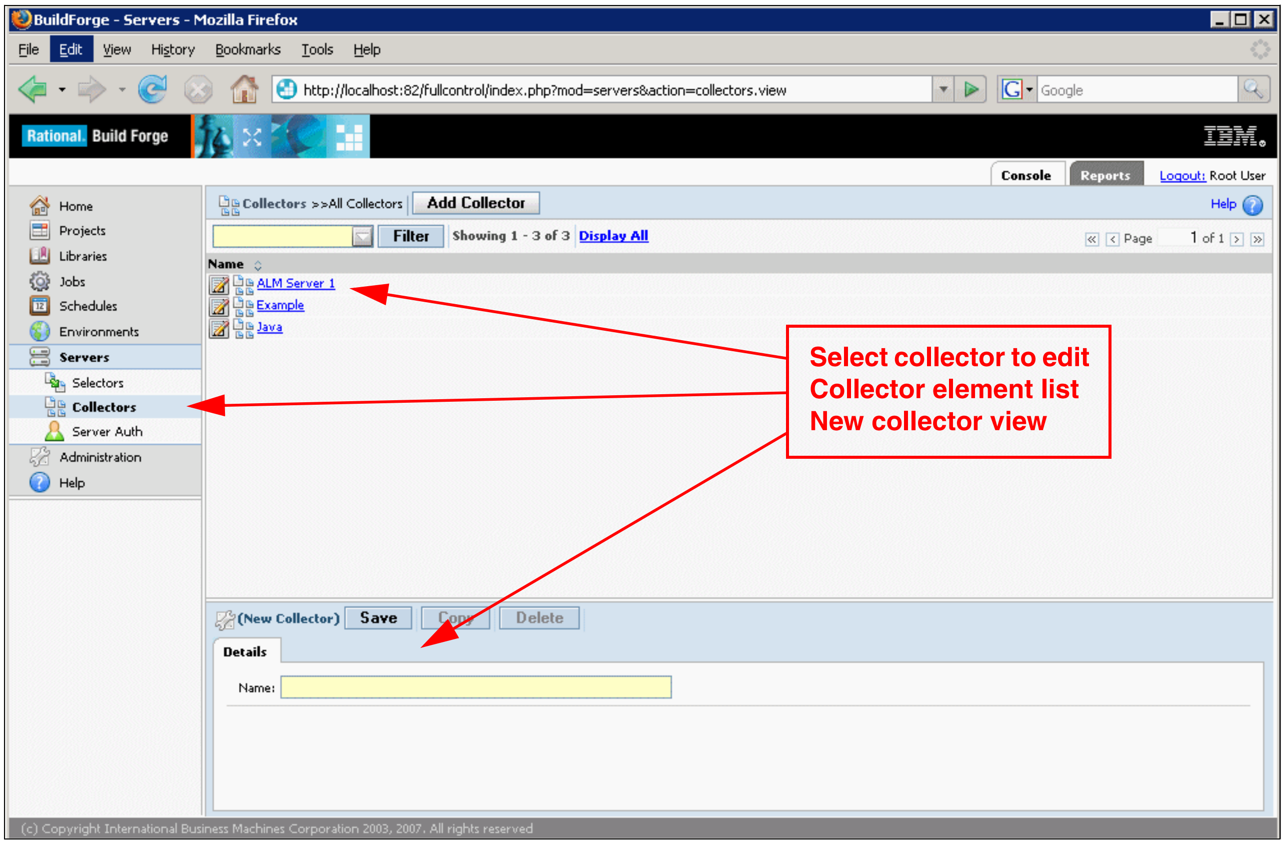


Figure 9-12 Collectors module

Consider this simple example. You create a selector named Mercury, which selects servers whose BF_NAME is equal to Mercury. When you run the project, the system selects the server named Mercury. Because this property is unique, the system always chooses the same server, and if that server is not available, the project fails for lack of an available server.

Here is a better example. You create a collector named Stats that collects RAM, the number of processors, and hard-disk space available. You assign the collector to several servers, one named Mercury (with 512 MB RAM), one named Mars (with 1 GB RAM), and one named Jupiter (3 GB RAM). Then you create two selectors named MuchRam (selects servers with at least 2 GB RAM) and NotMuchRam (selects servers with at least 256 MB RAM). Finally you create two projects, each of which uses one of these selectors.

When you run the projects, the system chooses the server Jupiter for the project that uses the MuchRam selector, because it is the only one that matches. The project that uses the NotMuchRam selector might end up with any of the available servers.

Later you add a server, Neptune (2 GB RAM), to the system. The next time you run a project that uses the MuchRam selector, the system might choose either Neptune or Jupiter. If Jupiter is down for some reason, the system uses Neptune, because it is the only one left that fits the selector.

Manifests

A *manifest* is a list of the properties for a specific server. Where a collector specifies a property, such as memory size or Perl version as something to collect or assign, a manifest stores the result of the collection operation. That is, a manifest for a server is the set of values that the Management Console collects from or assigns to that server and stores as a record in the database. When it chooses a server for a project or a step, the system compares a selector against its set of manifests and chooses a matching server.

You can view the manifests for your servers in the Servers module by selecting the server name and clicking the Manifest tab.

You cannot directly change the manifest for a server. Instead, you must edit the collector assigned to the server or assign a different collector to the server. The collector defines the kinds of properties that the system assigns to a server or attempts to collect from it. The manifest is the resulting set of property values.

Figure 9-13 on page 356 illustrates the relationship between servers, selectors, collectors, and manifests. This structure has crucial importance in the following capabilities of Rational Build Forge:

- Repeatability for consistent usage of systems and tools based on project system and tool requirements
- Abstraction so that projects and steps are not tied to specific physical systems
- Dynamic server management, which allows dynamic additions and removals of physical systems
- Scalability
- Improved resource utilization
- Fault tolerance

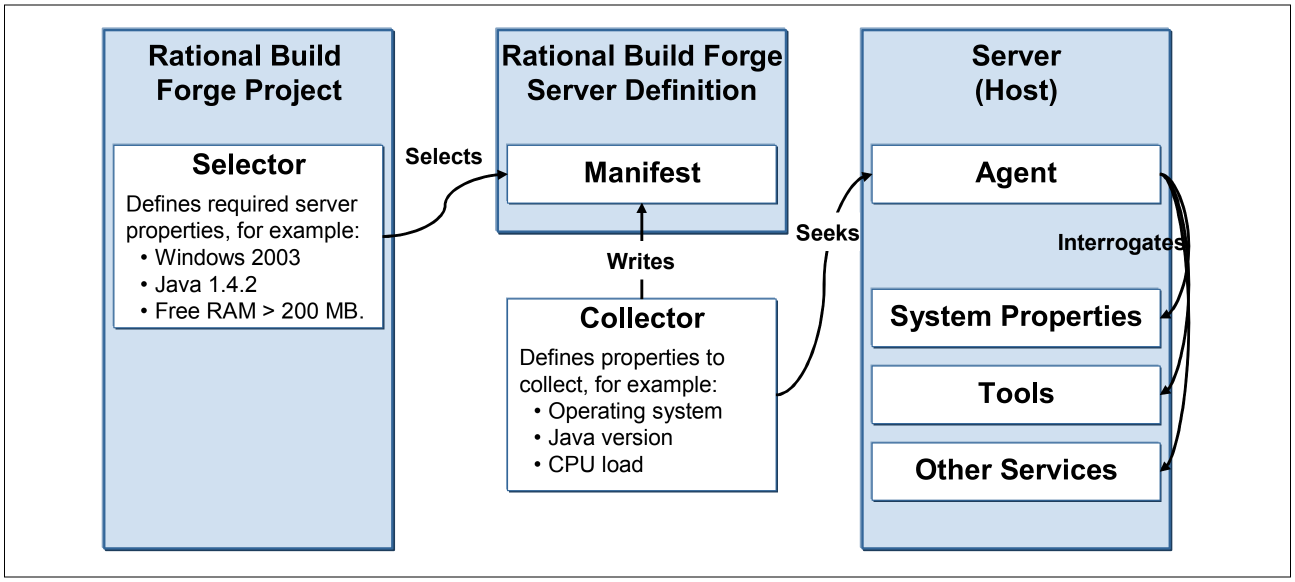


Figure 9-13 Servers, collectors, selectors, and manifests

Server authentication

You use server authentication to associate login credentials to a server. You can use the same credentials for many servers and update the credentials globally, by managing a set of server authentications.

Server authentication stores a login name and password as a single named object that you can associate with one or several servers. Use the Server Authentication page to create and edit server authentications.

You can force the system to use your Management Console login credentials instead of the server authentication that is assigned to the server, by using a special environment variable. To override the normal authentication, add a variable named _USE_BFCREDS, with a value of 1, to an environment used by your project or step. If you add the variable to the project environment, the system uses the override on every step in the project.

When the system attempts to run a step whose environment contains _USE_BFCREDS=1, the system uses the console login credentials of the user who started the project to execute the step's command.

Having separate server authentication is highly recommended to prevent an unprivileged user from performing tasks in Rational Build Forge as a privileged user without needing the credentials.

9.2.7 Libraries

The reference scenario does not use a library, but it is good to know that you can use libraries to modularize common steps into a reusable unit. The Libraries module (Figure 9-34 on page 384) displays library projects. When a project does not have a selector specified, it appears in the Libraries.

9.3 Using Rational Build Forge for an enterprise integration build

In Act 3, Rebecca (the build engineer) is responsible for the solution build of the AccountOpening project. Act 3 builds on the previous act where Diedrie delivered her component changes to the solution build. Rebecca schedules the integration build weekly, she monitors the jobs, and in case of any failure, she also inspects the build runs. After fixing the issue that caused the failure, she runs the integration build project again as shown in Figure 9-14.

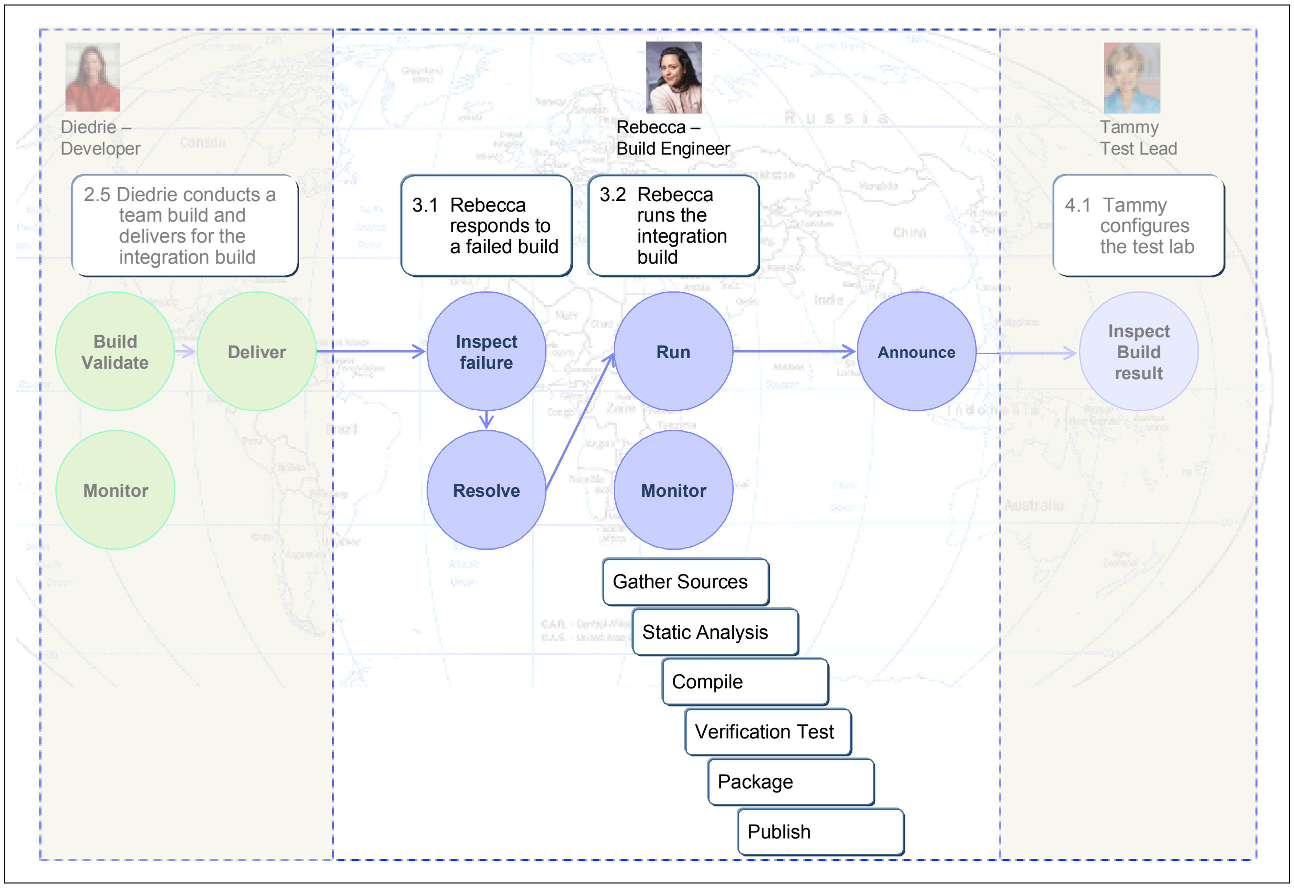


Figure 9-14 Enterprise integration build

The act starts with Rebecca inspecting and resolving a build failure. Then she starts the build and monitors the result. While the build is running, the following important tasks are automated:

- The script gathers the sources and prepares the build environment.
- Static analysis is run against the source code base.
- The solution is compiled.
- Verification tests are run.
- The solution is packaged into a distribution archive.
- The solution is published.

  This task involves staging the distribution archive, archiving and baselining the build project, and creating records in Rational ClearQuest to publish the build result and baseline information.

This act ends with Rebecca notifying the team of the build result.

9.3.1 Rebecca inspects the build

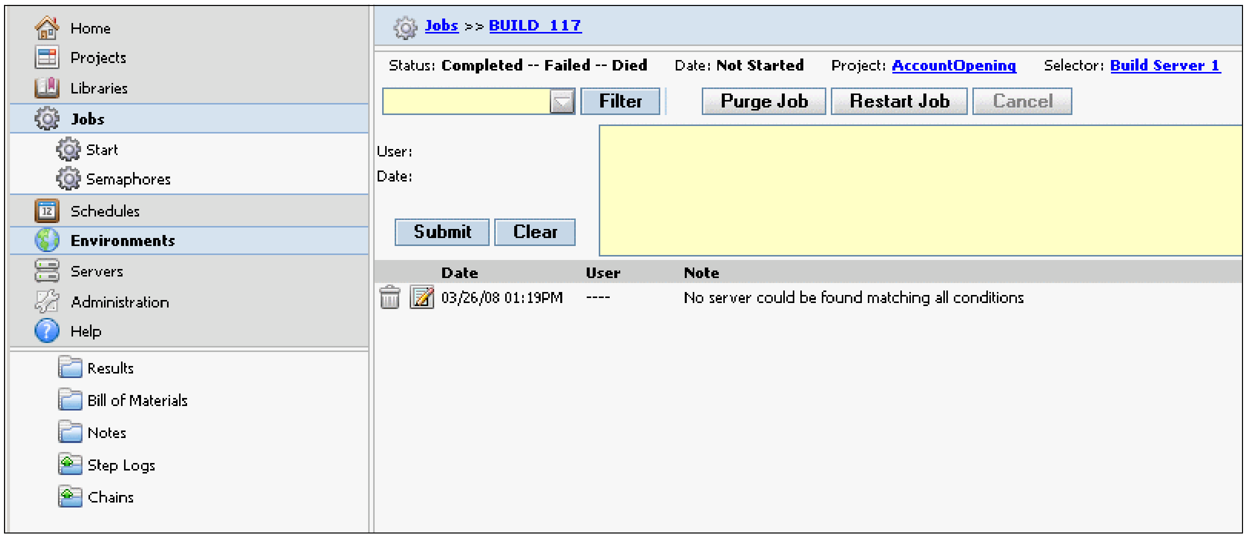
Synopsis: Rebecca has received a notification that the integration build failed. She inspects the build, resolves the problem, and starts the build process again. She monitors the build to ensure that it completes successfully.

Rebecca performs the following steps:

1. Rebecca logs into Rational BuildForge Enterprise Edition.
2. She inspects the build step details by clicking **Jobs** → **BUILD_117** (Figure 9-15).

   The Rational Build Forge dashboard indicates that the build failed in step 1 (Example 9-2 on page 364) with the message "`No server could be found matching all conditions`."

   Rebecca clicks that step to view the execution information. As soon as she reads the step execution information, she realizes that the build servers are busy. The build has a timeout error in selecting a build server according to the rules defined in the Selector, and this job has exceeded the timeout. She has configured the Selector that allows a maximum of two build executions at a time. She has run into this error several times and decides the project must have an additional build server.



Status: **Completed -- Failed -- Died** Date: **Not Started** Project: **AccountOpening** Selector: **Build Server 1**

Date	User	Note
03/26/08 01:19PM	----	No server could be found matching all conditions

Figure 9-15 Jobs → BUILD_117 to view the step execution information

3. Rebecca adds a new build server by defining it in the Build Forge Server Module:
   a. She clicks **Servers** → **Add Server** to open the Server module (Figure 9-10 on page 352).
   b. In the Name field, she types `Build Server 2`.
   c. In the Host field, she types the host information.
   d. She verifies that `C:\buildarea` is in the Path field.
   e. From the Authentication box, she chooses the proper authentication definition.
   f. She can still choose "Build Server 1" Collector as a Collector from the box. Alternatively, she can define a new collector. If she decides to have a separate new collector, she must add a new rule that recognizes the new server to the Selector.

g. She clicks **Selector** → **Add Selector Variable**. Then she chooses **BF_NAME** as the name and **==** as the operator. She also types `Build Server 2` as the value.

h. From the Access box, she selects **Build Engineer**.

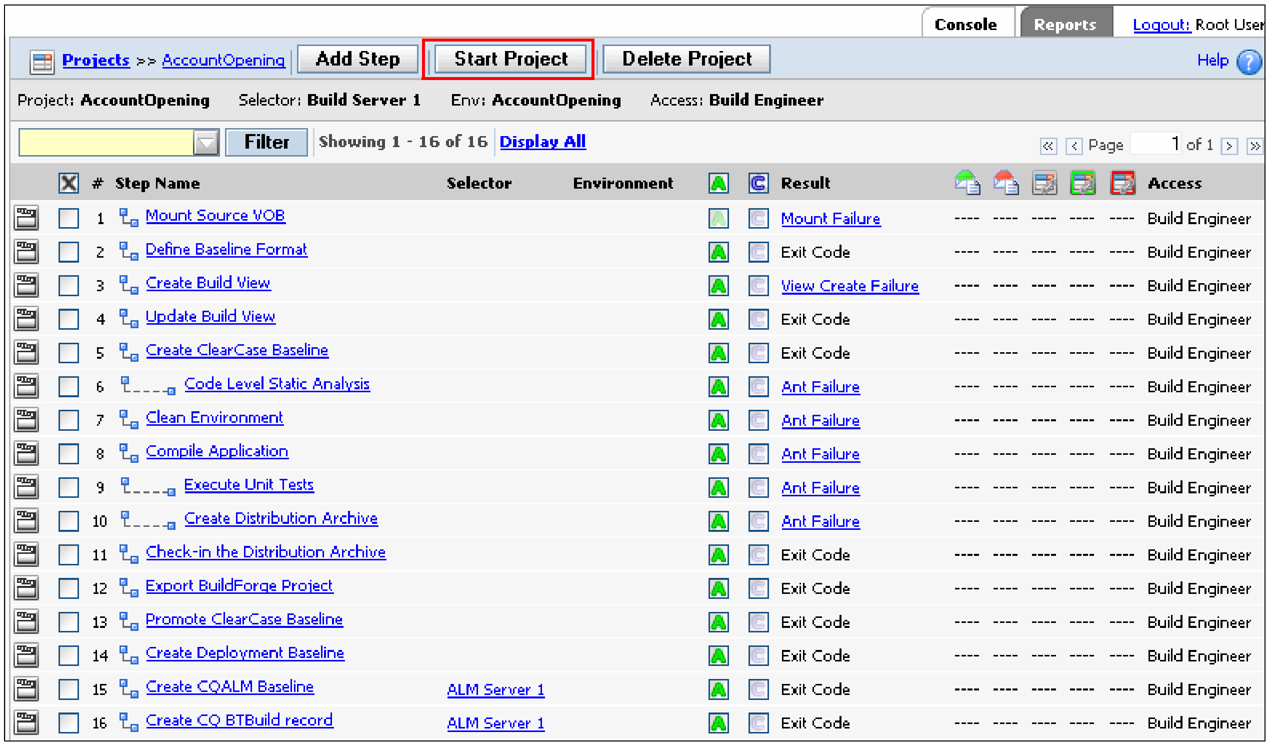
Now Rebecca is ready to run the build project again, which is explained in the following section.

9.3.2 Rebecca runs the integration build

Synopsis: Rebecca opens Rational Build Forge and logs in with her credentials. She opens the Projects module and selects the AccountOpening project.

Rebecca runs the integration build by using the following steps:

1. By using the Build Forge Management Console, she clicks the **Projects** module.
2. She opens the **AccountOpening** project and clicks the **Start Project** button (Figure 9-16).



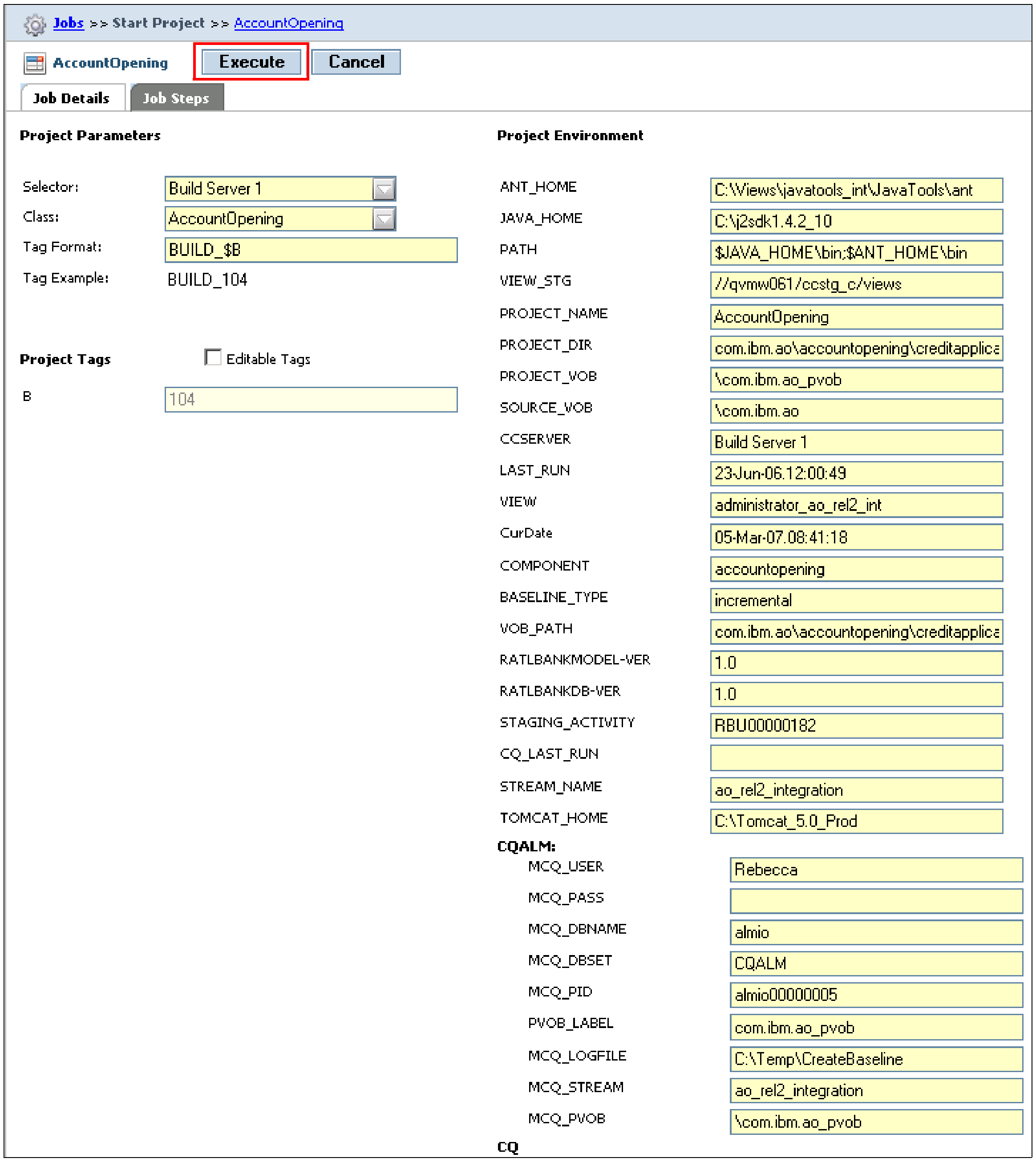
Project: **AccountOpening** Selector: **Build Server 1** Env: **AccountOpening** Access: **Build Engineer**

Showing 1 - 16 of 16

#	Step Name	Selector	Environment	Result						Access
1	Mount Source VOB			Mount Failure	----	----	----	----	----	Build Engineer
2	Define Baseline Format			Exit Code	----	----	----	----	----	Build Engineer
3	Create Build View			View Create Failure	----	----	----	----	----	Build Engineer
4	Update Build View			Exit Code	----	----	----	----	----	Build Engineer
5	Create ClearCase Baseline			Exit Code	----	----	----	----	----	Build Engineer
6	Code Level Static Analysis			Ant Failure	----	----	----	----	----	Build Engineer
7	Clean Environment			Ant Failure	----	----	----	----	----	Build Engineer
8	Compile Application			Ant Failure	----	----	----	----	----	Build Engineer
9	Execute Unit Tests			Ant Failure	----	----	----	----	----	Build Engineer
10	Create Distribution Archive			Ant Failure	----	----	----	----	----	Build Engineer
11	Check-in the Distribution Archive			Exit Code	----	----	----	----	----	Build Engineer
12	Export BuildForge Project			Exit Code	----	----	----	----	----	Build Engineer
13	Promote ClearCase Baseline			Exit Code	----	----	----	----	----	Build Engineer
14	Create Deployment Baseline			Exit Code	----	----	----	----	----	Build Engineer
15	Create CQALM Baseline	ALM Server 1		Exit Code	----	----	----	----	----	Build Engineer
16	Create CQ BTBuild record	ALM Server 1		Exit Code	----	----	----	----	----	Build Engineer

Figure 9-16 AccountOpening project

3. On the next page (Figure 9-17), she ensures that the environment variable values are correct. She clicks the **Execute** button at the top of the page to run the build. The build starts when the state of the execution transitions from Waiting to Running.



Project Parameters

Selector:	Build Server 1
Class:	AccountOpening
Tag Format:	BUILD_$B
Tag Example:	BUILD_104

Project Tags ☐ Editable Tags

B	104

Project Environment

ANT_HOME	C:\Views\javatools_int\JavaTools\ant
JAVA_HOME	C:\j2sdk1.4.2_10
PATH	$JAVA_HOME\bin;$ANT_HOME\bin
VIEW_STG	//qvmw061/ccstg_c/views
PROJECT_NAME	AccountOpening
PROJECT_DIR	com.ibm.ao\accountopening\creditapplica
PROJECT_VOB	\com.ibm.ao_pvob
SOURCE_VOB	\com.ibm.ao
CCSERVER	Build Server 1
LAST_RUN	23-Jun-06.12:00:49
VIEW	administrator_ao_rel2_int
CurDate	05-Mar-07.08:41:18
COMPONENT	accountopening
BASELINE_TYPE	incremental
VOB_PATH	com.ibm.ao\accountopening\creditapplica
RATLBANKMODEL-VER	1.0
RATLBANKDB-VER	1.0
STAGING_ACTIVITY	RBU00000182
CQ_LAST_RUN	
STREAM_NAME	ao_rel2_integration
TOMCAT_HOME	C:\Tomcat_5.0_Prod

CQALM:

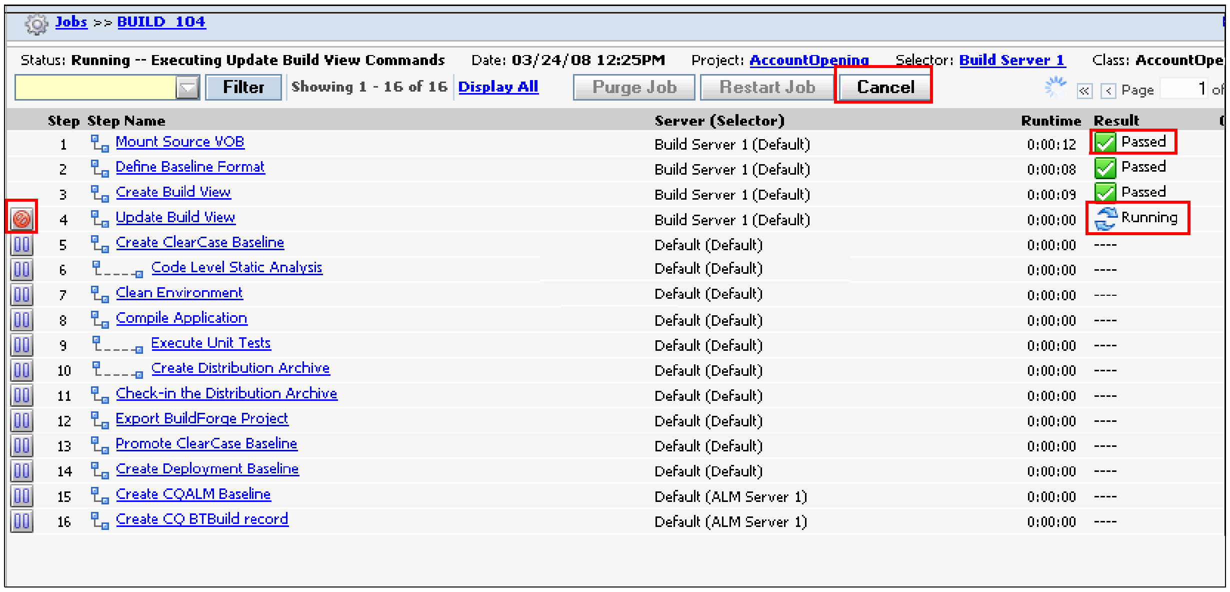
MCQ_USER	Rebecca
MCQ_PASS	
MCQ_DBNAME	almio
MCQ_DBSET	CQALM
MCQ_PID	almio00000005
PVOB_LABEL	com.ibm.ao_pvob
MCQ_LOGFILE	C:\Temp\CreateBaseline
MCQ_STREAM	ao_rel2_integration
MCQ_PVOB	\com.ibm.ao_pvob

CQ

Figure 9-17 AccountOpening project run details

As each step executes, the result column indicates the Pass/Fail status of the step as shown in Figure 9-18.

4. During the monitoring of execution, Rebecca can click any step name to drill down into any of the steps to display more detail about what Rational Build Forge is doing. If she thinks something is wrong, she can cancel the step running by clicking the **Cancel** icon.



Jobs >> BUILD_104

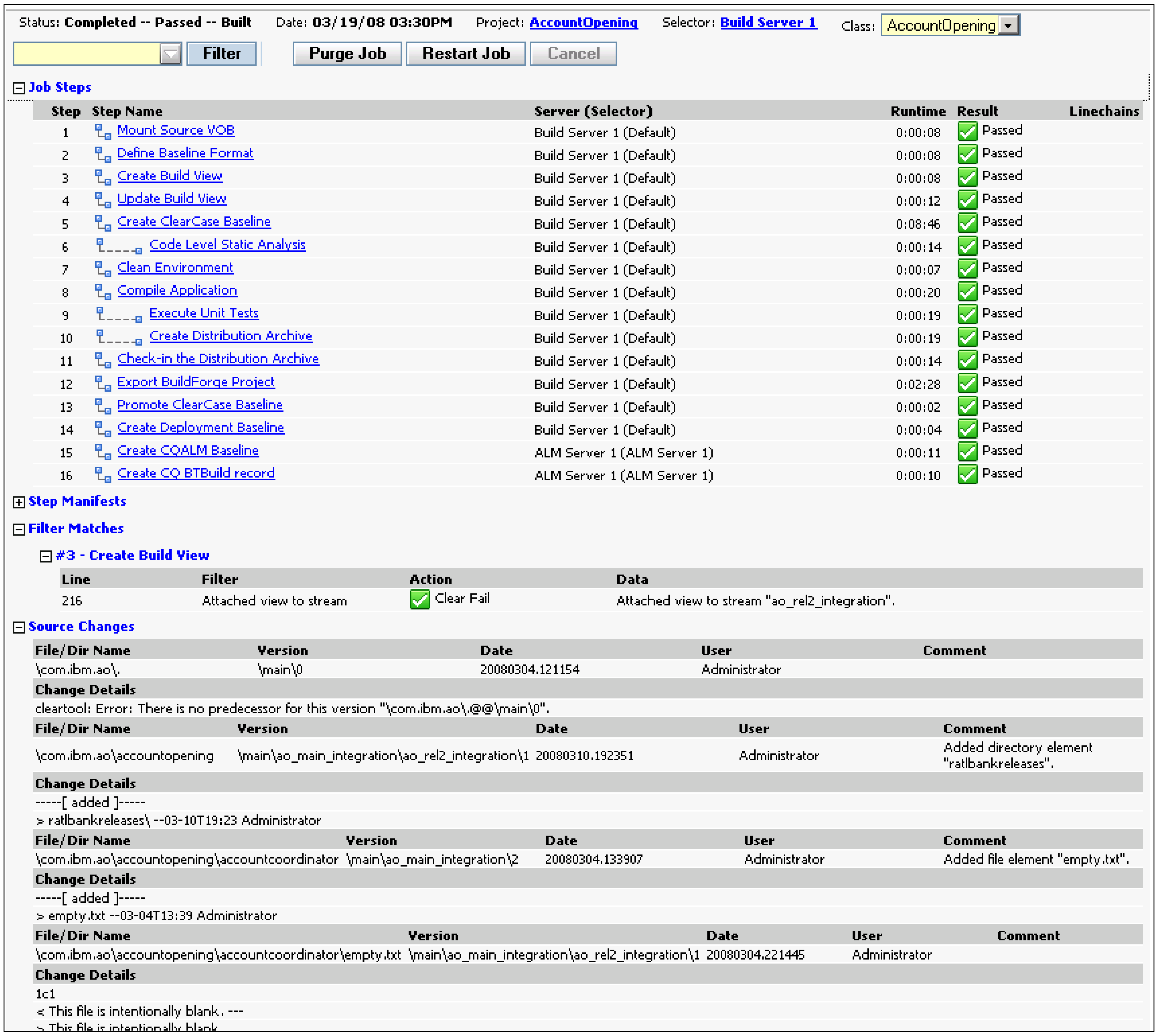
Status: **Running -- Executing Update Build View Commands** Date: **03/24/08 12:25PM** Project: **AccountOpening** Selector: **Build Server 1** Class: **AccountOpe**

Filter | Showing 1 - 16 of 16 | Display All | Purge Job | Restart Job | Cancel | Page 1 of

Step	Step Name	Server (Selector)	Runtime	Result
1	Mount Source VOB	Build Server 1 (Default)	0:00:12	✓ Passed
2	Define Baseline Format	Build Server 1 (Default)	0:00:08	✓ Passed
3	Create Build View	Build Server 1 (Default)	0:00:09	✓ Passed
4	Update Build View	Build Server 1 (Default)	0:00:00	Running
5	Create ClearCase Baseline	Default (Default)	0:00:00	----
6	Code Level Static Analysis	Default (Default)	0:00:00	----
7	Clean Environment	Default (Default)	0:00:00	----
8	Compile Application	Default (Default)	0:00:00	----
9	Execute Unit Tests	Default (Default)	0:00:00	----
10	Create Distribution Archive	Default (Default)	0:00:00	----
11	Check-in the Distribution Archive	Default (Default)	0:00:00	----
12	Export BuildForge Project	Default (Default)	0:00:00	----
13	Promote ClearCase Baseline	Default (Default)	0:00:00	----
14	Create Deployment Baseline	Default (Default)	0:00:00	----
15	Create CQALM Baseline	Default (ALM Server 1)	0:00:00	----
16	Create CQ BTBuild record	Default (ALM Server 1)	0:00:00	----

Figure 9-18 Job started

5. Because the build succeeds this time, Rebecca examines the bill of materials (Figure 9-19). She expands **Source Changes** to display the version or versions of Rational ClearCase source controlled elements that were included in the build. The bill of materials includes the differences between the version that is used in the build and its previous version.


Status: **Completed -- Passed -- Built** Date: **03/19/08 03:30PM** Project: **AccountOpening** Selector: **Build Server 1** Class: AccountOpening

Job Steps

Step	Step Name	Server (Selector)	Runtime	Result	Linechains
1	Mount Source VOB	Build Server 1 (Default)	0:00:08	✓ Passed	
2	Define Baseline Format	Build Server 1 (Default)	0:00:08	✓ Passed	
3	Create Build View	Build Server 1 (Default)	0:00:08	✓ Passed	
4	Update Build View	Build Server 1 (Default)	0:00:12	✓ Passed	
5	Create ClearCase Baseline	Build Server 1 (Default)	0:08:46	✓ Passed	
6	Code Level Static Analysis	Build Server 1 (Default)	0:00:14	✓ Passed	
7	Clean Environment	Build Server 1 (Default)	0:00:07	✓ Passed	
8	Compile Application	Build Server 1 (Default)	0:00:20	✓ Passed	
9	Execute Unit Tests	Build Server 1 (Default)	0:00:19	✓ Passed	
10	Create Distribution Archive	Build Server 1 (Default)	0:00:19	✓ Passed	
11	Check-in the Distribution Archive	Build Server 1 (Default)	0:00:14	✓ Passed	
12	Export BuildForge Project	Build Server 1 (Default)	0:02:28	✓ Passed	
13	Promote ClearCase Baseline	Build Server 1 (Default)	0:00:02	✓ Passed	
14	Create Deployment Baseline	Build Server 1 (Default)	0:00:04	✓ Passed	
15	Create CQALM Baseline	ALM Server 1 (ALM Server 1)	0:00:11	✓ Passed	
16	Create CQ BTBuild record	ALM Server 1 (ALM Server 1)	0:00:10	✓ Passed	

Step Manifests

Filter Matches

#3 - Create Build View

Line	Filter	Action	Data
216	Attached view to stream	✓ Clear Fail	Attached view to stream "ao_rel2_integration".

Source Changes

File/Dir Name	Version	Date	User	Comment
\com.ibm.ao\.	\main\0	20080304.121154	Administrator	

Change Details

cleartool: Error: There is no predecessor for this version "\com.ibm.ao\.@@\main\0".

File/Dir Name	Version	Date	User	Comment
\com.ibm.ao\accountopening	\main\ao_main_integration\ao_rel2_integration\1	20080310.192351	Administrator	Added directory element "ratlbankreleases".

Change Details

-----[added]-----
> ratlbankreleases\ --03-10T19:23 Administrator

File/Dir Name	Version	Date	User	Comment
\com.ibm.ao\accountopening\accountcoordinator	\main\ao_main_integration\2	20080304.133907	Administrator	Added file element "empty.txt".

Change Details

-----[added]-----
> empty.txt --03-04T13:39 Administrator

File/Dir Name	Version	Date	User	Comment
\com.ibm.ao\accountopening\accountcoordinator\empty.txt	\main\ao_main_integration\ao_rel2_integration\1	20080304.221445	Administrator	

Change Details

1c1
< This file is intentionally blank. ---
> This file is intentionally blank

Figure 9-19 Bill of materials

Rational Build Forge generates a bill of materials after each project run. The bill of materials contains information about the steps in the run and the changes to files that resulted from it. The bill of materials can be provided to consumers of the project run, such as the quality assurance department, for help in understanding the contents of a new build. It can also serve as an audit solution for your build and release process.

With the bill of materials, you can easily include other relevant information as in the following examples:

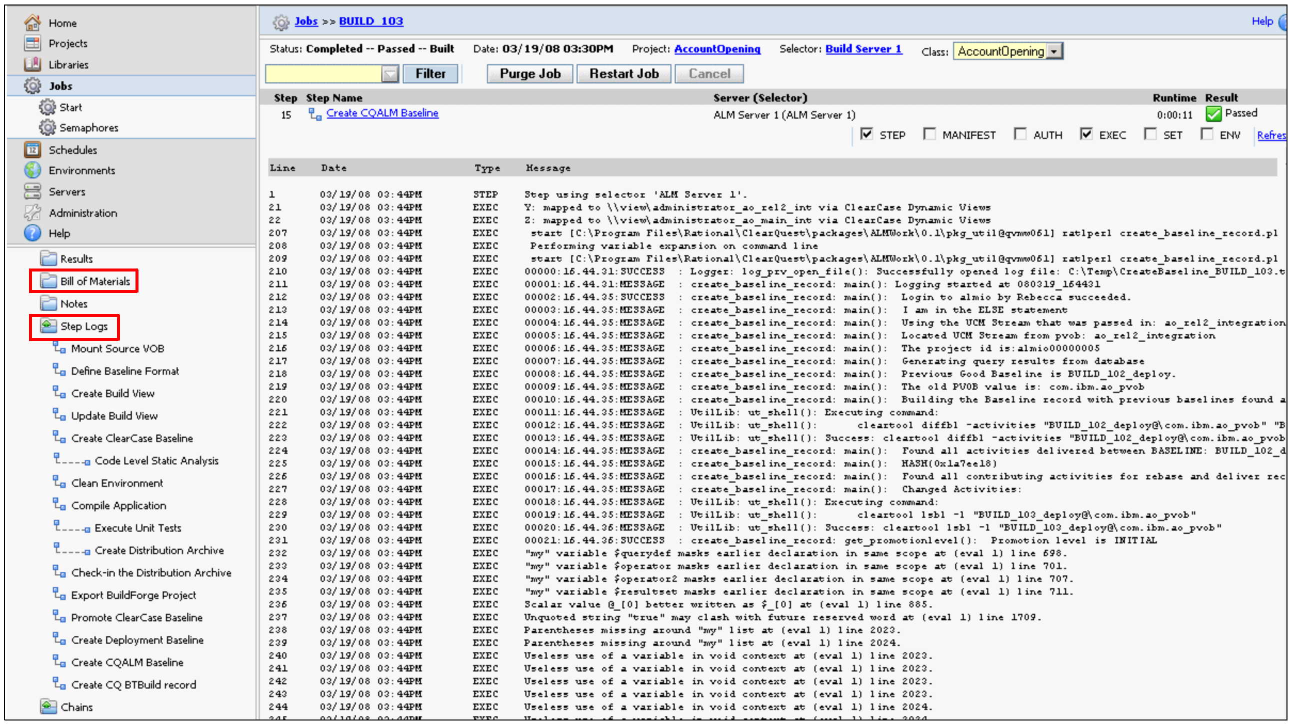
- ► A complete list of all files in the baseline used for this build
- ► The differences between this build and the last build (date/date, time/time, and label/label)
- ► Modified views that the process execution is based upon
- ► A list of all defects that were included in the process run

These extensive tracking mechanisms make the Rational Build Forge system a valuable tool for documenting processes, providing an audit trail to assist with regulatory compliance requirements. With the bill of materials, you achieve the following benefits:

- ► Reduced time spent reading log data, with a lesser chance of errors being undetected, which increases quality
- ► Quick diagnosis of build output and location of items of interest
- ► Great reduction in time spent analyzing builds and troubleshooting errors
- ► Faster build turnarounds, greater productivity, and faster time to market
- ► Better support to monitor systems and aid in compliance

With the bill of materials, you get complete documentation of a build's contents. It can include the build results, notes, environments, lists of build files, and code changes. This information can be used to compare and summarize the state of builds across the enterprise. The system generates a bill of materials for each build automatically.

In Rebecca's build, regardless of the result, success or failure, she has a related e-mail notification that informs her of the build result in accordance with our scenario. If any failure is indicated, she can either see the details in the bill of materials or click Step Logs (Figure 9-20).



Status: Completed -- Passed -- Built   Date: 03/19/08 03:30PM   Project: AccountOpening   Selector: Build Server 1   Class: AccountOpening

Step	Step Name	Server (Selector)	Runtime	Result
15	Create CQALM Baseline	ALM Server 1 (ALM Server 1)	0:00:11	✓ Passed

[✓ STEP] MANIFEST AUTH [✓ EXEC] SET ENV

```
Line    Date              Type    Message

1       03/19/08 03:44PM  STEP    Step using selector 'ALM Server 1'.
21      03/19/08 03:44PM  EXEC    Y: mapped to \\view\administrator_ao_rel2_int via ClearCase Dynamic Views
22      03/19/08 03:44PM  EXEC    Z: mapped to \\view\administrator_ao_main_int via ClearCase Dynamic Views
207     03/19/08 03:44PM  EXEC     start [C:\Program Files\Rational\ClearQuest\packages\ALMWork\0.1\pkg_util@qvmw061] ratlperl create_baseline_record.pl
208     03/19/08 03:44PM  EXEC     Performing variable expansion on command line
209     03/19/08 03:44PM  EXEC     start [C:\Program Files\Rational\ClearQuest\packages\ALMWork\0.1\pkg_util@qvmw061] ratlperl create_baseline_record.pl
210     03/19/08 03:44PM  EXEC    00000:16.44.31:SUCCESS  : Logger: log_prv_open_file(): Successfully opened log file: C:\Temp\CreateBaseline_BUILD_103.t
211     03/19/08 03:44PM  EXEC    00001:16.44.31:MESSAGE  : create_baseline_record: main(): Logging started at 080319_164431
212     03/19/08 03:44PM  EXEC    00002:16.44.35:SUCCESS  : create_baseline_record: main():  Login to almio by Rebecca succeeded.
213     03/19/08 03:44PM  EXEC    00003:16.44.35:MESSAGE  : create_baseline_record: main():  I am in the ELSE statement
214     03/19/08 03:44PM  EXEC    00004:16.44.35:MESSAGE  : create_baseline_record: main():  Using the UCM Stream that was passed in: ao_rel2_integration
215     03/19/08 03:44PM  EXEC    00005:16.44.35:MESSAGE  : create_baseline_record: main():  Located UCM Stream from pvob: ao_rel2_integration
216     03/19/08 03:44PM  EXEC    00006:16.44.35:MESSAGE  : create_baseline_record: main():  The project id is:almio00000005
217     03/19/08 03:44PM  EXEC    00007:16.44.35:MESSAGE  : create_baseline_record: main():  Generating query results from database
218     03/19/08 03:44PM  EXEC    00008:16.44.35:MESSAGE  : create_baseline_record: main():  Previous Good Baseline is BUILD_102_deploy.
219     03/19/08 03:44PM  EXEC    00009:16.44.35:MESSAGE  : create_baseline_record: main():  The old PVOB value is: com.ibm.ao_pvob
220     03/19/08 03:44PM  EXEC    00010:16.44.35:MESSAGE  : create_baseline_record: main():  Building the Baseline record with previous baselines found a
221     03/19/08 03:44PM  EXEC    00011:16.44.35:MESSAGE  : UtilLib: ut_shell(): Executing command:
222     03/19/08 03:44PM  EXEC    00012:16.44.35:MESSAGE  : UtilLib: ut_shell():      cleartool diffbl -activities "BUILD_102_deploy@\com.ibm.ao_pvob" "B
223     03/19/08 03:44PM  EXEC    00013:16.44.35:MESSAGE  : UtilLib: ut_shell(): Success: cleartool diffbl -activities "BUILD_102_deploy@\com.ibm.ao_pvob
224     03/19/08 03:44PM  EXEC    00014:16.44.35:MESSAGE  : create_baseline_record: main():  Found all activities delivered between BASELINE: BUILD_102_d
225     03/19/08 03:44PM  EXEC    00015:16.44.35:MESSAGE  : create_baseline_record: main():  HASH(0x1a7ee18)
226     03/19/08 03:44PM  EXEC    00016:16.44.35:MESSAGE  : create_baseline_record: main():  Found all contributing activities for rebase and deliver rec
227     03/19/08 03:44PM  EXEC    00017:16.44.35:MESSAGE  : create_baseline_record: main():  Changed Activities:
228     03/19/08 03:44PM  EXEC    00018:16.44.35:MESSAGE  : UtilLib: ut_shell(): Executing command:
229     03/19/08 03:44PM  EXEC    00019:16.44.35:MESSAGE  : UtilLib: ut_shell():      cleartool lsbl -l "BUILD_103_deploy@\com.ibm.ao_pvob"
230     03/19/08 03:44PM  EXEC    00020:16.44.36:MESSAGE  : UtilLib: ut_shell(): Success: cleartool lsbl -l "BUILD_103_deploy@\com.ibm.ao_pvob"
231     03/19/08 03:44PM  EXEC    00021:16.44.36:SUCCESS  : create_baseline_record: get_promotionlevel():  Promotion level is INITIAL
232     03/19/08 03:44PM  EXEC    "my" variable $querydef masks earlier declaration in same scope at (eval 1) line 698.
233     03/19/08 03:44PM  EXEC    "my" variable $operator masks earlier declaration in same scope at (eval 1) line 701.
234     03/19/08 03:44PM  EXEC    "my" variable $operator2 masks earlier declaration in same scope at (eval 1) line 707.
235     03/19/08 03:44PM  EXEC    "my" variable $resultset masks earlier declaration in same scope at (eval 1) line 711.
236     03/19/08 03:44PM  EXEC    Scalar value @_[0] better written as $_[0] at (eval 1) line 885.
237     03/19/08 03:44PM  EXEC    Unquoted string "true" may clash with future reserved word at (eval 1) line 1709.
238     03/19/08 03:44PM  EXEC    Parentheses missing around "my" list at (eval 1) line 2023.
239     03/19/08 03:44PM  EXEC    Parentheses missing around "my" list at (eval 1) line 2024.
240     03/19/08 03:44PM  EXEC    Useless use of a variable in void context at (eval 1) line 2023.
241     03/19/08 03:44PM  EXEC    Useless use of a variable in void context at (eval 1) line 2023.
242     03/19/08 03:44PM  EXEC    Useless use of a variable in void context at (eval 1) line 2023.
243     03/19/08 03:44PM  EXEC    Useless use of a variable in void context at (eval 1) line 2023.
244     03/19/08 03:44PM  EXEC    Useless use of a variable in void context at (eval 1) line 2024.
```

Figure 9-20 Step logs

How Rebecca defined the project steps

Rebecca's project is quite sophisticated. To understand what is happening in the integration build project in Rational Build Forge, we look at each step in the AccountOpening project:

1. Mount the source versioned object base (VOB).

   Rational ClearCase is our source control system, and the VOB server should be mounted in this step as shown in Example 9-2. The sources for the build are obtained from the source control system.

   The VOB might already be mounted. Therefore, our build project might fail in this step. To prevent this, Rebecca created a filter pattern as "already mounted" that clears the fail in that condition. For similar situations, use filter patterns as a best practice.

   Example 9-2 Mounting the source VOB

   ```
   cleartool mount ${SOURCE_VOB}
   ```

2. Define the baseline format.

   As we discussed earlier, naming conventions are important for a continuous build integration. The BASELINE name is defined as the component name and Rational Build Forge tag, which increases by 1 for each run. Naming conventions are crucial for reuse and continuous build integration at all levels of software development (such as versioning, component, streams, and so on) in addition to baselines as shown in Example 9-3.

   Example 9-3 Defining the baseline format

   ```
   .bset env "BASELINE=${COMPONENT}_$BF_TAG"
   ```

3. Create a build view.

   It is a best practice to create and use a view before you produce a build or release as shown in Example 9-4.

   Example 9-4 Creating a build view

   ```
   cleartool mkview -snapshot -tag "${PROJECT_NAME}_$BF_TAG" -tcomment
   "${PROJECT_NAME} Build" -stream ${STREAM_NAME}@$PROJECT_VOB -vws
   $VIEW_STG\${PROJECT_NAME}_$BF_TAG.vws $BF_SERVER_ROOT\${PROJECT_NAME}_$BF_TAG
   ```

4. Update the build view.

   In Example 9-5, Rational Build Forge loads the files from the VOB server locally to the view target directory that was created in a previous step.

   Example 9-5 Updating the build view

   ```
   cleartool update -add_loadrules com.ibm.ao\accountopening\creditapplication
   cleartool update -add_loadrules com.ibm.ao\accountopening\ratlbankreleases
   ```

5. Create the Rational ClearCase baseline.

   We create the baseline as shown in Example 9-6 on page 365 by means of ClearCase adapter. See 9.6.2, "How Rational Build Forge is configured for this scenario" on page 380. A ClearCase adapter fails if no changes have been made. It is similar to a linked adapter for continuous integration. Basically, if a new baseline cannot be created (because no additional changes have been made since the last source baseline), then the step fails and the project stops.

Example 9-6 Creating the Rational ClearCase baseline

```
.source "CC Interface" ByBaseline
```

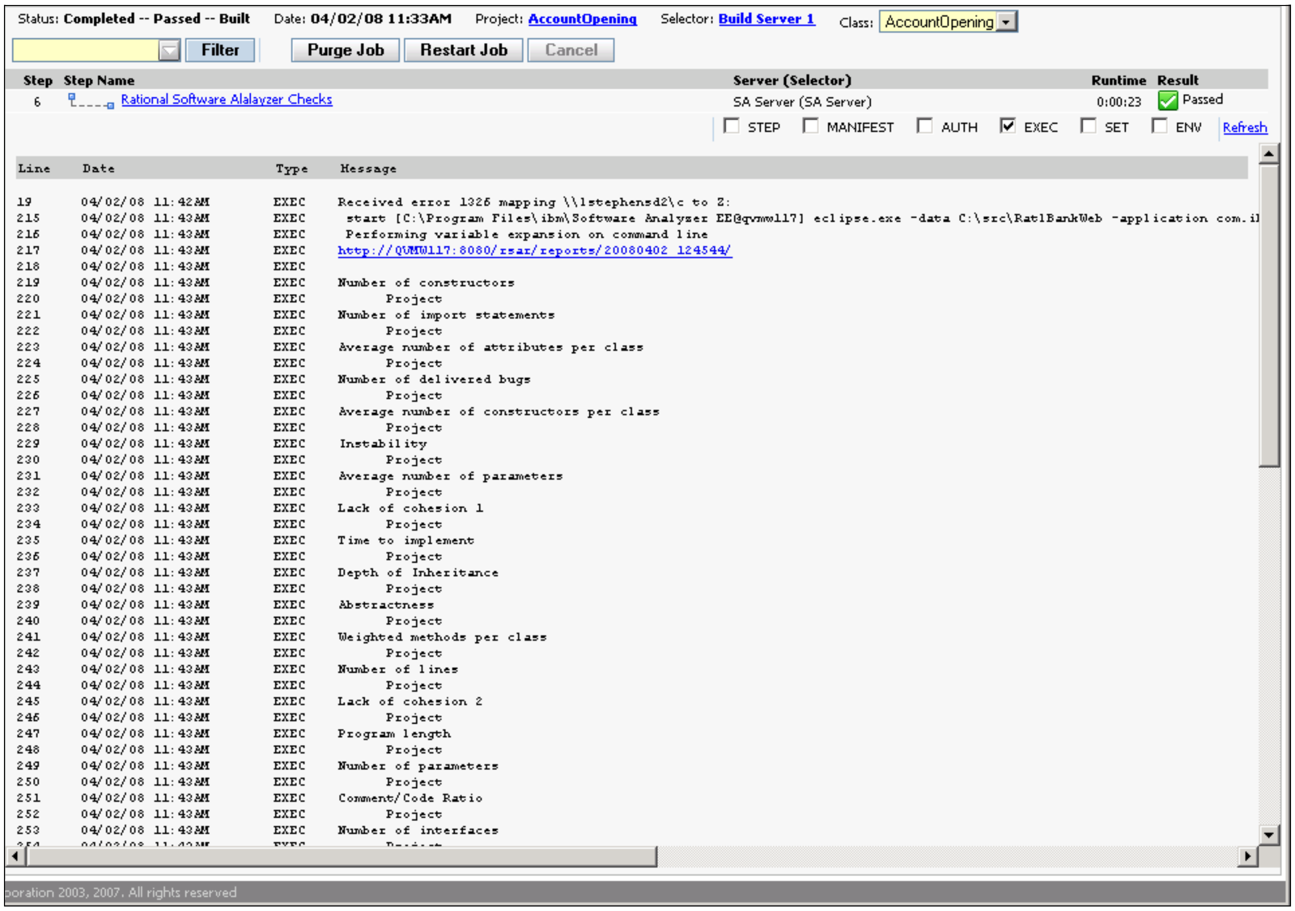
6. Perform Rational Software Analyzer checks.

   For reliability and assurance of quality, static analysis must be applied to all software artifacts. In Example 9-7, Rebecca uses Rational Software Analyzer to implement the code-level static analysis, as discussed in “Code review and analysis” on page 330. Figure 9-21 shows the results of using Rational Software Analyzer.

> **Note:** Rational Software Analyzer results include a URL link to the report server of Rational Software Analyzer (Figure 9-21).

Example 9-7 Rational Software Analyzer checks

```
eclipse.exe -data C:\src\RatlBankWeb -application
com.ibm.rsaz.analysis.commandline.AnalyzeApplication  -rulefile
c:\src\rules.dat  -exportdirectory c:\results -verbose
```



Step	Step Name	Server (Selector)	Runtime	Result
6	Rational Software Alalayzer Checks	SA Server (SA Server)	0:00:23	✓ Passed

```
Line    Date                  Type    Message

19      04/02/08 11:42AM      EXEC    Received error 1326 mapping \\lstephensd2\c to Z:
215     04/02/08 11:43AM      EXEC     start [C:\Program Files\ibm\Software Analyzer EE@qvmwl17] eclipse.exe -data C:\src\RatlBankWeb -application com.il
216     04/02/08 11:43AM      EXEC     Performing variable expansion on command line
217     04/02/08 11:43AM      EXEC    http://QVMW117:8080/rsar/reports/20080402_124544/
218     04/02/08 11:43AM      EXEC
219     04/02/08 11:43AM      EXEC    Number of constructors
220     04/02/08 11:43AM      EXEC          Project
221     04/02/08 11:43AM      EXEC    Number of import statements
222     04/02/08 11:43AM      EXEC          Project
223     04/02/08 11:43AM      EXEC    Average number of attributes per class
224     04/02/08 11:43AM      EXEC          Project
225     04/02/08 11:43AM      EXEC    Number of delivered bugs
226     04/02/08 11:43AM      EXEC          Project
227     04/02/08 11:43AM      EXEC    Average number of constructors per class
228     04/02/08 11:43AM      EXEC          Project
229     04/02/08 11:43AM      EXEC    Instability
230     04/02/08 11:43AM      EXEC          Project
231     04/02/08 11:43AM      EXEC    Average number of parameters
232     04/02/08 11:43AM      EXEC          Project
233     04/02/08 11:43AM      EXEC    Lack of cohesion 1
234     04/02/08 11:43AM      EXEC          Project
235     04/02/08 11:43AM      EXEC    Time to implement
236     04/02/08 11:43AM      EXEC          Project
237     04/02/08 11:43AM      EXEC    Depth of Inheritance
238     04/02/08 11:43AM      EXEC          Project
239     04/02/08 11:43AM      EXEC    Abstractness
240     04/02/08 11:43AM      EXEC          Project
241     04/02/08 11:43AM      EXEC    Weighted methods per class
242     04/02/08 11:43AM      EXEC          Project
243     04/02/08 11:43AM      EXEC    Number of lines
244     04/02/08 11:43AM      EXEC          Project
245     04/02/08 11:43AM      EXEC    Lack of cohesion 2
246     04/02/08 11:43AM      EXEC          Project
247     04/02/08 11:43AM      EXEC    Program length
248     04/02/08 11:43AM      EXEC          Project
249     04/02/08 11:43AM      EXEC    Number of parameters
250     04/02/08 11:43AM      EXEC          Project
251     04/02/08 11:43AM      EXEC    Comment/Code Ratio
252     04/02/08 11:43AM      EXEC          Project
253     04/02/08 11:43AM      EXEC    Number of interfaces
```

poration 2003, 2007. All rights reserved

Figure 9-21 Rational Software Analyzer execution results

7. Perform code-level static analysis.

   As shown in Example 9-8, Rebecca verifies the quality with **ant** static analysis.

   Example 9-8 Code-level static analysis

   ```
   ant checkstyle
   ```

8. Ensure a Clean Environment section.

   It is best practice to have a Clean Environment section in build or release projects to avoid interference with other runs and builds as shown in Example 9-9. A build or release should be performed in a clean and controlled environment. The clean environment ensures that a base can be reused for all builds and releases. You might prefer to have a Clean Environment section just before getting the source files, but it is really a matter of approach.

   Example 9-9 Clean Environment

   ```
   ant clean
   ```

9. Compile the application.

   Example 9-10 shows how we compile the code.

   Example 9-10 Compiling the application

   ```
   ant compile
   ```

10. Execute unit tests.

    To ensure the quality of the build, automate the test steps or a section in the build or release projects as a best practice. In Example 9-11, the code is tested against JUnit scripts after the compile.

    Example 9-11 Executing unit tests

    ```
    ant junit-all
    ```

11. Create a distribution archive.

    When the build artifacts are produced, they must be placed in a target directory for distribution and future reuse. In Example 9-12, we create a binary distribution of the build to the target directory.

    Example 9-12 Creating a distribution archive

    ```
    ant dist
    ```

12. Check in the distribution archive.

    The distribution is packaged and checked in Rational ClearCase in Example 9-13 from the distribution directory.

    Example 9-13 Check-in of the distribution archive

    ```
    cleartool setact $STAGING_ACTIVITY
    clearfsimport RatlBankWeb.war
    ..\..\..\..\${PROJECT_NAME}_$BF_TAG\RatlBankReleases\web\dist\
    cleartool co -nc
    ..\..\..\..\${PROJECT_NAME}_$BF_TAG\RatlBankReleases\web\build.xml
    copy ..\build.xml ..\..\..\..\${PROJECT_NAME}_$BF_TAG\RatlBankReleases\web\.
    ```

```
cleartool ci -nc -identical
..\..\..\..\${PROJECT_NAME}_$BF_TAG\RatlBankReleases\web\build.xml
cleartool co -nc
..\..\..\..\${PROJECT_NAME}_$BF_TAG\RatlBankReleases\web\default.properties
copy ..\default.properties
..\..\..\..\${PROJECT_NAME}_$BF_TAG\RatlBankReleases\web\.
cleartool ci -nc -identical
..\..\..\..\${PROJECT_NAME}_$BF_TAG\RatlBankReleases\web\default.properties
cleartool setact -none
```

13. Export the Rational Build Forge project.

    Even though the Rational Build Forge project is in Rational ClearCase source control, as a best practice, the project that we run should be exported to the target project directory, in case it is needed for audit or other reasons as shown in Example 9-14. Otherwise Rebecca must trace the version of the build or release project for each run by time stamp.

    Example 9-14 Exporting the Build Forge project

    ```
    .export ${PROJECT_NAME}_$BF_TAG/$PROJECT_DIR/dist/project.xml
    ```

14. Promote the Rational ClearCase baseline.

    Add a label to the delivery baseline in order to reflect a higher degree of stability. In Example 9-15, the promotion level is set with the baseline name and the project VOB name.

    Example 9-15 Promoting the Rational ClearCase baseline

    ```
    cleartool chbl -level BUILT ${BASELINE}@$PROJECT_VOB
    ```

15. Create the deployment baseline.

    The baseline is created in Rational ClearCase as shown in Example 9-16.

    Example 9-16 Creating the deployment baseline

    ```
    cleartool mkbl -nc -identical ${BF_TAG}_deploy
    ```

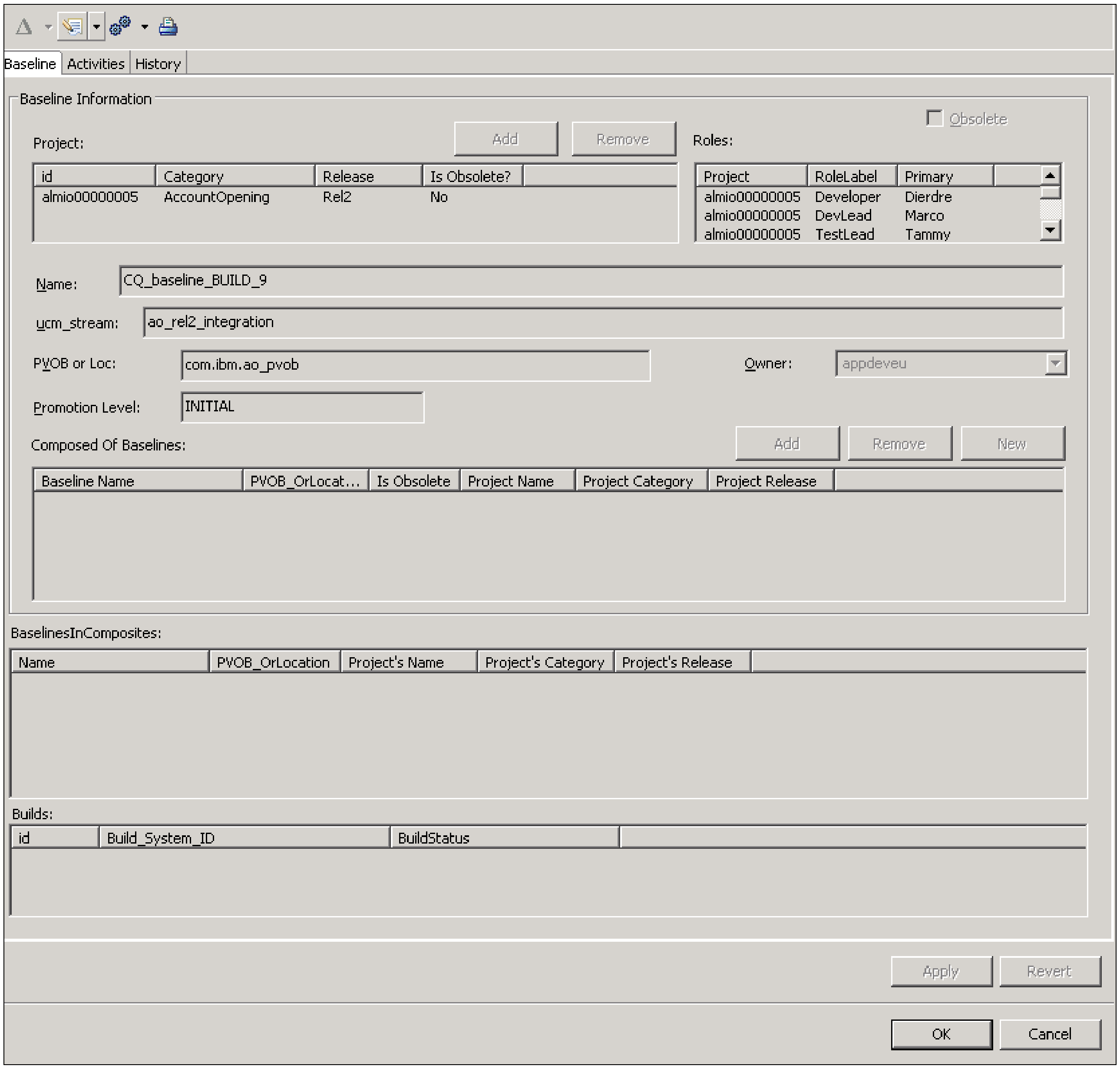
16. Create an ALMBaseline record in Rational ClearQuest.

    Example 9-17 integrates the build project with a new record in ClearQuest ALM called the *ALMBaseline record*. The step runs a Perl script that is provided by the ClearQuest ALM package. The script creates the baseline record and determines which activities were delivered between baselines. The list of activities is added to the ALMBaseline record. The URL link of the build job is also placed in the build record, in case any team member needs to see the job details.

    Example 9-17 Creating the ALMBaseline record in Rational ClearQuest

    ```
    ratlperl create_baseline_record.pl -user ${CQAL_USER} -pw "${CQALM_PASS}"
    -dbname ${CQALM_DBNAME} -dbset ${CQALM_DBSET} -pvob ${CQALM_PVOB} -ucmstream
    ${CQALM_STREAM} -baseline "${BF_TAG}_deploy" -projectid ${CQALM_PROJECT_ID}
    -logfile "${CQALM_LOGFILE}_${BF_TAG}.txt" -url
    "http://9.34.119.61:82/fullcontrol/index.php?mod=jobs&action=edit&bf_id=${BF_ID
    }"
    ```

You can see the details of the baseline record created in Figure 9-22 and Figure 9-23 on page 369.



Project:

id	Category	Release	Is Obsolete?
almio00000005	AccountOpening	Rel2	No

Roles:

Project	RoleLabel	Primary
almio00000005	Developer	Dierdre
almio00000005	DevLead	Marco
almio00000005	TestLead	Tammy

Name: CQ_baseline_BUILD_9

ucm_stream: ao_rel2_integration

PVOB or Loc: com.ibm.ao_pvob

Owner: appdeveu

Promotion Level: INITIAL

Composed Of Baselines:

Baseline Name	PVOB_OrLocat...	Is Obsolete	Project Name	Project Category	Project Release

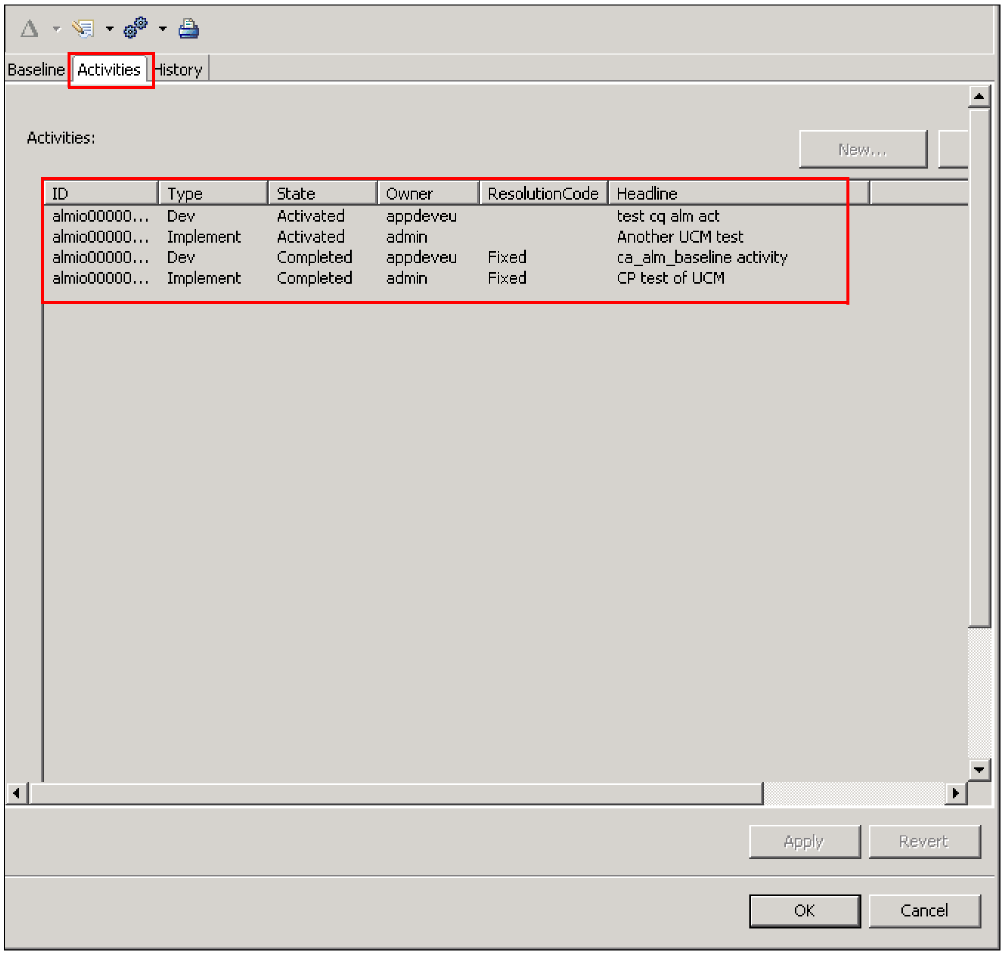
BaselinesInComposites:

Name	PVOB_OrLocation	Project's Name	Project's Category	Project's Release

Builds:

id	Build_System_ID	BuildStatus

Figure 9-22 Baseline record



ID	Type	State	Owner	ResolutionCode	Headline
almio00000...	Dev	Activated	appdeveu		test cq alm act
almio00000...	Implement	Activated	admin		Another UCM test
almio00000...	Dev	Completed	appdeveu	Fixed	ca_alm_baseline activity
almio00000...	Implement	Completed	admin	Fixed	CP test of UCM

Figure 9-23 Activities in the baseline record

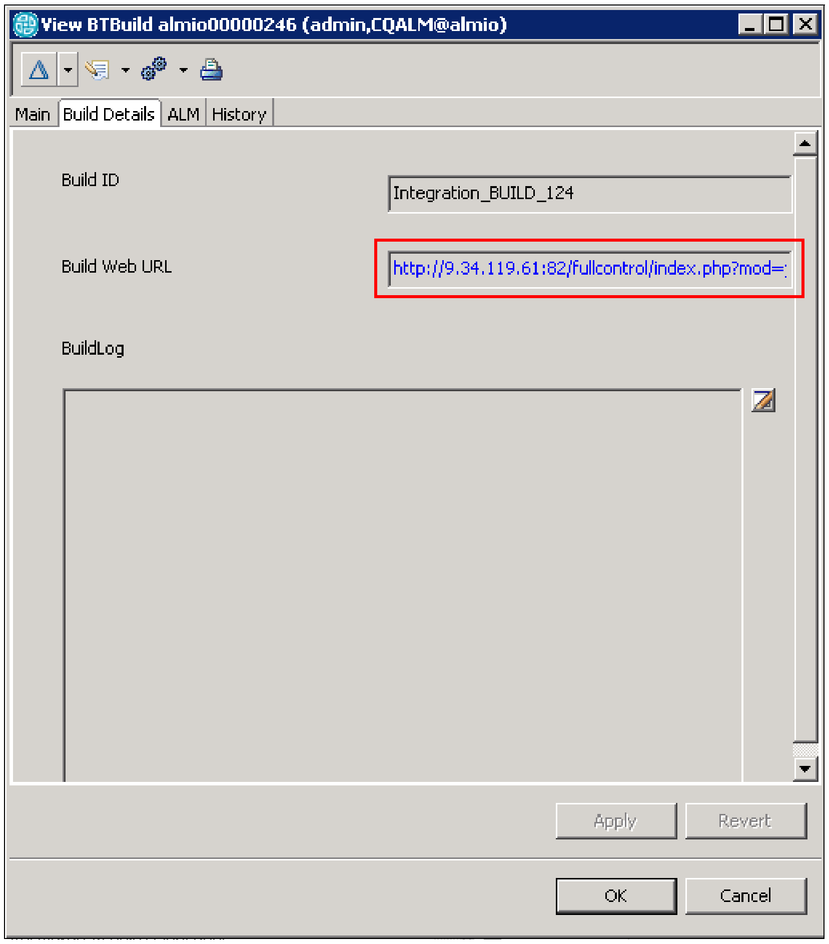
17. Create the Rational ClearQuest BTBuild record.

    The build project also creates a BTBuild record in the ClearQuest ALM package. The step runs a Perl script that is provided by the ClearQuest ALM package. The script populates the required information on the ALM tab and creates a reference to the ALMBaseline record that was created in the previous step. Example 9-18 shows how this is performed and some of the environment variables that are defined for this project by using the Environments module.

Example 9-18 Creating a Rational ClearQuest BTBuild record

```
ratlperl create_build_record.pl -user ${CQALM_USER} -pw "${CQALM_PASS}" -dbname
${CQALM_DBNAME} -dbset ${CQALM_DBSET} -pvob ${PVOB_LABEL} -baseline
"${BF_TAG}_deploy" -build "Integration_${BF_TAG}" -buildstatus "Passed"
-buildtype "Integration" -projectid ${CQALM_PROJECT_ID} -logfile
"${CQALM_LOGFILE}_${BF_TAG}.txt" -url
"http://9.34.119.61:82/fullcontrol/index.php?mod=jobs&action=edit&bf_id=${BF_ID
}
```

In Figure 9-24, you can see Build Web URL property that is set by this step. Team members can reach the build job execution details by clicking the URL from ClearQuest ALM.



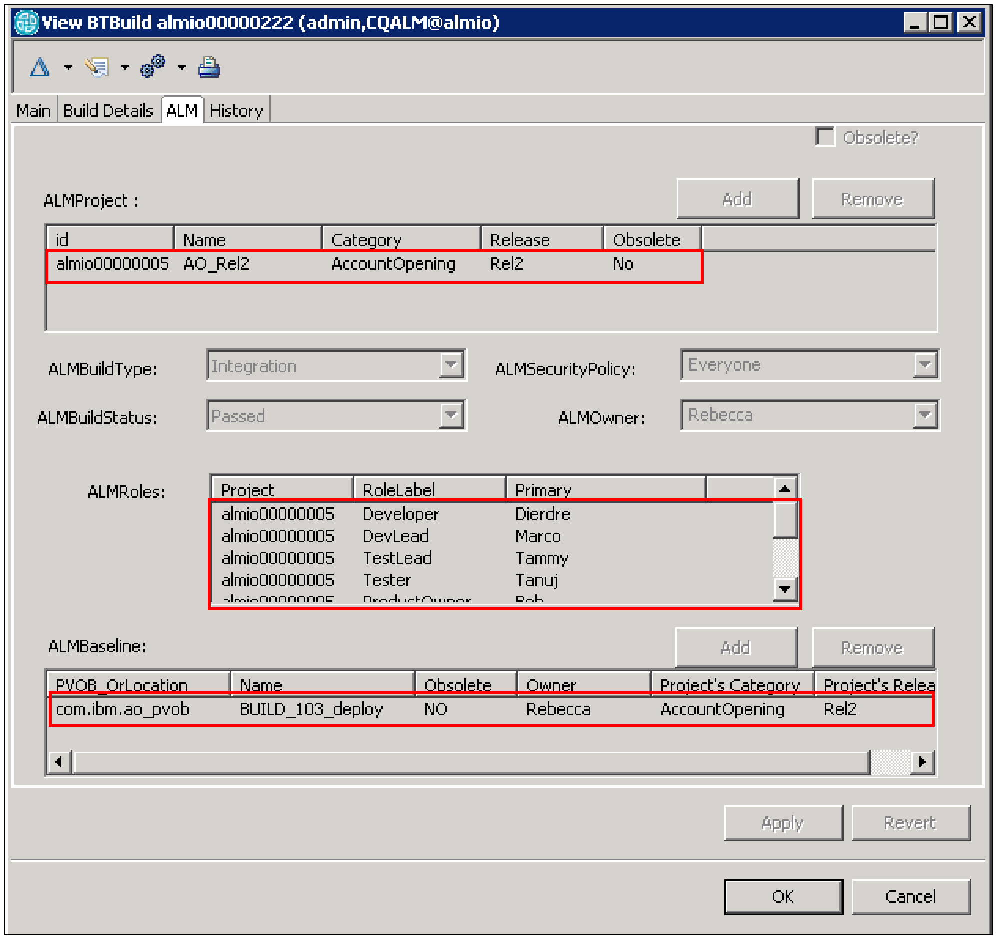
Build ID: Integration_BUILD_124

Build Web URL: http://9.34.119.61:82/fullcontrol/index.php?mod=:

BuildLog

Figure 9-24 Build Details tab showing the BTBuild record created in ClearQuest ALM

The ClearQuest ALM package also extends the BTBuild record by adding an ALM tab to the record. Figure 9-25 shows the details of the ALM properties of the build record that was created.



ALMProject :

id	Name	Category	Release	Obsolete
almio00000005	AO_Rel2	AccountOpening	Rel2	No

ALMBuildType: Integration

ALMSecurityPolicy: Everyone

ALMBuildStatus: Passed

ALMOwner: Rebecca

ALMRoles:

Project	RoleLabel	Primary
almio00000005	Developer	Dierdre
almio00000005	DevLead	Marco
almio00000005	TestLead	Tammy
almio00000005	Tester	Tanuj

ALMBaseline:

PVOB_OrLocation	Name	Obsolete	Owner	Project's Category	Project's Relea
com.ibm.ao_pvob	BUILD_103_deploy	NO	Rebecca	AccountOpening	Rel2

Figure 9-25 ALM properties showing the BTBuild record created in ClearQuest ALM

At the end of each project run or each step, Rational Build Forge notifies the group selected in the build project in the project level or in step level according to the need. In the AccountOpening build project, Rebecca is notified on failure only. On success, Al, Tammy, Patricia, Rebecca, and Marco are all notified according to our scenario.

Create user groups in Rational Build Forge to enable the best distribution of notifications and to ensure that the right notification is assigned to the right step or project.

How Rebecca used the environment variables

The build steps that Rebecca defined in 9.3.2, “Rebecca runs the integration build” on page 359, use many environment variables in commands. This approach is highly recommended to increase the reuse of the project and increase the ease of maintenance.

Example 9-19 shows some of AccountOpening environment variables.

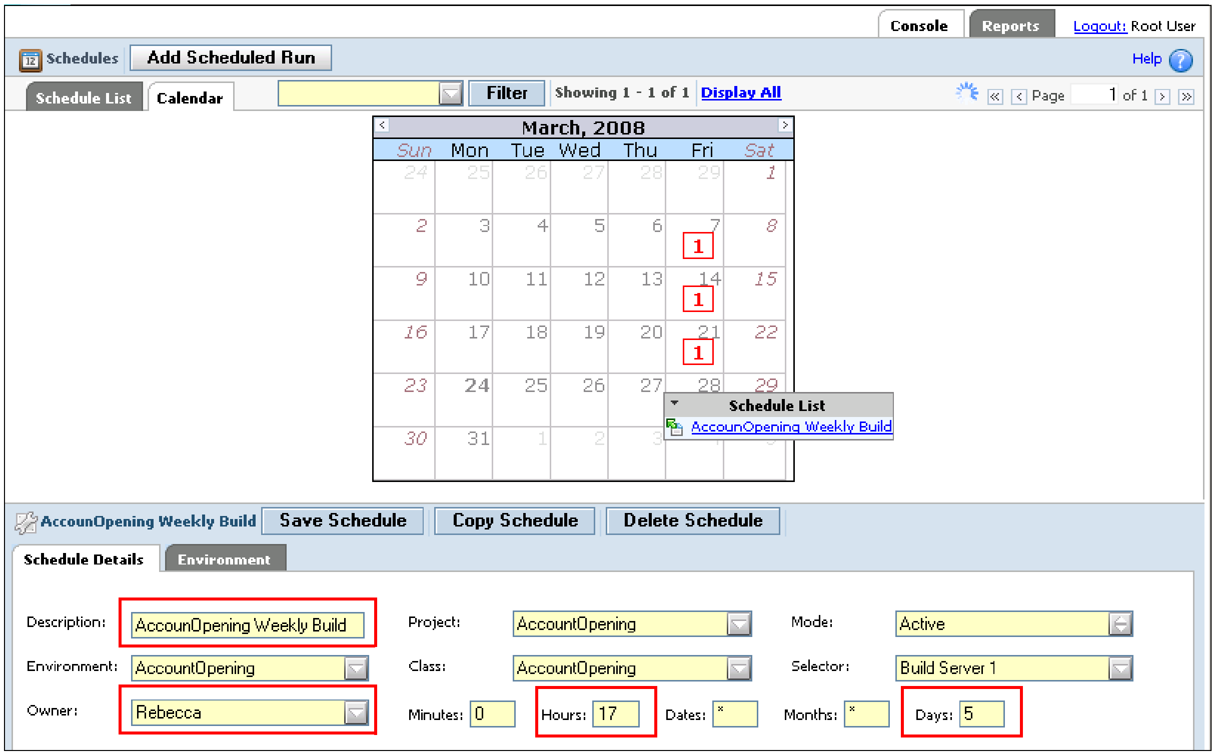
Example 9-19 Some of the AccountOpening environment variables

```
ANT_HOME=C:\Views\javatools_int\JavaTools\ant
JAVA_HOME=C:\j2sdk1.4.2_10
PATH=$JAVA_HOME\bin;$ANT_HOME\bin
VIEW_STG=//qvmw061/ccstg_c/views
PROJECT_NAME=AccountOpening
PROJECT_DIR=com.ibm.ao\accountopening\creditapplication\RatlBankWebPROJECT_VOB=\co
m.ibm.ao_pvob
SOURCE_VOB=\com.ibm.ao
CCSERVER=Build Server 1
VIEW=administrator_ao_rel2_int
COMPONENT=accountopening
BASELINE_TYPE=Incremental
VOB_PATH=com.ibm.ao\accountopening\creditapplication\RatlBankWeb
STREAM_NAME=ao_rel2_integration
CQALM_PROJECT_ID=almio00000005
PVOB_LABEL=com.ibm.ao_pvob
CQALM_LOGFILE=C:\Temp\CreateBaseline
CQALM_STREAM=ao_rel2_integration
CQALM_PVOB=\com.ibm.ao_pvob
.......
```

How Rebecca scheduled the integration build

In our scenario, the builds are produced weekly. Rebecca opens Rational Build Forge and opens the Schedules module. She schedules the AccountOpening builds every Friday at 17:00 p.m. (Figure 9-26) and performs the following actions:

1. In the Description field, she types the `AccountOpening Weekly Build` expression.
2. From the Project box, she selects the **AccountOpening** project.
3. From the Owner box, she selects the user **Rebecca**.
4. For every Friday, in the Days field, she types `5`.
5. In the Hours field, she types `17` for 17:00 p.m.



Description:	AccounOpening Weekly Build	Project:	AccountOpening	Mode:	Active
Environment:	AccountOpening	Class:	AccountOpening	Selector:	Build Server 1
Owner:	Rebecca	Minutes: 0	Hours: 17	Dates: *	Months: * Days: 5

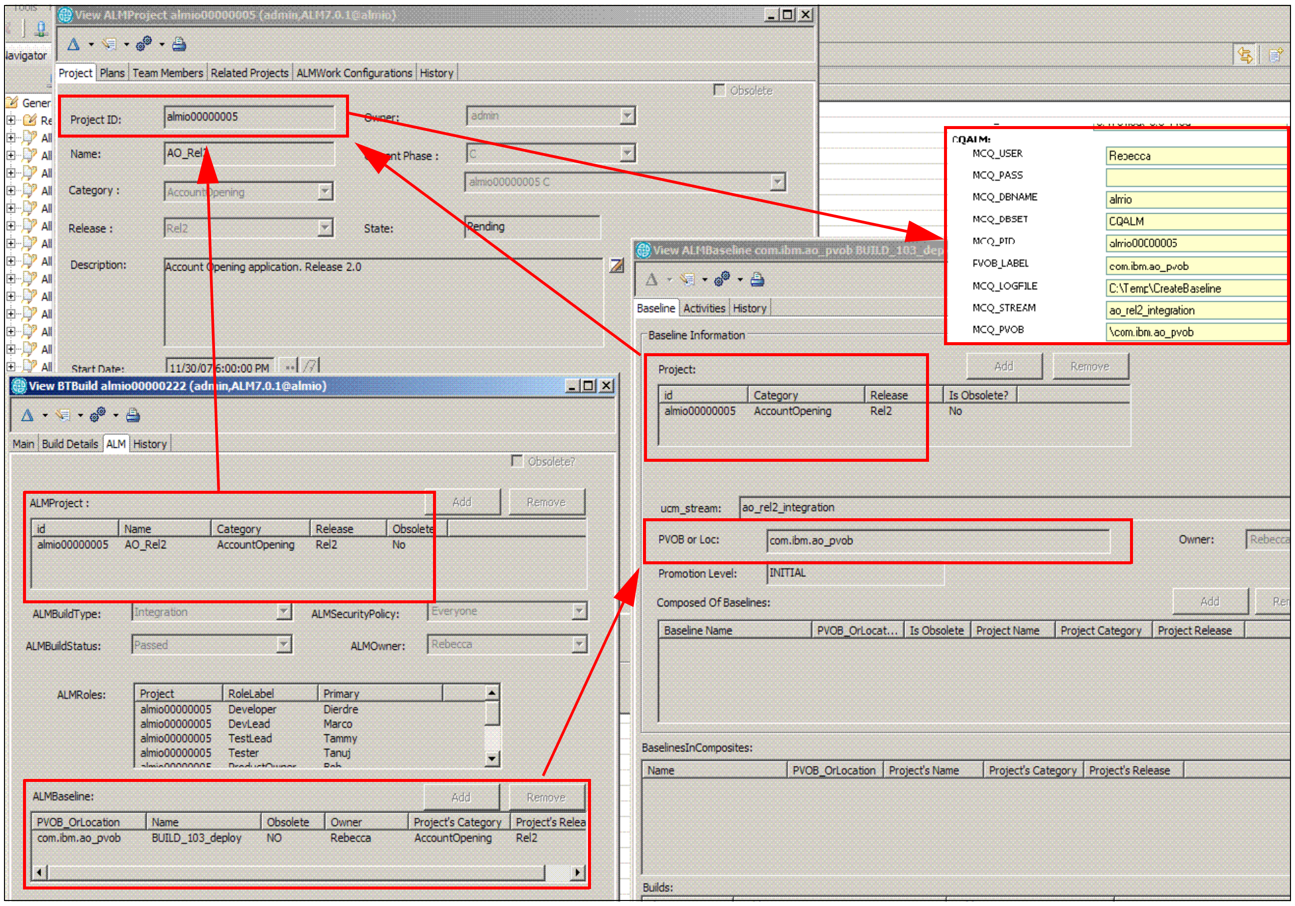
Figure 9-26 Rebecca scheduling the build

How Rebecca integrated with Rational ClearQuest

Integration with Rational ClearQuest is fairly straight forward for Rebecca. The ALMProject 1.0 and ALMWork 1.0 packages add all the necessary record types, fields, and forms needed to integrate Rational ClearQuest with Rational Build Forge. In addition, the ALMProject 1.0 package provides a ready-to-use Perl script that creates the BTBuild and ALMBaseline records in Rational ClearQuest. Rational Build Forge environment variables are created to pass information to the Perl script.

The Rational ClearQuest ALMProject ID is needed to identify the Rational ClearQuest project to which this build belongs. By having this ID, the ALMProject record can be linked to the BTBuild and ALMBaseline records by using reference list fields as shown in Figure 9-27. The Perl script ensures that all the relevant data from Rational Build Forge is populated in the appropriate fields in the Rational ClearQuest records.

The ALMProject record type also has a reference to the Unified Change Management (UCM) project on the ALMWorkConfigrations tab, which can be used to link the specific project to the UCM assets in Rational ClearCase. For more information about the ClearCase-ClearQuest UCM integration, see *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM*, SG24-6399.


View ALMProject almio00000005 (admin,ALM7.0.1@almio)

Project | Plans | Team Members | Related Projects | ALMWork Configurations | History

Project ID:	almio00000005	Owner:	admin
Name:	AO_Rel2	Current Phase :	C
			almio00000005 C
Category :	AccountOpening		
Release :	Rel2	State:	Pending
Description:	Account Opening application. Release 2.0		
Start Date:	11/30/07 6:00:00 PM		

View BTBuild almio00000222 (admin,ALM7.0.1@almio)

Main | Build Details | ALM | History

ALMProject :

id	Name	Category	Release	Obsolete
almio00000005	AO_Rel2	AccountOpening	Rel2	No

ALMBuildType:	Integration	ALMSecurityPolicy:	Everyone
ALMBuildStatus:	Passed	ALMOwner:	Rebecca

ALMRoles:

Project	RoleLabel	Primary
almio00000005	Developer	Dierdre
almio00000005	DevLead	Marco
almio00000005	TestLead	Tammy
almio00000005	Tester	Tanuj

ALMBaseline:

PVOB_OrLocation	Name	Obsolete	Owner	Project's Category	Project's Relea
com.ibm.ao_pvob	BUILD_103_deploy	NO	Rebecca	AccountOpening	Rel2

CQALM:

MCQ_USER	Rebecca
MCQ_PASS	
MCQ_DBNAME	almio
MCQ_DBSET	CQALM
MCQ_PID	almio00000005
FVOB_LABEL	com.ibm.ao_pvob
MCQ_LOGFILE	C:\Temp\CreateBaseline
MCQ_STREAM	ao_rel2_integration
MCQ_PVOB	\com.ibm.ao_pvob

View ALMBaseline com.ibm.ao_pvob BUILD_103_dep

Baseline | Activities | History

Baseline Information

Project:

id	Category	Release	Is Obsolete?
almio00000005	AccountOpening	Rel2	No

ucm_stream:	ao_rel2_integration		
PVOB or Loc:	com.ibm.ao_pvob	Owner:	Rebecca
Promotion Level:	INITIAL		

Composed Of Baselines:

Baseline Name	PVOB_OrLocat...	Is Obsolete	Project Name	Project Category	Project Release

BaselinesInComposites:

Name	PVOB_OrLocation	Project's Name	Project's Category	Project's Release

Builds:

Figure 9-27 Relationships between the BTBuild, ALMBaseline, and ALMProject records in Rational ClearQuest

9.4 Life-cycle collaboration

In this scenario, the actors not only created new artifacts, but relied on the artifacts of their team members from previous acts. In addition, the work produced by these team members will impact future scenes of the storyboard. Figure 9-28 shows the life-cycle assets that are used by the characters in this act.

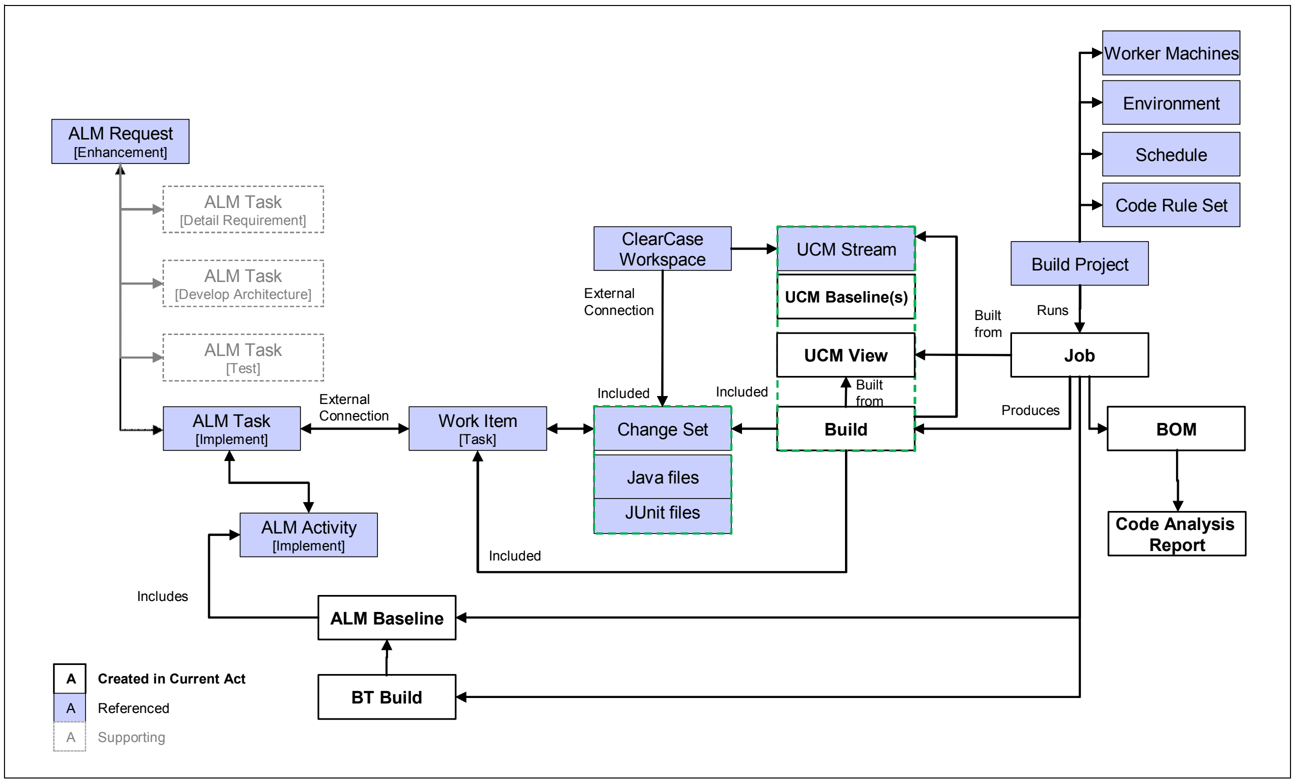


Figure 9-28 Life-cycle assets involved in this enterprise integration build scenario

The assets that are created in this act are indicated in white. The assets that are leveraged by this act are indicated in blue. This scenario includes the following life-cycle assets:

- Job (when the project is executed and its result)
- Bill of materials and Code Analysis report
- UCM baseline
- Build and the UCM stream
- ALMBaseline record in ClearQuest ALM
- ALMBuild record in ClearQuest ALM
- Build project, which refers to the AccountOpening project, which defines all information that pertains to this automation.
- Environment, which establishes the environment variables used for this project.
- Worker machines, which are a set of machines that are used by the automation, which are defined by servers, collectors, and selectors.
- Schedule, which determines when to run the job.
- Code rule set used by Rational Software Analyzer

9.5 Measuring success

Rational Build Forge has several report features in addition to the bill of materials, which we mention in "9.3.2, "Rebecca runs the integration build" on page 359. It provides ready-to-use reports that help development teams analyze their build and release processes over time. Detailed statistics about each step of the process highlight process bottlenecks or abnormalities so that teams can take corrective action. Additionally, server utilization reports pinpoint ways that hardware can be used more efficiently.

Rational Build Forge also tracks critical information to help companies meet their compliance requirements. The system captures each change that is made, by whom, and why in a comprehensive bill of materials for the release that can be generated on demand (Figure 9-29).

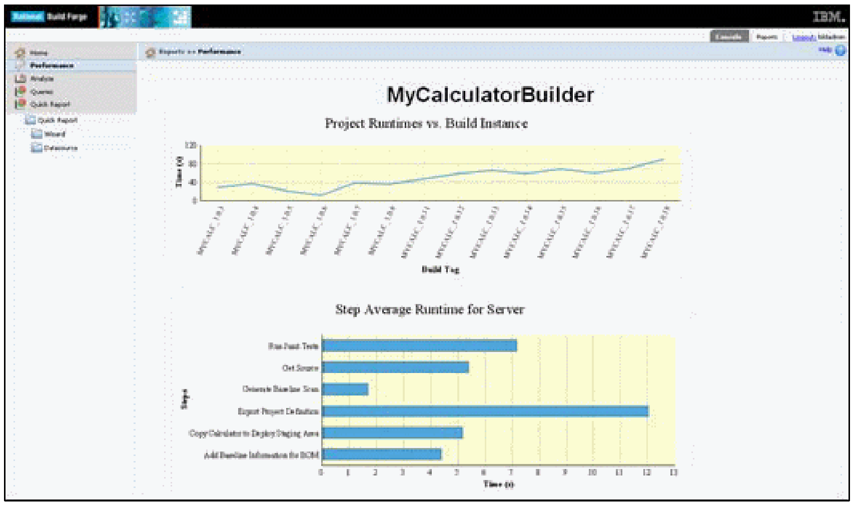
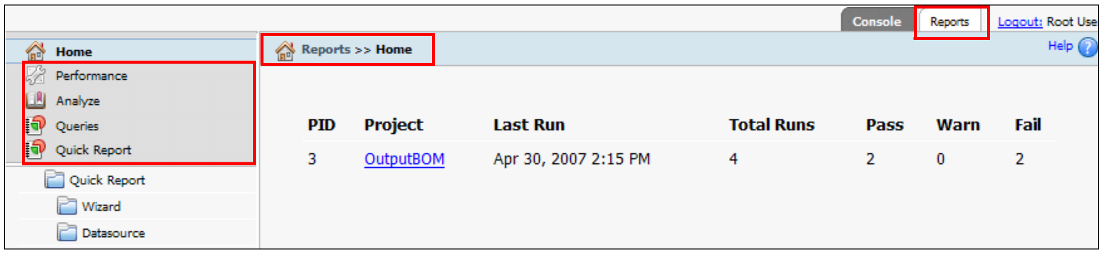

MyCalculatorBuilder
Project Runtimes vs. Build Instance
Time (s)
Build Tag
Step Average Runtime for Server
Run Junit Tests
Get Source
Generate Baseline Scan
Export Project Definition
Copy Calculator to Deploy Staging Area
Add Baseline Information the BOM
Steps
Time (s)

Figure 9-29 A sample report in Rational Build Forge

When you click the Report tab in the upper right of the main pane to display reports about your system (Figure 9-30), you can choose from the following reports:

- Home
- Performance
- Analyze
- Queries
- Quick Report



PID	Project	Last Run	Total Runs	Pass	Warn	Fail
3	OutputBOM	Apr 30, 2007 2:15 PM	4	2	0	2

Figure 9-30 Reports in Rational Build Forge

Home

The reporting Home module shows the same report as the Performance module. It shows the last job time for each project and data on the total number of jobs and how many jobs passed, failed, or passed with warnings. Click a project name in the list to display the project performance detail page, which graphs run times for all the jobs of the project.

Performance

The Performance module shows the last job time for each project and data on the total number of jobs and how many jobs passed, failed, or passed with warnings. Click a project name in the list to display the project performance detail page, which graphs run times for all the jobs of the project. Performance reports give you an overall idea for all runs. If you need more detailed information for your build project, you can run Analyze Reports or Queries Reports instead.

Analyze

The Analyze module displays information about the run times and numbers of passing or failing jobs for each project. Click an individual project name to display additional information. When you do, the system displays a comparison of the time that is required to perform each step in different runs and on different servers. The system displays the probability of encountering the longest and shortest run times for each step.

Queries

You can run the following reports from the Queries module:

- Identify the project selectors and step servers for each project.

  Click the **Run** button to display a list of projects and their steps. For each project, the system lists the selector. For each step, the server lists all the servers that the step has ever used in a job.

- Identify the current manifest for each server.

  Choose whether to include BF_ properties in the report. With this report, you can also compare the manifests for your servers.

- Build results historic data.

  With this report, you can select a range of dates. Then the report displays the jobs in that range, showing the number of passing and failing jobs for each project that had a job in the range. A selected date starts at midnight (00:00) on that date. Therefore, to specify a day's data, select that day for the beginning and the next day for the end.

- See Server and Selector utilization historic data.

  Use this option to view server usage over time.

- Locate a file based on its MD5 value.

  You can search through all completed jobs for a file if you have the MD5 value for the file. You can get an MD5 value from the bill of materials for a job, by running a `.scan` command.

Quick Report

Quick Report is a licensed option in the Rational Build Forge system. It appears and functions only if you have installed the license key for Quick Report. The installation includes a Tomcat server running on port 8080. Therefore, the Management Console should not be set to run on 8080.

Running existing reports

To run an existing report:

1. Click the **Wizard** selection in the navigator on the left.
2. Select **Start** from an existing report (or click a report in the list).
3. Click the report to use. Reports are organized under Public and My Reports headings.
4. Click **Run** to run the report.

You also have other options to manage the selected report:

- Use *Edit* to edit the report in the same way as you create a new report. Remember to save the report after you edit it.
- Use *Delete* to delete the report.

Creating new reports

To create a new report:

1. Click the **Wizard** selection in the navigator on the left.
2. Select **Create a new report**.
3. Define the report:
   a. Click a format, such as Table, Bar Chart, Pie Chart, or Line Chart, for the report. When you click a format, an illustration of it is displayed to the right of the format list.
   b. For Title, type a title for the report. You cannot change the title after you specify it here.
   c. Click **Next**.
4. Select a report type.

   Types are organized under the Build Forge and Custom headings. Custom reports come from configuring data sources from XML files, including exported bill of materials' data.

   For Tables, select one or more columns to use as fields in the table. For Bar Charts, Pie Charts, and Line Charts, choose columns to use for X Series Selection and Y Series Selection (X and Y axis of the chart). Click **Next**.
5. Populate the Group Ordering list.

   Note: Any report that includes a calculated field, such as Total, Percent, and so on, must have a grouping for that field.

   The report shows groupings and subgroupings as you specify in the list. A grouping is performed in order from the top of the list downward. Use the arrow controls to move fields to and from the Group Ordering list. Single arrows move a selected item to or from the list. Double arrows move all items to or from the list. Use the Top, Up, Down, and Bottom buttons to sequence items in the list as you want. Click **Next**.
6. Populate the Sort Ordering list.

   The report sorts within groups by using the fields in the list. Use the arrow controls to move fields to and from the Sort Ordering list. Use the arrow control to the right of fields in the Sort Ordering to change the sort order (ascending and descending). Clicking the arrow moves a field in the direction indicated. Click **Next**.
7. Build one or more filters. Each filter consists of selections:

Field	The field in which to filter results based on value.
Operator	The operator on the field.
Value	Either select a field or check the Text box and type a value.

You can add more filters to the list or delete them. You can limit output by selecting **Limit output to** and providing a value for rows. This option can be helpful when running tests on designs for large reports. Click **Next**.

8. Run the report. It is displayed in the panel. The following options are available:

   Save this Report Design
   Type a name and click **Save**. The report is displayed in the list of reports.

   HTML Download the report output in HTML format. You are prompted for a location.

   Cancel Exit this report and go back to the wizard.

9.6 Reference architecture and configuration

In this section, we describe how the products used in this act of storyboard fit into the overall solution architecture and how they are configured.

9.6.1 Fitting into the enterprise ALM solution

The Build Forge repository is installed at the corporate headquarters. Build Forge Agents should be running at the build servers and at the targets that Rational Build Forge uses. In this scenario, we have agents at the Rational ClearCase server, Rational ClearQuest server, Software Analyzer, and two separate build servers (Figure 9-31).

Rebecca used the Rational Build Forge adapters for Rational ClearCase, which we discuss in the next section.

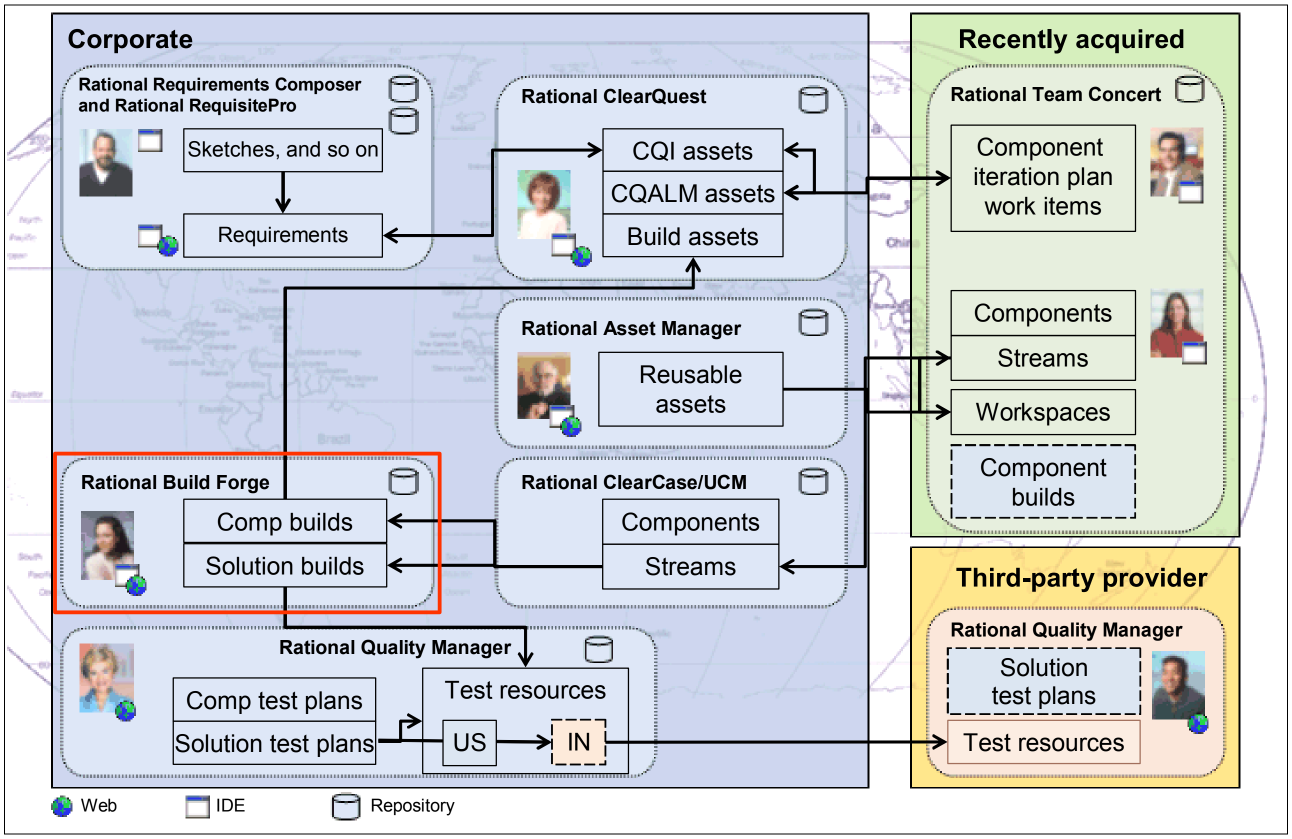


Figure 9-31 Configuration for this scenario

9.6.2 How Rational Build Forge is configured for this scenario

The configuration that is used by Rebecca is explained in 9.3.2, “Rebecca runs the integration build” on page 359, and illustrated in Figure 9-32. In this section, we discuss the use configuration of Rational Build Forge adapters.

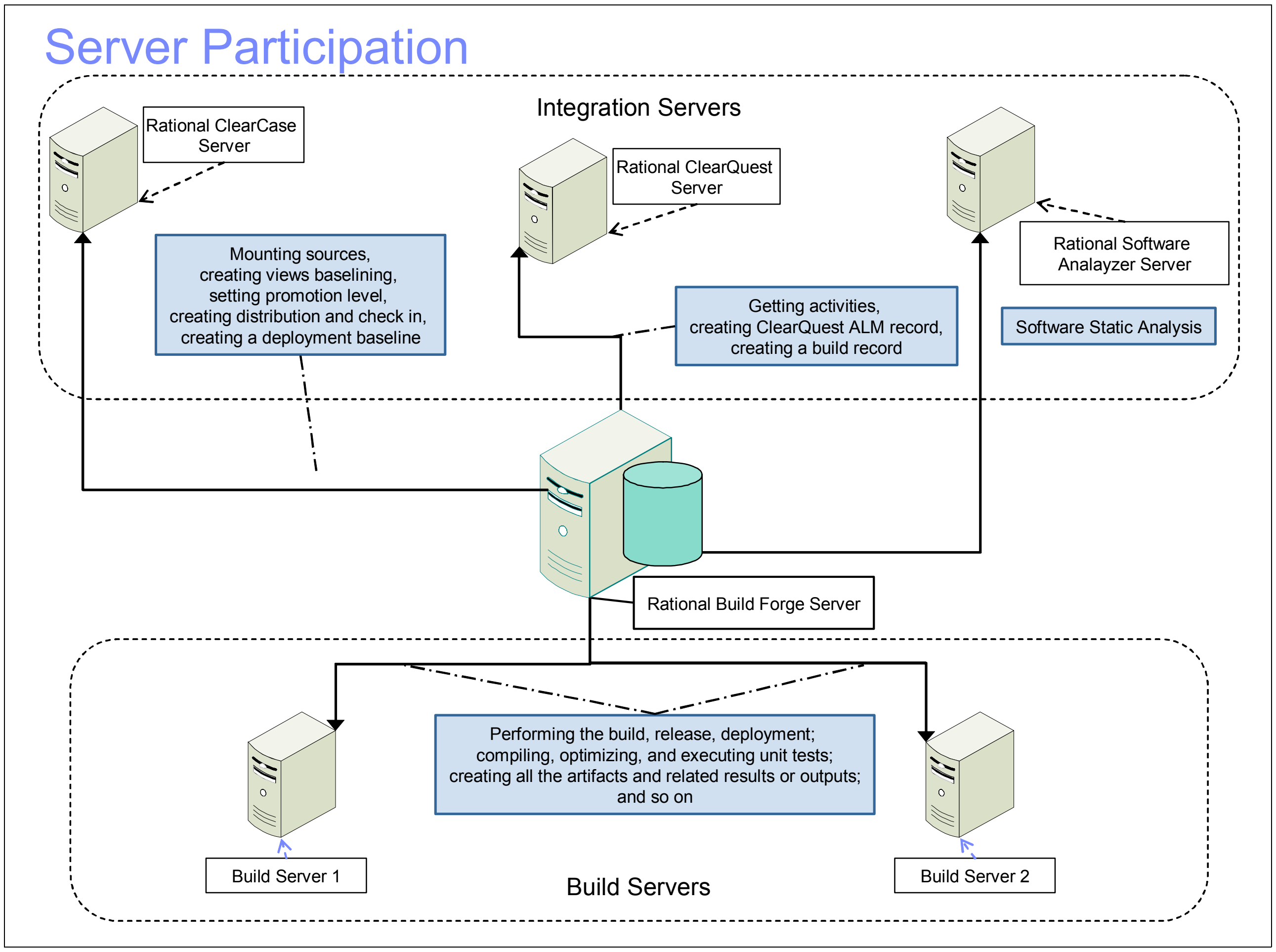

Server Participation
Integration Servers
Rational ClearCase Server
Rational ClearQuest Server
Rational Software Analayzer Server
Software Static Analysis
Mounting sources, creating views baselining, setting promotion level, creating distribution and check in, creating a deployment baseline
Getting activities, creating ClearQuest ALM record, creating a build record
Rational Build Forge Server
Performing the build, release, deployment; compiling, optimizing, and executing unit tests; creating all the artifacts and related results or outputs; and so on
Build Server 1
Build Server 2
Build Servers

Figure 9-32 Server participation

Rational ClearCase, Rational ClearQuest, and Rational Software Analyzer adapters

An *adapter* is an add-on that links the system to other, external information systems. You can use adapters to extract information from other data sources in your software development environment, such as source code control systems, change management systems, and test management systems. You can also store that information in the bill of materials for each project run. Rational Build Forge comes with adapters for the most popular source code control systems. You can also create your own adapters to connect to other information systems within your enterprise.

An adapter is a bi-directional interface between your project and another system, so that it can also send information and commands to other programs. For example, the adapter for the Rational ClearQuest change management system lists all the defect records that are associated with the project run in the bill of materials. It also updates each defect record with information about the project run and resolves each defect record within Rational ClearQuest.

You can directly use the **cleartool** command in the project steps to have connectivity with Rational ClearCase or you can use the Rational Build Forge ClearCase adapter.

For ClearQuest and ClearQuest ALM, you can use Perl scripts or Java directly in your commands or the Rational Build Forge ClearQuest Adapter.

For Rational Software Analyzer, you can use the adapter that comes with Rational Software Analyzer with installation, or you can directly use Rational Software Analyzer API in Rational Build Forge. If you prefer to use the adapter for Rational Software Analyzer, it already has a bill of materials section in it. Therefore, reporting is much easier than using the API. If you use the API, you must use additional **.bom** commands to create the bill of materials section in the job steps or you must look at the target "export directory."

Adapters in Rational Build Forge are crucial for connectivity. In this book, we used Rational ClearCase, Rational ClearQuest, and Rational Software Analyzer adapters as examples.

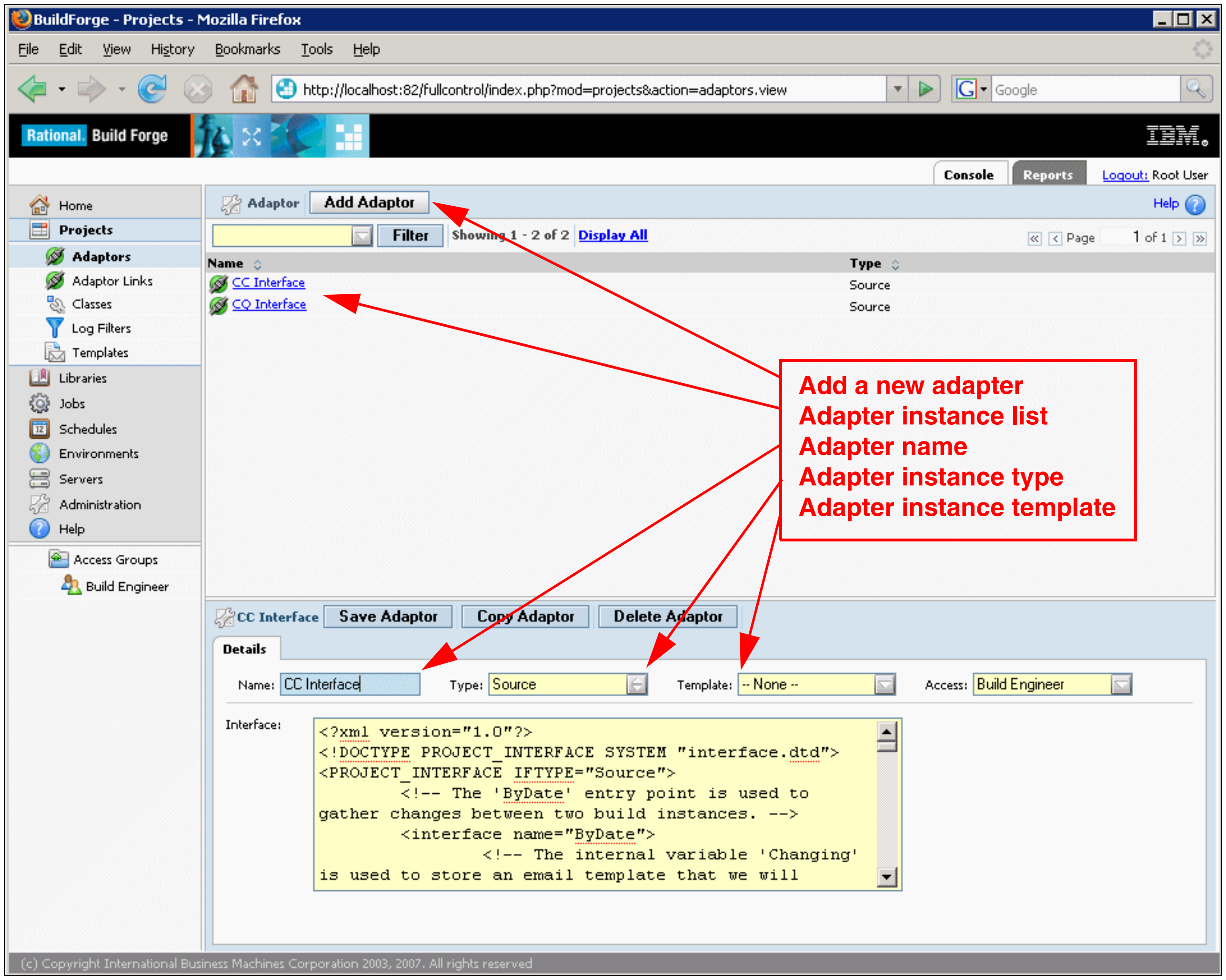
Adapters

An *adapter* is an interface to an external application. Adapters allow a Rational Build Forge project to exchange information with an external application to accomplish a goal. For example, this might involve checking to see if there are source code changes in a source code management (SCM) system. If there are changes, the adapter annotates the bill of materials with source code change information. If there are no changes, the step aborts the remainder of the execution. This feature is especially helpful for those running "continuous integration," because it avoids unnecessary re-builds of the project.

An adapter is a mechanism that encapsulates actions that integrate with an external system and reports information to the bill of materials. Adapters are the crucial bit of functionality that allows Rational Build Forge to represent multiple branches from a single bit of input. Without adapters, you only use the regular steps in Rational Build Forge, which function much in the same way as a makefile. The steps execute something on a client machine, and output is returned, a binary action. That is the basic kind of integration that is easy to set up in Rational Build Forge. More than integration, it allows a rudimentary branching logic and looping ability.

An adapter is an instance of an adapter template (Figure 9-33 on page 382), but you can also create your own adapter from scratch without using any adapter template as long as you obey the tags that are described in the Rational Build Forge Help file.

When you create an adapter, you assign it a unique name and associate it with a template. The template is an XML file. The XML file contains application commands to gather information, instructions for analyzing information, and format details for displaying results in the Bill of Materials report. The templates that are provided by Rational Build Forge are designed to be used without modification. However, you can modify templates or use templates as a model for creating a new adapter template. The adapter templates are installed in the <bf-install>\interface directory.



```
<?xml version="1.0"?>
<!DOCTYPE PROJECT_INTERFACE SYSTEM "interface.dtd">
<PROJECT_INTERFACE IFTYPE="Source">
        <!-- The 'ByDate' entry point is used to
gather changes between two build instances. -->
        <interface name="ByDate">
                <!-- The internal variable 'Changing'
is used to store an email template that we will
```

Figure 9-33 Adapters

The adapter requires environment variables to execute application commands. In the adapter templates, environment variables are listed in the <env> elements in the <template> section of the XML file. For example, for the ClearCaseBaseline adapter, the following environment variables are listed in the ClearCaseBaseline.xml file:

- `<template>`
- `<!-- Template section. These variables are parsed out of the final XML.Use the following syntax to help identify the variables that are needed to run this interface if you are integrating it during a regular Build Forge step:- →`
- `<env name="VIEW" value="my_adapter_view" />`
- `<env name="VOB_PATH" value="\adapterVob" />`
- `<env name="CCSERVER" value="BFServerName" />`
- `</template>`

Environment variables: In Rational Build Forge, environment variables are stored in environments. Before creating an adapter, create an environment for application environment variables.

Most adapter templates send e-mail notification to users. For example, when the ClearCaseByDate adapter executes, it sends a pass e-mail notification to users who changed source code files. If no files were changed, it sends a fail e-mail notification.

The Rational Build Forge product provides adapter templates for the all kinds of applications as listed in Appendix C, “Rational Build Forge adapter templates” on page 623. The templates for Rational ClearCase and Rational ClearQuest do not require a separate license key, but other application templates are licensed through the Rational Build Forge Adapter Toolkit.

Adapters are an important part to the functionality of Rational Build Forge. Implementing your own adapters to fit your workflow needs is the key to increasing connectivity with other tools.

Enabling continuous integration with Rational Build Forge source adapters

With source code adapters, the system can monitor and track changes in source code control systems and perform actions based on those changes. When properly linked to a project, an adapter can be activated along with a scheduled project run. The system runs the adapter in a special step it inserts before the first step of the project.

A link between a project and an adapter can be created to let a source control adapter know where specifically within your source control system it should look to check for changes. When the system runs a project that is linked to a source control adapter, the system runs the adapter commands as though they were contained in a special step that is inserted before the first step of the project. The ClearCaseSnapshot adapter template checks for changes since the last time the project ran. If the step succeeds, the project proceeds as usual. If it fails, the run is cancelled and deleted. The build tag is not incremented if no changes are found in the source control.

.source command: A source adapter can also be executed from a step by using the `.source` command. You can see an example of it when Rebecca creates a Rational ClearCase baseline in Example 9-6 on page 365.

Additional capabilities in Rational Build Forge not used by this scenario

This section provides a brief overview of additional capabilities that are provided by Rational Build Forge. While these are not demonstrated in the reference scenario, the topics covered may be of interest when creating Rational Build Forge projects.

Projects

The *Pass/Fail Chain property* selects the project that is executed when the project build processor fails. By setting a pass/fail chain at the project level, you can invoke separate actions based on the pass/fail status of the project. This capability is similar to setting pass/fail actions at the step level within a project.

At the project level, the pass/fail actions are triggered by the project run status that is not the step status. You can link projects together by using a feature called *chaining*. You can use this feature to maintain frequently used groups of steps separately from projects that depend on them. Other uses include executing automated test and deployment projects upon completion of certain steps. Chaining can also be used to clean up files that are no longer needed by development teams, by assigning a project to be run at the completion of a job of a specific class.

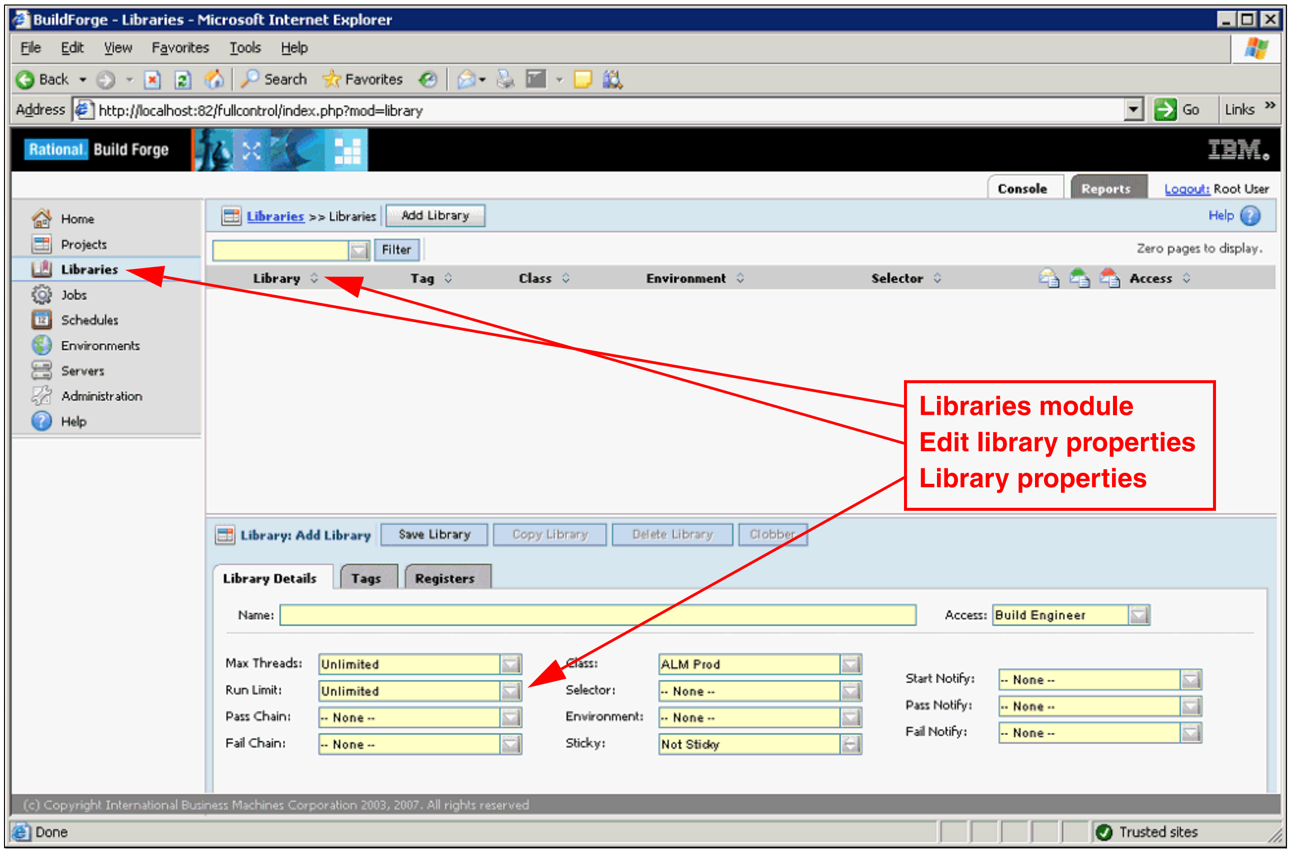
Tag Sync is used to synchronize the tag variables for two projects. Select the project whose tag variable you want to synchronize with the current project. When two projects are synchronized, their variables are drawn from the same pool, so that when they run in sequence, one project gets the value 1, the next gets the value 2, and so on.

Libraries

You can use libraries to modularize common steps into a reusable unit. This reusability makes libraries an important component for streamlining the creation of new projects. Rather than creating the same set of steps over again, you define a library that projects can call. Use care when making changes in libraries, because if the library changes, all projects that use it also change.

The Libraries module (Figure 9-34) shows library projects. When a project does not have a selector specified, it is displayed in the Libraries module. These projects absorb the selector of any project that calls them. They are typically called by other projects through inline chains or pass/fail chains. Libraries are run from Project Step Chains or Inlines. A library inherits the environment from the parent project.

From the Libraries module, you can view, edit, create, or launch library projects. You can execute a library project by itself, but you must specify a selector when you do so. You can change a library project into a normal project by editing the project and choosing a selector for it. When you save a library project with a selector, it becomes an ordinary project, disappearing from the Libraries list. Aside from the lack of a selector, libraries are treated just like any other project.


BuildForge - Libraries - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:82/fullcontrol/index.php?mod=library
Go Links
Rational Build Forge
IBM.
Console Reports Logout: Root User
Home
Projects
Libraries
Jobs
Schedules
Environments
Servers
Administration
Help
Libraries >> Libraries Add Library
Help
Filter
Zero pages to display.
Library Tag Class Environment Selector Access
Libraries module
Edit library properties
Library properties
Library: Add Library Save Library Copy Library Delete Library Clobber
Library Details Tags Registers
(c) Copyright International Business Machines Corporation 2003, 2007. All rights reserved
Done
Trusted sites

Name:				Access:	Build Engineer
Max Threads:	Unlimited	Class:	ALM Prod	Start Notify:	-- None --
Run Limit:	Unlimited	Selector:	-- None --	Pass Notify:	-- None --
Pass Chain:	-- None --	Environment:	-- None --	Fail Notify:	-- None --
Fail Chain:	-- None --	Sticky:	Not Sticky		

Figure 9-34 Libraries module

Selectors

A selector contains a list of property/value pairs called *variables*. For each variable, you can specify a value and a comparison. For example, you can specify a property “CompilerVersion = 1.1” to select only servers that have that property, but you can also specify “CompilerVersion >= 1.1” to select servers with versions 1.1, 1.3, 2, and 2.0. Selectors support numeric and string comparison operations:

- A variable can be required or optional. When multiple servers match the required variables, the system chooses the one that matches the most optional variables.
- You can repeat optional variables in a selector, to increase the score of a server that matches them. For example, you might require MEM_TOTAL> = 1 GB but repeat MEM_TOTAL >= 2 GB three times to bias the system to choose servers with memory of at least 2 GB. See the following list of actions for details about how the system makes its choice.
- You can use the `.include` statement to add environment variables from another selector. The `.include` statement references an environment. If duplicate variables are in environments, the system counts each instance of the variables when it assigns scores to servers.

The system chooses a server by using the following sequence of actions:

1. It compiles a list of the servers that contain all the required variables in the selector.
2. It rates each server, granting the server a point for each optional variable that it matches.
   - If the selector contains more than one copy of the same variable, the extra copies grant extra points to servers that match them.
   - The system assigns one extra point to the server with the lowest BF_LOADRATIO value.
3. It chooses the server that received the most points.

9.7 Problem determination

The build can be broken for several reasons. The most important issue is to determine the type of failure.

System-level failures

System-level failures originate in the operating system, network, database, or tool level. For example, if you do not have a Build Forge Agent running on the target server that the build steps use, the build project fails at the concerned step. If your network connection is not available for that instance for a particular target server for any reason, the result is the same. Also the tools that you command from Rational Build Forge should be up and running during the execution of build.

Before the build engineer asks the team to do any task to fix a failure, the engineer must be certain that the issue is not related to system-level failure.

Project-level failures

Project-level failures are the most valuable and most appreciated ones, because they are the only ones that indicate if something is wrong with the build or release.

For example, if the test scripts in our build project have errors, the project is failed by Rational Build Forge. When any filter pattern results in a failure, the project is also flagged as failed.

Conditional failures are also project-level failures. If the conditions that we set for the build project are not fulfilled as discussed in “Selectors” on page 353, Rational Build Forge generates a timeout failure after a certain period of time that is defined by build engineer in the project.

For instance, in our selection of the AccountOpening build project, we require a maximum of two actively running build projects at a time on the defined “Build Server 1.” Therefore, if we have more than two executions of a build when we run the third project, then Rational Build Forge waits for the defined timeout period and lets the build project fail.

Unmaintained project failures

Generally unmaintained project failures are encountered because of weak communication environment between team members. A build project expects every software artifact that will be used in the build or release is where it is meant to be as defined in the environment variables. Rebecca should be notified of any change so that she can update the related environment variable before the next scheduled project runs.

For instance, consider that Rebecca changes her password in Rational ClearQuest, but forgets to change it in the CQALM_PASS environment variable in Rational Build Forge. Obviously, she will have a failure when she creates the ALMBaseline record in Rational ClearQuest in Example 9-17 on page 367.

Part E

Act 4: Managing quality

In this part, we highlight how our customers and IBM are applying Application Lifecycle Management (ALM) to manage quality. Act 4 of the storyboard begins when the integration build is complete. The test team deploys the build onto the test servers and tests the solution. Two chapters are provided. Chapter 10, “The solution test team manages quality” on page 389, provides information about quality management as it relates to the scenario. Chapter 11, “Rational Quality Manager for managing quality” on page 409, provides detailed information about the Rational products that are used to support this act of the story.

Role-based guide: To understand how the content in this part applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

10

The solution test team manages quality

In this chapter, we provide an overview of quality management along with a reference scenario for how it can be applied by an enterprise team.

This chapter provides, describes, discusses, or contains the following topics:

- An introduction to quality management
- A reference scenario for quality management
- Information about how this scenario relates to the previous scenarios and how it can impact the subsequent scenario in the life cycle.
- Considerations for quality management and variations on the scenario

This chapter includes the following sections:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

10.1 Introduction to managing quality

The testing discipline has evolved from manual testing into a sophisticated battery of testing techniques and testing gates. In this chapter, we discuss the evolution of testing from a defect-driven perspective into a more holistic approach called *quality management*.

10.1.1 The changing test market

Testing was once viewed as an afterthought to the "more meaningful" act of developing software. The testing organization was established to find defects. This was considered an important step in managing the quality of the application. However, it placed the testing and development teams at psychological odds with each other. Developers take pride in not having defects in their source code, while the testers job is to find them. Therefore, a natural tension arises between the two organizations when their function is viewed from this perspective.

Software has proliferated into common household devices, appliances, and machines, in addition to complex systems and critical IT applications. As software has become a part of our daily life, in turn, it is driving more of the business' bottom line. The quality of the software has a direct impact on the business, and the reputation of the business is based on it. The need to deliver quality software brings a new spotlight on the testing discipline. As such, testing techniques have evolved from purely manual testing to a blend of manual and automated functional, performance, services, security, and compliance testing to respond to the critical role software plays in our every day life (Figure 10-1).

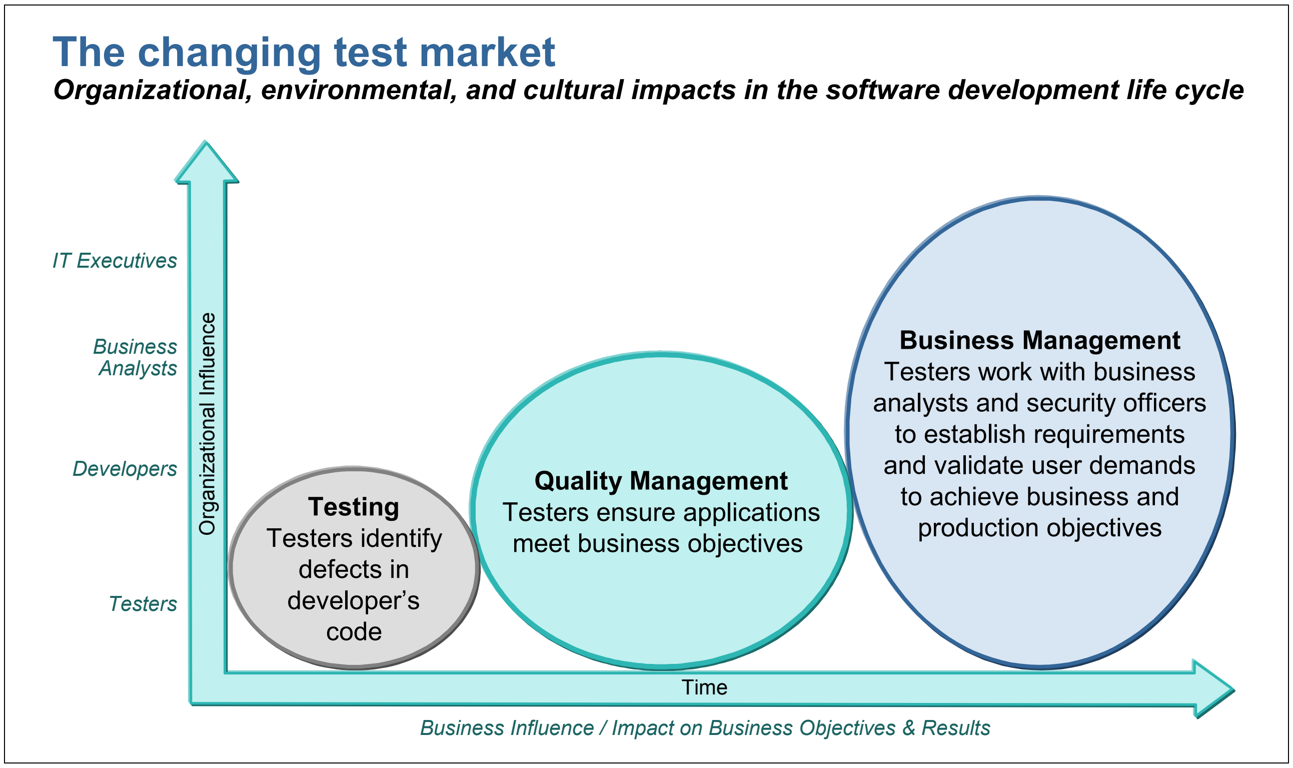

The changing test market
Organizational, environmental, and cultural impacts in the software development life cycle
IT Executives
Business Analysts
Developers
Testers
Organizational Influence
Testing
Testers identify defects in developer's code
Quality Management
Testers ensure applications meet business objectives
Business Management
Testers work with business analysts and security officers to establish requirements and validate user demands to achieve business and production objectives
Time
Business Influence / Impact on Business Objectives & Results

Figure 10-1 The changing test market

The market focus has moved from "testing" to quality management and is now trending toward business-driven quality management. The move to quality management involves understanding and delivering what the business needs. Quality management is a shared discipline across the organization, where the development team (developers and testers alike) seek to understand the business and deliver what the business needs.

Traditional testing made sure that the software did what the developers said it would do. Quality and business management moves the focus to ensure that the team delivers software that the business has defined. Rather than placing the development and testing organizations at odds, quality management creates a common goal between the teams. The goal is to have a common understanding of the user requirements and use these requirements to validate the developed application. The move to quality management has the business analysts, test and development teams closely working together. Achieving customer satisfaction is the focus of all organizations.

10.1.2 Quality management blueprint

With quality management, every team member contributes to the quality of the release throughout the development life cycle (Figure 10-2). The business analyst contributes the business requirements by using the language and expectations of the user, with the intent of defining what the user needs. A system analyst or modeler works with the business requirements to understand and develop the application requirements to define how the application will be implemented to satisfy user needs. Developers refine the design, implement, build, and test the solution against the requirements and expectations of the user.

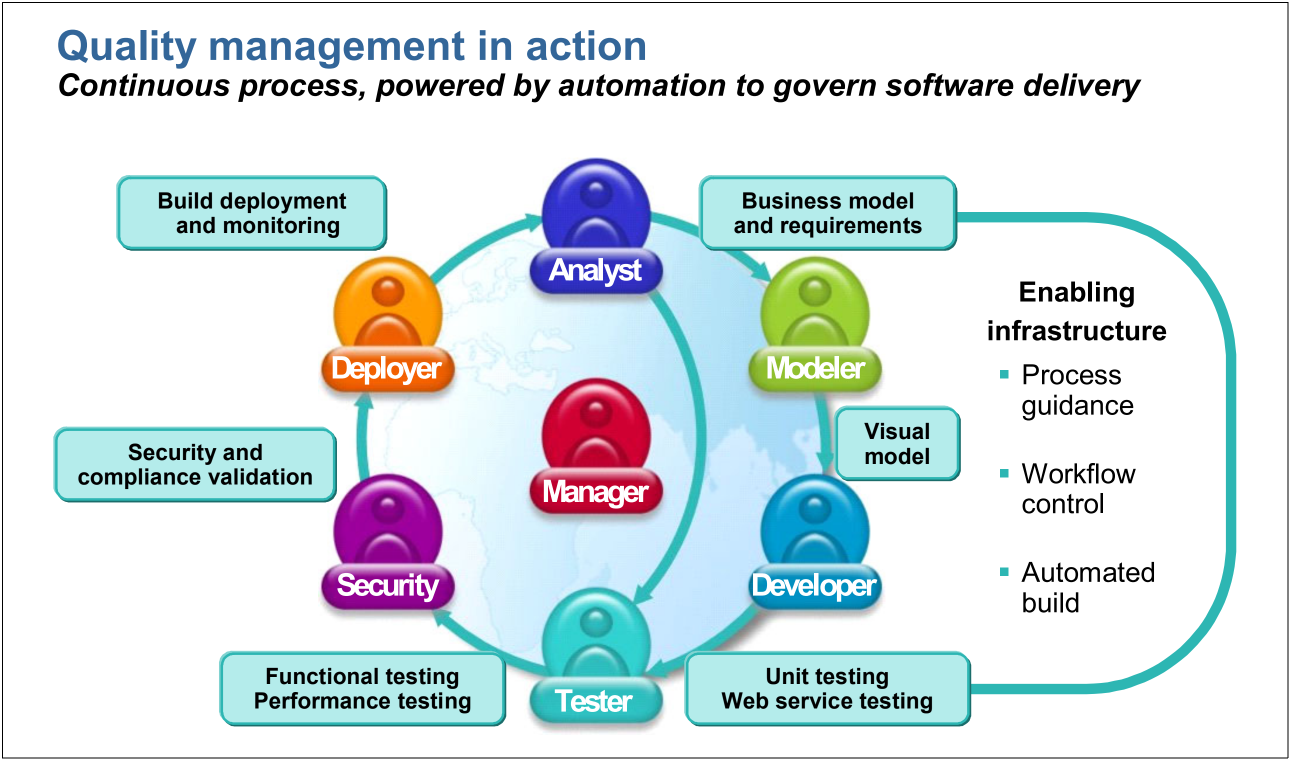

Quality management in action
Continuous process, powered by automation to govern software delivery
Build deployment and monitoring
Business model and requirements
Analyst
Deployer
Modeler
Enabling infrastructure
Process guidance
Workflow control
Automated build
Visual model
Security and compliance validation
Manager
Security
Developer
Tester
Functional testing Performance testing
Unit testing Web service testing

Figure 10-2 Quality management in action

Even the build system contributes to quality management through the automation of manual tasks and the running of build verification tests. Quality management can also involve monitoring the team build in a continuous build scenario to ensure. at delivery time. that developer changes have not broken the team build. To have a successful build is not enough in a quality management initiative. Automated build systems include build verification testing to provide insight into the quality of the build. Obvious errors can be fixed early by the development team, leaving the test team to focus on running system-wide tests that might uncover the not-so-obvious defects with the system.

The test team is where we traditionally look for quality management, and with the shift toward quality management, their role is expanded. Test plans that were once overlooked or ignored now form a type of contract back to the business by stating what the business expects. Test teams have a battery of testing techniques to increase the quality of the application, from traditional manual testing, to automated functional and performance testing. Security and compliance tools are used to scan a running application by using a predefined set of criteria to catch vulnerabilities. Service-oriented architecture (SOA) quality management is an important aspect of service lifecycle management. It is one that reflects the need to address multiple aspects of service quality across multiple SOA service implementations.

To address the needs of the quality management market, IBM Rational has produced and delivered on a quality management blueprint (Figure 10-3).

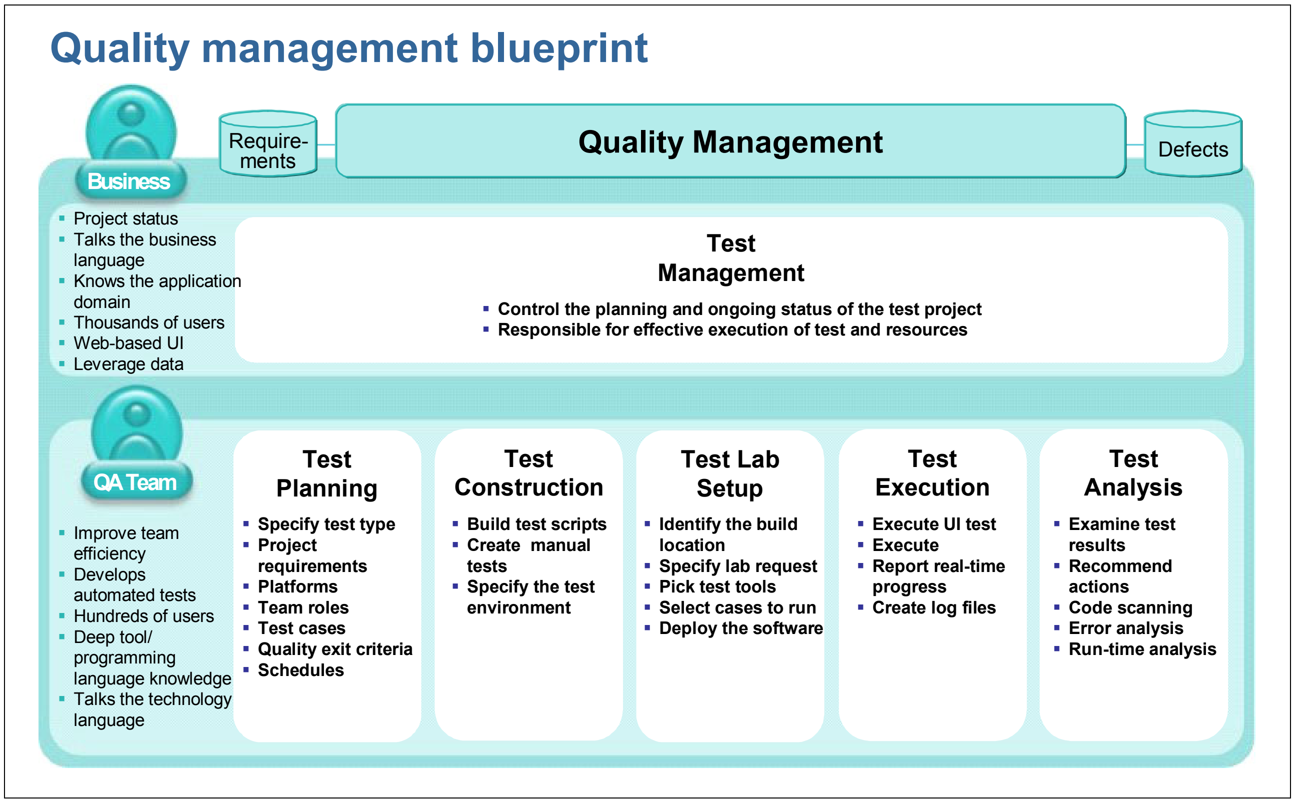


Figure 10-3 IBM Rational software - Quality management blueprint

Test management

Quality management starts with test management. Test management coordinates the disciplines of test planning, test construction, test lab setup, test execution, and test analysis. A test management system provides a repository for managing and organizing the test effort. By placing all assets in the test management system, the testing team has a clearer indicator of their progress and a common repository for managing and sharing test cases and scripts. Requirements are linked to test plans and test cases, test cases are managed in the context of a test plan, and test scripts are placed under version control. Test execution is tracked for the project, and defects are reported. In addition, tests can be created and reused across multiple projects, and reports become much easier to create and manage.

Test planning

Before the shift in quality, the emphasis was on delivering test cases. Now, the focus has shifted to targeting *which* test cases to run to meet business expectations. In quality management, the test effort revolves around the test plan.

Instead of writing a test plan in a document or spreadsheet, the test plan is managed by the quality management system. Test management organizes the test plans, cases, scripts, and execution results, thus giving the entire team clarity into the plan and progress of the testing effort.

During test planning the team is established and roles are defined. The project requirements and exit criteria are identified. Test types, such as functional or performance tests, are planned and scheduled, and the platforms are determined. Test plans also align test cases with the business requirements.

In addition, the test plan provides the team with something to measure against, because it provides insight into what needs to be done and how the team is progressing against the plan. The test plan states what the business expects, communicates how the team plans to meet them, and helps teams to determine when they are done. In the new paradigm, the team is done when the exit criteria for the test plan has been met.

The test plan can range from a simple statement or grow into a comprehensive testing strategy. Every project is different, and thus, the test plans from project to project differ to adapt to the needs of the team. There are fundamental components to a test plan that are worth considering for every software development project. The components, as illustrated in Figure 10-4, are test cases, builds, reports, strategy, quality process, requirements, test schedule, and test Environment. Each project can use as few or as many as makes sense for the situation.

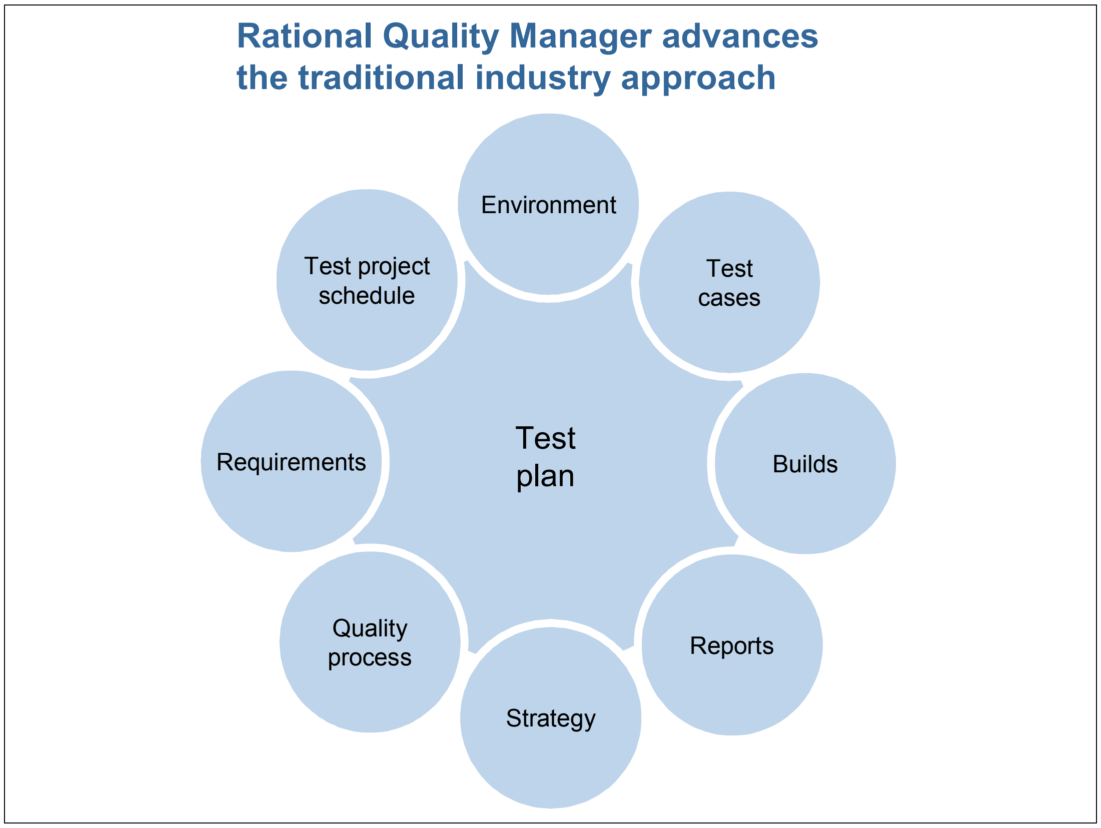


Figure 10-4 The test plan at the heart and soul of quality management

The test plan contains the following components:

- Requirements

  Requirements are added to the test plan to help the test team focus their efforts and understand exactly what is expected of the solution. By bringing the requirements into the test plan, the test team moves closer to working with the business to ensure that the application meets the business needs. Testers can review each requirement and build the test cases for each of the requirements. This gives the team insight into their requirements coverage by having direct links between requirements and test cases.

- Test schedules

  The test schedule gives clarity into what is expected when. Clearly you cannot test everything all at once. Therefore, the test schedule plans the testing over time.

- Test environment

  The test environment defines what systems and software are needed to test the software application. By defining the environments early, the test team can determine if new hardware or software is needed. Having an inventory of the lab assets and how they are configured is crucial to the success of the test team. The ability to automatically discover the current configuration of a lab resource streamlines the team's ability to locate a server and deploy a build. As part of the test plan, servers can be reserved by the team. When reserved, the team can configure machines with the appropriate software stack while waiting for a build from the development team. Additionally, consideration is given to the environments that will be used for integration, system, performance and user acceptance testing.

- Test cases

  Test cases are written to verify that the requirements have been implemented as expected. By linking test cases back to requirements, the test team has much greater visibility into the testing progress. For example, the number of test cases per requirement and across all requirements provides an indicator to the size of the effort. As test scripts are written, the team has an indicator of progress against the plan. Finally, as tests are executed, the team has insight into the testing progress and the quality of the solution.

- Builds

  Test execution cannot happen without a build. Knowing when the builds are available and what has changed in the build helps the test team to decide when to deploy a new build into the lab. Teams that run build verification tests also have insight into the quality of the build before spending precious time deploying it into the lab. Additionally, linking test results and defects with a specific build helps teams to reduce the time for problem resolution. When a defect is tied to a specific build, the developer can more easily recreate the defect as reported by the tester.

- Reports

  Reports are generated against the plan. Without a plan, the reports that a team can produce are limited. However, with a plan, the reports become much clearer. The team can now generate reports that tell them the following information and more:

  - How many requirements have test cases
  - How many requirements have been tested
  - How many test cases have been implemented
  - How many test cases have been run

  Additionally, trend reports can be generated to gain insight into the reality of reaching the plan. For example, are the number of requirements being added to the project trending up or down? Understanding the direction that the defect rate is trending provides insight into the overall quality.

- Strategy

  Strategy gives insight into the scope and approach to the testing effort. The team decides items such as how much of a test plan will they develop, the level of testing that will be executed (JUnit, build verification testing, system verification testing), how many test environments to use, the process for advancing a build through the environments of integration, system, user acceptance test, and so forth. The strategy defines how the team will ensure that the solution aligns with the business objectives. The strategy creates the definition of when quality is achieved in terms of the business objectives.

- Quality process

  The quality process brings clarity to the process that is used for testing. The teams agree on the process for defect reporting, whether test plan and test cases will be reviewed and approved by senior members of the team, and so forth.

Each team and each project might take a different approach to applying test plans. Some examples include creating a test plan for the entire release, for each iteration, for each test type (functional, performance, and so on), for each test environment such as integration, system, and user acceptance test. Agile teams can create test plans that align with the story for the iteration, and the definition of the story affects the test plan. A test plan can even include developer unit testing and build verification testing.

It is not expected that all of these components come into play on every project. Rather, consideration of each of these areas helps a team to determine their strategy and approach to each project.

Test construction

During construction, the test cases are defined and developed. The work that is completed in the Construction phase comes from the test plan. During test planning, test cases are identified. During the Construction phase, the test cases are developed.

All tests are managed by the quality management system. Manual tests are core to the system rather than written in separate documents and stored on a file system. Automated tests can be created by using any tool, and the resultant scripts are managed by the system.

A common challenge for testers is to build a test matrix to determine what tests to run against a series of test environments. During test planning, the test environments are identified. During construction, these environments are used to define test executions. The quality management system should help to simplify this task by using the environments that are defined in the test plan to guide the tester in choosing the test configurations, where each choice is stored in the system as work item for execution. These execution work items indicate which tests need to be run and serve as an indicator of progress.

As tests are used and refined, all changes go back into the system so that all team members benefit from the updated tests. Tests can be shared and reused across multiple projects, thus reducing redundancy and improving a team’s performance.

Because the test cases are managed by the quality management system, and associated with a test plan, the team can measure progress against the plan. For example, the test lead can ask the following questions:

- Do I have test cases for all of the requirements or for all of the test types?
- How many test scripts are written and how many are left to write for this iteration?
- How many execution items exist for this test case?
- Do all test cases have test executions?

These types of questions give the test lead additional information for measuring the team's progress. By planning the test cases, there is now a means to measure how many are written and how many need to be executed. In doing so, the test team has visibility into the size of the testing effort.

Test lab setup

Lab reservation and management systems contribute to managing quality by helping the test team to identify properly configured servers that are available for use. By managing the servers, test teams can reduce conflicts or accidental changes to a system that is in use. Teams can also identify servers that are either properly configured or close to the configuration that is needed for testing, thus reducing the turnaround time in preparing the servers for testing. By having insight and visibility into the test lab, test teams can intelligently plan for and choose properly configured servers.

Test lab setup becomes more streamlined when server configurations are understood as part of the test plan. By having a catalog of the servers in the lab, the test lead can ask: "Do I have the equipment and software needed?" When the lead discovers the current equipment, the lead can follow up with these questions: "Is it available when I need it?" and "Does it have the software I need installed on it?" The lead can interrogate the current test lab to determine if new hardware or software must be ordered.

When the test configurations are understood, the test lab preparation can begin. The lab servers are configured, and test tools are deployed. When an integration build is available, it is deployed to the servers for test execution to begin. Over time, teams gain insight into server utilization with an understanding of which servers are being reserved and used, and which are left idle.

Test execution

After a build is deployed on the test servers, test execution can be begin. But how does the tester know what tests to run? This information is stored in the test plan. The tester can reference the plan and determine which tests are supposed to be run. User interface tests, functional tests, performance tests, and security and compliance scans are run. However each of these test types are executed with a specific test execution engine. The quality management platform coordinates the test execution effort and provides the means to plug-in a wide variety of test execution engines.

For example, functional tests are a common source for automated testing. By recording a series of events against a software solution, a tester can save time and ensure that the same test is run against every build or iteration. Functional testing ensures the system functions as expected.

In a similar manner, performance tests involve running a series of tests while simulating a specified number of users who are using the system. Performance tests can be designed to simulate a wide range of user activity over a period time. Such testing helps the team to identify performance bottlenecks prior to going to production.

In addition, SOAs introduce a new level of complexity for testers because of the nature of composite applications. Composite applications are composed of many services that are often developed and deployed independently by separate development teams on different schedules, which creates unique challenges in ensuring a high level of quality throughout the development cycle.

Security and compliance tools are used to scan a running application by using a predefined set of criteria. *Security scans* test against a known set of tactics that are used by hackers, while *compliance scans* test compliance to a specified regulation. These scans work in a manner similar to virus scanning applications on a home PC. The definition files are created and maintained by the vendor. Organizations that use the scanning software keep their definition files up to date and run scans against their applications. The scan identifies potential problems thus enabling the team to respond during the testing phase. By using this type of automation, an enterprise can consistently identify potential threats and close them prior to going to production.

With test execution built into the quality management system, groups of tests can be selected and run against a system without human intervention. The testers monitor the result of the automated tests as they are run, but are free to develop additional test scripts while the automated tests run. For example, in some organizations, performance tests have grown in sophistication where they run for several days with thousands of virtual users fully stress the application.

With integrations to test execution engines, the results of the test effort are captured in the quality management platform, enabling progress to be reported against the test plan. Teams have insight into which tests have been executed and with what result.

Test analysis

Continuous improvement applies to every discipline and every step in the life cycle. Test analysis implies that teams seek to improve their test strategy, test plans, test cases, and test execution.

The quality management must provide a full set of reports to analyze the test team's progress and performance. The test plan and progress are evaluated, and action is recommended to improve quality. Test teams can begin to analyze the quality by asking questions such as: "Which test cases am I using the most often?" and "Which are not being used?" The heavily used test cases are candidates for automation. The test cases that are not used might expose a gap in the test effort or might indicate excess that is no longer needed. By analyzing the answers to these questions, test leads can take the appropriate action to improve their test coverage.

Many times test teams find problems that should have been found much earlier in the cycle. Test analysis encourages teams to continually seek improvement to avoid similar problems in future releases.

10.2 A reference scenario for managing quality

In this section, we provide an overview of the steps that are taken by the testing team to manage the quality of the weekly integration build (Figure 10-5 on page 398). In Chapter 11, "Rational Quality Manager for managing quality" on page 409, this workflow is demonstrated in the Rational products. This scenario continues to build off the previous acts in this book.

In 4.2, "A reference scenario for responding to a change request" on page 105, Tammy was notified of a changed requirement. She contributed to the sizing of the test effort for the requirement and then updated her test plan when the team agreed to absorb the change in their current iteration. She also reserved a group of servers and created a request to prepare the servers with her test configuration.

In Chapter 6, "An agile team implements a change" on page 213, the development team implemented the change. The implementation included a developer running unit tests prior to delivery and a team build that also conducted build verification tests.

In Chapter 8, “The release engineer conducts the integration build” on page 315, the release engineer, Rebecca, conducted the integration build, which brought together changes from multiple teams including the team that is described in Part C, “Act 2: Collaborative development” on page 211. The build is automated and includes source code analysis and build verification testing. The build is staged and published for the test team.

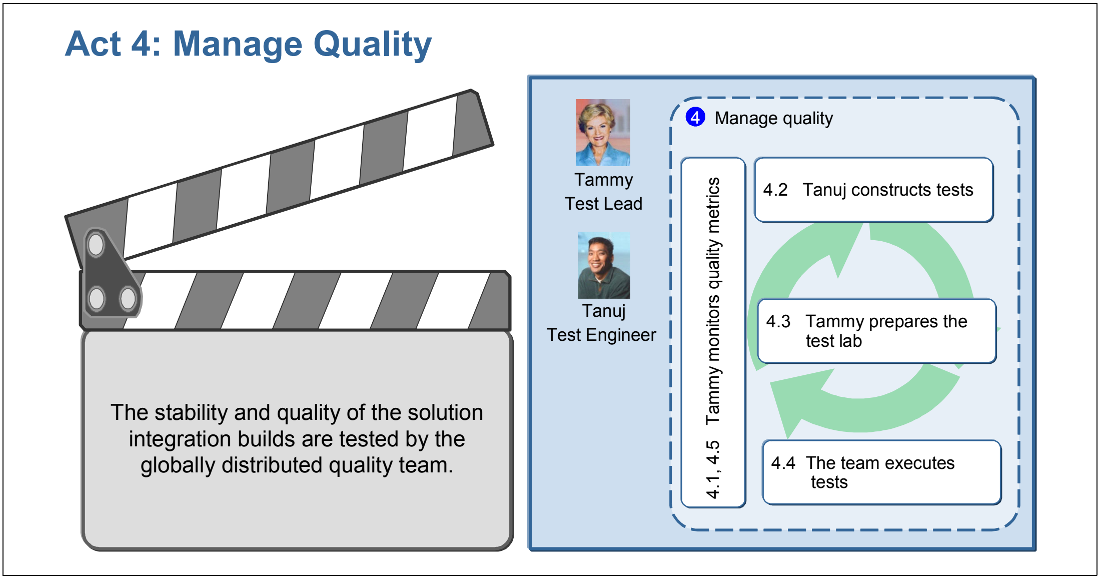

Act 4: Manage Quality
The stability and quality of the solution integration builds are tested by the globally distributed quality team.
Tammy
Test Lead
Tanuj
Test Engineer
4 Manage quality
4.1, 4.5 Tammy monitors quality metrics
4.2 Tanuj constructs tests
4.3 Tammy prepares the test lab
4.4 The team executes tests

Figure 10-5 Act 4 illustrating how the team tests the solution

The quality management story continues with the test team. After her test plan was updated, Tammy assigned a test case to Tanuj. While the development team is implementing the change, Tanuj is able to construct the tests. However, he needs a running application to execute his tests.

This scenario begins immediately after Tammy updated the plan in Part B, “Act 1: Responding to a change request” on page 77, and then pauses while the development team implements the change. It resumes again with Tammy when the weekly integration build is announced.

10.2.1 The actors

This scenario includes the several key actors as described in this section.

Tammy is the name of the test manager. She has a global quality team distributed over multiple sites. Her team is responsible for conducting solution testing, which includes functional, performance, and security testing at the solution level. The testing does not include JUnit or component level testing, which is the responsibility of each of the development teams. Tammy’s team conducts solution testing as part of the iteration. They take a new solution build each week, thus providing an early feedback loop to the development team when there are defects at the solution level. She also provides a globally distributed test environment for the project. She is the gatekeeper and owns staging.

Tanuj is the name of the tester. He is responsible for creating the test cases and test scripts. He also executes the tests and analyzes the results. He logs defects when needed. Tanuj and his teammates employ a full battery of tests including manual, functional, performance, and security tests.

Diedrie is the developer who implemented the change request that Tanuj is testing. She takes great pride in her work and is astute in fixing defects. In this part of the scenario, Diedrie and Tanuj collaborate on resolving a defect in Diedrie's solution.

10.2.2 The workflow

In Part C, "Act 2: Collaborative development" on page 211, of the Application Lifecycle Management (ALM) scenario, we capture the scenes and steps of a single request that is being tested by Tammy and her team. The testing team links to the requirements, constructs the tests, prepares the test lab, and executes the test cases as defined by the plan. Tammy monitors the test team's progress. At the end of the act, the solution is fully tested.

The steps shown in Figure 10-6 are performed by the globally distributed testing team in Act 4: Managing quality.

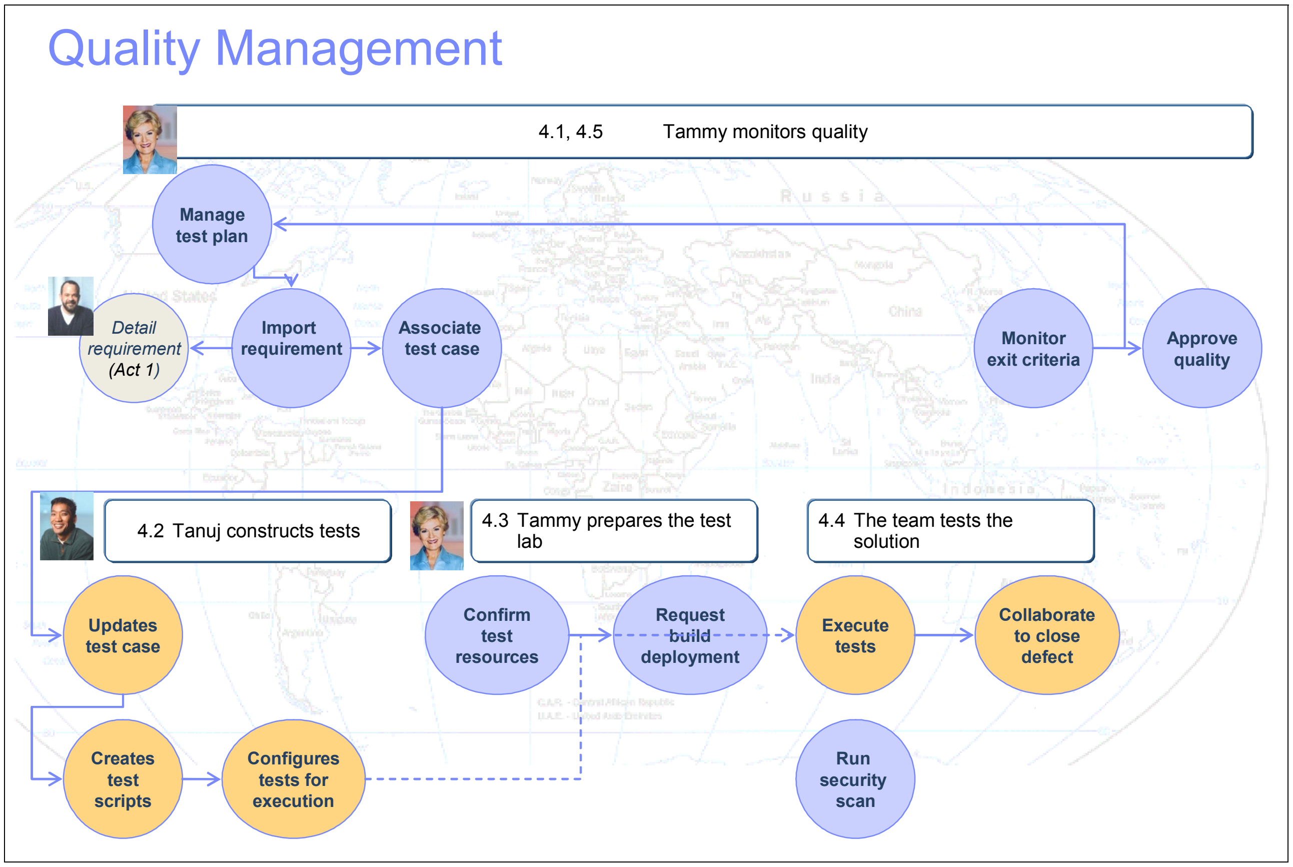


Figure 10-6 The flow of steps for the Act 4: Manage quality

This act includes the following scenes:

- Tammy monitors quality.
  - At the end of Act 1, Bob detailed the requirements. In this act, Tammy creates a link to the requirements in her test plan.
  - She associates the test case and the requirement.
- Tanuj constructs tests.
  - Tanuj updates the test case that Tammy created.
  - He creates a manual test script for the test case.
  - Using the environments defined in the test plan, he determines the test configurations that are needed to fully test the requirement and creates execution work items for each test to execute.
- Tammy prepares the test lab.
  - Tammy confirms that she has reserved the needed test configurations.
  - She submits a request to have the integration build deployed to the test lab.
- The team tests the solution.
  - Tanuj executes his manual tests.
  - Tammy runs a security scan.
  - They both evaluate their results.
- Tammy monitors quality.
  - Tammy evaluates the exit criteria.
  - Tammy confirms testing is complete and the solution has met its quality goals.

10.2.3 Tammy monitors quality

Synopsis: When Tammy updated the test plan with the new request from Bob, he had not yet detailed the requirements. The next day Bob notified Tammy that the requirements were complete. In this scene, Tammy links the requirements to her test plan. This scene begins immediately after the completion of Act 1 and occurs in parallel with Act 2, where Marco and Diedrie design and implement the request.

The workflow in this scene captures how Tammy completes the following tasks:

- Links to requirements in Rational RequisitePro
- Associates the requirement with a test case

10.2.4 Tanuj constructs tests

Synopsis: When Tammy updated the test plan with the new request from Bob, Tanuj began constructing tests, while Diedrie and Marco were implementing the change. This scene starts the day after Tammy updated the test plan, and Bob detailed the requirements in Act 1.

In this scene, Tanuj completes the following tasks:

- Updates the test case with additional detail
- Writes a manual test script that is associated with the test case
- Configures tests for execution

10.2.5 Tammy prepares the test lab

Synopsis: Rebecca notifies the team that a new integration build is available. Tammy inspects the build and decides to deploy it to the test lab.

The workflow in this scene captures how Tammy completes the following tasks:

- Confirms that the test resources are reserved
- Inspects the build
- Deploys the build

10.2.6 The team executes the tests

Synopsis: Tanuj and the team learn from Tammy that the new integration build has been deployed. He executes the test scripts, the results are logged, and the test status is updated.

The workflow in this scene captures how Tanuj and Tammy complete the following tasks:

- Execute tests
- Run a security scan

10.2.7 Tammy monitors quality

Synopsis: One week later, Tammy reviews the test plan and evaluates the exit criteria. She uses the exit criteria that is established in the test plan to measure their progress and make adjustments. The usual cycle of defect submission and resolution occurs until the test exit criteria is met and is complete

On the final day of the iteration, Tammy confirms that the solution meets the exit criteria. She informs Patricia that the quality has met the expectations.

The workflow in this scene captures how Tammy and her team perform the following tasks:

- Monitor quality
- Approve quality

10.3 Considerations in quality management

As discussed, quality management spans the entire life cycle. The scenario provided for this IBM Redbooks publication provides a view into the testing discipline, but does not cover every aspect. Additional considerations for quality management are discussed in this section.

10.3.1 Automated testing

The reference scenario showed how Tammy's team conducted manual functional and automated security scanning as part an iteration. Automated test tools can also be used with Rational Quality Manager, which has an execution engine that allows for the execution of automated tests from external test tools. Adapters are provided for Rational Functional Tester and Rational Performance Tester.

Automated functional testing

Functional testing of a system ensures that the system behaves as expected according to the requirements. Some functional tests are performed manually, as described in the reference scenario. Other functional tests can be automated by recording the series of steps and saving the script. The script can be run as needed to confirm that the system behaves as expected. The automated test tool executes the test and returns the results to the user.

Rational Quality Manager can be used to manage and execute the scripts from automated functional testing tools. An adapter is provided to execute the automated test scripts of Rational Functional Tester. In the reference scenario, Tammy runs an automated security scan. The execution of an automated functional test occurs in a similar manner.

See the Rational Functional Testing page at the following address for a complete list of functional testing products, including Rational Functional Tester and Rational Tester for SOA Quality:

http://www-306.ibm.com/software/rational/offerings/quality/functional.html

Performance testing

Performance testing is another critical type of testing to take into consideration. There are multiple approaches to performance testing. In some organizations, the development team (developers and testers) perform the performance tests. In other organizations, performance testing occurs as a critical test prior to production deployment.

Performance tests are established to determine how well the system performs under various circumstances such as having a large number of users perform the same action on the system at the same time or over long periods of time. A series of tests are created that mimic user behavior, involving common user paths in a lab that either mimics or is scaled-down version of the production environment. The purpose is to ensure that the solution tested against the same hardware and software configuration as the final production environment. The testing includes a set of virtual users that are configured to run through a series of tests. In many cases, these users are defined to mimic users from multiple geographies by using the system.

By running performance tests prior to going to production, the team reduces the chances of down time due to customer load. It also gives the team the opportunity to optimize performance and identify potential bottlenecks before the users do.

Rational Quality Manager can be used to manage and execute the scripts from performance testing tools. An adapter is provided to execute Rational Performance Tester scripts. In the reference scenario, Tammy runs an automated security scan. The execution of a performance test occurs in a similar manner.

See the Rational Performance Testing page for a complete list of performance testing products, including Rational Performance Tester and its extensions for SAP, Siebel, SIP, Citrix Presentation Server, and SOA Quality:

http://www-306.ibm.com/software/rational/offerings/quality/performance.html

10.3.2 Automated scanning

Automated scans provide a unique value to test teams by scanning software using a predefined set of rules.

Security and compliance scans

Web site security and compliance should be a top priority for organizations that are intent on protecting sensitive company, customer, and employee data, on meeting regulatory and corporate compliance requirements, and on defending against the high cost of a data breach. Web sites and their applications are high focus targets for hackers because they provide a direct route to corporate or personal data regardless of network security implementations.

In many cases organizations cannot keep up with the details and changes to regulatory compliance, new threats, and security mandates. For example, Payment Card Industry Data Security Standards (PCI DSS) Subsection 6.6 became a mandatory requirement on 30 June 2008. PCI DSS requires that companies ensure that all Web-facing applications are protected against known attacks by applying either of the following methods:

- Having all custom application code reviewed for common vulnerabilities by an organization that specializes in application security
- Installing an application layer firewall in front of Web-facing applications

Using scanning technology frees IT teams from learning and implementing tests for every detail of these regulations by relying on experts in security or compliance provide a scanning service.

IBM Rational AppScan and Rational Policy Tester are scanning and testing solutions that automate application and content analysis to help organizations identify vulnerabilities, assess compliance requirements, and improve the accuracy and reliability of online systems.

Security and compliance tools are used to scan a running application by using a predefined set of criteria. *Security scans* test against a known set of tactics that are used by hackers, while *compliance scans* test compliance to a specified regulation. These scans work in a manner similar to virus scanning applications on a home PC. The definition files are created and maintained by the vendor. Organizations that use the scanning software keep their definition files up to date and run scans against their applications. The scan identifies potential problems, thus enabling the team to respond during the testing phase. By leveraging this type of automation, an enterprise can consistently identify potential threats and close them prior to going to production.

Rational offers Web application security solutions through the Rational AppScan product family for all stages of development and for all types of testers. The Rational AppScan family includes the following products:

- Rational AppScan Standard Edition for IT Security, auditors, and penetration testers
- Rational AppScan Reporting Console, which provides centralized reporting on Web application vulnerability data
- Rational AppScan Enterprise Edition, which is a Web-based, multi-user Web application vulnerability testing and reporting solution that is used to scale security testing across the enterprise
- Rational AppScan Tester Edition, which integrates Web application security testing into the current QA environment

Rational Web site compliance solutions include IBM Rational Policy Tester, which automates Web site privacy, quality, and accessibility reviews to help identify issues impacting compliance and site effectiveness. The Policy Tester family includes the following products:

- Rational Policy Tester Accessibility Edition, which helps ensure the accessibility of Web sites to all users, including those who access sites by using assistive devices
- Rational Policy Tester Privacy Edition, which uncovers and reports online privacy oversights that might expose the organization to undue risk
- Rational Policy Tester Quality Edition, which automates the scanning, analysis, and reporting for online privacy, quality, and accessibility compliance

Static analysis

Scanning source code is an important aspect of an overall quality management program. Developers can run source analysis during development, release engineers can incorporate it into the build, or a final scan might take place prior to release.

The Rational Software Analyzer product is designed to analyze code and report any areas where a set of selected rules have been broken. These rules can take almost any form including basic code review, code complexity, or detection of common code patterns and anti-patterns. Rational Software Analyzer provides the following benefits:

- Software development teams can consistently catch potential software defects in real time earlier in the software development life cycle.
- Project team leads can more effectively manage governance and compliance IT objectives through a customizable reporting framework.
- It automates code reviews and incorporates static analysis into the existing software prebuild process across the life cycle, empowering teams to deliver more value.
- With a common static analysis framework, you can create a customizable and consistent workflow for all forms of static analysis rules within a single session.

Rational Software Analyzer ships in two editions: one for the developer and one for the enterprise:

- Rational Software Analyzer Developer Edition is a standalone Eclipse-based tool that targets developers. As the develop code, developers can quickly run the analyzer tools to detect early problems in their code before it is delivered to the code management system.
- The Rational Software Analyzer Enterprise Edition runs as a central service and can be integrated with Rational Build Forge or any other build management system to perform regular analysis of the source code in a project. The tool plays a part in the ALM and governance processes within the enterprise. The Enterprise Edition can generate refined analysis reports for both developers and others in the enterprise.

10.3.3 Approaches to iterations

In some organizations, it is common to conduct integration or system testing at the end of the project. The development team works for weeks before the solution is “thrown over the wall” for testing to begin. All too often, the development schedules take longer than expected, and as a result, the amount of time dedicated to testing is cut short. The problems with this approach have been well documented. The bottom line is that this practice typically leads to lower quality solutions and a lot of blame to determine the root cause of the failure. This section describes strategies for incorporating the test effort into development iterations.

Testing at the end of an iteration

Agile and Iterative development encourage a time-boxed deliverable, where a shorter testing time is better. Thirty-day iterations tend to be a common theme, while some teams strive for shorter iterations, and others use longer iterations. The solution team described in this scenario rallied around four-week iterations, while the smaller more agile team used 2x2 week iterations for each one of the solution team's iterations.

With a thirty-day deliverable, it makes sense to system test right after the iteration. When the development team completes iteration 1, the test team begins system testing. The development team continues on to iteration 2. As testers find and report defects, the developers can fix the defects in their current iteration. When that iteration is complete, the testers deploy the build, confirm the fixed defects from the previous iteration, and test the new functionality. To accommodate this offset model, the last iteration is dedicated to fixing the defects that are found in the previous iteration. This approach (Figure 10-7) dramatically shortens the amount of time between development and system test.

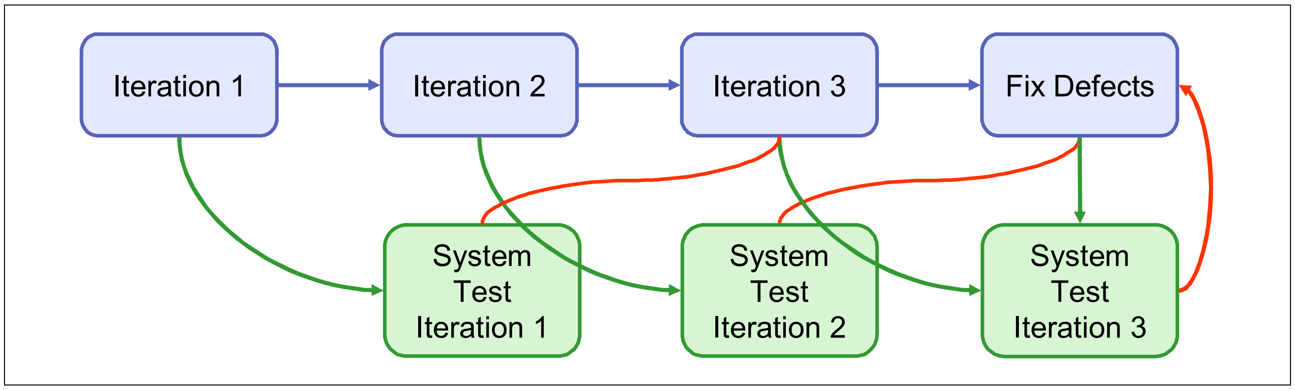


Figure 10-7 System testing after each iteration

This approach includes the following benefits among others:

- Developers gain insight into defects soon after developing the source code. The code is still fresh in their minds, and they can more easily get to the root cause.
- Testers and project managers gain insight into the solution health much sooner in the cycle. The feedback loop for the user perspective begins early in the project and can continue to be finessed as the team progresses through their iterations.
- The testing team is more engaged in the product development and improvement. Earlier access to the system allows the team to become more familiar with the features and expectations of the solution.
- With the testers aligning closer to the business, this helps to shorten the feedback loop from the business, thus giving the entire team a better gauge to measure themselves against.
- Questions or disputes about expected behavior can be resolved early in the cycle, and if need be, stakeholders can be contacted to gain insight into the resolution.

Any time that there is a change in approach, there is likely going to be a downside:

- By system testing after each iteration, the test team must also become more agile. The builds are produced more often and must be deployed to test servers. Test cases and test scripts must be ready in time for the iteration, and it is likely that they will be re-written many times as the team learns more about the system iteration after iteration.
- The test effort for the iteration must align with the stories and requirements implemented by the development team. The scope for each test effort becomes narrower and more

targeted. To accomplish this, the teams need to communicate often, and the test team must have insight into the development iteration plans and user stories.

- The test iteration lasts as long as the development iteration. After all, when the development iteration ends, there is a new build available for testing. And if testing of the previous iteration is not complete, it sits idle, causing a landslide effect for the rest of the project.
- The last iteration must focus on fixing defects, which few developers enjoy doing, but at least time is finally dedicated to the task.

Testing incorporated into the iteration

Agile teams can take this idea one step further by including system test as part of the iteration. In this approach, the integration build occurs on a weekly basis. Each week a new build is made available to the test team for testing. The cycle between finding and fixing defects is shortened even further, thus bringing the quality of the iteration to a higher level. The teams that use this model (Figure 10-8) can set exit criteria for the iteration, holding themselves accountable for a high quality iteration before moving on to the next one.

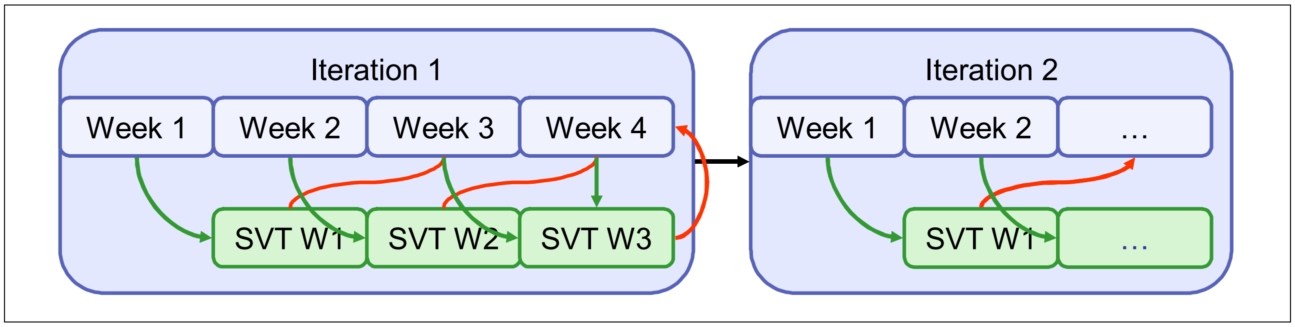


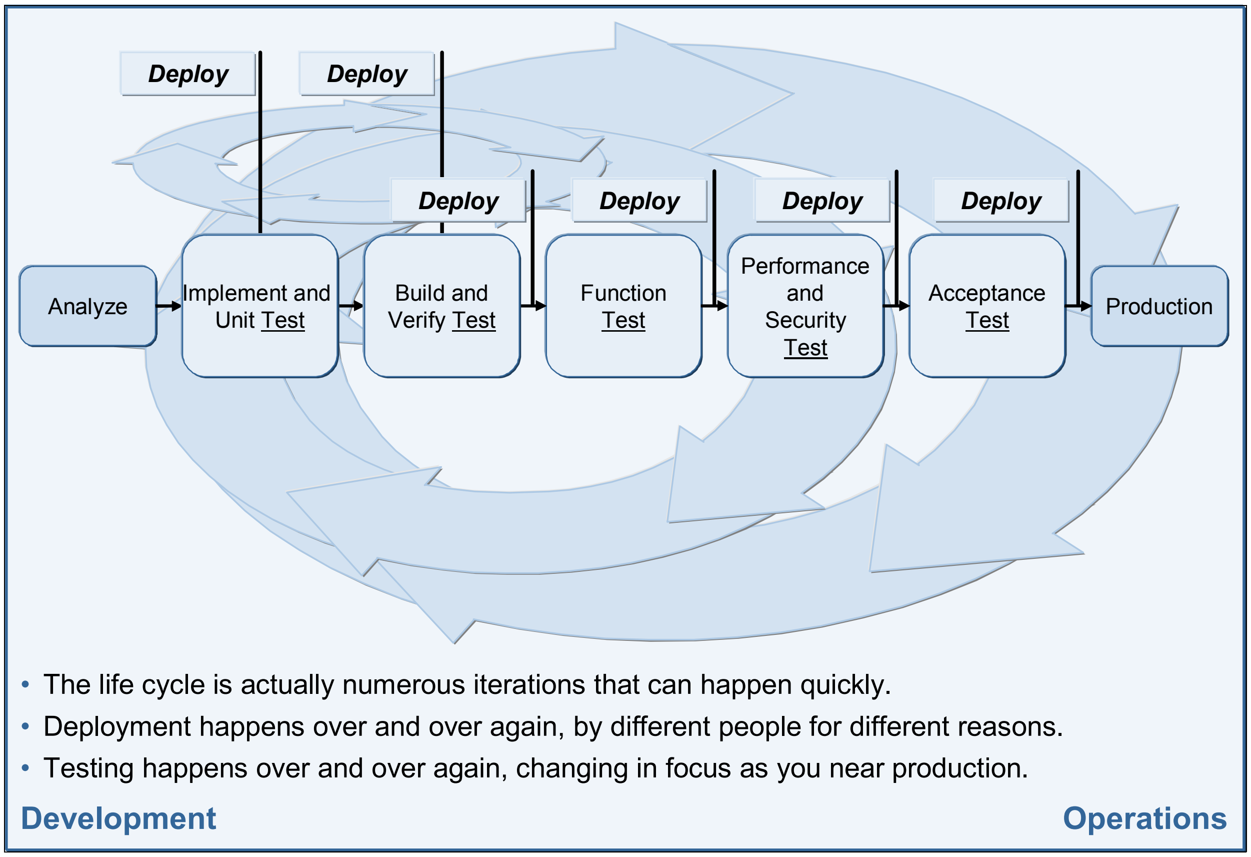
Figure 10-8 System testing incorporated into the iteration

As seen in Figure 10-8, the same offset between development and system testing occurs. However this time, the cycle is much shorter and integrated into the iteration. After the first week's integration build is ready, the test team begins testing. Defects are reported and fixed in the same week. The next week's integration build contain the fixes along with new features. The last week of the iteration is dedicated to defect fixing.

This approach requires more discipline and communication between the teams and fundamentally requires that the imaginary wall between development and test is removed. All members of the team are expected to respond to changes as they occur. The test team is tightly aligned with the development team, and both teams are aligned with the business needs. Together they move through each iteration developing, testing and confirming user stories intent on delivering what the business needs. The pros and cons to this approach are similar to those in "Testing at the end of an iteration" on page 405, but somewhat amplified. The defect cycle is extremely short, the test scope is extremely focused, and all members of the team produce assets that are likely to be revised and improved upon in the next iteration.

10.3.4 Many test phases on the path to production

Testing is no longer a discipline that occurs at the end of the development cycle. Testing is a vital part of every iteration plan, as shown in the reference scenario. While the reference scenario focuses on a single iteration in the Construction phase, the testing of the solution continues through multiple testing phases before the solution is approved for production (Figure 10-9).



- The life cycle is actually numerous iterations that can happen quickly.
- Deployment happens over and over again, by different people for different reasons.
- Testing happens over and over again, changing in focus as you near production.

Figure 10-9 Managing quality at all levels of deployment

As illustrated in Figure 10-9, testing is completed by developers, by release engineering, and then by more formal test phases of function, performance, and acceptance testing. As an integration build becomes more stable, it is promoted to the next level of testing, until it is accepted and approved for production.

Every organization differs in the names and definitions of these test phases. Figure 10-10 on page 408 provides a representative set of test environments and their definition, which are explained as follows:

- Unit testing is lowest level of granularity and typically occurs in a developers' sandbox or during build verification testing.
- Function testing confirms that components of the solution function as expected. A component must pass the function test before being promoted to an integration test.
- Integration testing combines components into groups, where the cross-component function is confirmed to work as expected. When the group of components passes the integration test, it is promoted to a system test, performance test, or both.

- System testing, performance testing, and security scans are sometimes performed in parallel. At this point, the application is reasonably stable and ready for more intensive testing. The tests and server configuration are the most sophisticated, and where possible, the server configuration either mimics or is a scaled-down version of the production configuration:
  - A system test exercises the system the way a user might.
  - A performance test analyzes the system's ability to respond to increased user load and transaction volume over an extended period of time. Performance testing is used to identify possible bottlenecks and to fine-tune the performance.
  - Security and compliance scans are run against the system during this time to seek possible security holes and incomplete regulatory compliance.
  - An acceptance test involves the stakeholders' acceptance of the solution. The acceptance criteria is reviewed and confirmed complete.

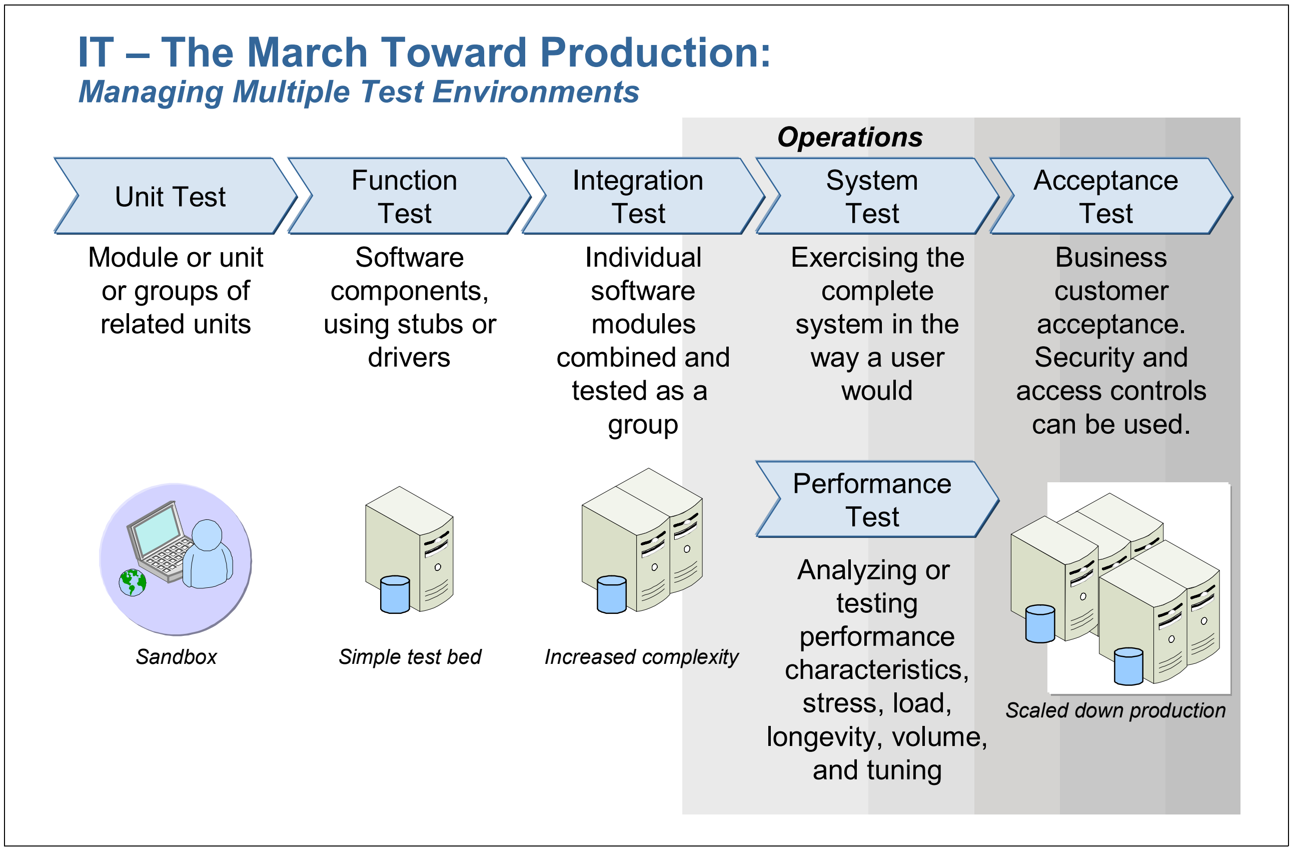

IT – The March Toward Production:
Managing Multiple Test Environments
Operations
Unit Test
Function Test
Integration Test
System Test
Acceptance Test
Module or unit or groups of related units
Software components, using stubs or drivers
Individual software modules combined and tested as a group
Exercising the complete system in the way a user would
Business customer acceptance. Security and access controls can be used.
Performance Test
Analyzing or testing performance characteristics, stress, load, longevity, volume, and tuning
Sandbox
Simple test bed
Increased complexity
Scaled down production

Figure 10-10 Test phases to ensure quality in production

11

Rational Quality Manager for managing quality

In this chapter, we provide detailed "how to" information for Act 4: Managing quality in the storyboard, where the test team manages the quality of the solution.

This chapter contains the following sections:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

11.1 Act 4: Managing quality

In Act 1, the team responded to a new request. In Act 2, the agile team delivered the change, and in Act 3, the integration build occurred. In this chapter, Act 4, we include a step-by-step discussion about how the test team manages and executes tests against the integration build using the storyboard illustrated in Figure 11-1.

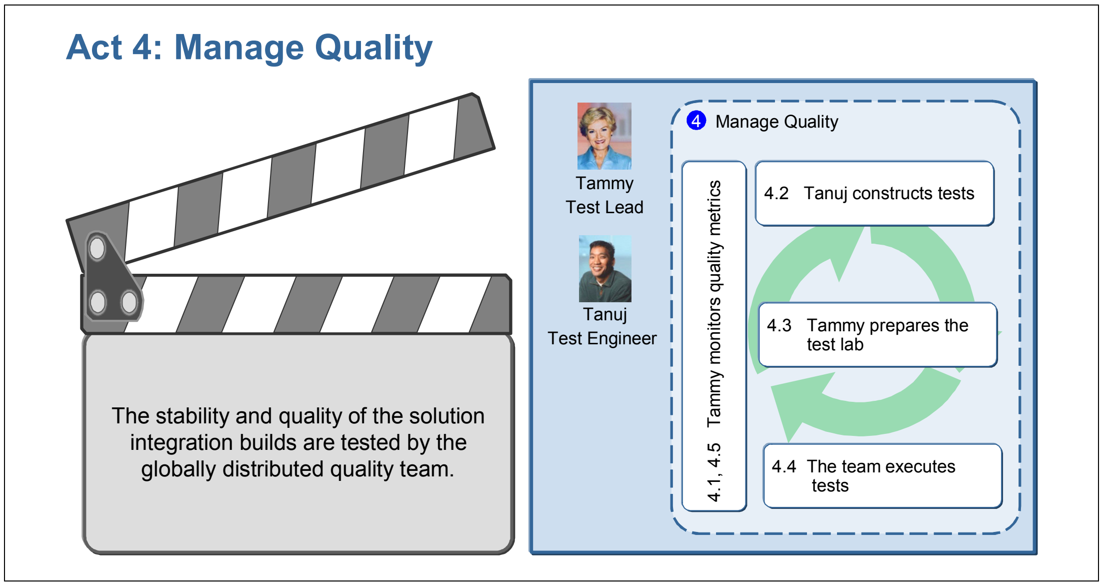

Act 4: Manage Quality
The stability and quality of the solution integration builds are tested by the globally distributed quality team.
Tammy
Test Lead
Tanuj
Test Engineer
4 Manage Quality
4.1, 4.5 Tammy monitors quality metrics
4.2 Tanuj constructs tests
4.3 Tammy prepares the test lab
4.4 The team executes tests

Figure 11-1 Act 4: The solution test team manages quality

The act has the following scenes:

- Tammy monitors quality.
- Tanuj constructs the tests.
- Tammy prepares the test lab.
- The team executes the tests.

The following Rational products are used in this act:

- Rational Quality Manager 8.0
- Rational Functional Tester 8.0
- Rational AppScan Tester Edition for Rational Quality Manager 5.6

11.2 Rational Quality Manager

Rational Quality Manager provides a test management system that puts the team in total control of their test efforts. Everything you need to do in test planning, construction, and execution can be completed in Rational Quality Manager. Decisions are managed in a database that enables the team to track against the plan.

One of the most critical questions the test team needs to answer is: “When are we done testing?” With Rational Quality Manager, teams are better equipped to answer that question by placing an emphasis on the test plan, which enables a team to track their progress against the plan. A team can do as much or as little as they need.

The Rational Quality Manager product provides the following features:

- A collaborative Web-based quality management solution
- A central repository for test planning, construction, deployment, and execution
- The ability to align the test effort with project requirements
- Quantifiable metrics for project tracking and decision making
- Keyword-driven manual test authoring and execution

Rational Quality Manager works for simple test management needs and yet is sophisticated enough to scale to larger teams that share test assets across multiple releases of software. Just about every item managed by the system can be assigned to a team member. A test plan can be broken down into sections and assigned to different owners. Test case and test script construction can be assigned and tracked to gauge the level of effort and progress in building the tests. Test execution can be assigned and tracked through the execution results. Test lab requests can be used for configuring test servers.

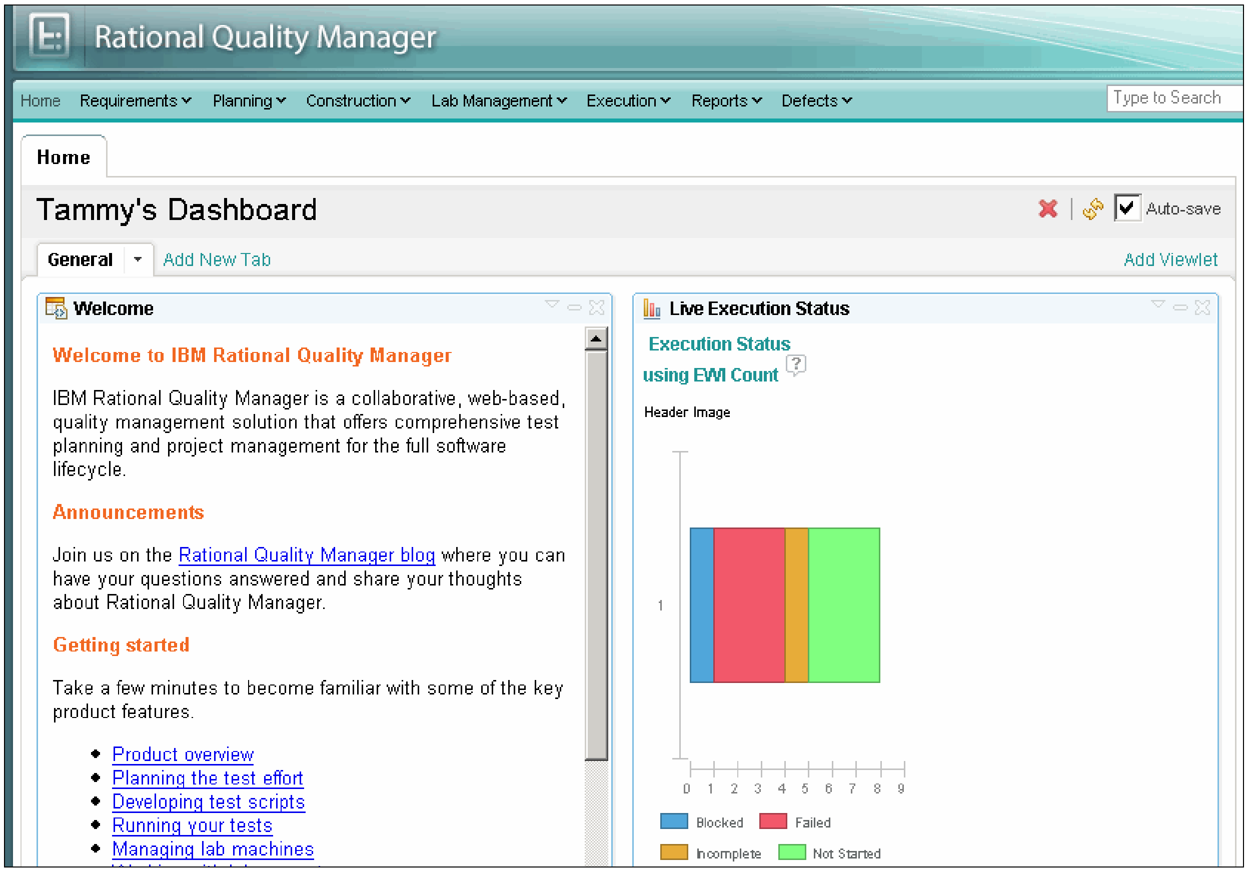
This assignment of work in all aspects of the test effort helps the team to ensure that all of the expected work is completed. It also gives them insight into their progress against the work effort.

Earlier we defined the software quality management blueprint as including test management, test planning, test construction, lab management, test execution, and analysis. Rational Quality Manager implements this blueprint.

Dashboard

Rational Quality Manager comes with a customizable dashboard on the home page. Dashboards can be created for each project managed.

Upon first use of the dashboard, a welcome viewlet (Figure 11-2 on page 412) instructs users on how to customize it. Additional viewlets are provided that can be changed to fit each user's needs.


Rational Quality Manager

Home | Requirements | Planning | Construction | Lab Management | Execution | Reports | Defects | Type to Search

Home

Tammy's Dashboard — Auto-save

General | Add New Tab | Add Viewlet

Welcome

Welcome to IBM Rational Quality Manager

IBM Rational Quality Manager is a collaborative, web-based, quality management solution that offers comprehensive test planning and project management for the full software lifecycle.

Announcements

Join us on the Rational Quality Manager blog where you can have your questions answered and share your thoughts about Rational Quality Manager.

Getting started

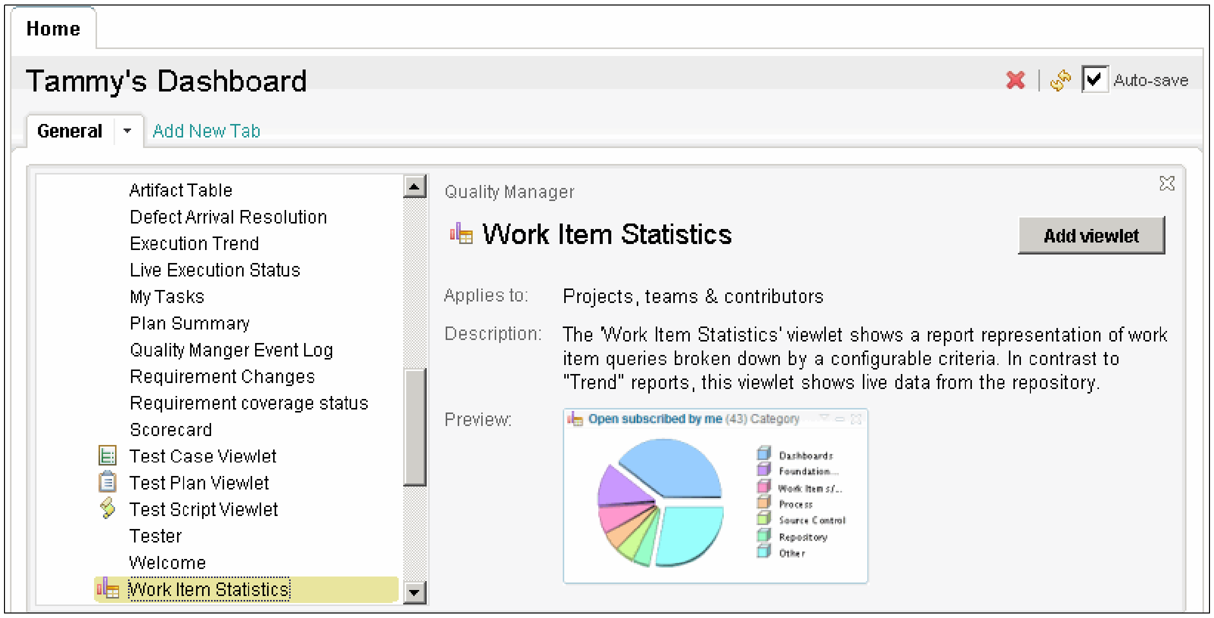
Take a few minutes to become familiar with some of the key product features.

- Product overview
- Planning the test effort
- Developing test scripts
- Running your tests
- Managing lab machines

Live Execution Status

Figure 11-2 Rational Quality Manager Dashboard 'Welcome' viewlet

Users can add or remove viewlets as needed by clicking the **Add Viewlet** link on the home page. As shown in Figure 11-3, a wide variety of viewlets are provided that are ready to use. It is simply a matter of selecting a viewlet and clicking the **Add viewlet** button.


Home

Tammy's Dashboard — Auto-save

General | Add New Tab

Artifact Table
Defect Arrival Resolution
Execution Trend
Live Execution Status
My Tasks
Plan Summary
Quality Manger Event Log
Requirement Changes
Requirement coverage status
Scorecard
Test Case Viewlet
Test Plan Viewlet
Test Script Viewlet
Tester
Welcome
Work Item Statistics

Quality Manager

Work Item Statistics — Add viewlet

Applies to: Projects, teams & contributors

Description: The 'Work Item Statistics' viewlet shows a report representation of work item queries broken down by a configurable criteria. In contrast to "Trend" reports, this viewlet shows live data from the repository.

Preview:

Figure 11-3 Adding a viewlet to the dashboard

After the viewlets are added to the dashboard, you simply drag them to a location on the page. Tabs can be added to the dashboard to organize viewlets.

Requirements

The testing effort is often linked directly to validating the software requirements. Therefore, Rational Quality Manager provides a means for linking to requirements from an external repository, such as IBM Rational RequisitePro, or for storing requirements directly in the database. The requirements menu (Figure 11-4) is used to import or locate existing requirements. These requirements can be linked to test plans or test cases.

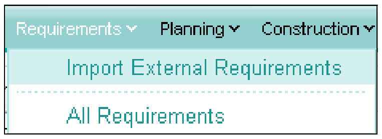


Figure 11-4 The Requirements menu

Planning

The Planning menu (Figure 11-5) is used to create and manage test plans. New test plans can be created, and existing plans can be modified from this menu. Additionally test plans can be imported into the system or exported for sharing with external stakeholders.

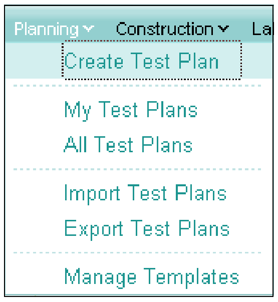
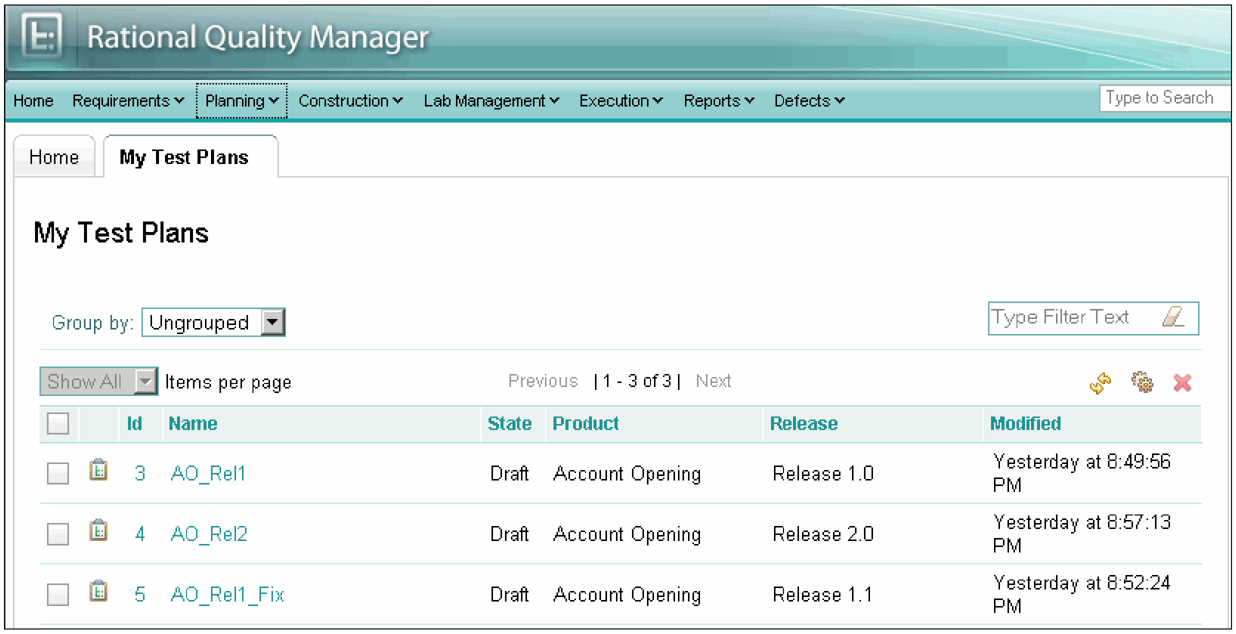


Figure 11-5 The Planning menu

A test plan is used to organize the test effort. Some teams create a single test plan for the entire project. Others create a test plan for each iteration or for each test environment such as integration test, system test, and performance test. Yet others use test plans for each test type such as functional, performance, and security tests. Figure 11-6 on page 414 shows a set of test plans where each test plan targeted for each release of the "Account Opening" solution.



My Test Plans

Group by: Ungrouped

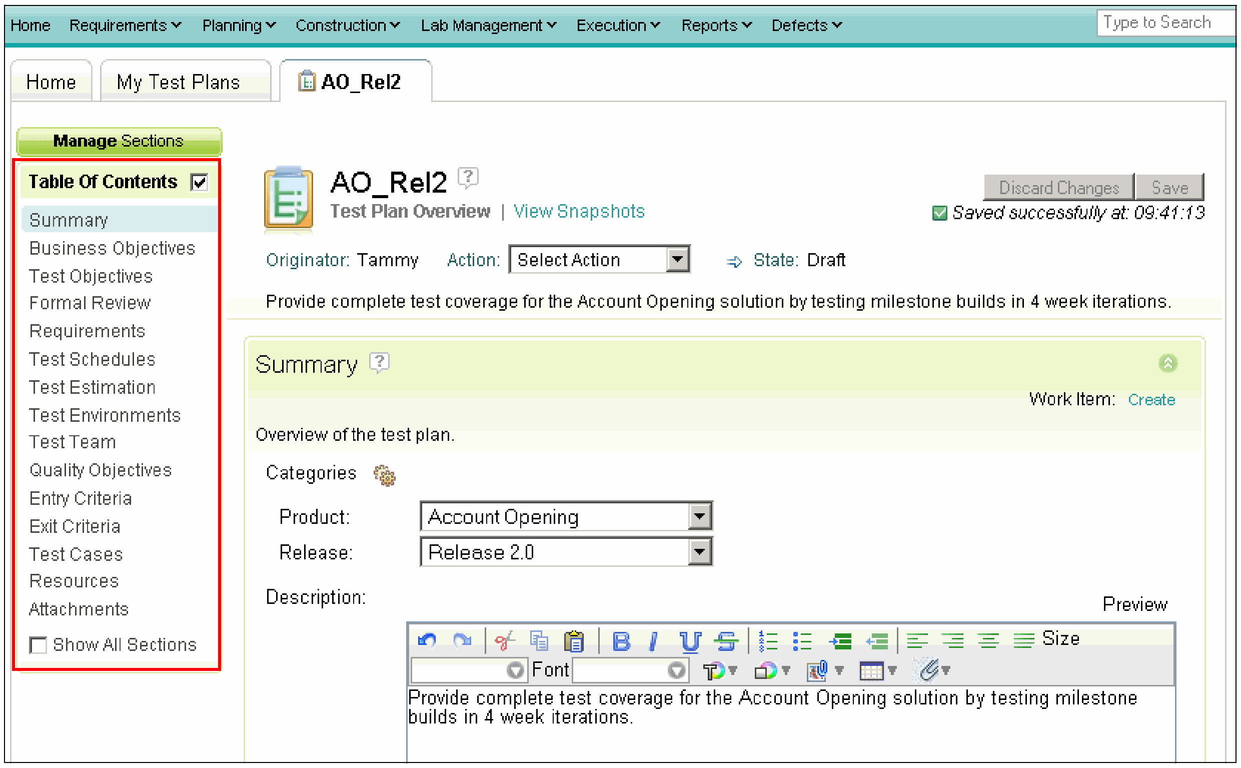
Type Filter Text

Show All Items per page Previous | 1 - 3 of 3 | Next

	Id	Name	State	Product	Release	Modified
	3	AO_Rel1	Draft	Account Opening	Release 1.0	Yesterday at 8:49:56 PM
	4	AO_Rel2	Draft	Account Opening	Release 2.0	Yesterday at 8:57:13 PM
	5	AO_Rel1_Fix	Draft	Account Opening	Release 1.1	Yesterday at 8:52:24 PM

Figure 11-6 A set of sample test plans

The test plan is divided into multiple sections as shown in the Table of Contents in Figure 11-7. A team can decide which of these sections to use for each testing effort. Additionally each section can be assigned to a different team member by creating a Work Item for that section. In the upcoming scenario, some, but not all of these sections are used by the Account Opening team.



AO_Rel2

Test Plan Overview | View Snapshots

Saved successfully at: 09:41:13

Originator: Tammy Action: Select Action State: Draft

Provide complete test coverage for the Account Opening solution by testing milestone builds in 4 week iterations.

Summary

Work Item: Create

Overview of the test plan.

Categories

Product: Account Opening

Release: Release 2.0

Description:

Provide complete test coverage for the Account Opening solution by testing milestone builds in 4 week iterations.

Figure 11-7 A test plan with sections listed in the Table of Contents

By using the test plan in Rational Quality Manager, you can perform the following tasks:

- Set up a review and approval process so that you can track completed and outstanding reviews.
- Import project requirements from external requirements management tools and associate these requirements with test cases. You can then run reports to track those requirements that are not yet covered. For teams that do not use external requirements management tools, you can add requirements directly to the test plan and still run the same reports.
- Save a read-only snapshot of a test plan or test case at any point in time. Later, you can make a copy of the snapshot and use it as the basis for a new test plan or test case.
- Attach existing documents, such as previous test plans, schedules, and other supporting material, by using the Attachments section.
- Copy portions of existing documents into test plan sections. Certain sections of the test plan include a rich text editor that you can use much as you might use any word processing document. If you have content in external documents that you want to reuse, copy the content into the appropriate test plan section.
- Create test cases and associate them with the test plan. Because test cases can also be associated with various kinds of test scripts, you can manage your test cases and test scripts within the test plan.
- List the various environments supported and tested by the test plan. You can add various platforms such as browsers, databases, operating systems, and other items. This list is then used to generate test configurations.
- Estimate the overall test planning and test execution effort.
- Define schedules for each test iteration.
- Define business objectives, test objectives, quality goals, and entrance and exit criteria.

The list of sections can be modified by clicking the **Manage Sections** button. Additionally Test Plan templates can be created to define the set of sections to use in the test plan. You can start with one of the templates and modify it as needed. If there are sections in the test plan that you do not use, you can remove them. If the section names do not match what you are accustomed to, you can change them to something more familiar. If there are missing sections that you want to add to the test plan, you can create your own test plan sections and add them to the template.

This flexibility makes the test plan suitable for both agile and formal test teams and for teams that perform different kinds of testing, such as functional regression testing, performance testing, system verification testing, globalization testing, and so on.

Construction

The Construction menu (Figure 11-8 on page 416) is used to work with test cases, scripts, suites, and data. New tests can be created, and existing tests can be located for modification. Test execution records can also be created from this menu. Additionally existing cases and scripts can be imported and managed by Rational Quality Manager. Each of these items, managed by the database, can be assigned to specific owners, tracked, and reused across multiple software releases.

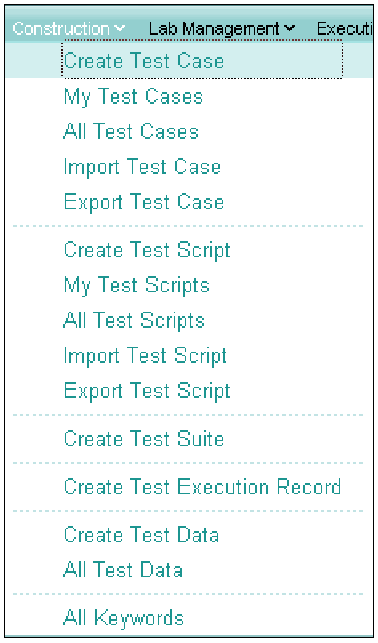
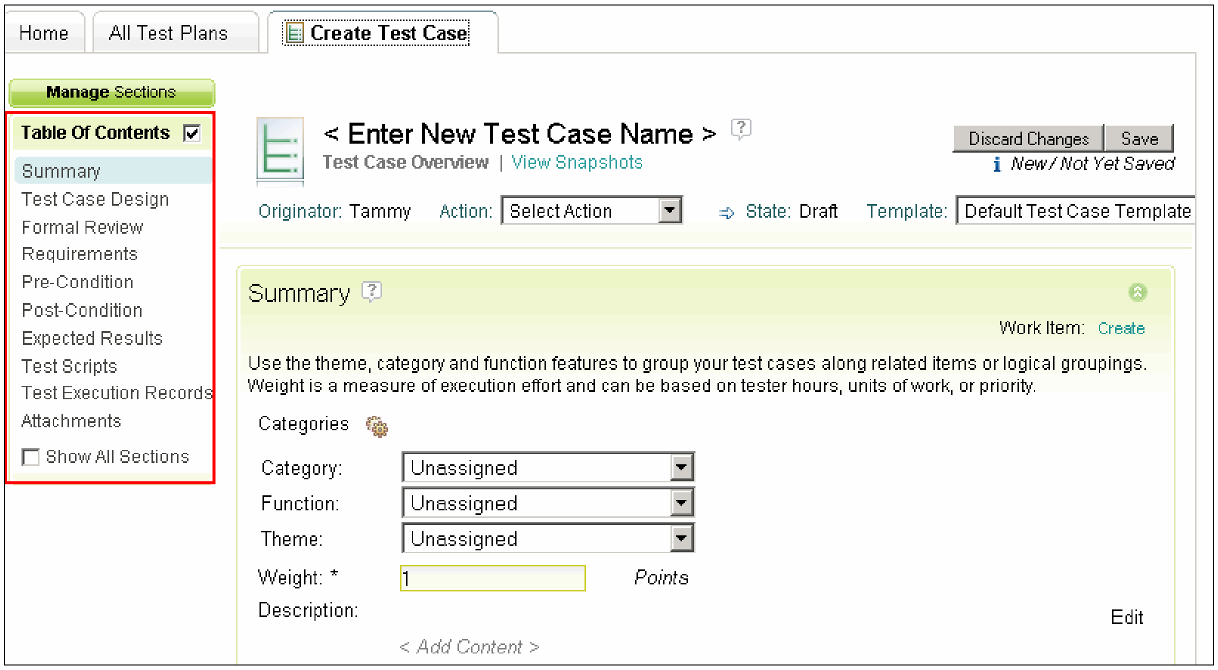


Figure 11-8 The Construction menu

A *test case* defines the characteristics for an individual test. Clicking the **Create Test Case** tab opens the test case creation user interface. The Test Case Table of Contents shown in Figure 11-9 provides sections with additional detail that describes the test case. Similar to the test plan, teams can decide which sections to use and can modify the list of available sections by clicking the **Manage Sections** button. For example, a test case can reference one or more test scripts, or has pre- and post-conditions defined, with expected results. Requirements can be associated with test cases to help the tester ensure requirement coverage as part of the test creation process. Test Case templates can be created to define the set of sections for use by the team.



Originator: Tammy Action: Select Action State: Draft Template: Default Test Case Template

Summary

Work Item: Create

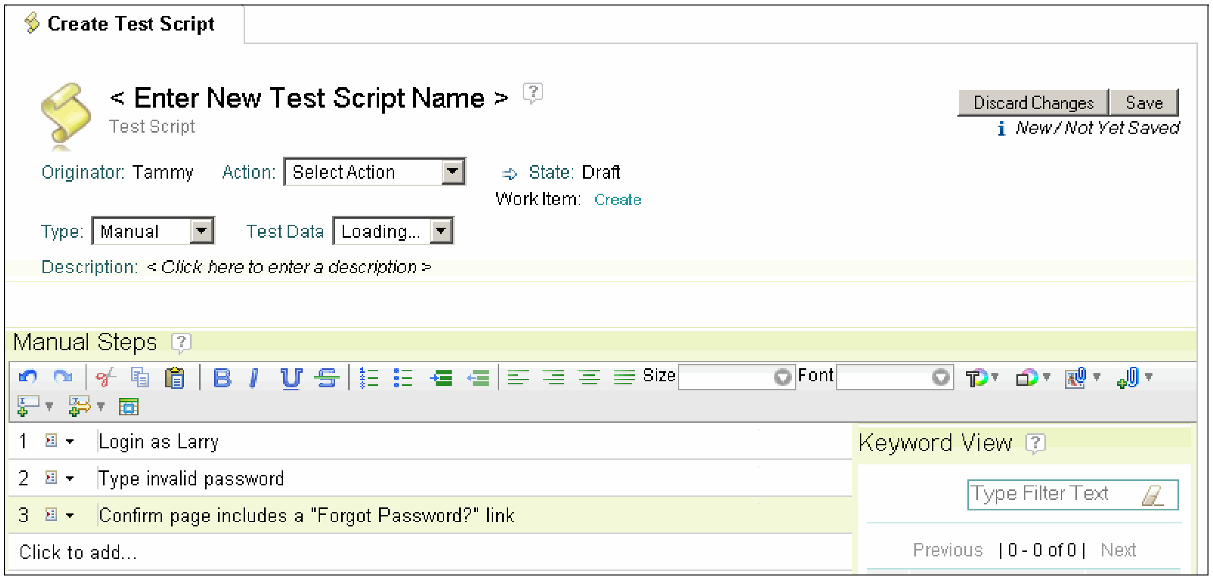
Use the theme, category and function features to group your test cases along related items or logical groupings. Weight is a measure of execution effort and can be based on tester hours, units of work, or priority.

Categories

Category:	Unassigned
Function:	Unassigned
Theme:	Unassigned
Weight: *	1 Points
Description:	

Figure 11-9 An example test case

A *test script* defines the steps that are involved in conducting the test. A test script can be a manual or automated test, or a combination. Manual test scripts are defined directly in Rational Quality Manager by using a rich text editor to detail each step as shown in Figure 11-10.



1 Login as Larry
2 Type invalid password
3 Confirm page includes a "Forgot Password?" link

Figure 11-10 Example of a manual test script

Test scripts can be written in context of the test case, or they can be created separately and later linked to the test case. This provides flexibility in work flow while also encouraging the reuse of existing test scripts.

Test suites can be created to group a set of related test cases. Test suites provide a list of related test cases, and a single test case can be included in more than one test suite. A new test suite is created by using the Create Test Suite menu item in the Construction menu. Figure 11-11 on page 418 shows the user interface for adding existing test cases to a new test suite.

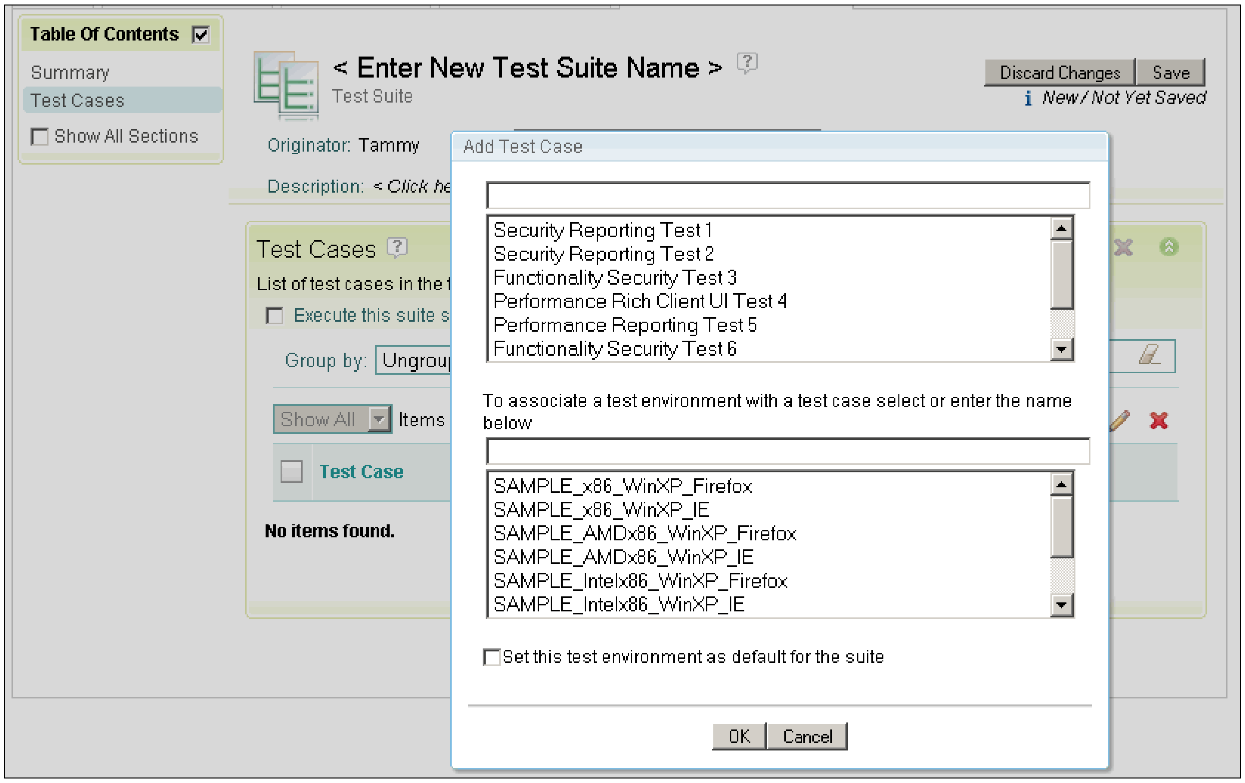


Table Of Contents

Summary

Test Cases

Show All Sections

< Enter New Test Suite Name >

Test Suite

Discard Changes | Save

New / Not Yet Saved

Originator: Tammy

Description: < Click he

Test Cases

List of test cases in the t

Execute this suite s

Group by: Ungrou

Show All | Items

Test Case

No items found.

Add Test Case

Security Reporting Test 1
Security Reporting Test 2
Functionality Security Test 3
Performance Rich Client UI Test 4
Performance Reporting Test 5
Functionality Security Test 6

To associate a test environment with a test case select or enter the name below

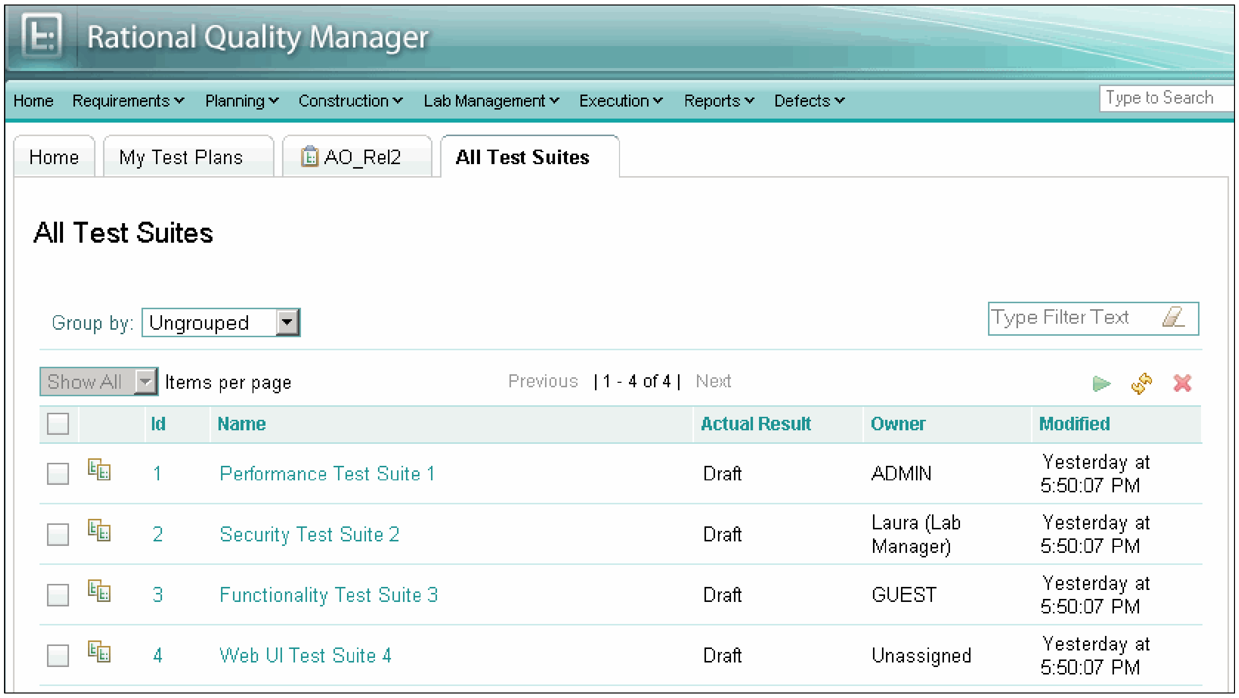
SAMPLE_x86_WinXP_Firefox
SAMPLE_x86_WinXP_IE
SAMPLE_AMDx86_WinXP_Firefox
SAMPLE_AMDx86_WinXP_IE
SAMPLE_Intelx86_WinXP_Firefox
SAMPLE_Intelx86_WinXP_IE

Set this test environment as default for the suite

OK | Cancel

Figure 11-11 Test cases linked to a new test suite

Existing test suites are located or modified from the Execution menu. Figure 11-12 shows the result of running the **Execution** → **All Test Suites**.


Rational Quality Manager

Home | Requirements | Planning | Construction | Lab Management | Execution | Reports | Defects | Type to Search

Home | My Test Plans | AO_Rel2 | All Test Suites

All Test Suites

Group by: Ungrouped

Type Filter Text

Show All | Items per page | Previous | 1 - 4 of 4 | Next

	Id	Name	Actual Result	Owner	Modified
	1	Performance Test Suite 1	Draft	ADMIN	Yesterday at 5:50:07 PM
	2	Security Test Suite 2	Draft	Laura (Lab Manager)	Yesterday at 5:50:07 PM
	3	Functionality Test Suite 3	Draft	GUEST	Yesterday at 5:50:07 PM
	4	Web UI Test Suite 4	Draft	Unassigned	Yesterday at 5:50:07 PM

Figure 11-12 Existing test suites

A single test case can be run on different environments. For example, in a Web user interface, different browsers can be tested. The test case is the same, but the environment is different. Therefore, an *execution record* is created to run the test case in each environment. In the previous example, an execution record is created for each browser that must be supported. As test cases are being constructed, the execution records can be created and assigned to team members. Test execution records are creating by using a menu item on the Construction menu. The Execution menu is used to locate and run existing test execution records.

Lab Management

Lab machines and virtual images are a critical resource in the testing effort. Rational Quality Manager has an optional add-on that helps a team to manage lab resources. When installed, an additional menu for Lab Management is added to the menu bar. A lab manager uses the menu shown in Figure 11-13 to manage the lab resources.

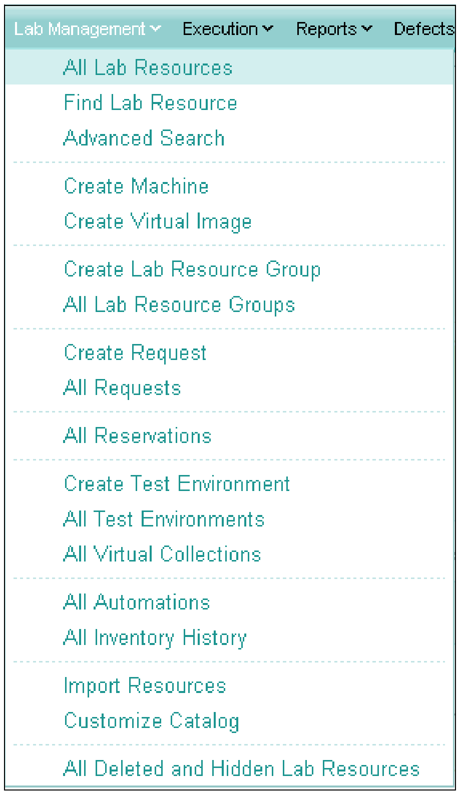
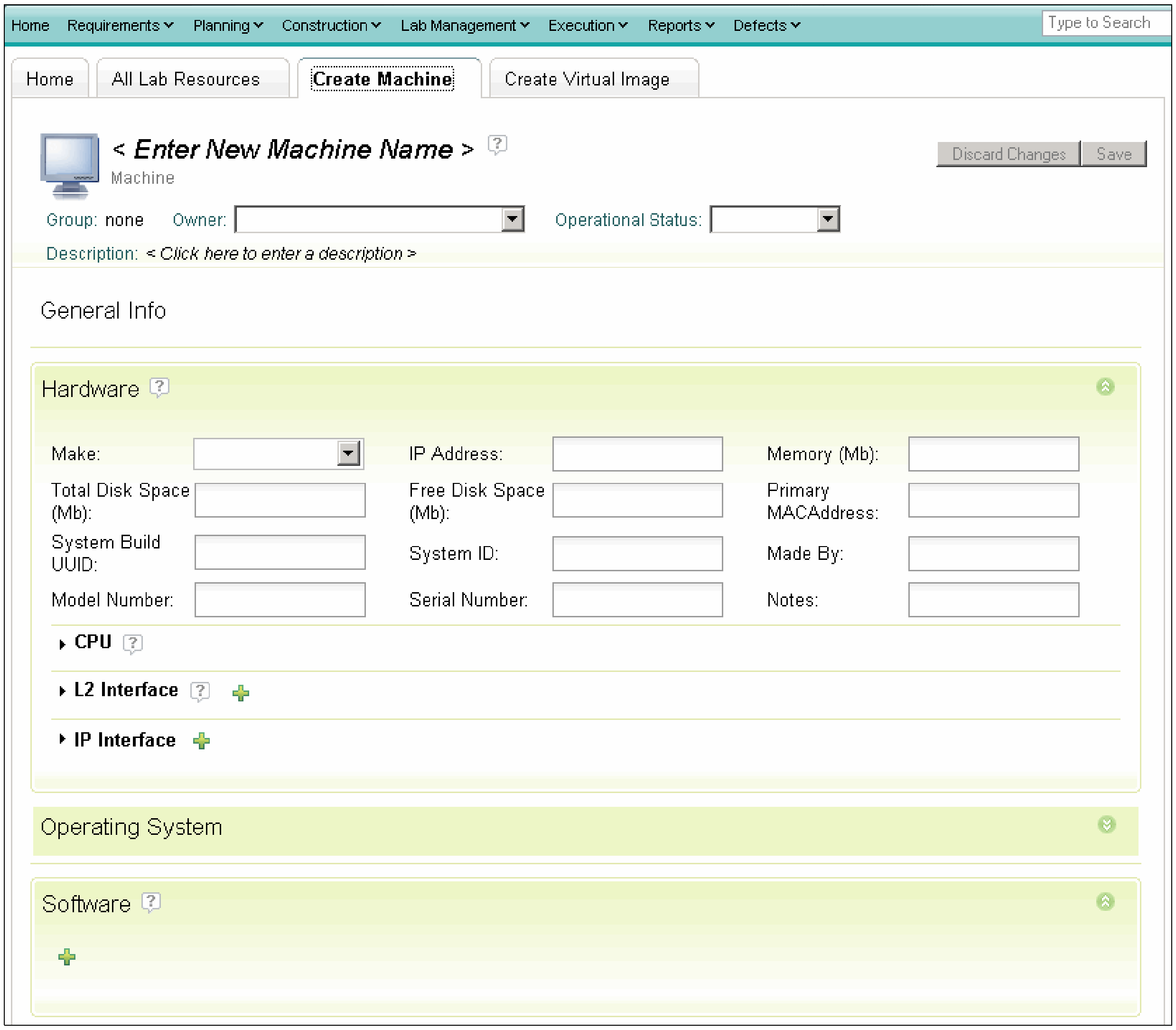


Figure 11-13 Lab Management menu

Machines can be added or imported into the lab management system. Characteristics of the machine, such as its hardware, software, and operating system, can be captured and stored in this system as shown in Figure 11-14. In a similar manner, virtual images can be recorded by the system. These recordings provide the team with an inventory of lab machines and virtual images to choose from by using the All Lab Resources, Find Lab Resource, or Advanced Search menu items.



< Enter New Machine Name >

Machine

Group: none Owner: Operational Status:

Description: < Click here to enter a description >

General Info

Hardware

Make:		IP Address:		Memory (Mb):	
Total Disk Space (Mb):		Free Disk Space (Mb):		Primary MACAddress:	
System Build UUID:		System ID:		Made By:	
Model Number:		Serial Number:		Notes:	

Operating System

Software

Figure 11-14 Adding a machine to the resource inventory

Lab resources can be allocated across test teams by using *resource groups*. A resource group is associated with a team area, and machines are added to the resource group. This creates a pool of machines that are specific to the members of the team area that is specified in the resource group.

Testers can use the lab management menu to reserve resources and submit requests to have a server configured with a specific environment or build.

Test environments define the machine characteristics, operating systems, and software needed for specific test functions. For example, a simple environment defines an x86 machine running a Microsoft Windows Vista operating system with Firefox 3.0 installed. More complex environments can also be defined, such as a three-tiered Web application configuration with a Web server, application server, and databases each running on different machines or virtual images. Defining environments simplifies the request process by enabling testers to choose predefined environments or define a new one. When defined, environments can be used to locate available resources that match the description in the test environment.

Test execution

In test construction, test cases are turned into execution records that are owned by members of the test team. Execution records are measured and tracked from the Execution menu (Figure 11-15).

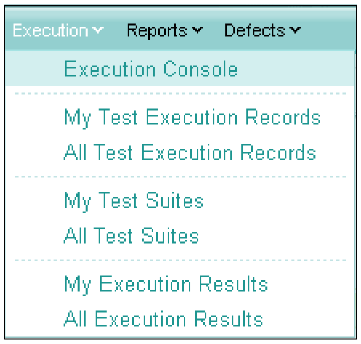
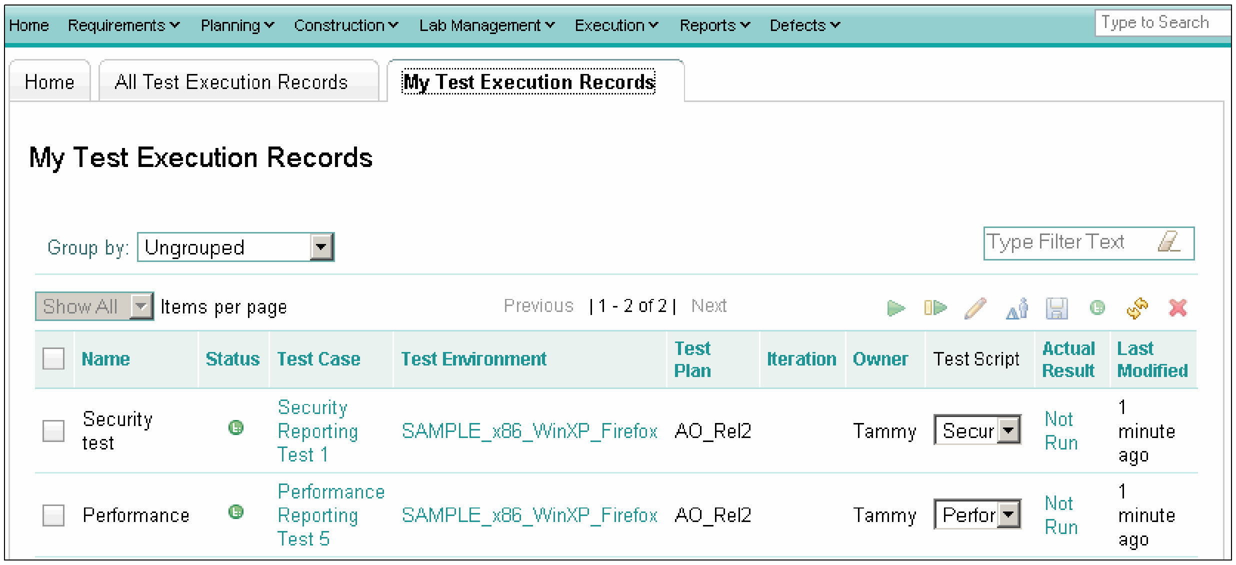


Figure 11-15 Execution menu

Executed tests have results and a result history. Queries can be run to view execution records or executed results. Figure 11-16 shows the list of records that are returned by choosing the My Test Execution Records menu item from the Execution menu.



My Test Execution Records

Group by: Ungrouped

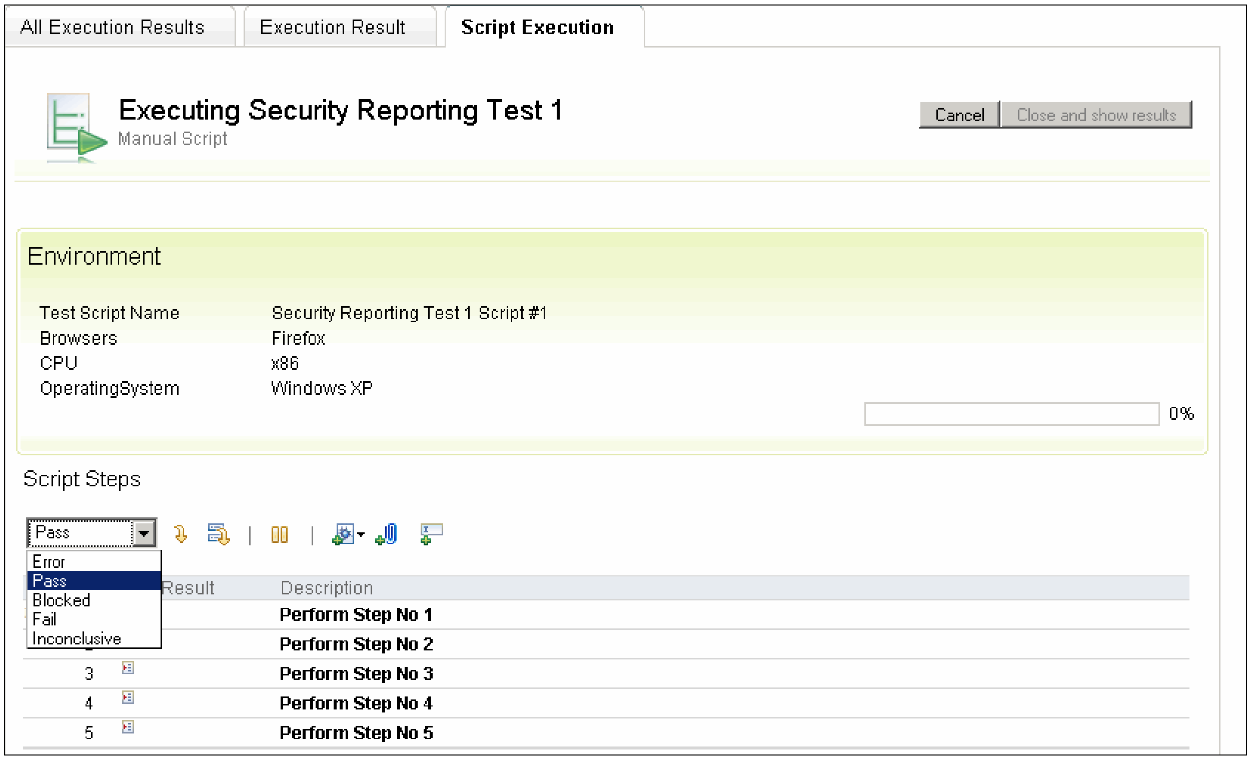
Type Filter Text

Show All Items per page   Previous | 1 - 2 of 2 | Next

	Name	Status	Test Case	Test Environment	Test Plan	Iteration	Owner	Test Script	Actual Result	Last Modified
☐	Security test	●	Security Reporting Test 1	SAMPLE_x86_WinXP_Firefox	AO_Rel2		Tammy	Secur	Not Run	1 minute ago
☐	Performance	●	Performance Reporting Test 5	SAMPLE_x86_WinXP_Firefox	AO_Rel2		Tammy	Perfor	Not Run	1 minute ago

Figure 11-16 Status of test execution for Execution work items

One or more records can be selected from the list and executed by clicking the green Play button on the toolbar. The script execution user interface is displayed, which is shown in Figure 11-17. For manual tests, each step of the test is performed by the tester and updated with a status. Step through the execution, one statement at a time, and click the **Apply verdict** icon after you perform it in the application that you are testing. Possible verdicts include Passed, Blocked, Failed, and Inconclusive.

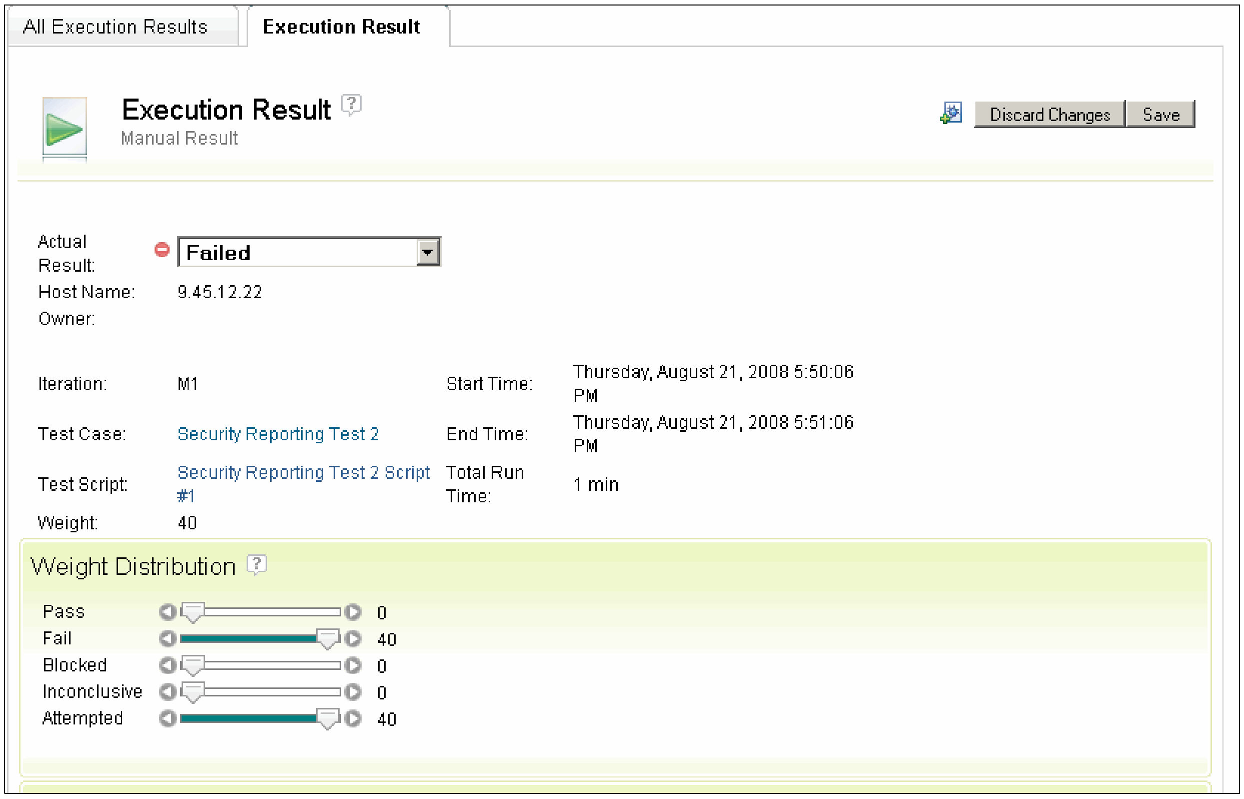


Test Script Name	Security Reporting Test 1 Script #1
Browsers	Firefox
CPU	x86
OperatingSystem	Windows XP

	Result	Description
		Perform Step No 1
		Perform Step No 2
3		Perform Step No 3
4		Perform Step No 4
5		Perform Step No 5

Figure 11-17 Script execution with Verdict choice to apply to each step

To view the Execution results, choose **My Execution Results** or **All Execution Results** from the Execution menu. The system returns the set of results in a list that can be quickly scanned. By clicking the **State** link, you see detailed information about the execution result, which includes information such as the verdict, and the environment on which the test was run, as shown in Figure 11-18.



Actual Result:	Failed		
Host Name:	9.45.12.22		
Owner:			
Iteration:	M1	Start Time:	Thursday, August 21, 2008 5:50:06 PM
Test Case:	Security Reporting Test 2	End Time:	Thursday, August 21, 2008 5:51:06 PM
Test Script:	Security Reporting Test 2 Script #1	Total Run Time:	1 min
Weight:	40		

Pass	0
Fail	40
Blocked	0
Inconclusive	0
Attempted	40

Figure 11-18 Execution Result details

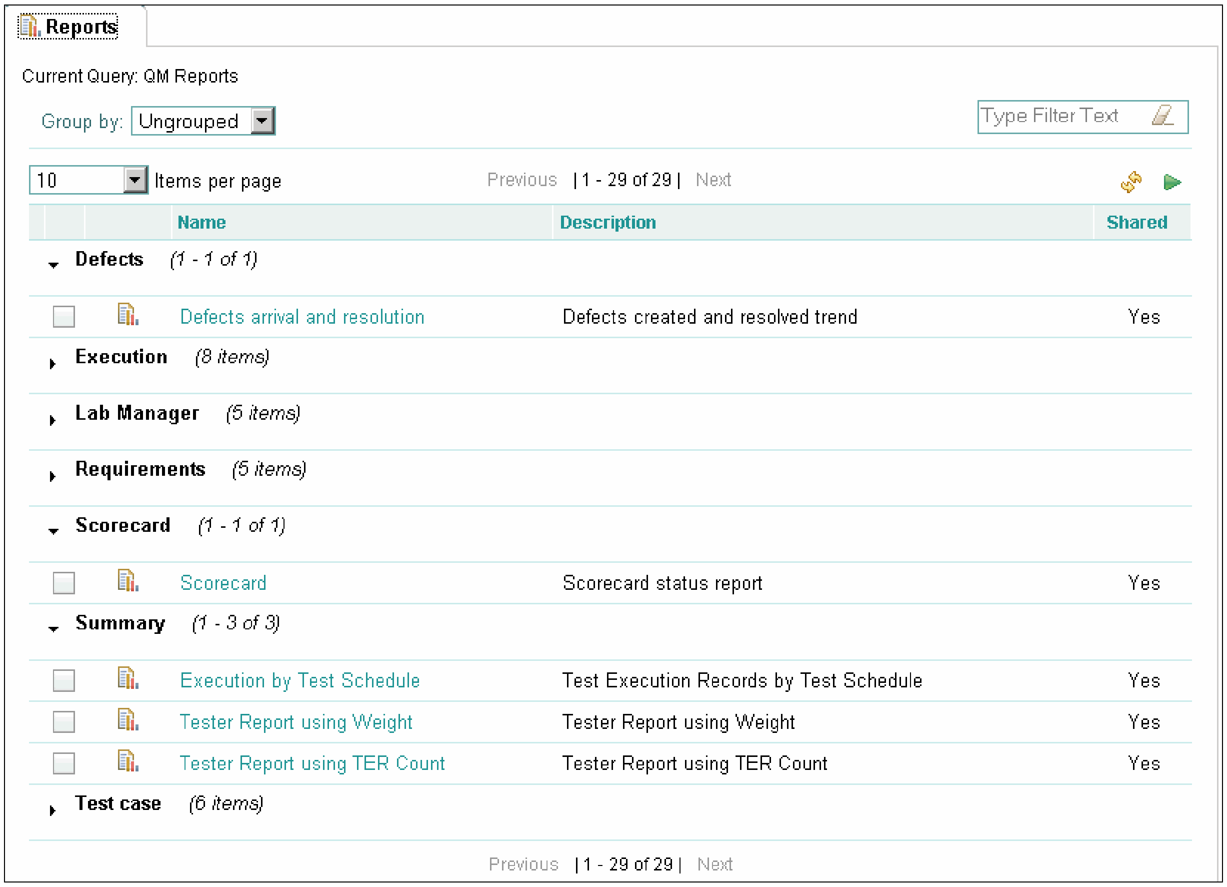
For automated test scripts, adapters can be added to an execution engine. Test scripts from automated software testing tools can be linked to test cases and managed by the system. When it is time for execution, the test script can be launched from Rational Quality Manager and run by the automated test tool, with the results captured and managed by Rational Quality Manager. You can run automated test scripts for the following types of tests:

- Functional tests by using tools such as Rational Functional Tester or Rational Robot
- Security tests by using Rational AppScan Tester Edition for Rational Quality Manager
- Performance tests by using Rational Performance Tester
- Service tests by using test scripts that are created with Rational Service Tester for SOA Quality

Adapters can be written to integrate additional automated test tools.

Test analysis (reports)

Up front test planning pays off when it comes to measuring success. By analyzing your test runs, you can identify trends and spot where bottlenecks exist in your project. From the Reports menu, choosing View Reports displays the reports that are available for measuring progress and quality. Figure 11-19 shows an example report for measuring the execution status by owner.



Name	Description	Shared
Defects (1 - 1 of 1)		
Defects arrival and resolution	Defects created and resolved trend	Yes
Execution (8 items)		
Lab Manager (5 items)		
Requirements (5 items)		
Scorecard (1 - 1 of 1)		
Scorecard	Scorecard status report	Yes
Summary (1 - 3 of 3)		
Execution by Test Schedule	Test Execution Records by Test Schedule	Yes
Tester Report using Weight	Tester Report using Weight	Yes
Tester Report using TER Count	Tester Report using TER Count	Yes
Test case (6 items)		

Figure 11-19 Reports provided to measure progress

The reports are grouped by function, such as defects, execution, summary, scorecard, and so forth. Further reports can be created by clicking the Create Report menu item.

11.3 Rational Functional Tester

Rational Functional Tester is used to automate functional testing, regression testing, GUI testing, and data-driven testing. In this scenario Rational Quality Manager manages the test scripts that are produced by Rational Functional Tester. When it is time to execute the test, Rational Quality Manager launches Rational Functional Tester, which runs the script. The results are captured and managed by Rational Quality Manager.

In the scenario, we do not detail how to write an automated functional test. Instead we assume that a test script is already written and managed by the system. We illustrate how that script is managed in Rational Quality Manager, how it is executed, and how the results are captured.

11.4 Rational AppScan Tester Edition for Rational Quality Manager

Rational AppScan Tester Edition is used to author and execute security scans. It provides testers with standard security policies and scan templates to test their Web applications.

In this scenario, users create a Rational AppScan Tester Edition test script in Rational Quality Manager. The test is executed from within Rational Quality Manager and displays pages and vulnerabilities as they are discovered. When execution is complete, high-level results are captured and used to determine whether the test script execution has passed. Testers can use Rational AppScan Tester Edition to drill down into scan results and then log vulnerabilities that are found by the security scan as Rational Quality Manager defects.

11.5 Rational Quality Manager

In this section, we provide detailed instructions about how the team uses the Rational Quality Manager tools to complete the scenario that was described in Chapter 5, “Rational ClearQuest, Requirements Composer, and RequisitePro to manage stakeholder requests” on page 115.

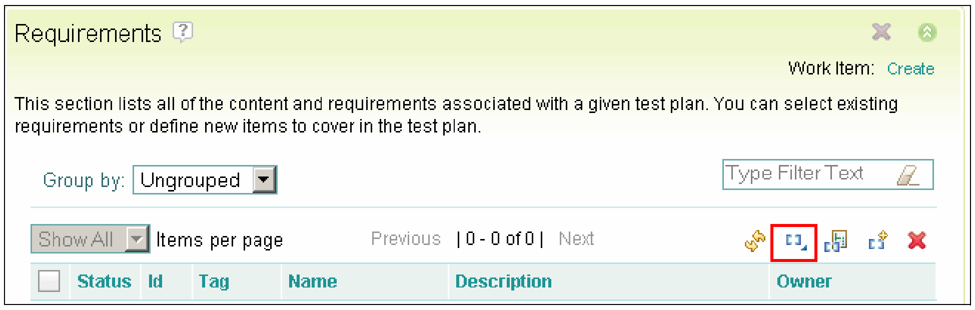
11.5.1 Tammy monitors quality

Synopsis: When Tammy updated the test plan with the new request from Bob, he had not yet detailed the requirements. The next day Bob notified Tammy that the requirements were complete. In this scene, Tammy links the requirements to her test plan. This scene begins immediately after the completion of Act 1 and occurs in parallel with Act 2, where Marco and Diedrie design and implement the request.

Tammy links to the requirements

Tammy has been notified by Bob that he has detailed the requirements for his new request. She logs into Rational Quality Manager and opens her test plan:

1. From the **Planning** menu, Tammy chooses the **My Test Plans** query.
2. She clicks the **Account Opening Rel2** test plan to open it.
3. In the **Table of Contents** for the test plan, she clicks **Requirements**.
4. In the requirements view, she clicks the **Import Requirements** button, which is highlighted in Figure 11-20.


Requirements

Work Item: Create

This section lists all of the content and requirements associated with a given test plan. You can select existing requirements or define new items to cover in the test plan.

Group by: Ungrouped

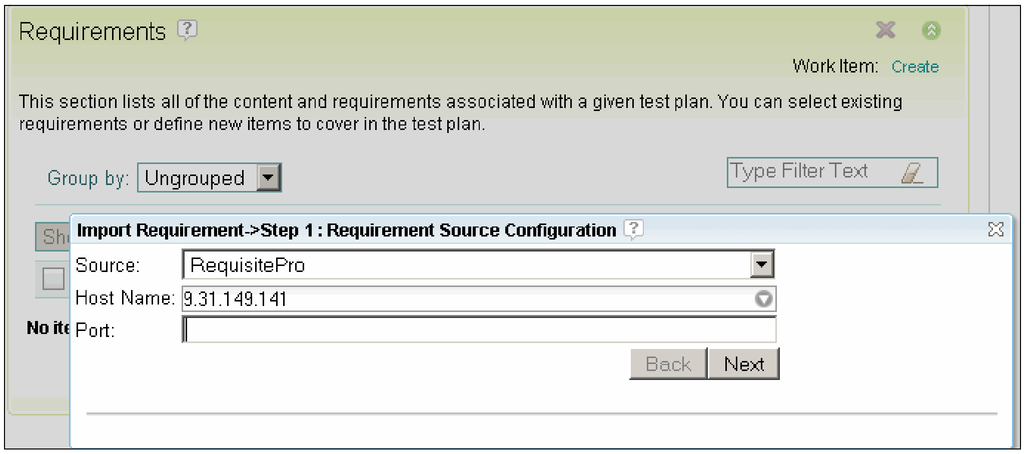
Type Filter Text

Show All Items per page Previous | 0 - 0 of 0 | Next

	Status	Id	Tag	Name	Description	Owner

Figure 11-20 Import Requirements button

5. In Step 1 of the Import Requirements wizard (Figure 11-21):
   a. For Source, she chooses **RequisitePro**.
   b. She types a host name.
   c. She clicks **Next**.


Requirements

Work Item: Create

This section lists all of the content and requirements associated with a given test plan. You can select existing requirements or define new items to cover in the test plan.

Group by: Ungrouped

Import Requirement->Step 1 : Requirement Source Configuration

Source: RequisitePro

Host Name: 9.31.149.141

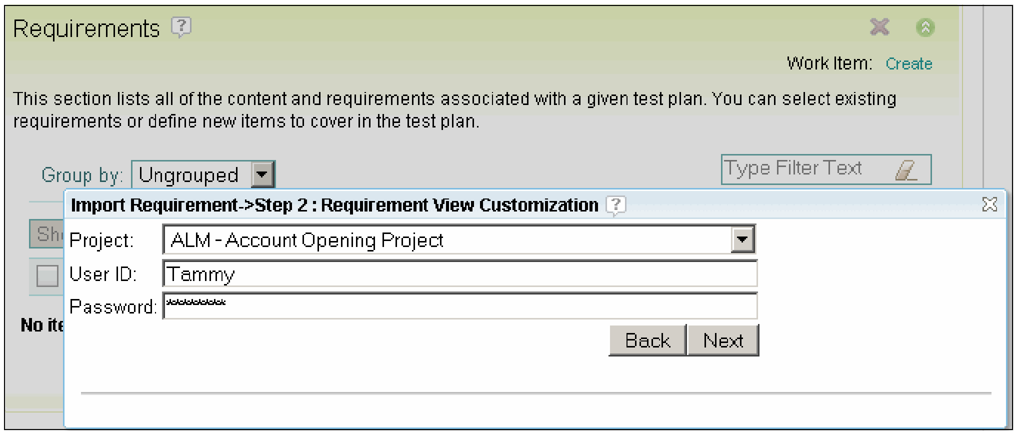
Port:

Figure 11-21   Step 1 of the Import Requirements wizard

6. In Step 2 of the Import Requirements wizard (Figure 11-22):
   a. For Project, she chooses the **ALM - Account Opening Project**.
   b. She types her Rational RequisitePro User ID and Password to log in to the repository.

   Important: In Rational Quality Manager 8.0 beta, we had to enable security on the Rational RequisitePro project and ensure that the user name had a password assigned.

   c. She clicks **Next**.


Requirements

Work Item: Create

This section lists all of the content and requirements associated with a given test plan. You can select existing requirements or define new items to cover in the test plan.

Group by: Ungrouped

Import Requirement->Step 2 : Requirement View Customization

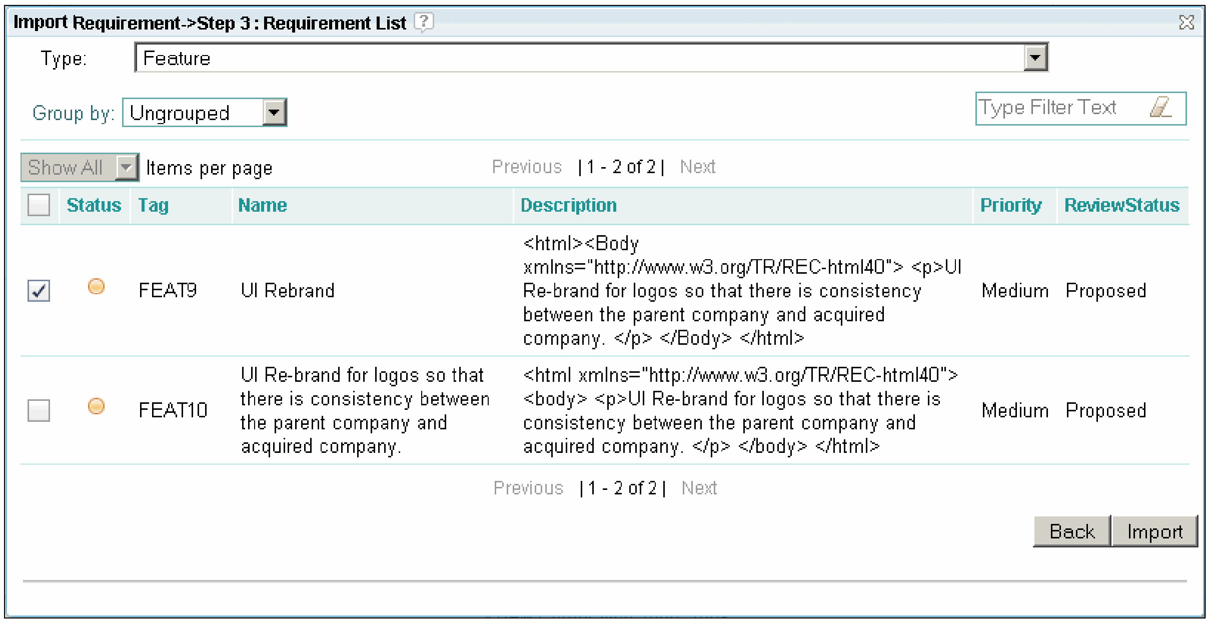
Project: ALM - Account Opening Project

User ID: Tammy

Password: ********

Figure 11-22   Step 2 of the Import Requirements wizard

7. In Step 3 of the Import Requirements wizard (Figure 11-23):
   a. She changes the requirement Type to **Feature**.
   b. She selects the check box next to the UI Branding requirement in the list.
   c. She clicks the **Import** button.



Import Requirement->Step 3 : Requirement List

Type: Feature

Group by: Ungrouped

Type Filter Text

Show All Items per page

Previous | 1 - 2 of 2 | Next

	Status	Tag	Name	Description	Priority	ReviewStatus
✓	●	FEAT9	UI Rebrand	<html><Body xmlns="http://www.w3.org/TR/REC-html40"> <p>UI Re-brand for logos so that there is consistency between the parent company and acquired company. </p> </Body> </html>	Medium	Proposed
	●	FEAT10	UI Re-brand for logos so that there is consistency between the parent company and acquired company.	<html xmlns="http://www.w3.org/TR/REC-html40"> <body> <p>UI Re-brand for logos so that there is consistency between the parent company and acquired company. </p> </body> </html>	Medium	Proposed

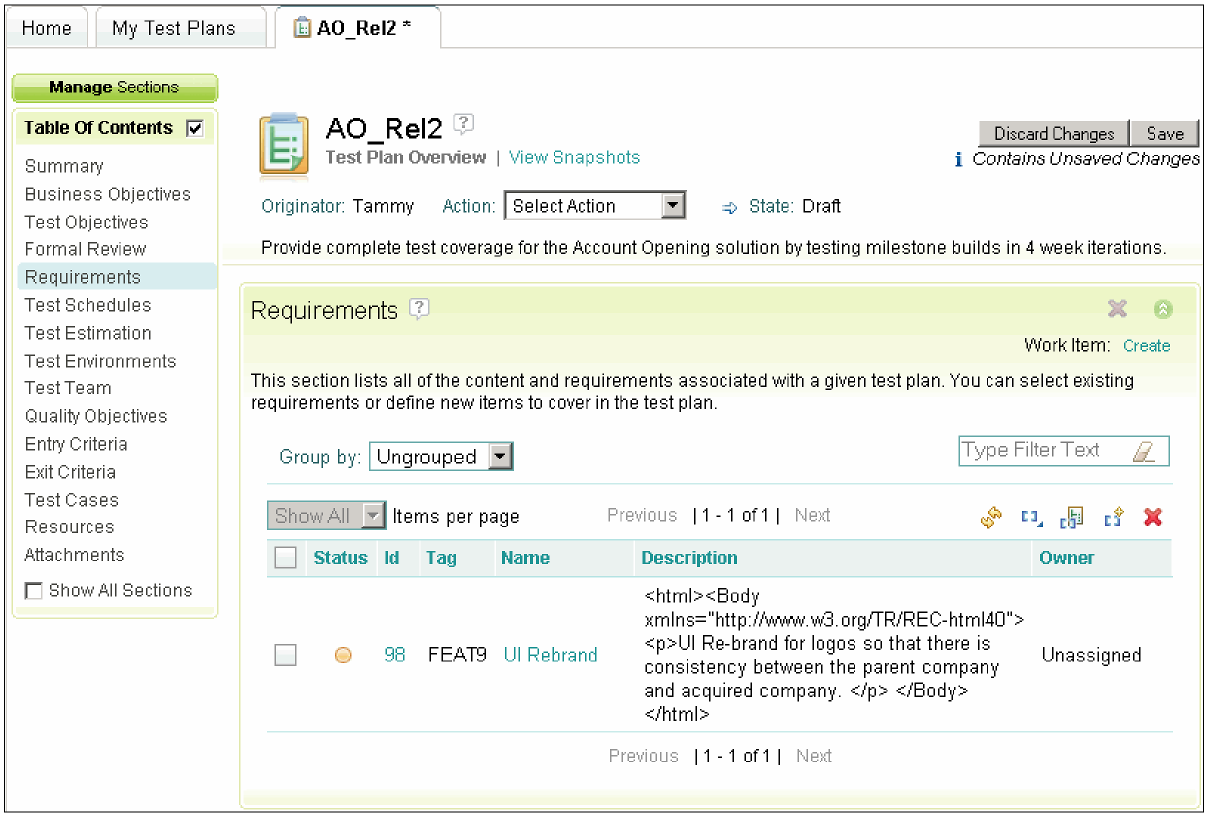
Previous | 1 - 2 of 2 | Next

Back Import

Figure 11-23 Step 2 of the Import Requirements wizard

The requirements are added to the Requirements section of the test plan as shown in Figure 11-24 and are now available for others to reference in test plans or test cases.

d. Tammy clicks **Save** to save the test plan.



Provide complete test coverage for the Account Opening solution by testing milestone builds in 4 week iterations.

This section lists all of the content and requirements associated with a given test plan. You can select existing requirements or define new items to cover in the test plan.

Status	Id	Tag	Name	Description	Owner
●	98	FEAT9	UI Rebrand	<html><Body xmlns="http://www.w3.org/TR/REC-html40"> <p>UI Re-brand for logos so that there is consistency between the parent company and acquired company. </p> </Body> </html>	Unassigned

Figure 11-24 Requirement imported into the test plan

Tip: The user interface uses the word “Import” for requirements. However, the requirements are not imported into Rational Quality Manager. Instead, URL links are created in Rational Quality Manager that reference the requirement that is managed in Rational RequisitePro.

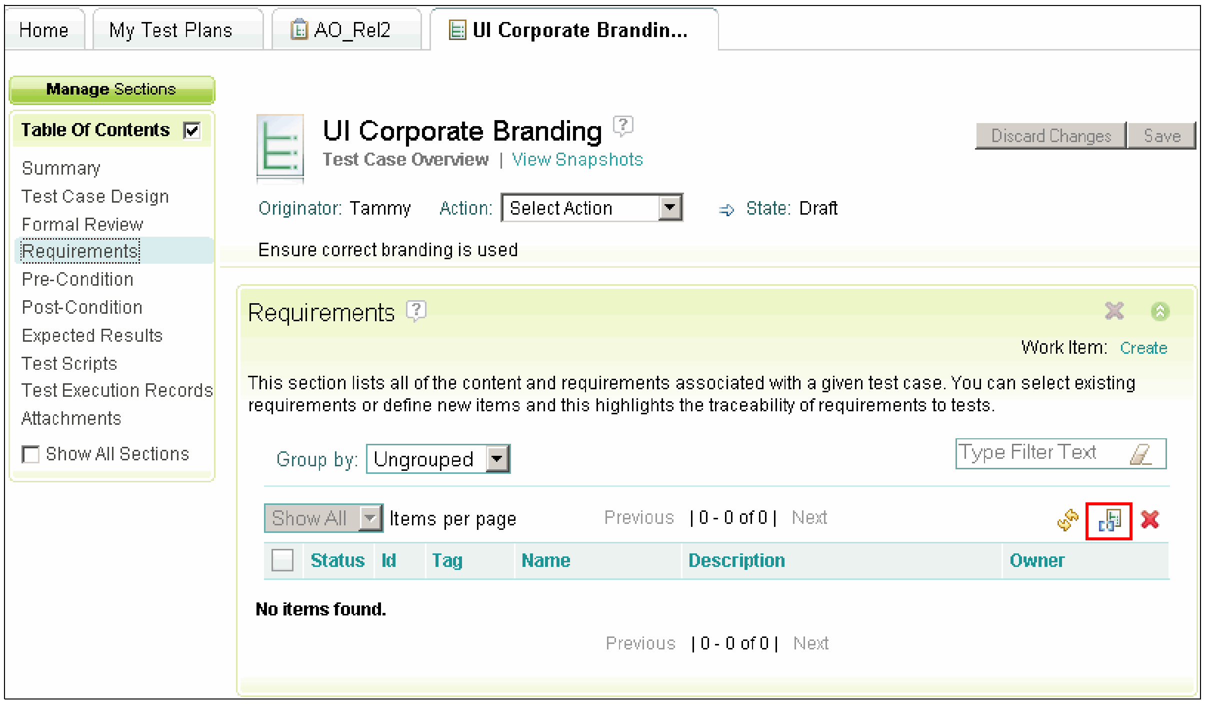
Tammy associates the test case with the requirement

In Act 1, Tammy created a placeholder test case in her test plan. Now that she has a link to the requirements that Bob created in Rational RequisitePro, she can create a link to the test case.

Tammy associates the test case to the requirement:

1. In the open test plan, Tammy clicks **Test Cases** in the Table of Contents.
2. She clicks the UI Corporate Branding test case.

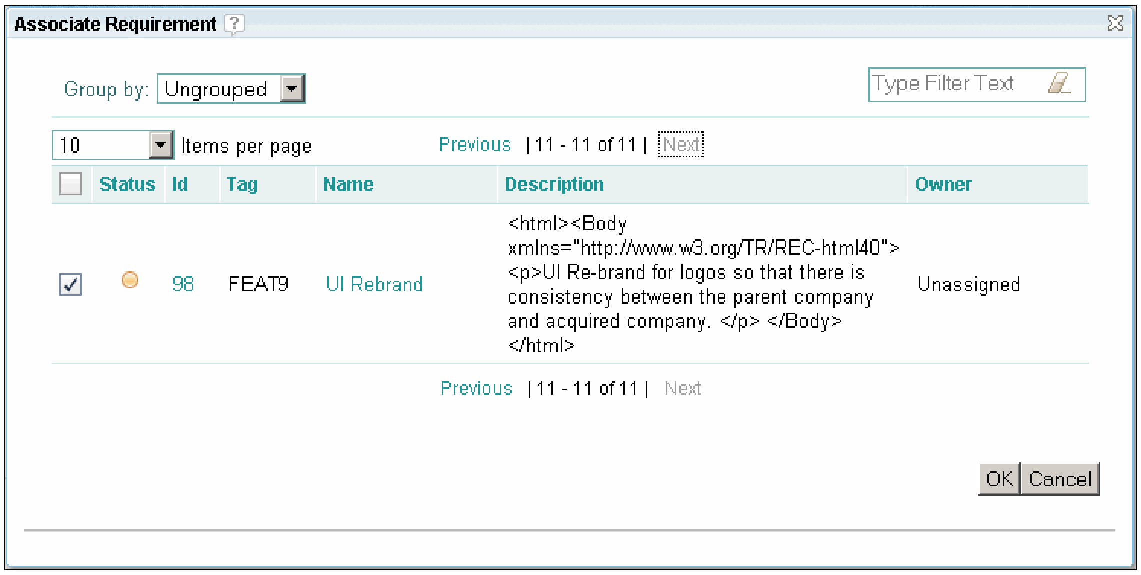
3. When the test case editor opens, Tammy clicks the **Requirements** menu item in the Test Case Table of Contents (Figure 11-25). In doing so, she provides information about the requirements that are associated with this test case. Because this is a new test case, no requirements are found.
4. She clicks the **Associate Requirements** button, which is highlighted in Figure 11-25.



This section lists all of the content and requirements associated with a given test case. You can select existing requirements or define new items and this highlights the traceability of requirements to tests.

Figure 11-25 Requirements associated with test cases

5. In the Associate Requirement panel, from the list of available requirements, she clicks the check box next to the UI Corporate Branding requirement (Figure 11-26). She has the option to select one or more requirements. However, in this case, she only needs this new requirement that Bob provided.



	Status	Id	Tag	Name	Description	Owner
✓	●	98	FEAT9	UI Rebrand	<html><Body xmlns="http://www.w3.org/TR/REC-html40"><p>UI Re-brand for logos so that there is consistency between the parent company and acquired company. </p> </Body> </html>	Unassigned

Figure 11-26 Selecting the requirements to associate with the test case

6. She clicks **OK**
7. She **Saves** the test plan.

Tip: When linking to requirements that are managed by Rational RequisitePro, Rational Quality Manager marks the requirements as “suspect” when they change in Rational RequisitePro. This gives testers a visual indication to review the requirements and ensure that the test cases are still valid.

11.5.2 Tanuj constructs the tests

Synopsis: When Tammy updated the test plan with the new request from Bob, Tanuj began constructing tests, while Diedrie and Marco were implementing the change. This scene occurs in parallel with Act 2.

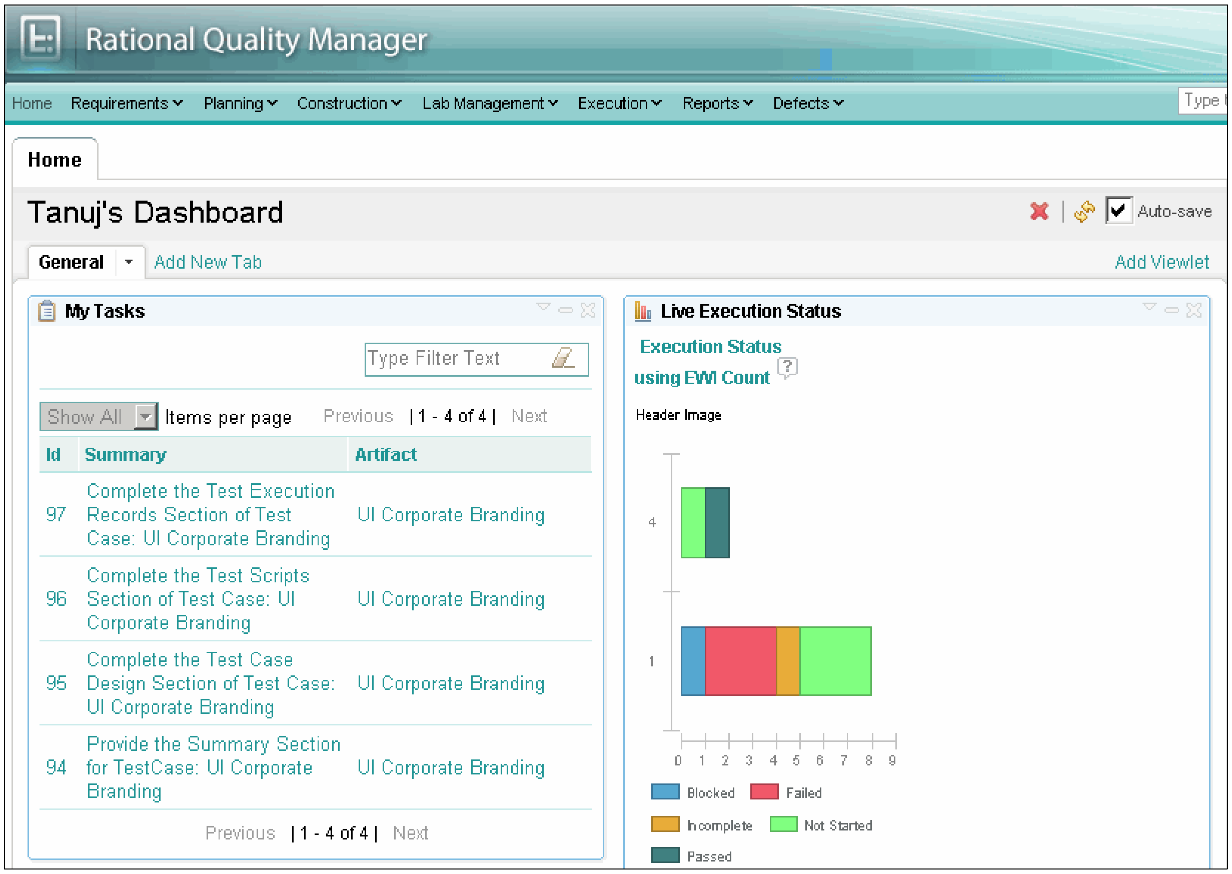
In this scene, Tanuj updates the test case, writes the test script, and creates the execution records to test Bob’s request on all test environments.

Tanuj updates the test cases

Goal: The goal is to create the test cases to cover the new requirement.

In “Tammy monitors quality” on page 425, Tammy updated the test plan by importing and associating a requirement with a test case that is owned by Tanuj. In this scene, Tanuj begins working on his assignments as explained in the following steps. He is working in parallel with Marco and Diedrie.

1. In Rational Quality Manager, Tanuj logs in and views his dashboard (Figure 11-27), where he sees the new tasks that are assigned to him by Tammy. His tasks are all related to a test case called *UI Corporate Branding*.



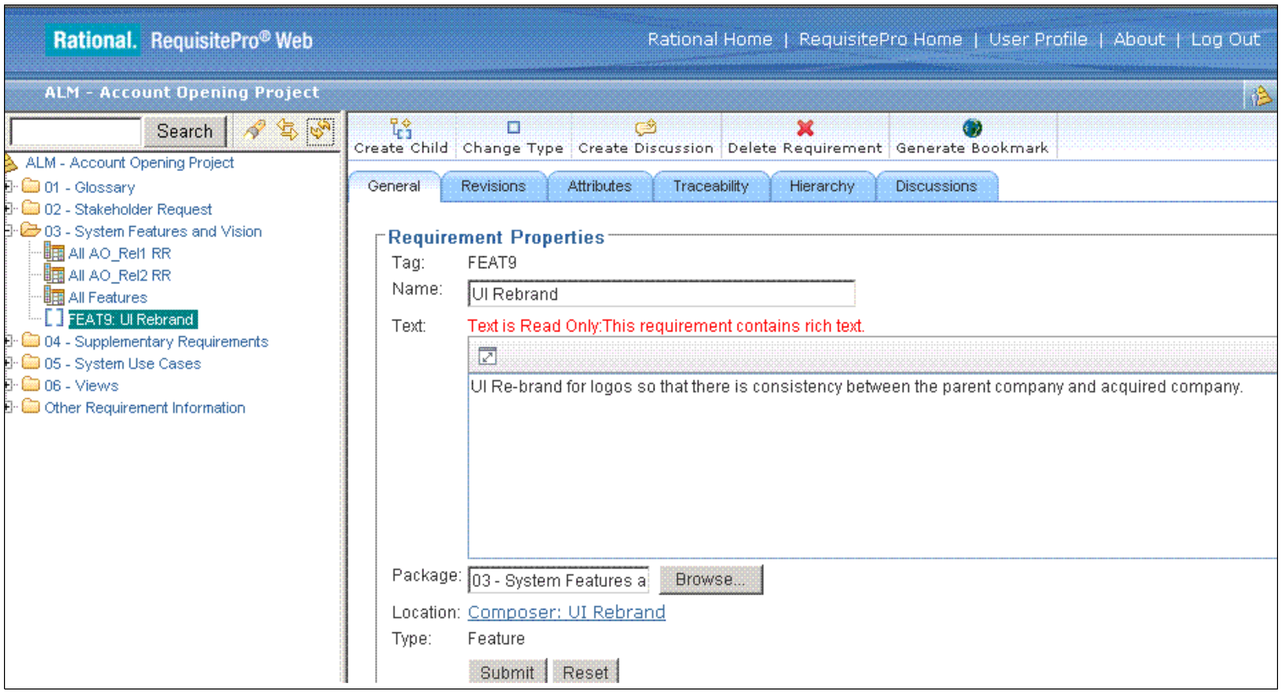
Id	Summary	Artifact
97	Complete the Test Execution Records Section of Test Case: UI Corporate Branding	UI Corporate Branding
96	Complete the Test Scripts Section of Test Case: UI Corporate Branding	UI Corporate Branding
95	Complete the Test Case Design Section of Test Case: UI Corporate Branding	UI Corporate Branding
94	Provide the Summary Section for TestCase: UI Corporate Branding	UI Corporate Branding

Previous | 1 - 4 of 4 | Next

Figure 11-27 Tanuj reviewing his tasks in the My Tasks viewlet

2. He selects **Construction** → **My Test Cases**.
3. In the results list, he selects the **UI Corporate Branding** test case from the list to view it.
4. He clicks **Summary** in the Table of Contents for the test case to review the summary and familiarize himself with this test case.
5. He clicks the **Requirements** tab to review the associated requirements for this test case.
6. He clicks the **Name** link for the associated requirement, and the Rational RequisitePro Web client launches either in another browser tab or browser window depending on how the browser is configured.
7. He selects the **Account Opening** project in the Project field, and enters his Rational RequisitePro login user name and password. He then clicks **Login**.

8. He reviews the requirement properties (Figure 11-28). Then in the Location field of the requirement properties, he clicks the link **Composer:UI Rebrand**.



Requirement Properties

Tag: FEAT9

Name: UI Rebrand

Text: Text is Read Only:This requirement contains rich text.

UI Re-brand for logos so that there is consistency between the parent company and acquired company.

Package: 03 - System Features a

Location: Composer: UI Rebrand

Type: Feature

Figure 11-28 Tanuj reviewing the requirement provided by Bob

9. After Rational Requirements Composer launches and opens the sketch that Bob created in Act 1 (Figure 11-29), he reviews the detail of the requirement and can now define the expected result for his test case. He closes Requirements Composer and returns to Rational Quality Manager.

Note: Tanuj uses the rich client that is installed on his desktop. While this book was being written, a Web client for Rational Requirements Composer was under development, but was not available for inclusion in this book.

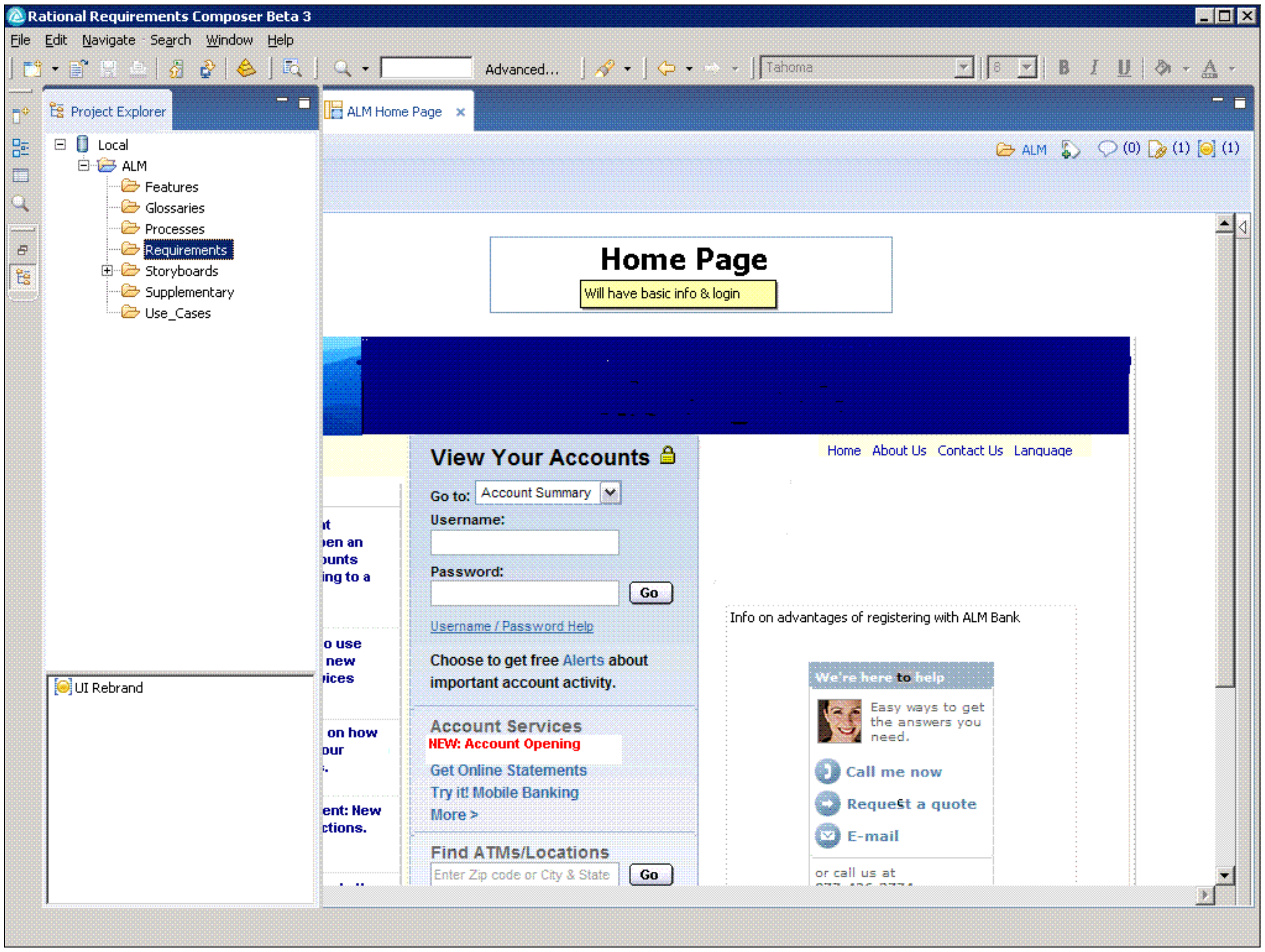


Figure 11-29 Tanuj reviewing the sketch provided by Bob

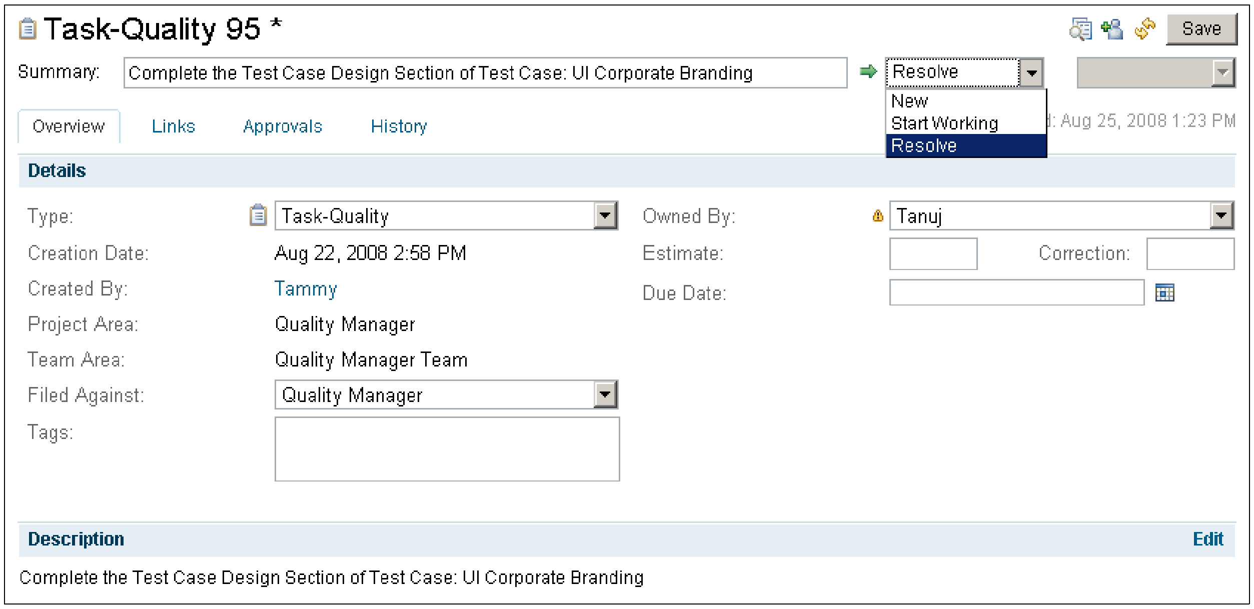
10. He clicks **Expected Results** in the Table of Contents, and adds additional information about the expected result for the test case. In this example, he can cite corporate guidelines or provide examples of properly re-branded Web pages as a comparison. He can also refer to the sketch that is provided by Bob.

11. He clicks **Test Case Design** to provide information about the test case. For this test case, he provides a description that includes the need to ensure that the UI complies with the corporate brand, and that because the elements are visual, this is a manual test.

12. He clicks **Save** to capture his changes to the test case. At this point, he decides that this single test case will cover Bob's requirement.

13. Tanuj reviews the tasks that are listed in the My Tasks viewlet on his dashboard. He selects the task that is related to completing the test case design and clicks to open it.

14. As shown in Figure 11-30, he changes the state to **Resolve** and clicks **Save**.


Task-Quality 95 *

Save

Summary: Complete the Test Case Design Section of Test Case: UI Corporate Branding

Resolve

New
Start Working
Resolve

Overview | Links | Approvals | History

: Aug 25, 2008 1:23 PM

Details

Type:	Task-Quality	Owned By:	Tanuj
Creation Date:	Aug 22, 2008 2:58 PM	Estimate:	
Created By:	Tammy	Due Date:	
Project Area:	Quality Manager		
Team Area:	Quality Manager Team		
Filed Against:	Quality Manager		
Tags:			

Correction:

Description Edit

Complete the Test Case Design Section of Test Case: UI Corporate Branding

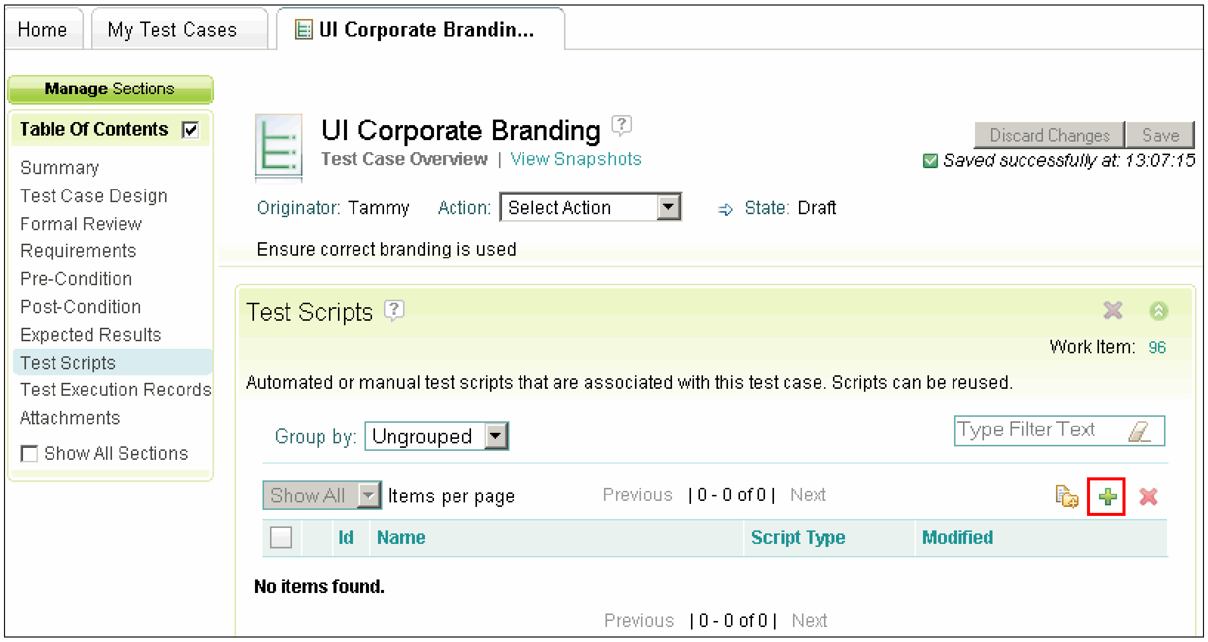
Figure 11-30 Tanuj resolving a task

Tanuj writes the test script

Goal: The goal is to capture the execution steps, verification points, and reporting points for the test case.

In this scene Tanuj writes a manual test script to use for testing Bob's request.

1. Tanuj clicks **Construction** → **My Test Cases**. The test case he's been working on is still visible in a tab. In the Test case editor, Tanuj clicks **Test Scripts** in the Table of Contents.
2. He clicks **Add New Test Script**, which is highlighted with a red box in Figure 11-31.


Home | My Test Cases | UI Corporate Brandin...

Manage Sections

Table Of Contents ☑

Summary
Test Case Design
Formal Review
Requirements
Pre-Condition
Post-Condition
Expected Results
Test Scripts
Test Execution Records
Attachments
☐ Show All Sections

UI Corporate Branding
Test Case Overview | View Snapshots

Discard Changes | Save

Saved successfully at: 13:07:15

Originator: Tammy Action: Select Action State: Draft

Ensure correct branding is used

Test Scripts

Work Item: 96

Automated or manual test scripts that are associated with this test case. Scripts can be reused.

Group by: Ungrouped

Type Filter Text

Show All Items per page Previous | 0 - 0 of 0 | Next

☐	Id	Name	Script Type	Modified

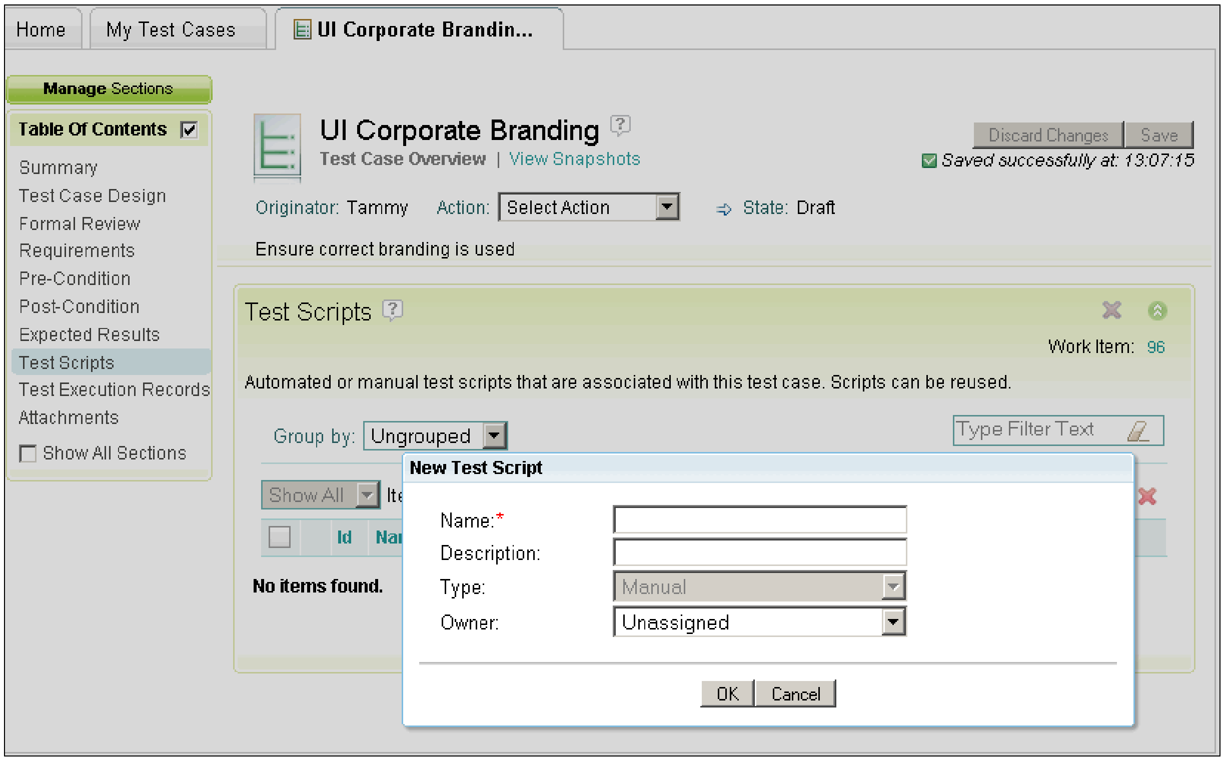
No items found.

Previous | 0 - 0 of 0 | Next

Figure 11-31 Adding a test script to a test case

3. In the New Test Scripts window (Figure 11-32):

   a. He types a Name and Description.
   b. He leaves Type set to **Manual**.
   c. For Owner, he selects himself.
   d. He clicks **OK**.



UI Corporate Branding

Test Case Overview | View Snapshots

Saved successfully at: 13:07:15

Originator: Tammy Action: Select Action State: Draft

Ensure correct branding is used

Test Scripts

Work Item: 96

Automated or manual test scripts that are associated with this test case. Scripts can be reused.

Group by: Ungrouped

No items found.

New Test Script

Name:*

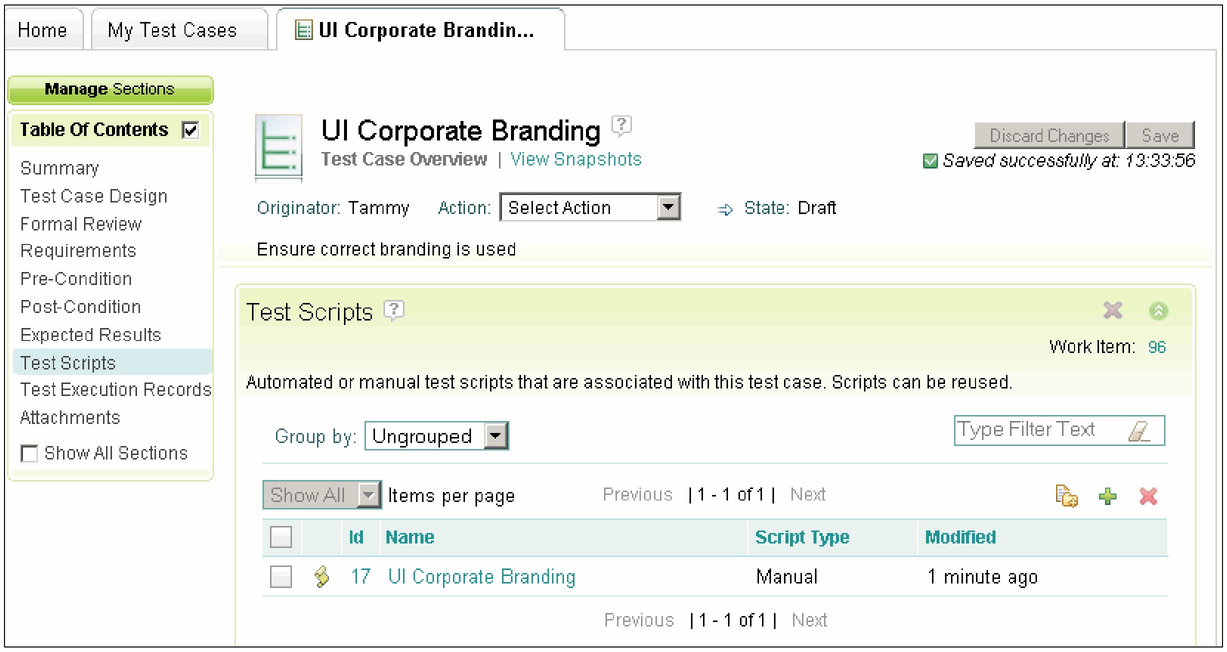
Description:

Type: Manual

Owner: Unassigned

Figure 11-32 New Test Script window

The window closes, and the test script is created and linked to the test case. He clicks **Save** to save the test case. The test script is now a clickable link in the test case, as shown in Figure 11-33.



UI Corporate Branding

Test Case Overview | View Snapshots

Saved successfully at: 13:33:56

Originator: Tammy   Action: Select Action   State: Draft

Ensure correct branding is used

Test Scripts

Work Item: 96

Automated or manual test scripts that are associated with this test case. Scripts can be reused.

Group by: Ungrouped

Show All Items per page   Previous | 1 - 1 of 1 | Next

	Id	Name	Script Type	Modified
	17	UI Corporate Branding	Manual	1 minute ago

Previous | 1 - 1 of 1 | Next

Figure 11-33 Test script linked to a test case

Tanuj can now begin writing the test script:

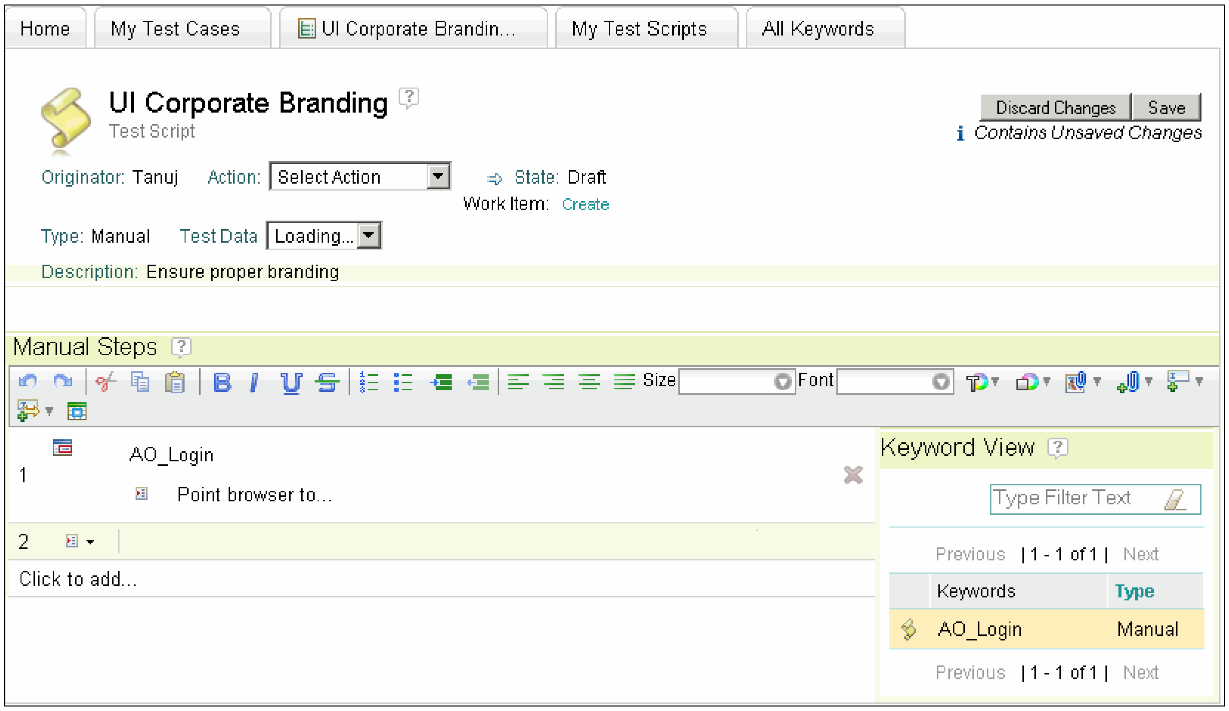
1. He clicks the link for the **UI Corporate Branding** test script, and the test script editor opens (Figure 11-34 on page 437).
2. Tanuj chooses to reuse an automated test as **Keyword** in this script. He drags the **AO_Login** script from the Keyword view to the first step of his script (Figure 11-34).

Tip: A *keyword* is any statement or group of statements that you can reuse in other test scripts. Many tasks in testing are composed of a sequence of steps, for example, logging in to a page. A simple task can be saved as a keyword. That keyword can be used in other scripts to create the full test script.

A team can build a library of keywords to streamline the writing of test scripts while ensuring consistency for frequent tasks. Additionally, manual keywords become candidates for automation, thus helping a team to move from manual testing to a mix of manual and automated tests. As seen in this example, Tanuj constructs a manual test that contains an automated login script. While this is a simple case, it is representative of using a keyword automation as part of a larger manual test.

3. Tanuj clicks the text **Click to add** in the bottom left corner of the editor, which adds a step in the test and places his cursor in the text edit field. Tanuj types the second step of the test script and presses Enter.

4. After another line is added to the test script, Tanuj types the second step in the test. He continues this procedure until he's added all steps of the step as shown in Figure 11-34.



UI Corporate Branding
Test Script

Contains Unsaved Changes

Originator: Tanuj Action: Select Action State: Draft
Work Item: Create
Type: Manual Test Data Loading...
Description: Ensure proper branding

Manual Steps

1 AO_Login
Point browser to...
2
Click to add...

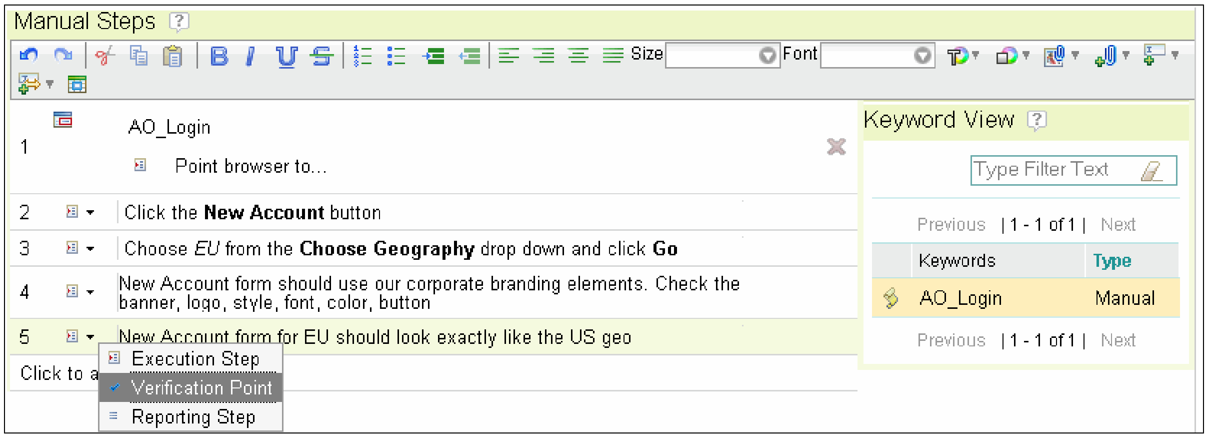
Keyword View
Type Filter Text
Previous | 1 - 1 of 1 | Next

Keywords	Type
AO_Login	Manual

Previous | 1 - 1 of 1 | Next

Figure 11-34 Creating a manual test by using an automated keyword test

5. He sets two of the steps to be verification points by clicking the **Execution step** icon, which is to the right of the step number. A menu is displayed as shown in Figure 11-35.
   – Execution Step is the action that you want the tester to perform when running the script, for example, "Start the application."
   – Verification Point asks questions about the application that you are testing, for example, "Did the User Login window open?"
   – Reporting Step is a higher-level verification point. It also asks questions, but the answers require higher visibility and often are included in reports. Reporting steps might summarize the result of several verification points, for example, "Were you able to log in?"



Manual Steps

1 AO_Login
Point browser to...
2 Click the **New Account** button
3 Choose *EU* from the **Choose Geography** drop down and click **Go**
4 New Account form should use our corporate branding elements. Check the banner, logo, style, font, color, button
5 New Account form for EU should look exactly like the US geo
Click to a

Keyword View
Type Filter Text
Previous | 1 - 1 of 1 | Next

Keywords	Type
AO_Login	Manual

Previous | 1 - 1 of 1 | Next

Figure 11-35 Setting a Verification Point in a manual test

6. He chooses **Verification Point** for steps 4 and 5. The icon changes to a blue check mark.
7. Tanuj clicks **Save** on the test script.

Tanuj configures the tests for execution

Goal: The goal is to determine how many test executions are needed to cover this requirement in all test configurations.

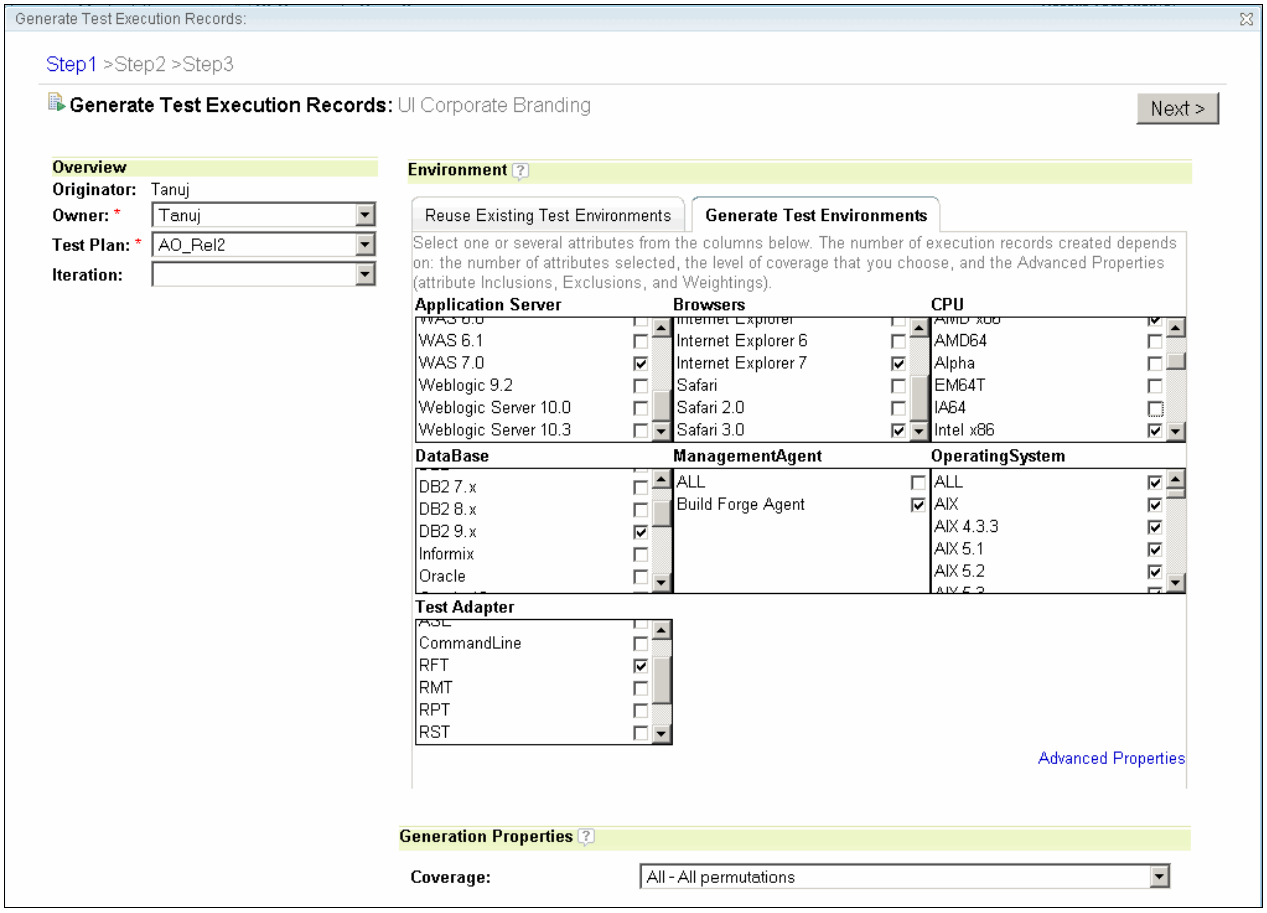
In this scene, Tanuj determines that the test matrix needed to fully test Bob's request. During test planning, Tammy defined the environments used for testing. Tanuj uses that environment list to create one test execution record for each environment on the test matrix. A common challenge for testers is to ensure that all combinations of test environments are tested for a solution. Rational Quality Manager simplifies this complex task by managing the expected environments and providing a wizard to automatically generate test execution records. The results of each execution record are also managed by the system. In this scene, Tanuj creates the test execution records to cover the environments that are defined in the test plan.

Test execution records specify the execution environments for the test case. Imagine a simple browser test. Multiple client machines with a specified browser are set up for a single test case. This ensures that the system behaves properly in a matrix of browser versions running on different operating systems and hardware. A test execution record is created to track the execution of each of these tests. Thus, a single browser-based test case might have one test execution record to verify the Firefox 2.x browser on a Windows XP machine and a different test execution record to run on a SUSE Linux machine or to run in a different browser.

1. Tanuj clicks the tab for the **UI Corporate Branding** test case. If a tab is not available, he can select **Construction** → **My Test Cases** query.
2. In the Table of Contents, he clicks **Test Execution Records** to view what is currently defined. Because this is a new test case, no test execution records exist.
3. He clicks the **Generate Test Execution Records** button from the toolbar. This starts the Generate Test Execution Records wizard, which is shown in Figure 11-36 on page 439. After the wizard has started, he can cancel it at any time by clicking the X in the upper right corner.
4. In Step 1 of the wizard, he defines the following settings:
   a. He sets Owner to himself.
   b. He sets the Test Plan to **Account_Opening Rel 2**.
   c. He sets Iteration to 2. Iterations are the various phases or milestones in your test plan. Iterations are only available when you select a particular test plan.
5. He clicks **Generate Test Environments** tab and selects one or several environment attributes from each section. His application always runs on the same server configuration by using IBM WebSphere Application Server and IBM DB2 running SUSE Linux Enterprise Server (SLES). Because this is a browser test, he is most interested in creating client execution environments. He selects three browsers, a CPU, and all operating systems that are defined in the test plan.

   The number of execution work items that are created depends on the number of attributes that is selected, the level of coverage that you choose, and the Advanced Properties (Inclusions, Exclusions, and Weightings) that you select.

   Advanced Properties: Advanced Properties may not be visible until you select at least two attributes from one of the lists of attributes.


Generate Test Execution Records:

Step1 >Step2 >Step3

Generate Test Execution Records: UI Corporate Branding

Next >

Overview

Originator:	Tanuj
Owner: *	Tanuj
Test Plan: *	AO_Rel2
Iteration:	

Environment

Reuse Existing Test Environments | **Generate Test Environments**

Select one or several attributes from the columns below. The number of execution records created depends on: the number of attributes selected, the level of coverage that you choose, and the Advanced Properties (attribute Inclusions, Exclusions, and Weightings).

Application Server

WAS 6.1	☐
WAS 7.0	☑
Weblogic 9.2	☐
Weblogic Server 10.0	☐
Weblogic Server 10.3	☐

Browsers

Internet Explorer 6	☐
Internet Explorer 7	☑
Safari	☐
Safari 2.0	☐
Safari 3.0	☑

CPU

AMD64	☐
Alpha	☐
EM64T	☐
IA64	☐
Intel x86	☑

DataBase

DB2 7.x	☐
DB2 8.x	☐
DB2 9.x	☑
Informix	☐
Oracle	☐

ManagementAgent

ALL	☐
Build Forge Agent	☑

OperatingSystem

ALL	☑
AIX	☑
AIX 4.3.3	☑
AIX 5.1	☑
AIX 5.2	☑

Test Adapter

CommandLine	☐
RFT	☑
RMT	☐
RPT	☐
RST	☐

Advanced Properties

Generation Properties

Coverage: All - All permutations

Figure 11-36 Step 1 in the Generate Execution Work Items wizard

6. For Coverage, he selects the desired level to **All Permutations**. He uses this setting and the Advanced Properties setting to fine-tune the execution work items that will be generated. Coverage includes the following options:

   – The *Minimal* option ensures that each selected attribute is covered at least once, with no attempt to cover specific combinations of attributes. For example, if you select one attribute from three of the columns, three execution work items are created, ensuring that each selected attribute is covered at least once.

   – The *Medium - pair-wise interaction* option ensures that each combination of paired attributes is covered at least once.

   – The *Large - three-way interaction* option ensures that each three-way combination of attributes is covered at least once.

   – The *All - all permutations* option ensures that all combinations of attributes are covered at least once.

   Tanuj has no need for Advanced Properties in this case. In more complex scenarios, Advanced Properties are used to display a window with following three tabs:

   – On the *Exclusions* tab, he can specify the attribute combinations to explicitly exclude, for example, the Safari browser running on Windows XP.

   – On the *Inclusions* tab, he can specify the attribute combinations to always include, for example, Windows Internet Explorer 7.x running on Windows XP.

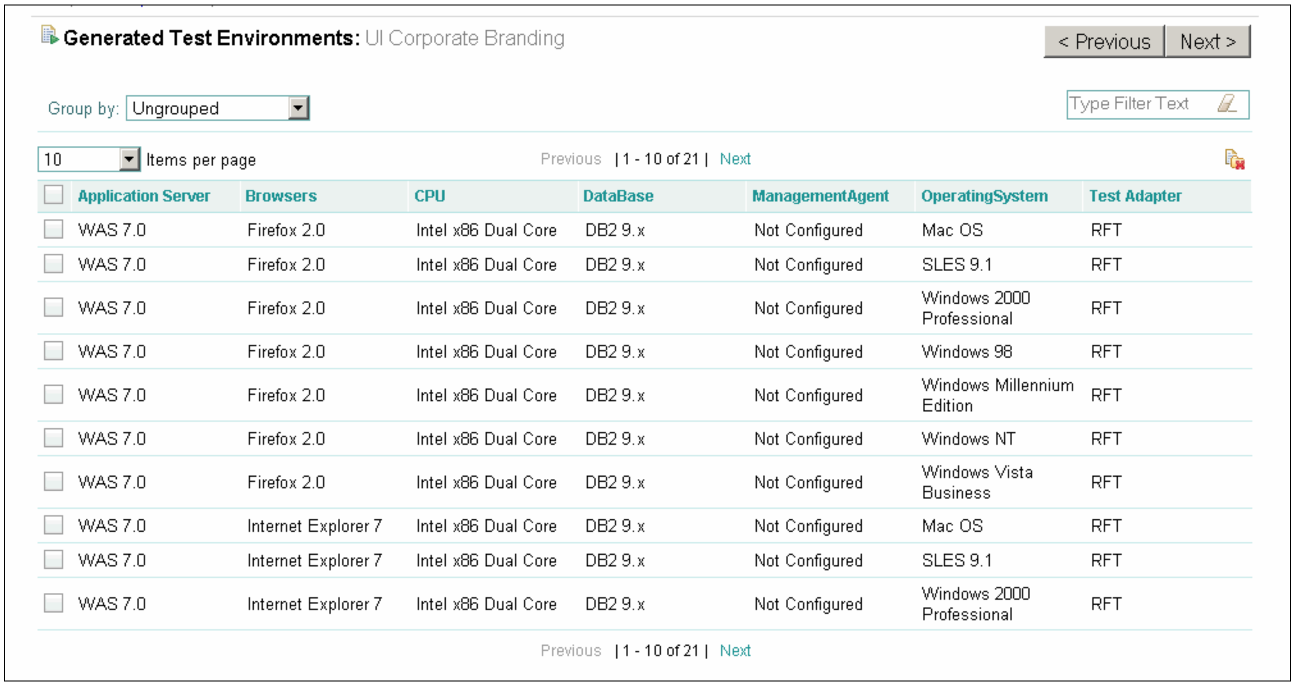
   – On the *Weightings* tab, he can to set the weight or importance of each attribute relative to the other values for that attribute.

For example, he can assign greater weight to Windows XP than SUSE Linux to ensure that at the least Windows XP is tested.

7. He clicks **Next**.

   The wizard creates a preview of the Generated Test Environments from which the execution work items will be generated. Figure 11-37 shows the environments that are presented to Tanuj.

8. He selects the configurations that he wants to keep and clicks **Next**.



Generated Test Environments: UI Corporate Branding

Group by: Ungrouped

10 Items per page   Previous |1 - 10 of 21| Next

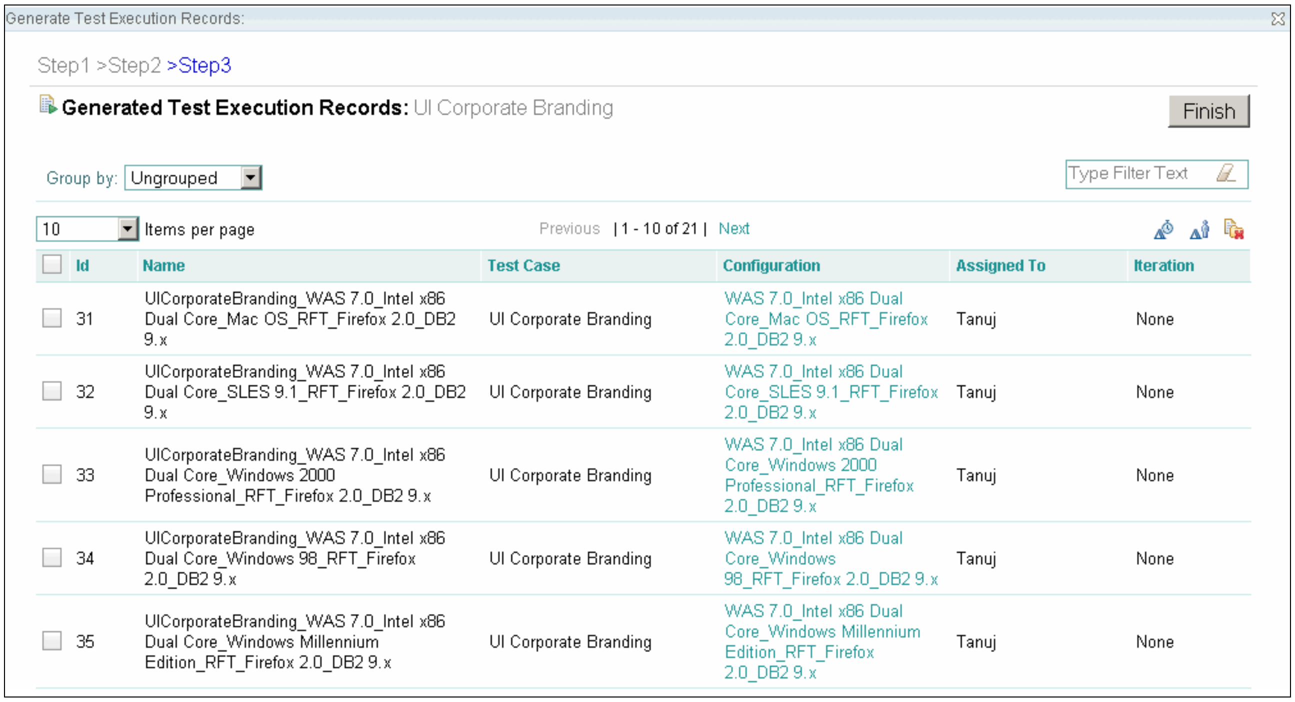
	Application Server	Browsers	CPU	DataBase	ManagementAgent	OperatingSystem	Test Adapter
☐	WAS 7.0	Firefox 2.0	Intel x86 Dual Core	DB2 9.x	Not Configured	Mac OS	RFT
☐	WAS 7.0	Firefox 2.0	Intel x86 Dual Core	DB2 9.x	Not Configured	SLES 9.1	RFT
☐	WAS 7.0	Firefox 2.0	Intel x86 Dual Core	DB2 9.x	Not Configured	Windows 2000 Professional	RFT
☐	WAS 7.0	Firefox 2.0	Intel x86 Dual Core	DB2 9.x	Not Configured	Windows 98	RFT
☐	WAS 7.0	Firefox 2.0	Intel x86 Dual Core	DB2 9.x	Not Configured	Windows Millennium Edition	RFT
☐	WAS 7.0	Firefox 2.0	Intel x86 Dual Core	DB2 9.x	Not Configured	Windows NT	RFT
☐	WAS 7.0	Firefox 2.0	Intel x86 Dual Core	DB2 9.x	Not Configured	Windows Vista Business	RFT
☐	WAS 7.0	Internet Explorer 7	Intel x86 Dual Core	DB2 9.x	Not Configured	Mac OS	RFT
☐	WAS 7.0	Internet Explorer 7	Intel x86 Dual Core	DB2 9.x	Not Configured	SLES 9.1	RFT
☐	WAS 7.0	Internet Explorer 7	Intel x86 Dual Core	DB2 9.x	Not Configured	Windows 2000 Professional	RFT

Previous |1 - 10 of 21| Next

Figure 11-37 Step 2 in the Generate Execution Work Items Wizard

9. After the wizard creates a preview of the generated execution work items, he selects the execution work items that he wants to keep.

10. From the Group By list (Figure 11-38), he groups the generated test execution records by selecting **Test Case** and then clicks **Finish**. The wizard generates the test execution records according to the criteria he has selected.


Generate Test Execution Records:
Step1 >Step2 >Step3
Finish
Type Filter Text
Previous | 1 - 10 of 21 | Next

Generated Test Execution Records: UI Corporate Branding

Group by: Ungrouped

10 Items per page

Id	Name	Test Case	Configuration	Assigned To	Iteration
31	UICorporateBranding_WAS 7.0_Intel x86 Dual Core_Mac OS_RFT_Firefox 2.0_DB2 9.x	UI Corporate Branding	WAS 7.0_Intel x86 Dual Core_Mac OS_RFT_Firefox 2.0_DB2 9.x	Tanuj	None
32	UICorporateBranding_WAS 7.0_Intel x86 Dual Core_SLES 9.1_RFT_Firefox 2.0_DB2 9.x	UI Corporate Branding	WAS 7.0_Intel x86 Dual Core_SLES 9.1_RFT_Firefox 2.0_DB2 9.x	Tanuj	None
33	UICorporateBranding_WAS 7.0_Intel x86 Dual Core_Windows 2000 Professional_RFT_Firefox 2.0_DB2 9.x	UI Corporate Branding	WAS 7.0_Intel x86 Dual Core_Windows 2000 Professional_RFT_Firefox 2.0_DB2 9.x	Tanuj	None
34	UICorporateBranding_WAS 7.0_Intel x86 Dual Core_Windows 98_RFT_Firefox 2.0_DB2 9.x	UI Corporate Branding	WAS 7.0_Intel x86 Dual Core_Windows 98_RFT_Firefox 2.0_DB2 9.x	Tanuj	None
35	UICorporateBranding_WAS 7.0_Intel x86 Dual Core_Windows Millennium Edition_RFT_Firefox 2.0_DB2 9.x	UI Corporate Branding	WAS 7.0_Intel x86 Dual Core_Windows Millennium Edition_RFT_Firefox 2.0_DB2 9.x	Tanuj	None

Figure 11-38 Step 3 in the Generate Execution Work Items wizard

11. He clicks **Save** on the test case. The execution records are now ready to be run. If necessary, he can also change their ownership and reassign them to new iterations.

At this point, Tanuj has completed all of his planning and construction work. He updated the test case that Tammy assigned to him in Act 1, wrote the associated test script, and created the execution work items to test all permutations of the browser test environments. When the build with Bob's change is ready, he is ready to execute his tests.

11.5.3 Tammy configures the test lab

Synopsis: Acts 2 and 3 of the storyboard are complete. Marco's team delivered the change, and Rebecca has conducted the integration build. Rebecca notifies the team that a new integration build is available. Tammy makes a request to deploy the build to the test lab.

This scene has the following objectives:

- Perform continual integration testing by refreshing the test servers with an updating integration build
- Reduce down-time for the test team by choosing a good build and managing the test lab

Tammy confirms the test resources

Goal: The goal is to validate the availability of test servers for use by the test team, and to allocate additional servers if needed.

In Act 1, Tammy submitted a request to a have an Apple Macintosh client, who is running Safari, added to her test lab. In this scene, she checks the status of her request and confirms that she has reserved the required resources for this iteration:

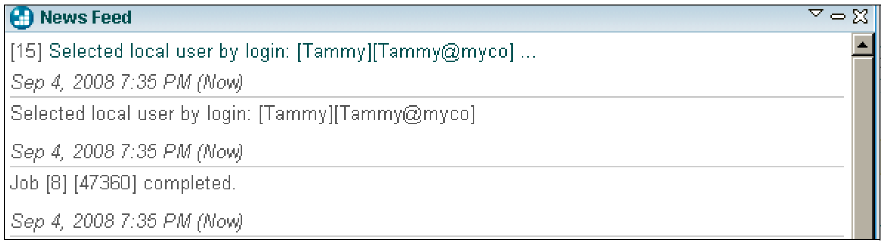
1. Tammy clicks a tab on her dashboard that contains viewlets for tracking her lab resources.
2. She focuses on the **Requests** viewlet and sees that her request for a new client machine is completed.
3. She reviews her reservations to confirm that she has all of the resources that are required for this iteration. In this case, Tammy has the Reservations viewlet on the dashboard. An alternative approach is to choose **Lab Management** → **All Reservations**.
4. She compares her reservations to the test plan.

Tammy inspects the build

Goal: The goal is to be aware of new integration builds and determine if they are suitable for testing.

This scene occurs right after Rebecca completes the integration build. As discussed in 9.6.2, “How Rational Build Forge is configured for this scenario” on page 380, Rebecca has set the Notify on Pass property for the project, which sends an e-mail notification to the team leads when the integration build passes. In addition, Rational Build Forge provides RSS feeds to which Tammy has subscribed.

1. Tammy receives an e-mail notification regarding the success of the build.
2. She also receives an RSS feed viewlet on her dashboard (Figure 11-39) that provides information about the events that are occurring on the build server.



[15] Selected local user by login: [Tammy][Tammy@myco] ...
Sep 4, 2008 7:35 PM (Now)
Selected local user by login: [Tammy][Tammy@myco]
Sep 4, 2008 7:35 PM (Now)
Job [8] [47360] completed.
Sep 4, 2008 7:35 PM (Now)

Figure 11-39 A news feed configured to view Build Forge server messages

3. She clicks the URL that opens the Build Forge Administration Messages window.

   Important: Clicking the link might open the Rational Build Forge user interface in the same browser window, losing the Rational Quality Manager user interface. Click the browser’s Back button to return to Rational Quality Manager.

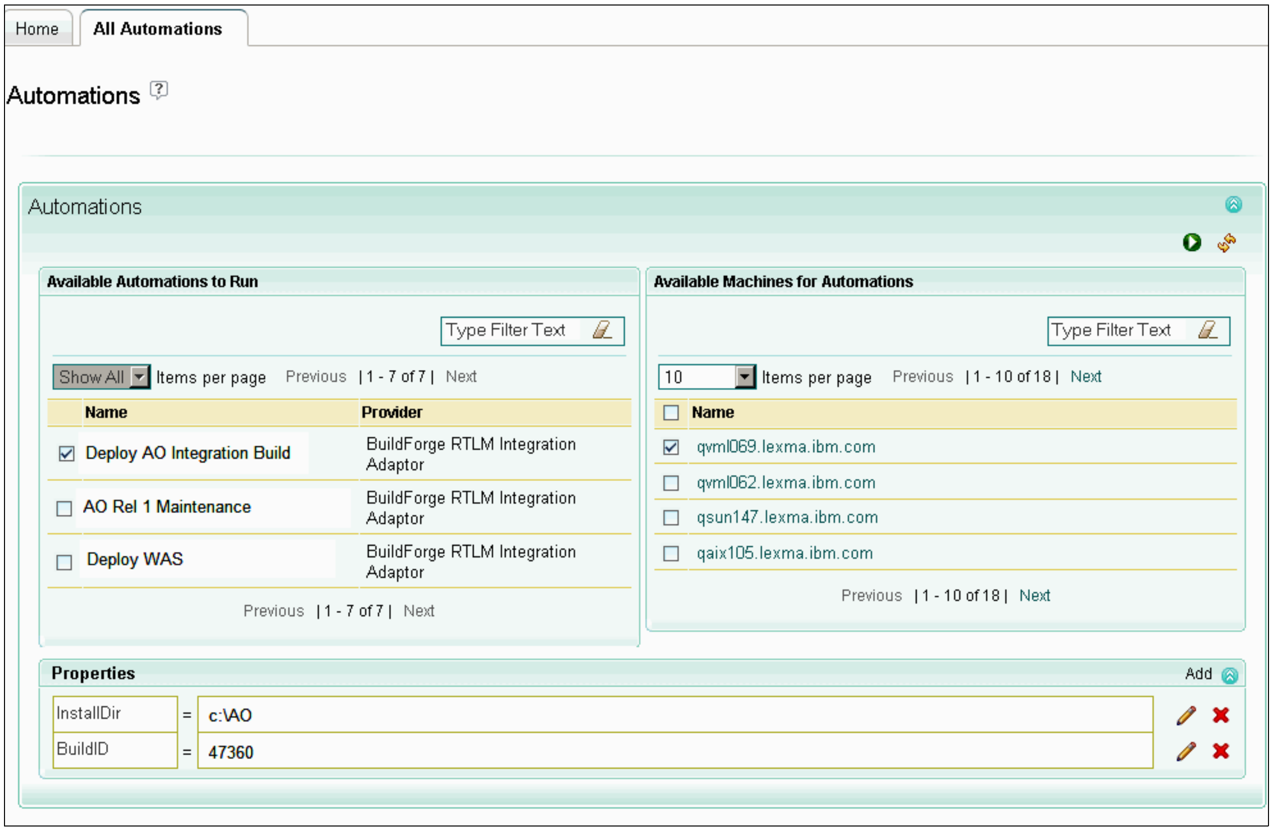
4. Tammy locates the integration build in the messages window and clicks the job link. From here, she can inspect any of the build steps, such as the step named *Execute unit tests*.
5. She can click the **Bill of Materials** menu to inspect the Job Steps or Step Manifests.
6. Satisfied, she deploys the build to the test servers.

Tip: See Chapter 9, “Rational Build Forge for enterprise integration build” on page 341, for information and images regarding the Rational Build Forge job results.

Tammy deploys the build

Goal: The goal is to deploy the latest build and test tools to the test environments.

Since Tammy has already reserved the machines, it is now a matter of deploying the latest build onto the resources. Tammy’s team has a set of automation jobs that streamline this process. In this scene, Tammy chooses an automation job and prepares the test server, as shown in Figure 11-40.



	Name	Provider
☑	Deploy AO Integration Build	BuildForge RTLM Integration Adaptor
☐	AO Rel 1 Maintenance	BuildForge RTLM Integration Adaptor
☐	Deploy WAS	BuildForge RTLM Integration Adaptor

☐	Name
☑	qvml069.lexma.ibm.com
☐	qvml062.lexma.ibm.com
☐	qsun147.lexma.ibm.com
☐	qaix105.lexma.ibm.com

InstallDir	=	c:\AO
BuildID	=	47360

Figure 11-40 Tammy deploying the build by using a predefined deployment automation

Tammy performs the following steps:

1. Tammy selects **Lab Management** → **All Automations**. The list of available automations is returned.
2. She selects the **Deploy AO Integration Build** automation.
3. She chooses an available machine from the list of Available Machines for Automations.
4. She provides a value for InstallDir.
5. She provides a value for the Build ID of the build that she just inspected in Rational Build Forge.
6. She clicks **Run**, which is represented by the green arrow in the upper right corner, and tracks the status as the automation runs.

When the automation is complete, Tammy executes a “smoke test” test suite against the new test environment. Assuming that all tests pass, Tammy notifies the team that the environment is ready for testing.

Tip: In this scenario, Tammy deploys the build by using an existing automation. An alternative approach is to send a request to a lab manager.

11.5.4 The team executes the tests

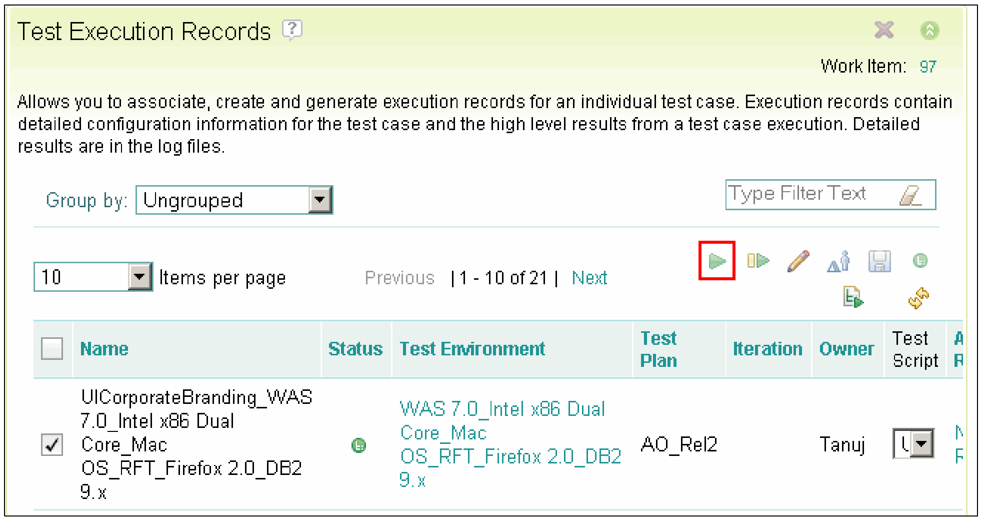
Synopsis: Tanuj and the team learn from Tammy that the new integration build has been deployed. He executes the test scripts. Then the results are logged, and the test status is updated.

Tanuj executes the tests

Tanuj learns from Tammy that the new integration build has been deployed. Tanuj is now ready to test the re-branded user interface from all permutations of browsers. Running a manual test requires a test case with an associated test script and one or more associated test execution records. In “Tanuj configures the tests for execution” on page 438, Tanuj created the test execution records. In this scene, he executes the tests.

To a run a manual test, Tanuj performs the following steps:

1. Tanuj selects **Execution** → **My Test Executions** to view the list of test execution records that he owns.
2. He selects one or more execution records, for example, Firefox 2.0 on Windows Vista, that he wants to run.
3. He clicks the **Execute Work Item** icon, which is highlighted in Figure 11-41.


Test Execution Records

Work Item: 97

Allows you to associate, create and generate execution records for an individual test case. Execution records contain detailed configuration information for the test case and the high level results from a test case execution. Detailed results are in the log files.

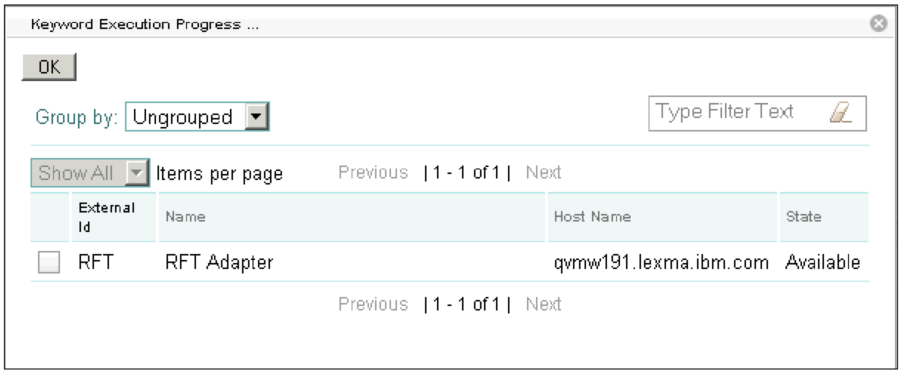
Group by: Ungrouped

10 Items per page   Previous | 1 - 10 of 21 | Next

	Name	Status	Test Environment	Test Plan	Iteration	Owner	Test Script
✓	UICorporateBranding_WAS 7.0_Intel x86 Dual Core_Mac OS_RFT_Firefox 2.0_DB2 9.x	●	WAS 7.0_Intel x86 Dual Core_Mac OS_RFT_Firefox 2.0_DB2 9.x	AO_Rel2		Tanuj	

Figure 11-41 Running a manual test in Rational Quality Manager

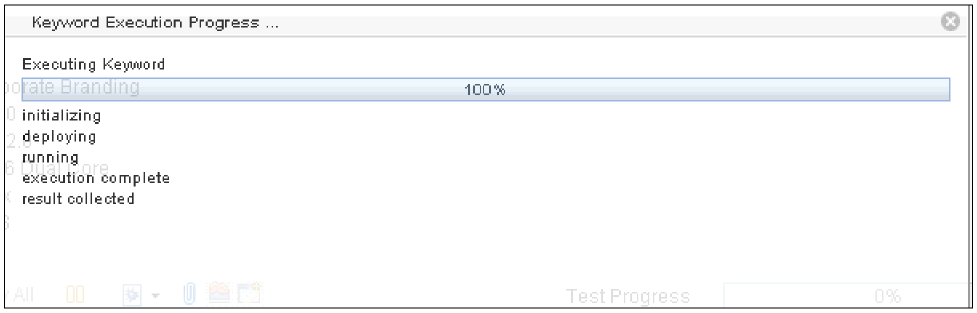
4. He performs an automated keyword test. When the system prompts Tanuj to select the adapter for running this automated test, he selects the check box next to **RFT**, which is the Functional Tester adapter in this example, that is running on his test server. Then he clicks **OK** (Figure 11-42).



	External Id	Name	Host Name	State
☐	RFT	RFT Adapter	qvmw191.lexma.ibm.com	Available

Figure 11-42   Choosing an automated test adapter

As the automated test execution occurs, a progress window is displayed similar to the example in Figure 11-43.



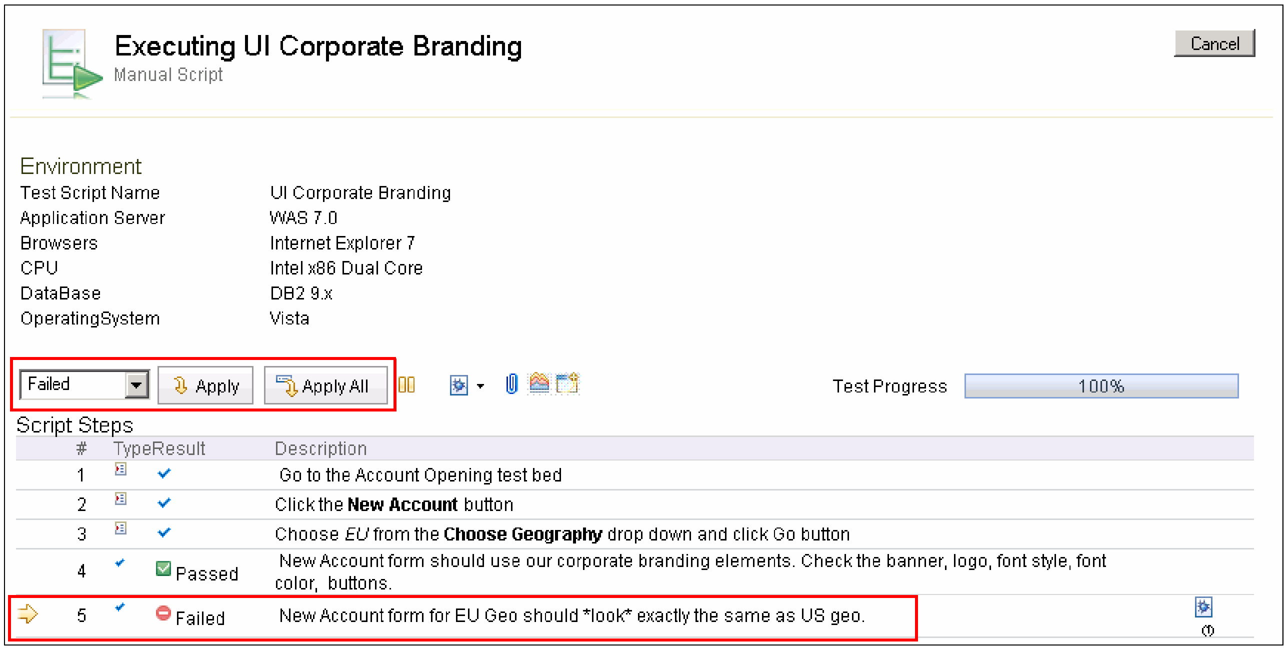
initializing
deploying
running
execution complete
result collected

Figure 11-43   Keyword Execution Progress window

When the test execution is complete, the result field for the first step is updated to indicate the verdict. From this point, the rest of the test involves manual testing.

5. He steps through the execution one statement at a time. As he completes each step, he selects a Verdict from the list and clicks **Apply** to assign a verdict to the step. Figure 11-44 shows a Verdict of Failed applied to Step 5 in the manual test. The following verdicts are possible:

   – Passed
   – Blocked
   – Failed
   – Inconclusive


Executing UI Corporate Branding
Manual Script

Environment

Test Script Name	UI Corporate Branding
Application Server	WAS 7.0
Browsers	Internet Explorer 7
CPU	Intel x86 Dual Core
DataBase	DB2 9.x
OperatingSystem	Vista

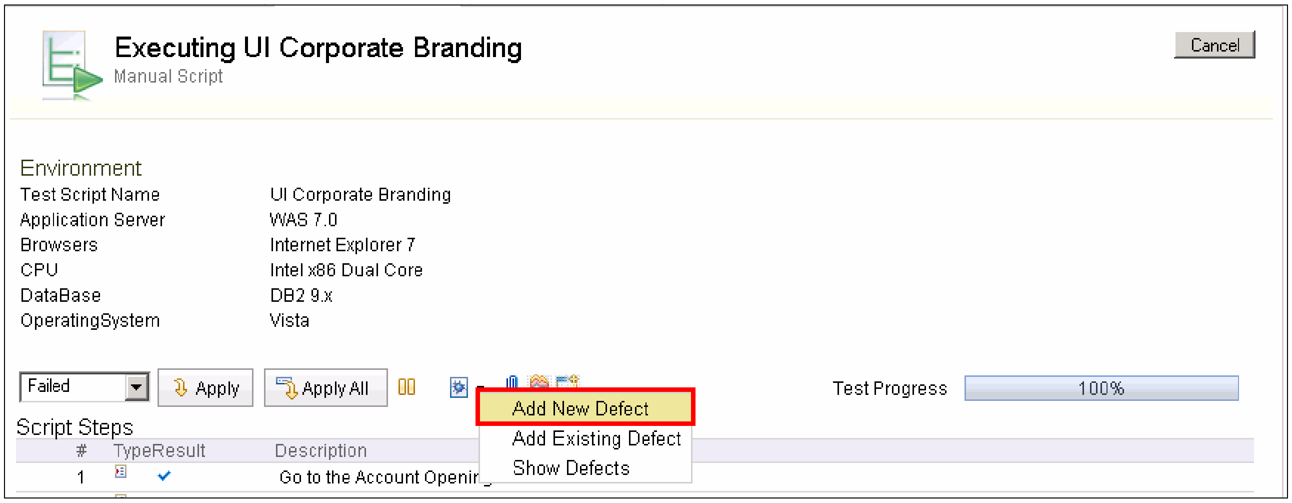
Failed | Apply | Apply All | Test Progress 100%

Script Steps

#	Type	Result	Description
1		✓	Go to the Account Opening test bed
2		✓	Click the **New Account** button
3		✓	Choose *EU* from the **Choose Geography** drop down and click Go button
4	✓	Passed	New Account form should use our corporate branding elements. Check the banner, logo, font style, font color, buttons.
5	✓	Failed	New Account form for EU Geo should *look* exactly the same as US geo.

Figure 11-44   Manual script execution showing a failed step and associated defect

6. In step 5 in his test script, he notices that the logo for the EU Account form is placed differently from the US form. To submit a defect, Tanuj clicks the **Show Defects** icon on the toolbar and chooses **Add New Defect** shown in Figure 11-45.


Executing UI Corporate Branding
Manual Script

Environment

Test Script Name	UI Corporate Branding
Application Server	WAS 7.0
Browsers	Internet Explorer 7
CPU	Intel x86 Dual Core
DataBase	DB2 9.x
OperatingSystem	Vista

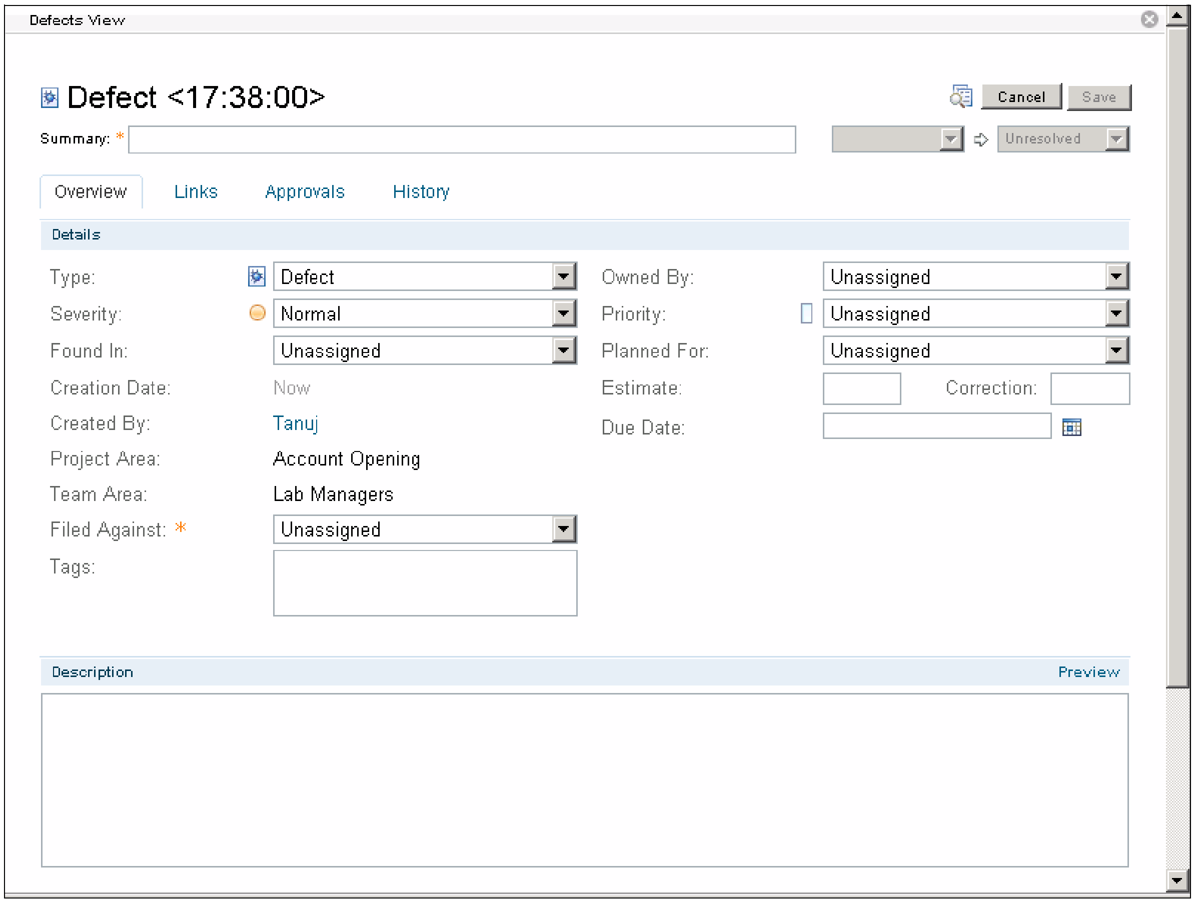
Failed | Apply | Apply All | Test Progress 100%

Script Steps

#	Type	Result	Description
1		✓	Go to the Account Openin

Figure 11-45   Adding and associating a defect with a line in a test execution

7. In the Defects View form (Figure 11-46):
   a. For Type, he keeps the setting **Defect**.
   b. For Severity, he selects **Normal**.
   c. For Filed Against, he chooses **Account Opening**.
   d. In the Tags box, he adds `Credit_check`.
   e. For Owned by, he selects **Diedrie**.
   f. For Priority, he selects **Medium**.
   g. For Due Date, he sets it to equal the last day of the iteration.
   h. For Description, he provides meaningful text.
   i. He clicks the **Links Tab** and click Brows to add an attachment. Then he chooses a .jpg file from his file system. Earlier he made a window capture of both forms by placing them side by side to identify the difference.
   j. When he is satisfied with the information, he clicks **Save**.



Type:	Defect	Owned By:	Unassigned
Severity:	Normal	Priority:	Unassigned
Found In:	Unassigned	Planned For:	Unassigned
Creation Date:	Now	Estimate:	Correction:
Created By:	Tanuj	Due Date:	
Project Area:	Account Opening		
Team Area:	Lab Managers		
Filed Against: *	Unassigned		
Tags:			

Figure 11-46 A New Defect form

8. He sets the Verdict to **Failed** for this step. When the verdict on the last step of the test is set, the Test execution is complete.
9. From the Execution Result tab that is displayed, Tanuj reviews the results and does not make any changes. If necessary, he can modify the verdict and the weight distribution from the execution results. After he makes an edit, he must click **Save** to save any changes.

In addition Tanuj can add comments or attachments:

- To add an optional comment to one or several statements:
  a. From the toolbar, he clicks the **Show Comments** icon, and the Comments View is displayed.
  b. He types the comment in the Write Comments field.
  c. He clicks the X in the upper right corner of the view to close it. The comments icon displays within each statement that has a comment.
- To add an optional attachment to one or several statements, for example, he can create a window capture of the application under test and attach it to the statement. To attach a file to a statement:
  a. From the toolbar, he clicks the **Show Attachments** icon, and the Attachments View is displayed.
  b. He browses to the file, and clicks **Open**.
  c. He clicks the X in the upper right corner of the view to close it. The attachment icon displays within each statement that has an attachment.

Restriction: The defect report that is created in this scenario is stored in Rational Quality Manager. Diedrie needs some form of notification to fix and resolve the defect. One approach is for her to have a license for Rational Quality Manager. She can log in to the Rational Quality Manager user interface to work with defects that are reported by the test team.

Another approach is to use the ClearQuest Connector to create a record in Rational ClearQuest, which then synchronizes to Rationa Team Concert. For this to work, the ClearQuest Connector must be configured to work with the Rational ClearQuest defect record. At the time of writing this book, the ClearQuest Connector did not work with ClearQuest Application Lifecycle Management (ALM) record types. The restriction occurs when a record must be created in Rational ClearQuest by the connector. Note that synchronization works when records are created in Rational ClearQuest and pushed to Rational Team Concert or Rational Quality Manager, as documented in this book.

Tammy runs a security scan

Goal: The goal is to identify security vulnerabilities before approving the release for production.

Tammy assigns the security and compliance scan test suites to herself. With the functional test coming to a close and performance tests underway, she runs the scans late in the evening after the test team has left for the day. The usual cycle of defect submission and resolution occurs until the test exit criteria is met and is complete

Tammy creates a security test script

In this scene, Tammy writes a Rational AppScan Tester Edition test script to run a security scan:

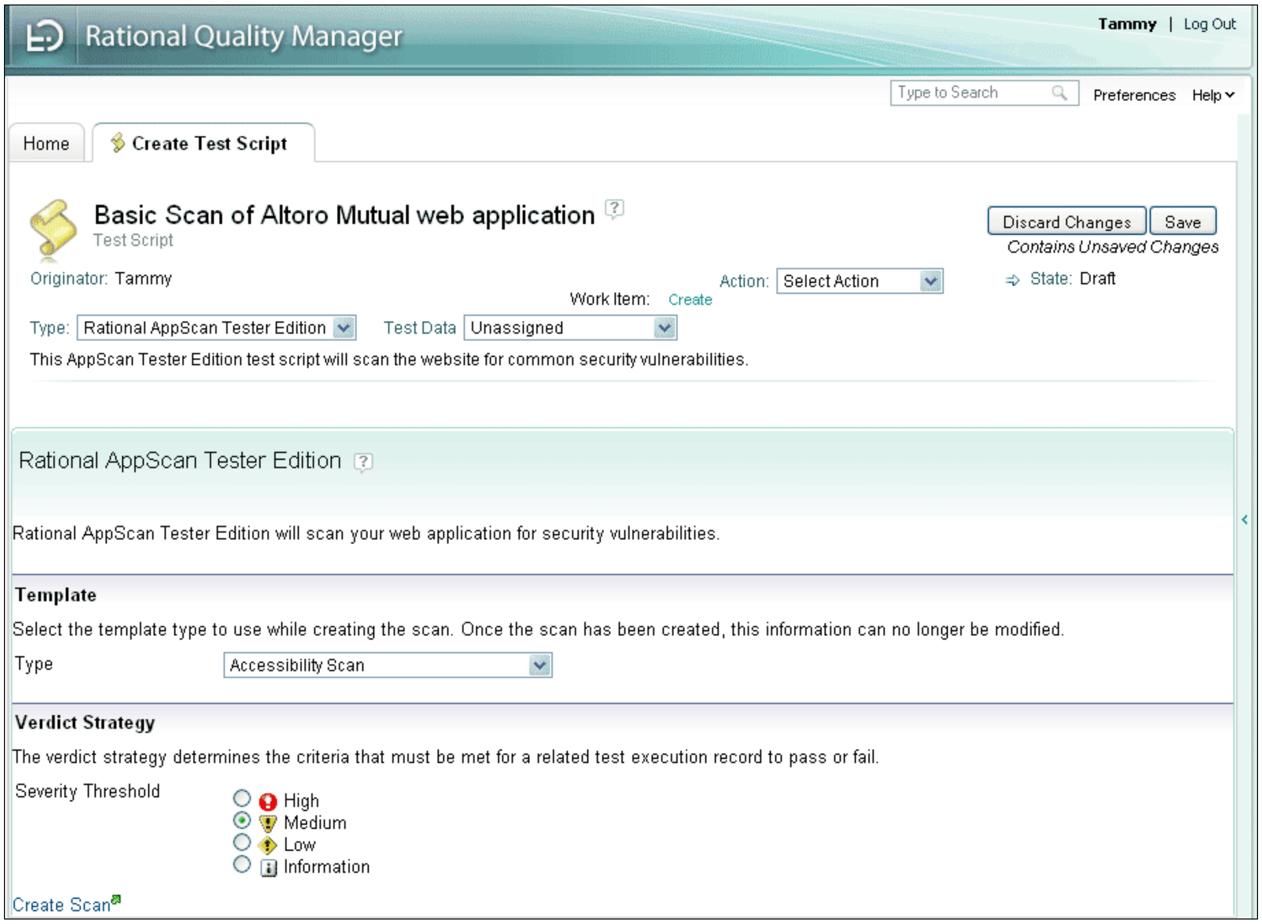
1. Tammy selects **Construction** → **Create Test Script**.
2. On the **Create Test Script** tab:
   a. She provides a name and description.
   b. For Owner, she selects herself.

c. For Type, she selects Rational AppScan Tester Edition, which populates the test script with information specific to performing security tests, as shown in Figure 11-47.

d. For Template, she selects one to use for the creation of the security scan. For this scan, she selects the **Altoro Mutual** template, which has been preconfigured with the URL and login information of the system under test.

e. For Verdict Strategy, she keeps the *default* setting of Severity Threshold, which is **Medium**. This setting means that the test execution will pass only if no Medium or High severity issues are found by the security scan.

f. She clicks the **Create Scan** link to create an associated security scan in Rational AppScan Tester Edition. Depending on Tammy's browser settings, either a new tab or a browser window opens where she can edit the scan.



Basic Scan of Altoro Mutual web application
Test Script
Contains Unsaved Changes
Originator: Tammy
Action: Select Action
Work Item: Create
State: Draft
Type: Rational AppScan Tester Edition
Test Data: Unassigned
This AppScan Tester Edition test script will scan the website for common security vulnerabilities.

Rational AppScan Tester Edition

Rational AppScan Tester Edition will scan your web application for security vulnerabilities.

Template

Select the template type to use while creating the scan. Once the scan has been created, this information can no longer be modified.

Type: Accessibility Scan

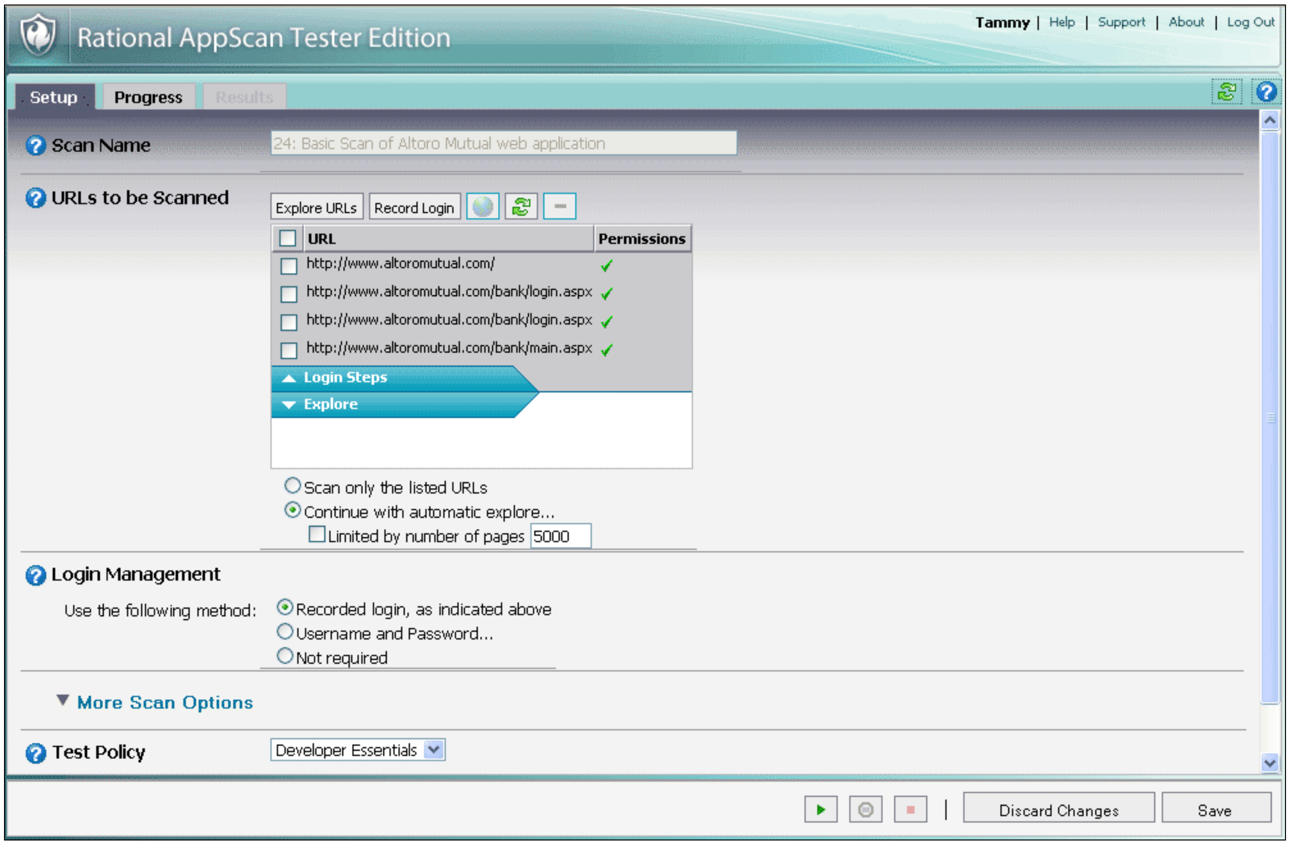
Verdict Strategy

The verdict strategy determines the criteria that must be met for a related test execution record to pass or fail.

Severity Threshold: High [✓ Medium] Low Information

Figure 11-47 Creating a Rational AppScan Tester Edition test script

3. Tammy reviews the scan in Rational AppScan Tester Edition (Figure 11-48):
   a. For URLs to be scanned, she confirms the URL for the system under test.
   b. For Test Policy, she confirms the one to use during the scan. The test policy is a predefined set of security tests that the scan executes.
   c. She determines that she has no need to edit the scan. She closes the window or tab to return to Rational Quality Manager.



Scan Name: 24: Basic Scan of Altoro Mutual web application

URLs to be Scanned

URL	Permissions
http://www.altoromutual.com/	✓
http://www.altoromutual.com/bank/login.aspx	✓
http://www.altoromutual.com/bank/login.aspx	✓
http://www.altoromutual.com/bank/main.aspx	✓

Scan only the listed URLs [✓ Continue with automatic explore...]
Limited by number of pages 5000

Login Management
Use the following method: [✓ Recorded login, as indicated above] Username and Password... Not required

Test Policy: Developer Essentials

Figure 11-48 Reviewing a Rational AppScan Tester Edition scan

Tammy runs a security scan

Tammy has already defined a test script of the Rational AppScan Tester Edition type, to perform a standard security scan of the Web application. A test case has also been defined to ensure that there are no security vulnerabilities with the security theme nor associated with the existing AppScan Tester Edition test script. The test case is part of her overall test plan. In this act, she reviews the settings of the test case, runs the security scan, and evaluates the results.

Tammy performs the following three tasks:

- Reviews the test case and its associated requirements
- Runs a security scan
- Evaluates the results and log a defect

To review the test case and its associated requirements, Tammy performs the following steps:

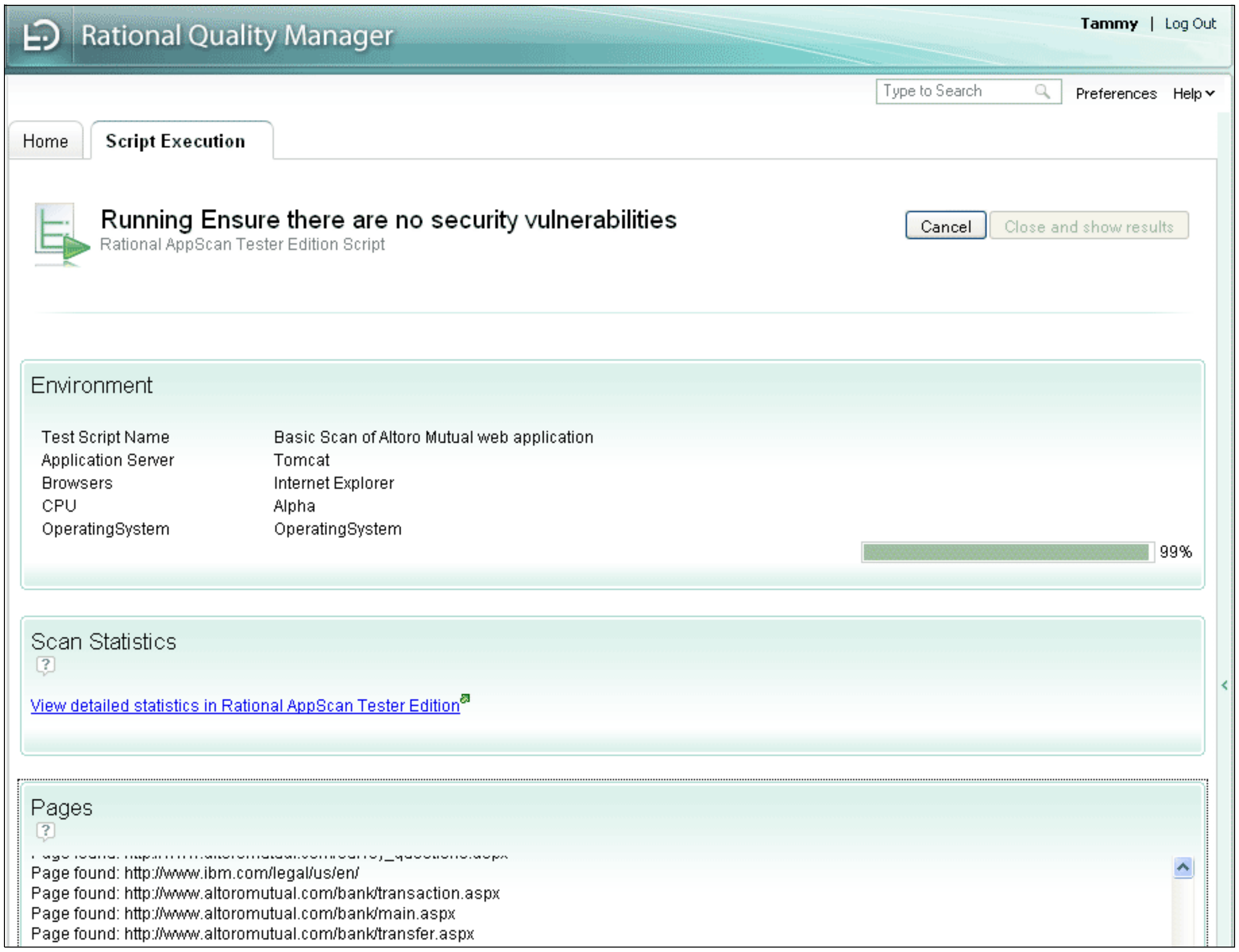
1. Tammy opens her test plan and clicks **Test Cases**.
2. She identifies the **Security** test case and clicks to open it.
3. She clicks **Requirements** to review the security requirements that are associated with the test case.

4. She clicks **Test Scripts** to review the test scripts that are associated with the test case.
5. She clicks the **AppScan Security Scan** test script to view it and confirm that it is correct.

To execute the security scan, Tammy does the following steps from within the open test case:

1. She clicks **Test Execution Records**.
2. She checks the security execution item and clicks **Execute**.

   The security scan runs against the system under test while reporting progress and pages that are scanned on the Script Execution tab (Figure 11-49).
3. Tammy clicks **View detailed statistics** in Rational AppScan Tester Edition.



Running Ensure there are no security vulnerabilities
Rational AppScan Tester Edition Script

Environment

Test Script Name	Basic Scan of Altoro Mutual web application
Application Server	Tomcat
Browsers	Internet Explorer
CPU	Alpha
OperatingSystem	OperatingSystem

Scan Statistics

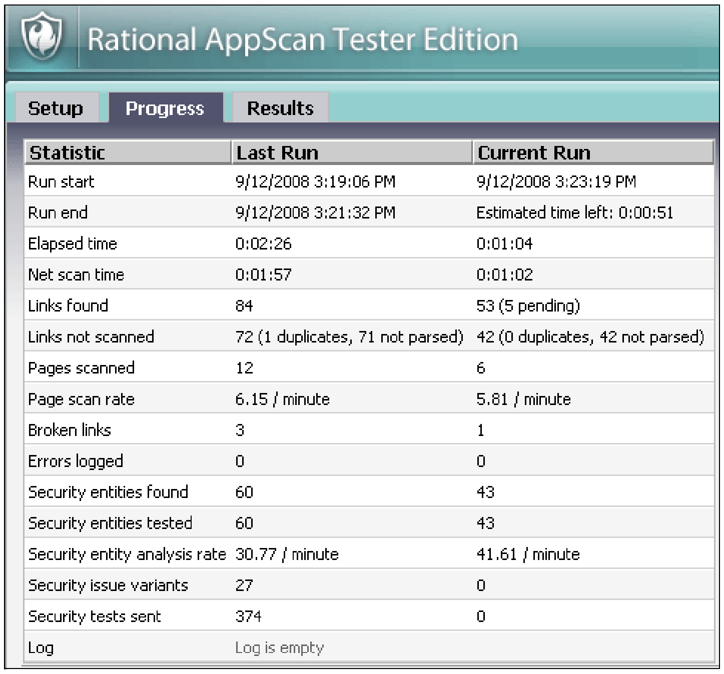
View detailed statistics in Rational AppScan Tester Edition

Pages

```
Page found: http://www.ibm.com/legal/us/en/
Page found: http://www.altoromutual.com/bank/transaction.aspx
Page found: http://www.altoromutual.com/bank/main.aspx
Page found: http://www.altoromutual.com/bank/transfer.aspx
```

Figure 11-49 Rational AppScan Tester Edition Test Script Execution

4. She views the information about the scan while it is running in Rational AppScan Tester Edition on the Progress tab as shown in Figure 11-50. When she is done reviewing the detailed statistics, she closes the window or tab to return to Rational Quality Manager.

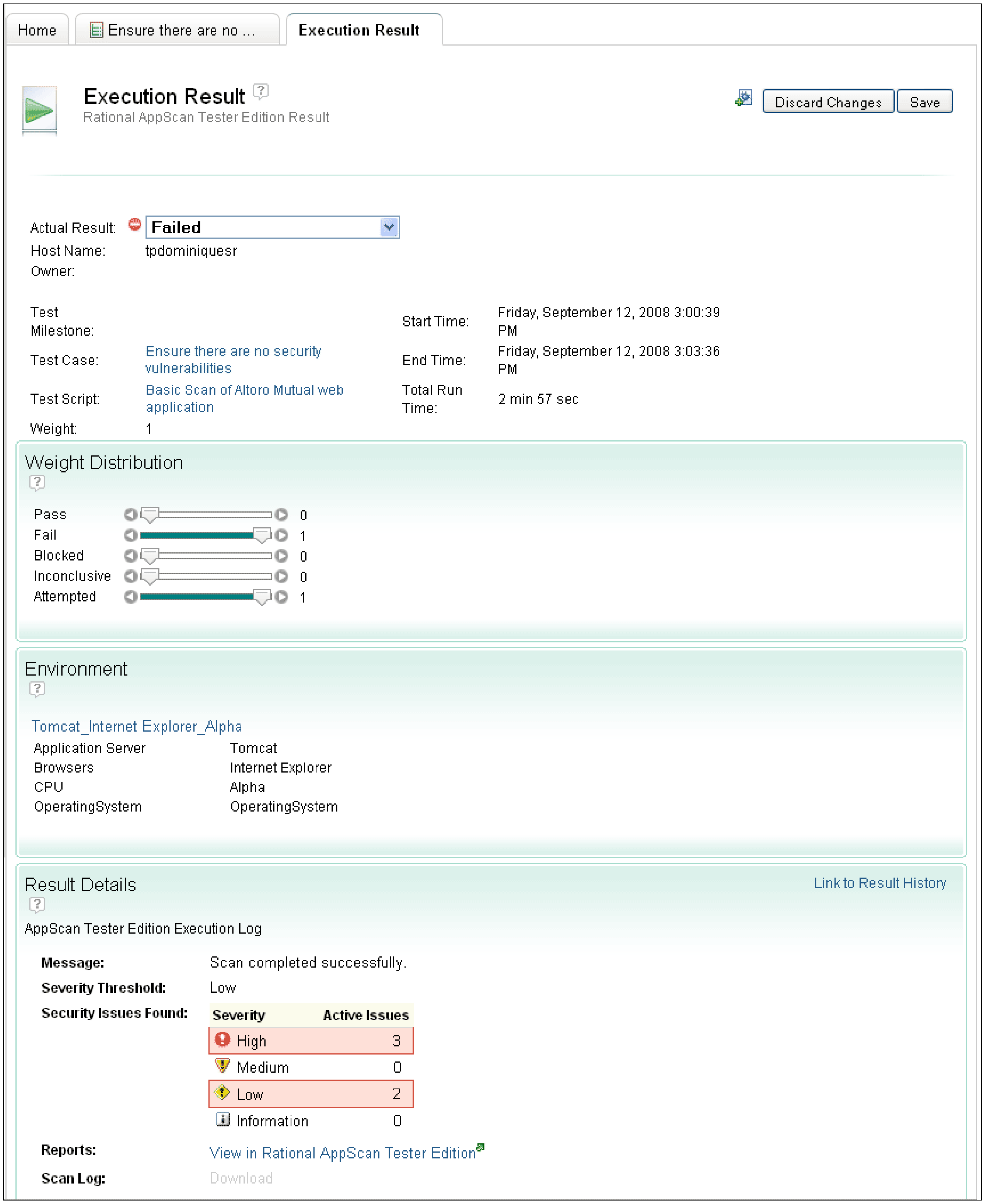


Statistic	Last Run	Current Run
Run start	9/12/2008 3:19:06 PM	9/12/2008 3:23:19 PM
Run end	9/12/2008 3:21:32 PM	Estimated time left: 0:00:51
Elapsed time	0:02:26	0:01:04
Net scan time	0:01:57	0:01:02
Links found	84	53 (5 pending)
Links not scanned	72 (1 duplicates, 71 not parsed)	42 (0 duplicates, 42 not parsed)
Pages scanned	12	6
Page scan rate	6.15 / minute	5.81 / minute
Broken links	3	1
Errors logged	0	0
Security entities found	60	43
Security entities tested	60	43
Security entity analysis rate	30.77 / minute	41.61 / minute
Security issue variants	27	0
Security tests sent	374	0
Log	Log is empty	

Figure 11-50 Rational AppScan Tester Edition scan statistics

After the execution is complete, Tammy does the following steps to review the results and log a defect:

1. She clicks **Close and show results**.
2. On the **Execution Result** tab (Figure 11-51 on page 453), Tammy notices that the result is **Failed** and reviews the number of issues that were found to see that both Medium and High severity issues were found.
3. She clicks the **View in Rational AppScan Tester Edition** link to view the detailed reports.



Execution Result
Rational AppScan Tester Edition Result

Actual Result:	Failed
Host Name:	tpdominiquesr
Owner:	
Test Milestone:	
Test Case:	Ensure there are no security vulnerabilities
Test Script:	Basic Scan of Altoro Mutual web application
Weight:	1
Start Time:	Friday, September 12, 2008 3:00:39 PM
End Time:	Friday, September 12, 2008 3:03:36 PM
Total Run Time:	2 min 57 sec

Weight Distribution

Pass	0
Fail	1
Blocked	0
Inconclusive	0
Attempted	1

Environment

Tomcat_Internet Explorer_Alpha

Application Server	Tomcat
Browsers	Internet Explorer
CPU	Alpha
OperatingSystem	OperatingSystem

Result Details

Link to Result History

AppScan Tester Edition Execution Log

Message: Scan completed successfully.

Severity Threshold: Low

Security Issues Found:

Severity	Active Issues
High	3
Medium	0
Low	2
Information	0

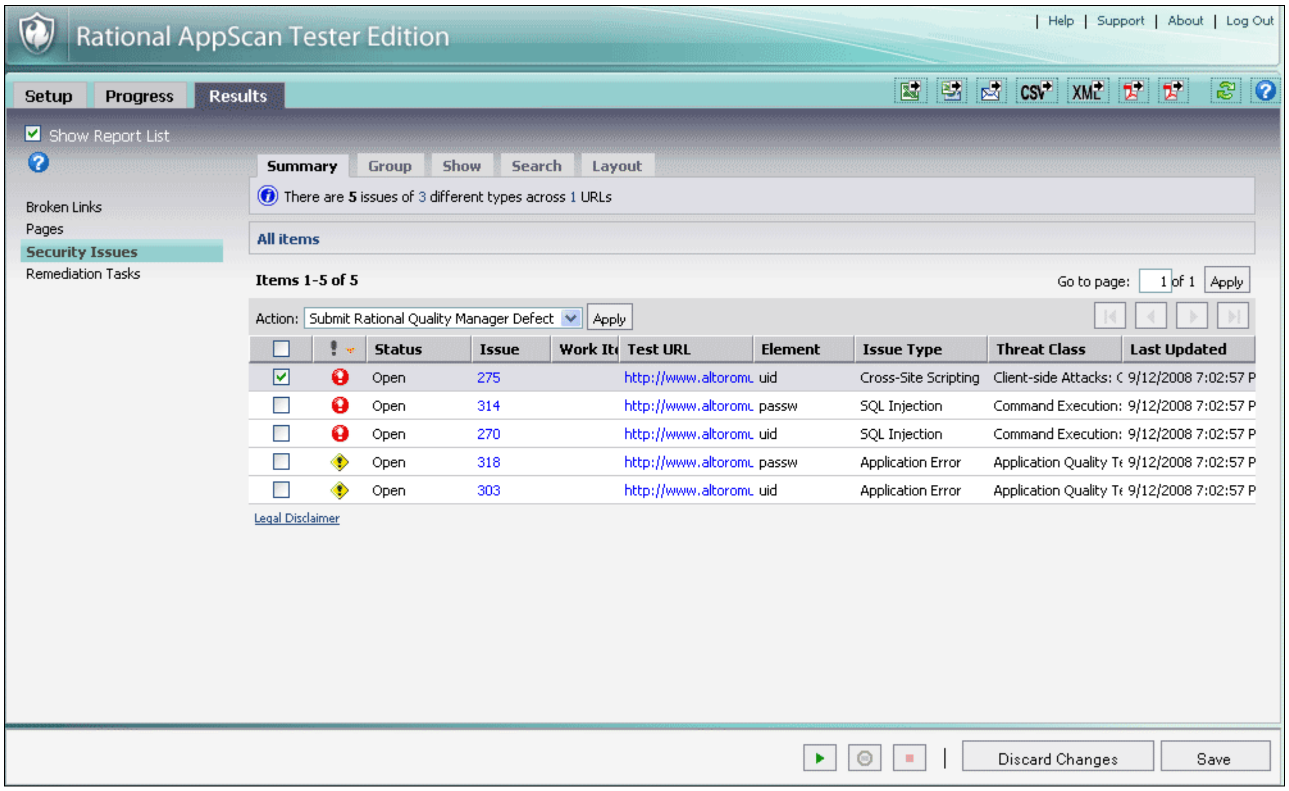
Reports: View in Rational AppScan Tester Edition

Scan Log: Download

Figure 11-51 Viewing the execution results

4. Tammy reviews the issues and decides to log a defect for a high severity cross-site scripting vulnerability that was found on the login page.

5. As shown in Figure 11-52, she selects the issue. For Action, she verifies that the **Submit Rational Quality Manager** option is selected and clicks **Apply**.



There are 5 issues of 3 different types across 1 URLs

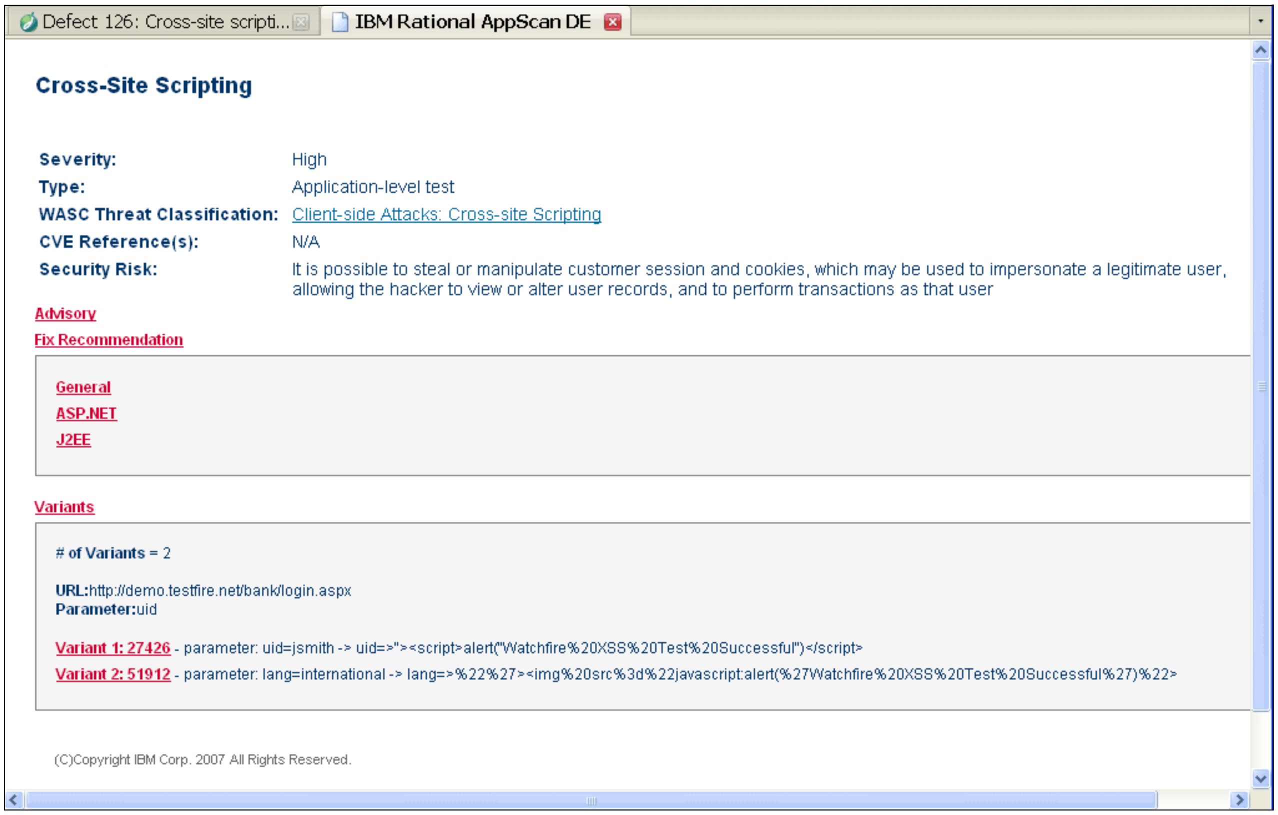
Items 1-5 of 5

☐	!	Status	Issue	Work Ite	Test URL	Element	Issue Type	Threat Class	Last Updated
✓	●	Open	275		http://www.altoromu	uid	Cross-Site Scripting	Client-side Attacks: C	9/12/2008 7:02:57 P
☐	●	Open	314		http://www.altoromu	passw	SQL Injection	Command Execution:	9/12/2008 7:02:57 P
☐	●	Open	270		http://www.altoromu	uid	SQL Injection	Command Execution:	9/12/2008 7:02:57 P
☐	◆	Open	318		http://www.altoromu	passw	Application Error	Application Quality Te	9/12/2008 7:02:57 P
☐	◆	Open	303		http://www.altoromu	uid	Application Error	Application Quality Te	9/12/2008 7:02:57 P

Figure 11-52 Rational AppScan Tester Edition test script execution results

6. When a link to the defect is added to the Work Item column, she clicks the link to review the created defect.
7. Tammy notices that the Severity field is set to **Major** and the Tags field contains the term `appscan`. Because the Owner field is unassigned, she sets it to the developer who is responsible for security. She also sets the Priority to **High**.

8. She looks at the Discussion header and clicks the attachment name to view the attachment that was automatically created to help the developer resolve the security vulnerability. The attachment includes information about the risk, a fix recommendation, and information about different variants of this issue (Figure 11-53).



Cross-Site Scripting

Severity:	High
Type:	Application-level test
WASC Threat Classification:	Client-side Attacks: Cross-site Scripting
CVE Reference(s):	N/A
Security Risk:	It is possible to steal or manipulate customer session and cookies, which may be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user

Advisory

Fix Recommendation

General
ASP.NET
J2EE

Variants

# of Variants = 2

URL:http://demo.testfire.net/bank/login.aspx
Parameter:uid

```
Variant 1: 27426 - parameter: uid=jsmith -> uid=>"><script>alert("Watchfire%20XSS%20Test%20Successful")</script>
Variant 2: 51912 - parameter: lang=international -> lang=>%22%27><img%20src%3d%22javascript:alert(%27Watchfire%20XSS%20Test%20Successful%27)%22>
```

(C)Copyright IBM Corp. 2007 All Rights Reserved.

Figure 11-53 Defect advisory and fix recommendation

9. She closes the tab or window that contains the attachment to return to the defect.
10. When she is satisfied with the information, she clicks **Save**.

11.5.5 Monitoring quality

Synopsis: One week later, Tammy reviews the test plan and evaluates the exit criteria. She uses the Exit Criteria that was established in the test plan to measure their progress and make adjustments. The usual cycle of defect submission and resolution occurs until the Test Exit Criteria is met and is complete.

On the final day of the iteration, Tammy confirms that the solution meets the exit criteria. She informs Patricia that the quality has met their expectations.

In this scene, we discuss how Tammy monitors the quality of the solution for this iteration. For more information about the use of metrics, see 11.7, “Planning and measuring success in quality management” on page 467.

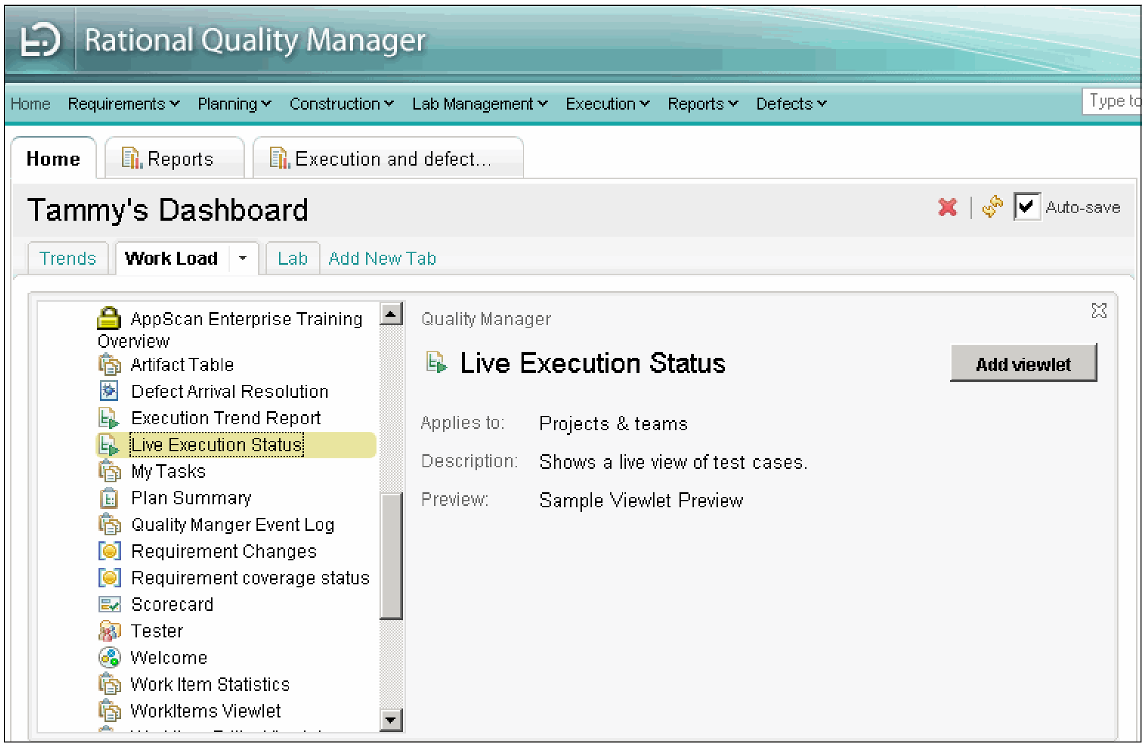
Rational Quality Manager Dashboard

Goal: The goal is to maintain a real-time view of quality and team health.

Tammy and the team use their dashboards to access a live overview of the quality health indicators. The dashboard helps the team identify significant changes in trends.

As shown in Figure 11-54, Tammy added tabs to her dashboard. She created a tab that she calls *Trends* and populated it with viewlets that provide trend and real-time information, such as the Execution Trend Report, Requirements changes, Requirement coverage status, and Defect Arrival Resolution.

She also created a tab called *Work Load*, which she uses to track work items for herself and her team. Finally, she created a tab called *Lab*, which she populated with viewlets from Rational Lab Manager that give her insight into the status of her reservations and deployment automations, along with information about the use of lab resources.



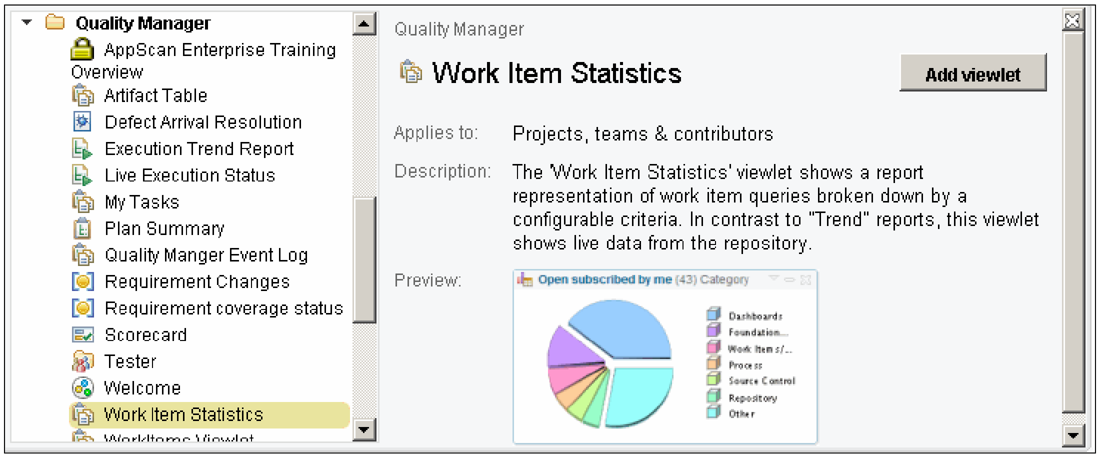
Applies to: Projects & teams

Description: Shows a live view of test cases.

Preview: Sample Viewlet Preview

Figure 11-54 Adding tabs to the dashboard to organize viewlets

Each of these tabs are populated with viewlets that are available with Rational Quality Manager. Figure 11-55 shows an example of some of the viewlets that are available for monitoring quality.


Quality Manager
AppScan Enterprise Training Overview
Artifact Table
Defect Arrival Resolution
Execution Trend Report
Live Execution Status
My Tasks
Plan Summary
Quality Manger Event Log
Requirement Changes
Requirement coverage status
Scorecard
Tester
Welcome
Work Item Statistics
WorkItems Viewlet

Quality Manager

Work Item Statistics

Add viewlet

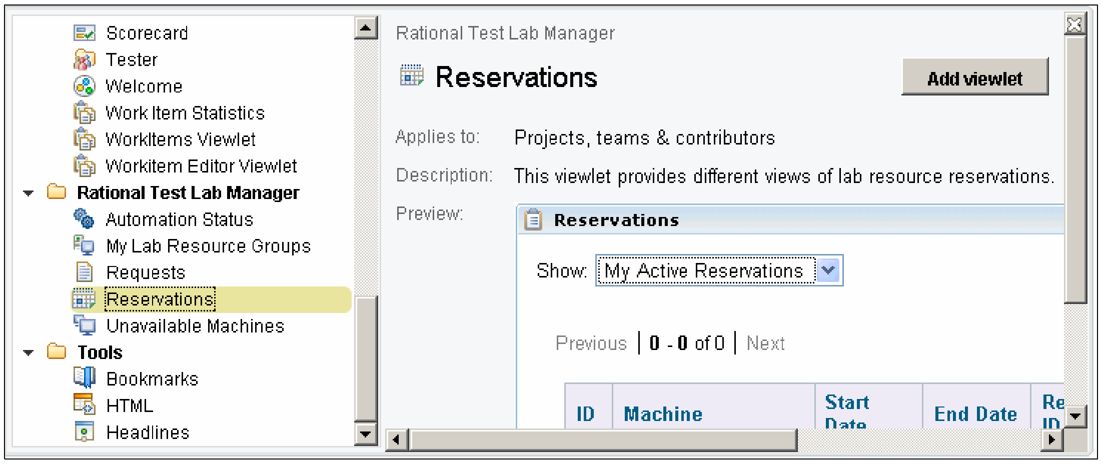
Applies to: Projects, teams & contributors

Description: The 'Work Item Statistics' viewlet shows a report representation of work item queries broken down by a configurable criteria. In contrast to "Trend" reports, this viewlet shows live data from the repository.

Preview:

Figure 11-55 Rational Quality Manager dashboard viewlets

In addition, when using Rational Test Lab Manager, viewlets are available for managing Lab Resources, shown in Figure 11-56.


Scorecard
Tester
Welcome
Work Item Statistics
WorkItems Viewlet
Workitem Editor Viewlet
Rational Test Lab Manager
Automation Status
My Lab Resource Groups
Requests
Reservations
Unavailable Machines
Tools
Bookmarks
HTML
Headlines

Rational Test Lab Manager

Reservations

Add viewlet

Applies to: Projects, teams & contributors

Description: This viewlet provides different views of lab resource reservations.

Preview:

Reservations

Show: My Active Reservations

Previous | 0 - 0 of 0 | Next

ID	Machine	Start Date	End Date	Re ID

Figure 11-56 Rational Test Lab Manager provides viewlets for managing lab resources

Tammy uses a combination of viewlets, reports, and exit criteria to determine when the iteration is complete. This is a combination that confirms that all planned work is completed, all requirements have been validated, and defects have been fixed to the agreed upon level. In her case, all Priority 1 defects have been closed. Satisfied with the quality of iteration, Tammy communicates the news to her team and the team leads.

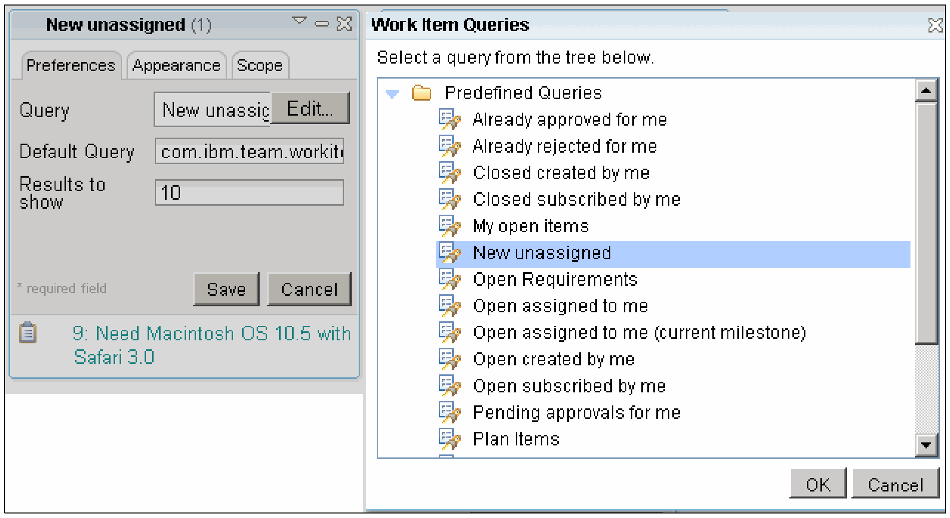
Rational Quality Manager reports

Rational Quality Manager includes a set of reports that Tammy uses to monitor the quality of the solution and along with the status of the test execution. Reports are found by selecting **Reports** → **View Reports**. The reports are grouped by type to help you craft your quality monitoring strategy. In addition, you can create and run your own reports.

Assessing team health

Tammy uses the test management system as the basis for the daily discussions. The team talks about current work assignments and work load per tester. Tammy and the team use viewlets on their dashboards to view real-time status of workload.

Tammy uses the work item statistics, WorkItems viewlet, and WorkItem Editor Viewlet to create views that give her insight into the team health. For example, she creates several WorkItems viewlets with different queries that provide her with insight into the status of the work items. For example, she configures one viewlet to show all new work items that are unassigned as shown in Figure 11-57.



Query: New unassig

Default Query: com.ibm.team.workit

Results to show: 10

* required field

9: Need Macintosh OS 10.5 with Safari 3.0

Select a query from the tree below.

Predefined Queries
- Already approved for me
- Already rejected for me
- Closed created by me
- Closed subscribed by me
- My open items
- New unassigned
- Open Requirements
- Open assigned to me
- Open assigned to me (current milestone)
- Open created by me
- Open subscribed by me
- Pending approvals for me
- Plan Items

Figure 11-57 A viewlet configured to show new unassigned work

As shown by the choice of queries, she can create a variety of viewlets that provide real-time transparent information regarding the work items on her team. She can click any of the work items to modify them, such as assigning the unassigned work items, or reassigning work to other team members to balance the load.

She also runs the Tester Report by using TER Count, which is in the Execution folder of the Reports to monitor test execution records (TER) per tester.

She uses the Test Case Review report, under the Test Case folder, to see the status (draft, review) of all test cases in her test plan. By doing this, she can ensure that all test cases are completed for the solution.

She can also run the Test Cases by Defect report to see which test cases are impacted by defects.

Monitoring quality

Tammy uses a combination of her dashboard and reports to measure quality and execution status. For Tammy, it is important to measure quality by tracking when the team meets the exit criteria. As you might recall, in 5.5.4, “Tammy updates the solution test plan” on page 139, Tammy defined the exit criteria as follows:

- 100% of all priority 1 defects fixed
- 100% requirements test coverage
- 100% user interface using the corporate branding

To monitor quality, she created three viewlets on her dashboard that provide information for the criteria.

100% of all priority 1 defects fixed

To track against the first exit criteria, Tammy created a viewlet on her dashboard to monitor defects:

1. For this specific metric, she creates a custom query, which is shown in Figure 11-58 by using the following steps:
   a. She selects **Defects** → **Go To Defects**.
   b. She clicks **Create Query** from the left navigation menu.
   c. She clicks the green plus sign to add a condition, and selects **Priority** from the list. Then she clicks **Add attribute condition**.
   d. For Priority, she selects **High**.
   e. In the Add Condition box, she selects **Status** and clicks **Add attribute condition**.
   f. She selects the **Unresolved** check box.
   g. In the Add Condition box, she selects **Type** from the list and clicks **Add attribute condition**.
   h. From the list box of values, she selects **Defect**.
   i. For Name, she types `High Priority Defects`.
   j. She clicks **Save**.

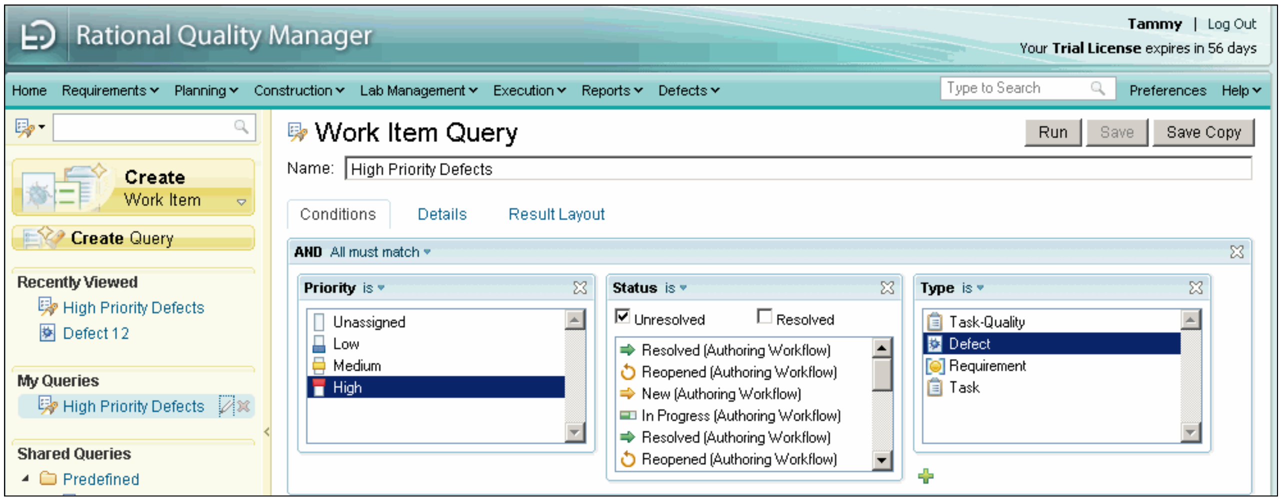


Figure 11-58 A custom query to find all unresolved, high priority defects

2. She clicks the **Home** tab to view the dashboard and used the following steps to add the viewlet:
   a. She clicks **Add Viewlet**
   b. She scrolls to the **Quality Management** section, selects **Workitems Viewlet**, and clicks **Add viewlet**.
   c. After the viewlet is added to the dashboard, she clicks the **X** above add viewlet to close it.

d. In the new viewlet, she clicks the downward arrow to expose the menu options, and selects **Edit Settings** (Figure 11-59).

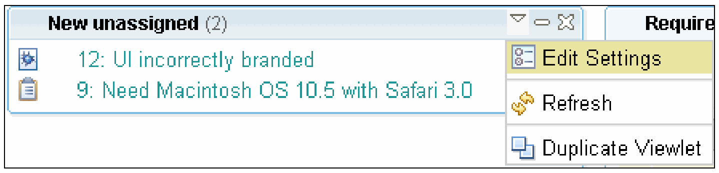
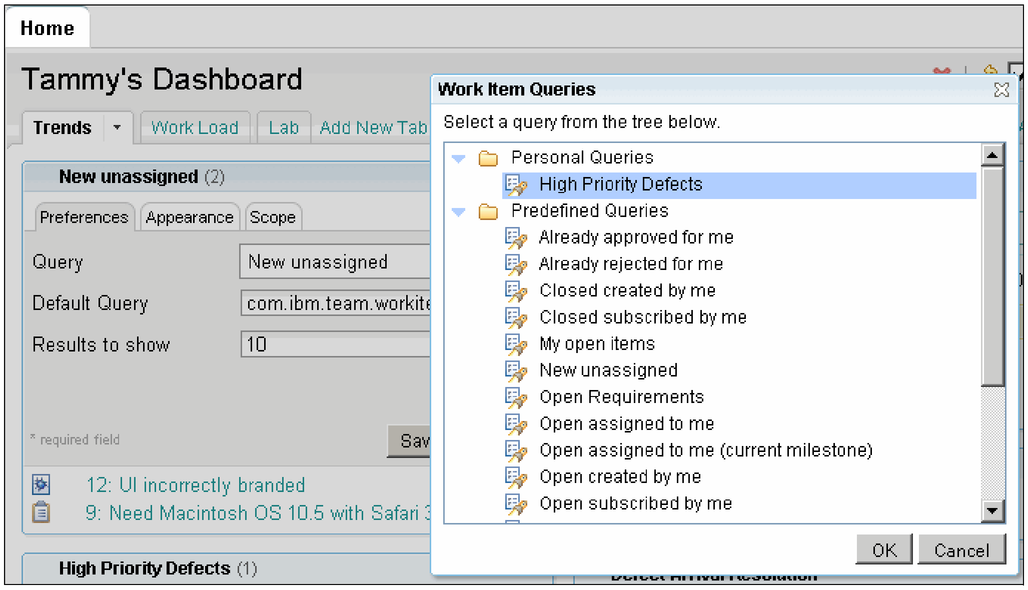

New unassigned (2)
12: UI incorrectly branded
9: Need Macintosh OS 10.5 with Safari 3.0
Require
Edit Settings
Refresh
Duplicate Viewlet

Figure 11-59 The menu for a viewlet

e. After the viewlet opens with the settings available for editing, she clicks the **Edit** button next to the Query field.

f. In the window that opens and lists the available queries (Figure 11-60), in the Personal Queries folder, she selects the query that she just created. In this case, she selects **High Priority Defects**. Then she clicks **OK**.


Home

Tammy's Dashboard

Trends | Work Load | Lab | Add New Tab

New unassigned (2)

Preferences | Appearance | Scope

Query: New unassigned

Default Query: com.ibm.team.workite

Results to show: 10

* required field

Sav

12: UI incorrectly branded

9: Need Macintosh OS 10.5 with Safari 3

High Priority Defects (1)

Work Item Queries

Select a query from the tree below.

- Personal Queries
  - High Priority Defects
- Predefined Queries
  - Already approved for me
  - Already rejected for me
  - Closed created by me
  - Closed subscribed by me
  - My open items
  - New unassigned
  - Open Requirements
  - Open assigned to me
  - Open assigned to me (current milestone)
  - Open created by me
  - Open subscribed by me

OK | Cancel

Figure 11-60 Editing the query used for a viewlet

g. In the viewlet:

i. She clicks the **Appearance** tab.

ii. She clears the **Use computed title** check box.

iii. In the Title field, she types a title, `High Priority Defects`, for the viewlet.

iv. She clicks **Save** and the editor closes. The query runs showing all unresolved, high priority defects.

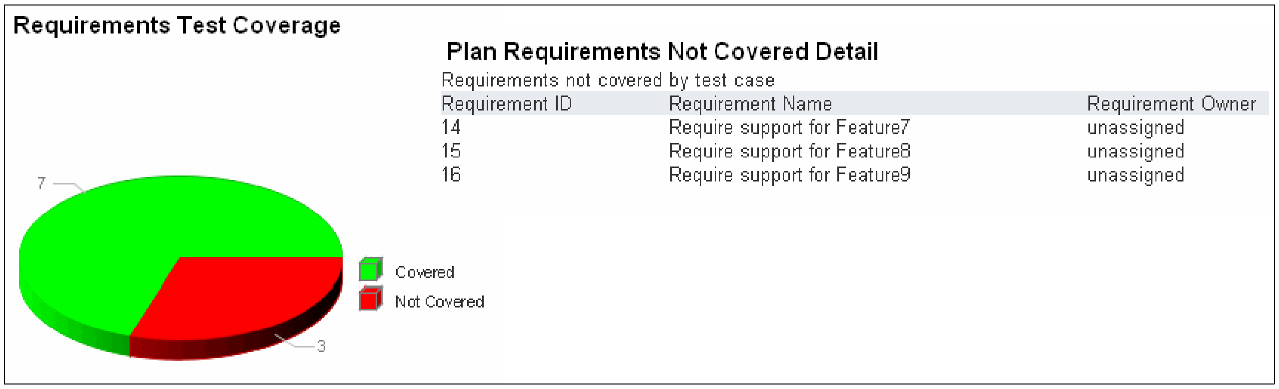
Requirements coverage

Rational Quality Manager provides real-time information about the status of requirements coverage. An important function of any test team is to validate that the working software meets the requirements. Tammy uses the Requirements Test Coverage and Requirements Status by Execution reports to ensure completeness in testing as one of her exit criteria.

Tammy uses a viewlet that is configured to display the Requirements Test Coverage report. To add this viewlet she does the following steps:

1. Tammy clicks the **Home** tab to view the dashboard.
2. She adds the viewlet:
   a. She clicks **Add Viewlet**.
   b. She scrolls to the Quality Management section, selects **Requirement Coverage Status,** and clicks **Add viewlet**. The viewlet is added to the dashboard.
   c. She clicks the **X** above the Add viewlet button to close it.
   d. In the new viewlet, she clicks the downward arrow and selects **Edit Settings**. The viewlet opens with the settings available for editing.
   e. She sets the Test Plan to the appropriate test plan, which is **AO_Rel2** in this case.
   f. In the viewlet, she clicks the **Appearance** tab to modify any settings or change the title. She deselects the **Use computed** title check box.
   g. On the **Scope tab,** no changes are needed.
   h. She click **Save** and the editor closes. The requirements coverage report is now available.

The Requirements Test Coverage report provides a pie chart that shows her the ratio of requirements that are covered and not covered (Figure 11-61).



Plan Requirements Not Covered Detail

Requirements not covered by test case

Requirement ID	Requirement Name	Requirement Owner
14	Require support for Feature7	unassigned
15	Require support for Feature8	unassigned
16	Require support for Feature9	unassigned

Figure 11-61 Example of a Requirements Test Coverage report

She can click the **Covered** section of the pie chart to view the details of all requirements that are covered by test cases. More importantly, she clicks the **Not Covered** section to view all requirements that are not covered by test cases, which is also shown in Figure 11-61. She clicks the requirement to open it, and from there, can create and assign a test case to a team member to ensure that it is covered.

By monitoring this report, Tammy can see when the team has created test cases for all requirements. She uses this report in combination with the Test Execution status reports to monitor when all tests have been executed. In particular, the Tester Using TER count report helps Tammy to see the number of test executions per tester along with a view of the unfinished test executions per tester. By seeing this information, she gains insight into the workload across the team so that she can reassign work.

100% user interface by using corporate branding

To track the test effort that is related to corporate branding, Tammy created a test suite that contains all re-branding test cases. She might have performed these steps at the beginning of the project. We explain them here to show how they can be used to measure a team's progress against exit criteria.

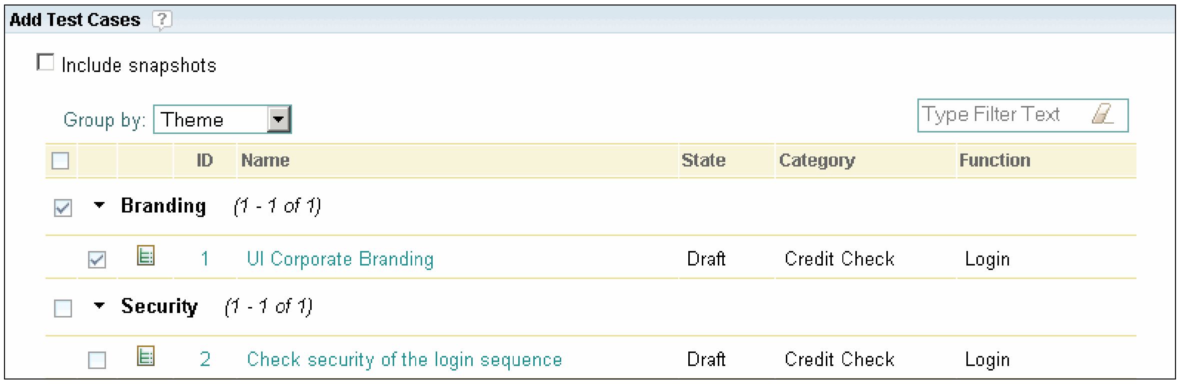
First Tammy created a *Branding* theme to identify branding related test cases. As an administrator, Tammy has the ability to modify the Category, Function, and Theme lists that are available to users of the system.

To add a Theme for organizing test cases:

1. Tammy selects **Admin** → **System Properties**.
2. On the **System** tab, in the list box, she selects **Test Case Categories**.
3. Under **Theme**, she clicks the green plus sign to add a theme.
4. In the text box that is displayed, she types the theme named `Branding`.
5. Optional: She can assign this theme to a tester.

As test cases are created, the Branding theme is selected, which makes it easier to identify related test cases. To create a test suite that uses this theme, Tammy completed the following steps:

1. She selects **Construction** → **Create Test Suite**.
2. For Name, she types `UI re-branding tests` and types a Description. Then she clicks **Save**.
3. From the Table of Contents, she clicks **Test Cases**.
4. She clicks the green plus sign to add a test case.
5. In the Add Test Case window, for Group by, she selects **Theme**, which re-orders the test cases by themes. All of the test cases with the Branding theme are now grouped.
6. She clicks the check box next to the **Branding** theme, which automatically selects all test cases with that theme (Figure 11-62), and clicks **OK**.



		ID	Name	State	Category	Function
✓	Branding (1 - 1 of 1)					
	✓	1	UI Corporate Branding	Draft	Credit Check	Login
☐	Security (1 - 1 of 1)					
	☐	2	Check security of the login sequence	Draft	Credit Check	Login

Figure 11-62 Adding test cases to a test suite, grouped by theme

Tammy can now use the test suite to monitor the progress of all tests that are related to the UI branding. The test suite provides a grouping mechanism for a set of test cases. At a glance, she can see which test cases have test scripts. She can also view the execution results to determine the quality of the effort.

Trend reports

Last Tammy uses trend reports to get an indication of how the quality is trending. Two reports in particular give her insight into the completeness of the test effort:

- The *Defect Arrival and Resolution report* provides an indication of the rate of defects that are found versus those that are fixed. She watches to see a decline in the arrival rate and eventually a leveling or decline in the resolution rate.
- The *Execution Trend report* (Figure 11-63) helps to estimate the actual test execution progress against the projected progress. It demonstrates what test case execution work is complete, how much work is left, and whether progress is being made as expected.

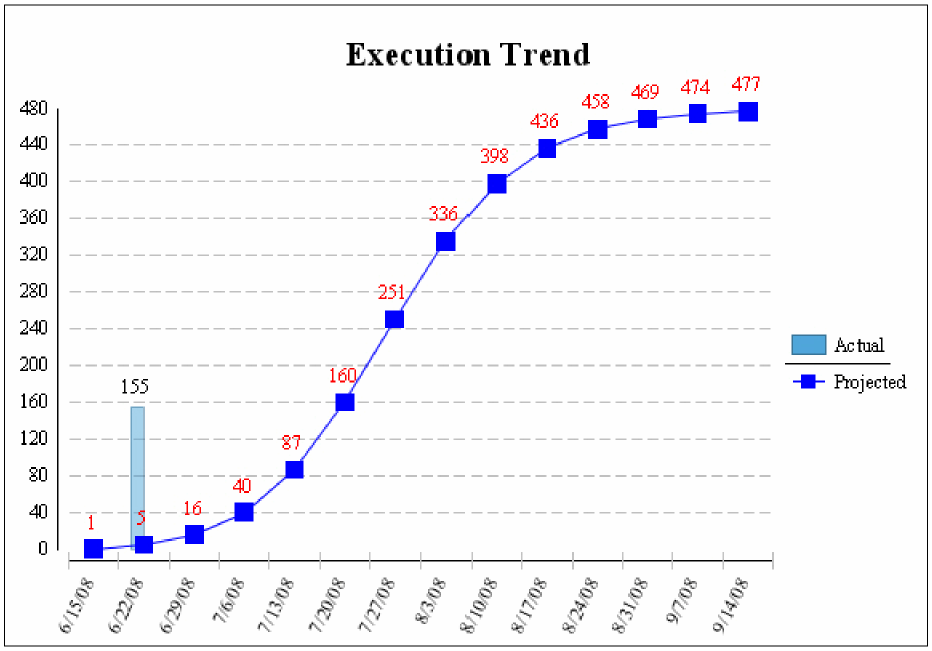


Figure 11-63 Execution trends comparing planned versus actual execution status

11.6 Life-cycle collaboration

As shown in this scenario the test team members collaborate and depend upon the contributions of other team members. Figure 11-64 on page 464 shows the life-cycle assets that are involved in the quality management scenario.

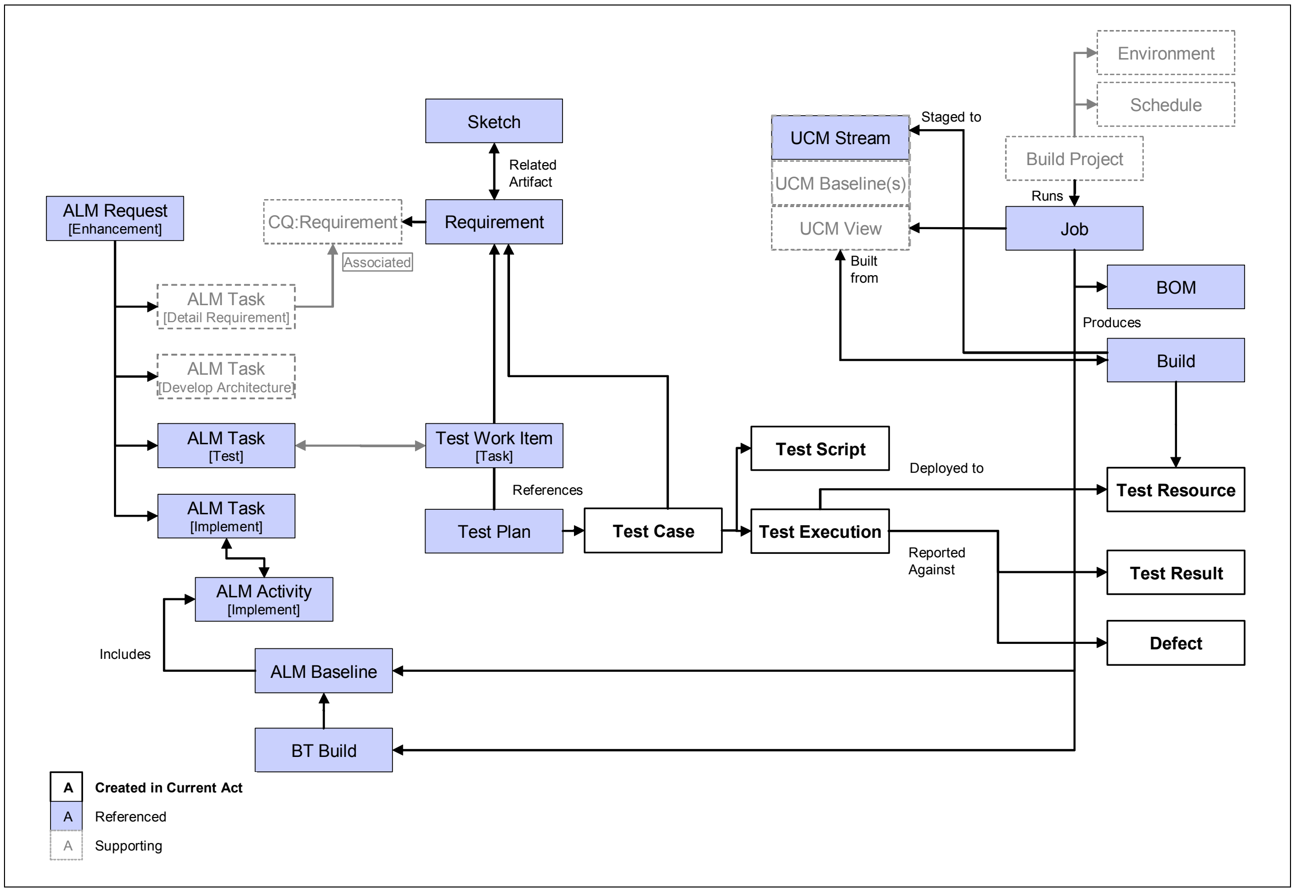


Figure 11-64 Life-cycle assets involved in this quality management scenario

The following assets are created in this act:

- The test plan is linked to requirements in Rational RequisitePro.
- A build is deployed to a managed resource in the lab.
- Test cases are updated with detail and include an association with the linked requirements from Rational RequisitePro.
- A test script is written and associated with the test case.
- Test execution records are generated for multiple browser configurations.
- Test results exist for each test execution record.
- A defect is associated with a line in the test execution.

The following assets are also referenced by this act:

- Test plan
- Build and the Unified Change Management (UCM) stream that stages it, which was created by Rebecca in Act 3
- Bill of materials from the integration build, which was created by Rebecca in Act 3
- Requirement and sketch created in Act 1, which is used to verify the expected result of the test case

11.6.1 Managing quality

The manage quality theme begins from the first act of the storyboard when the team responds to a change request (Figure 11-65). The theme continues through the last act when the team delivers the release. In this section, we highlight the areas where quality management comes into play in the reference scenario.

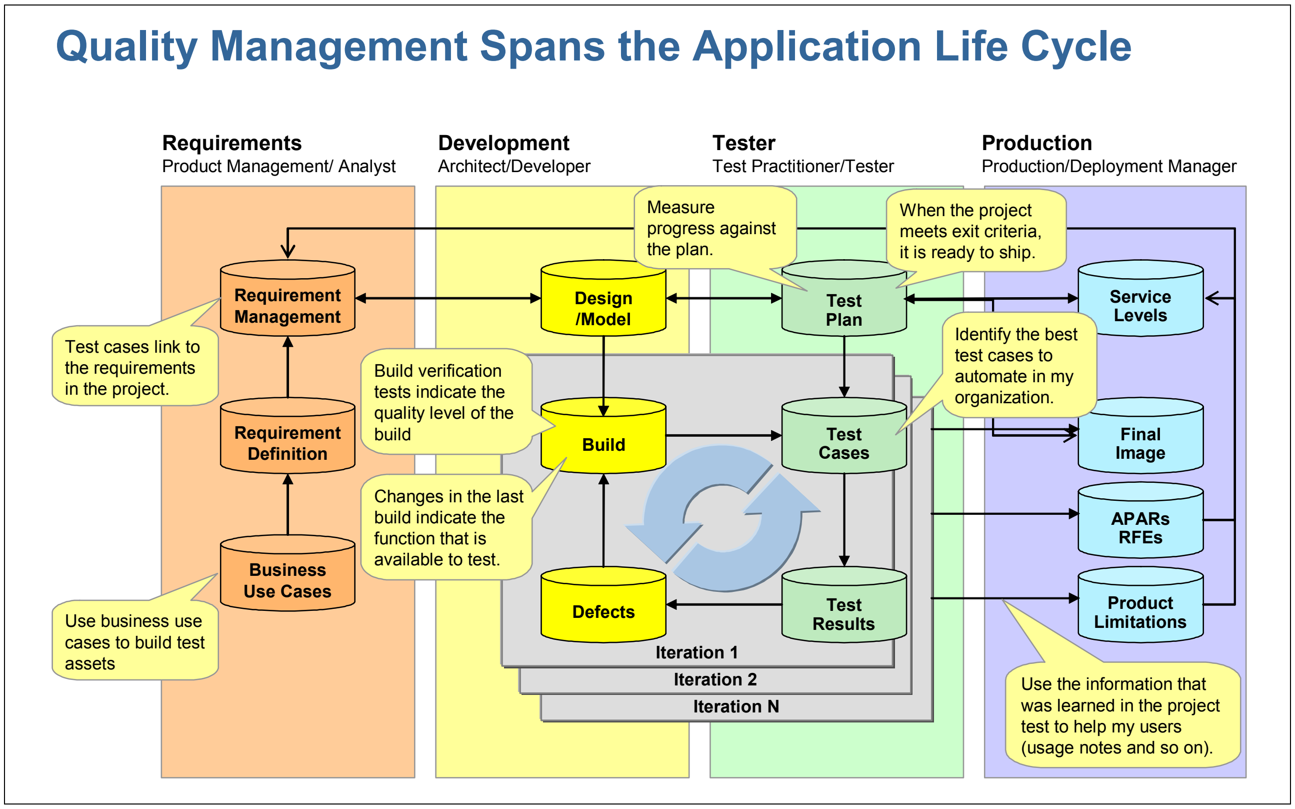

Quality Management Spans the Application Life Cycle
Requirements
Product Management/ Analyst
Development
Architect/Developer
Tester
Test Practitioner/Tester
Production
Production/Deployment Manager
Requirement Management
Requirement Definition
Business Use Cases
Test cases link to the requirements in the project.
Use business use cases to build test assets
Design /Model
Build
Defects
Build verification tests indicate the quality level of the build
Changes in the last build indicate the function that is available to test.
Measure progress against the plan.
Test Plan
Test Cases
Test Results
When the project meets exit criteria, it is ready to ship.
Identify the best test cases to automate in my organization.
Iteration 1
Iteration 2
Iteration N
Service Levels
Final Image
APARs RFEs
Product Limitations
Use the information that was learned in the project test to help my users (usage notes and so on).

Figure 11-65 Quality management throughout the life cycle

Iteration planning

Quality management starts by including the test effort in each iteration. By conducting integration tests in every iteration, the team can drive out defects with each iteration, which prevents the defects from piling up into an unmanageable list by the end of the project. Testing in each iteration also gives the test team more time to learn and work with the solution. This additional test time allows for increased test execution over the course of the project and increases the chances of driving down defects. Additionally as the team becomes more familiar with the functionality, the test cases can become more detailed and sophisticated.

Tammy the test lead is involved in planning the iterations. The test team has insight into the planned requirements and development effort, which enables Tammy to respond with a more meaningful and targeted test plan for each iteration.

Test planning

Test planning provides the heart beat for the test effort. While it might seem like additional work when beginning the project, having a plan in place saves significant time over the life of the project, whether you have a big, small, agile, or traditional plan. The test plan is worth the investment to ensure that the team is working toward a common goal. By creating and managing a test plan, Tammy and the rest of the team can measure their progress against the plan. Tammy has produced a test plan for each iteration. She also created test plans for each of the test environments (system test, acceptance test) at the end of the project.

The test plans include links to the requirements that will be implemented, thus enabling the team to ensure that there are test cases for every requirement. This also gives the test team direct access to the content and intent of each of the requirements, thus improving the quality of the test cases.

Tammy's plan includes exit criteria. By gaining agreement on the exit criteria, Tammy can determine when the team has completed the work. She can now measure and report against the exit criteria. All team leads can be aware of and measure the project against the exit criteria. This gives the project team and the business assurance over the quality of the release.

Tammy's test plan is managed by the test management system. By managing it in a database, all requirements and test cases can be linked to the plan. This makes the act of measuring progress much easier, as the system provides reports for specified metrics.

Developer testing

As we saw in Chapter 6, "An agile team implements a change" on page 213, Diedrie builds the source code with her changes in her local sandbox. This ensures that the source code works in her sandbox and reduces the chances of breaking the build.

Diedrie also runs unit tests against her changes before she delivers her source code. This initial set of testing catches the obvious defects before the source code is shared with the rest of the team.

Build automation and verification testing

In Chapter 8, "The release engineer conducts the integration build" on page 315, and Chapter 9, "Rational Build Forge for enterprise integration build" on page 341, we saw that the team has implemented an automated build strategy.

For Marco and his team, they ensure that their component builds without errors. They run build verification tests on their component to ensure quality in their build. When they are satisfied with the quality of their build, they deliver their changes into the integration stream. This ensures that each component has met with a predefined quality level prior to submitting it to the integration build.

Rebecca monitors the integration build. She has automated the integration builds and includes build verification testing at the integration level. She also runs static analysis on the source code base to catch inconsistencies in the source. This level of testing ensures that, when the components come together in the integration build, a predefined level of quality is met before deploying the build to the test servers.

By automating the builds, the team ensures that the exact same process is used for every build. This ensures consistency and reduces the chances of human error. In addition, the automation system keeps an audit trail of every step in the automation. At any point, the team can inspect the audit trail and determine what occurred.

Frequent integration tests

Rebecca conducts the integration build on a weekly basis, and Tammy chooses which of the builds to deploy into the test lab. Frequent integration tests help Tammy's team to identify defects early in the development cycle. These test are also more likely to catch regressions from week to week.

Lab management

Tammy and the rest of the team have a catalog of servers that are available for use. They manage the configurations and make requests to a lab manager to prepare the servers based on the environments that they have defined. This ensures a consistent approach toward lab reservation, reduces conflicts over server usage and helps the team to optimize server utilization.

Test construction and execution

Tammy's team uses a test management system to centralize their test cases, test scripts, test executions, and test results. This helps them to reuse existing test cases and scripts, thus reducing the amount of time of writing tests, which gives them more time to execute tests. By linking test cases to requirements, they can ensure that all requirements have test cases that have been executed. They can also track the planned tests versus executed tests to gauge progress and team health.

Quality monitoring

By managing these assets in a centralized repository, reports are easily generated to provide insight on progress. Additionally, the team established exit criteria for the iteration, which is used to determine when they have achieved the expected quality.

During a retrospective at the end of the project, Tammy can analyze the usage rates of the tests and test servers. This analysis helps her to continuously improve her testing effort.

11.6.2 Requirements-driven testing

The tests that are conducted by the test team tie directly back to the business requirements that Bob submitted at the beginning of the scenario. Ultimately the requirements drive much of the work that is completed by the team.

- Bob's request resulted in detailed requirements.
- Requirements are referenced by Al to locate a reusable asset.
- Diedrie references the detailed requirements to understand her development task.
- Tammy adds links to the requirements in her test plan to ensure that the team validates that the requirements have been met. She also adds the UI branding effort as an exit criteria in her test plan.
- Tanuj references the requirements to design and construct his tests. He also uses it when deciding if he needs to file a defect.

These are just some examples of how requirements impact the contributions of the rest of the team.

11.7 Planning and measuring success in quality management

The test plan serves as a linchpin for measuring success. By creating and managing against the plan, the team has a better sense of what is done and what remains to be done. The plan gives the team one metric of completion: You are done when you have completed everything that you planned. Reports and live metrics are then used to measure against the plan. Rational Quality Manager provides a set of reports that are accessed from the Reports menu (Figure 11-66 on page 468), which covers a range of information that helps the test teams confirm the quality of the solution.

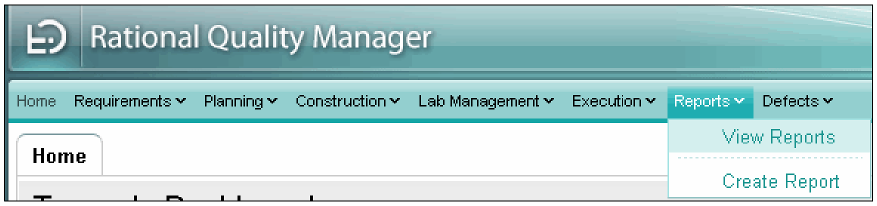
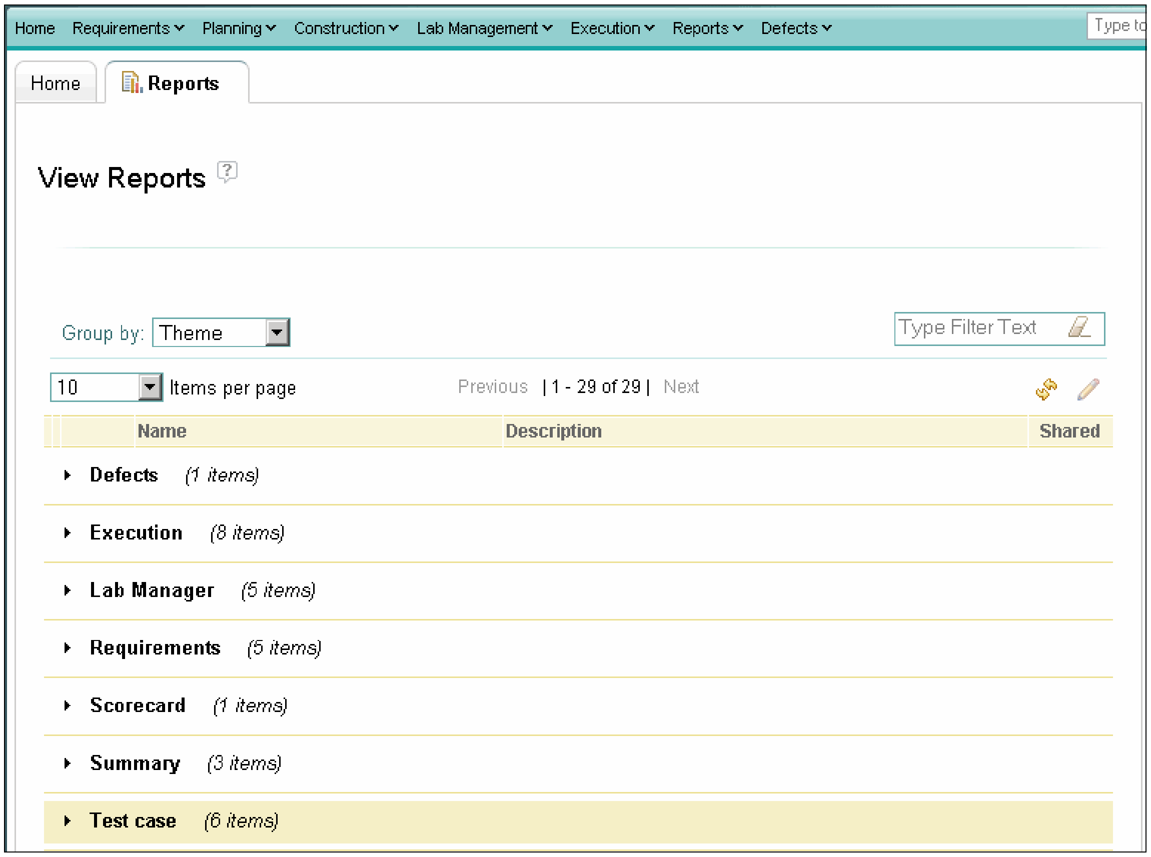


Figure 11-66 View or Create Reports by using the Reports menu

Choosing the View Reports menu item opens the set of predefined reports (Figure 11-67) that ship with the product.



Name	Description	Shared
Defects *(1 items)*		
Execution *(8 items)*		
Lab Manager *(5 items)*		
Requirements *(5 items)*		
Scorecard *(1 items)*		
Summary *(3 items)*		
Test case *(6 items)*		

Figure 11-67 Categorized reports in Rational Quality Manager

As shown in Figure 11-67, these reports are categorized into folders as follows:

- The *Defects folder* shows information and trends about defects, such as defect arrival versus resolution.
- The *Execution folder* provides the execution status of plans, machines, owners, trends, and defects.
- The *Lab Manager folder* provides information about lab usage, requests, and reservations,
- The *Requirements folder* contains the requirements coverage reports for test coverage information about your plan requirements.
- The *Scorecard folder* includes summary information that shows the status of test cases, test executions, and defects.

- The *Summary folder* includes individual summary reports such as Execution by Test Schedule and Tester Report using Weight.
- The *Test case folder* contains test case reports to list test cases by plan, configuration, or team.

In addition, you can create your own reports that are tailored to your needs. In the following sections, we highlight some of the report types that are available.

Trend reports

Use trend reports to gain insight on the direction that you are trending. For example, a trend report that shows the defect arrival rate slowing might indicate a sign that the solution is becoming stable. The Execution Trend report provides insight into the number of test executions that are performed by the team against the plan, which can indicate whether the schedule is a risk or on target.

Execution status

Tammy can obtain the execution status of plans, machines, owners, trends, and defects by clicking **Reports** → **View Reports**. She also uses viewlets on her dashboard to view the Execution Trend Report and Live Execution Status.

Click the report that you want to run, and select the parameters. Execution status reports by plan, owner, and machine display charts with data that is divided into six color-coded categories. All of these reports use the same status outcomes. The following reports, among others, are of interest to Tammy:

- The *Execution Status per Tester report* lists the status of execution work items by their testers or owners. You can select more than one plan to see the status of execution work items by owners across multiple plans. Click a section of the graph to view the execution work items that are associated with a particular status for that owner.
- The *Execution and Defects by Owner report* displays both the defects and the execution work items for each owner of a test plan. Click the ID number of an execution work item or a defect to open it.

Requirements coverage

Ensuring that all requirements are tested is an important task for the test team. Rational Quality Manager allows testers to link to requirements in Rational RequisitePro or create requirements directly in Rational Quality Manager. These requirements can then be associated with a test plan. By associating them with the test plan, the test manager can organize all requirements that will be validated by that plan. A report is provided to allow a test manager to see all requirements that are associated with a plan.

Additionally each requirement should have at least one test case written to validate it. Test cases have test scripts and one or more test execution records with results. A report is provided so that test managers can ensure that all requirements in plan have a test case.

Lab management

The lab management reports provide insight into the utilization of lab resources. This insight helps a team optimize their use of machines by seeing which machines are under or over utilized. The following reports are among those of interest:

- The *Lab Resource Utilization report* shows the average daily usage of a group of machines. You can also specify a time period to see the utilization rate during that period. This report can be based on a group of resources or for the entire lab.

- The *Machine Free Time Rate report* shows how much time each machine is idle. This is report is in the form of a bar chart with a threshold marker. You can specify a threshold, which highlights when you go under the threshold, indicating that the machine is too idle.
- The *Request Response Time report* shows the average response time per request to see how the lab managers are responding.

11.8 Reference architecture and configuration

In this section, we describe how Rational Quality Manager fits into the overall solution architecture and how the tools have been configured for this act of the storyboard.

11.8.1 Fitting into the enterprise ALM solution

Rational Quality Manager as used in this act illustrates part of an enterprise ALM solution with a globally distributed testing team integrated into a larger enterprise project. In this chapter, we have presented the workflows and tool usage in Rational Quality Manager for an integrated ALM solution that supports the team in alignment of work, iteration planning, reuse, delivery of change, and build integration. Figure 11-68 highlights the part with the enterprise ALM solution that is discussed in this chapter.

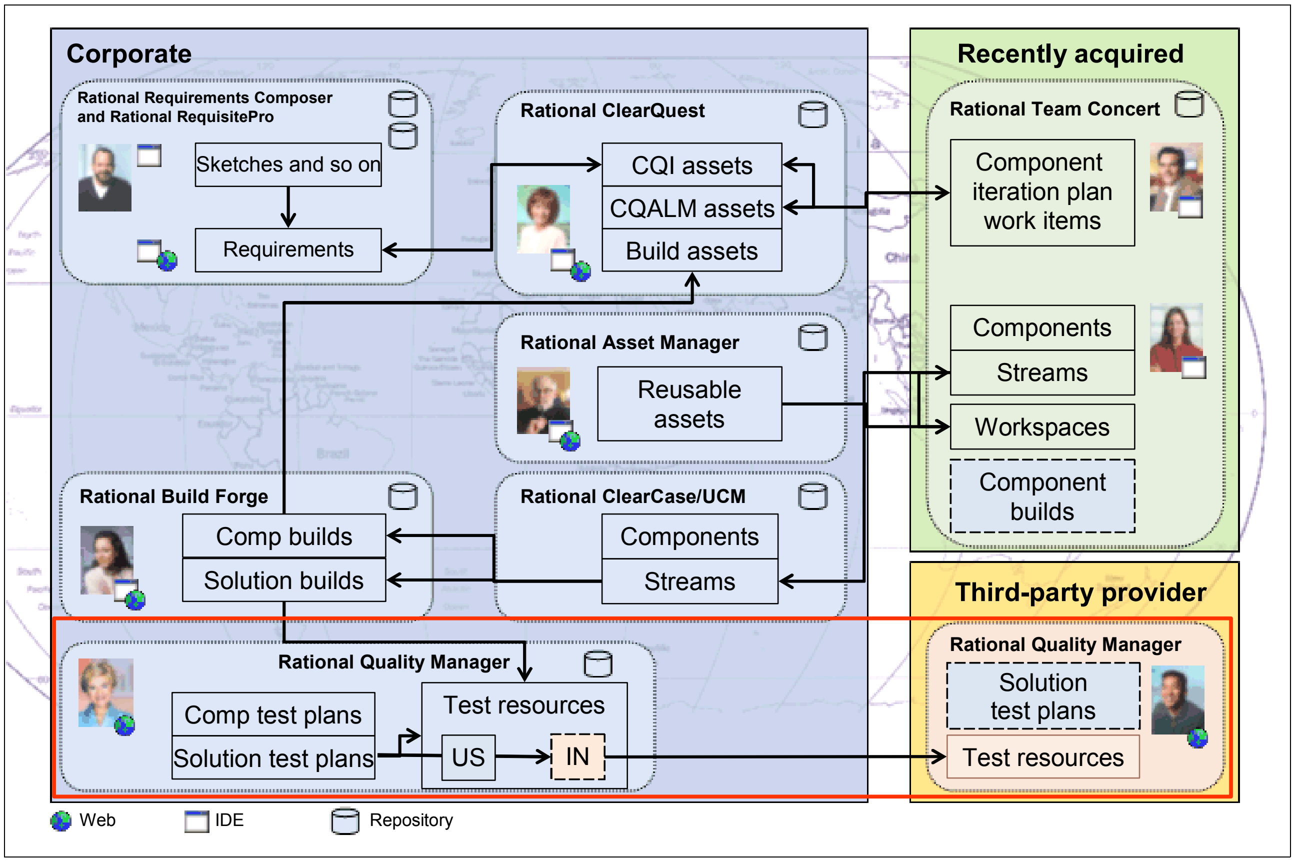


Figure 11-68 Rational Team Concert as one part of the enterprise ALM solution

Deploying Rational Quality Manager

Rational Quality Manager is built on the Jazz platform and provides a Web-based user interface. Rational Quality Manager was deployed to a single Windows 2003 server. The deployment of Rational Quality Manager is similar to that of Rational Team Concert as described in "Deploying Rational Team Concert" on page 302. The primary difference for Rational Quality Manager is that there is no Eclipse user interface and no build engine is involved.

A deeper description of the Jazz architecture is provided in *Jazz Platform Technical Overview* on the Jazz.net Web site at the following address (sign-on required):

`https://jazz.net/learn/LearnItem.jsp?href=content/docs/platform-overview/index.html`

Read the Rational Quality Manager blog for inside thoughts from the development team at the following Web address:

`http://qualitymanager.wordpress.com/`

Integrating Rational Quality Manager and Rational ClearQuest

Rational Quality Manager is integrated with Rational ClearQuest in the same way as Rational Team Concert. Work that is triaged in Rational ClearQuest must be aligned with the work that is performed by the users of Rational Quality Manager. For the reference scenario, we are only concerned with aligning work items that inform the test lead to add tests to the test plan. It assumed that all test-related work occurs in and is managed by Rational Quality Manager.

We are not concerned with aligning the results of the executed tests with items that are managed in Rational ClearQuest. Nor do we make any attempt in aligning Rational ClearQuest Test Manager with Rational Quality Manager. See "Integrating Rational Team Concert and Rational ClearQuest" on page 303 for more information.

For users of Rational Test Manager for ClearQuest, a utility is provided to migrate assets to Rational Quality Manager.

11.8.2 How the products are configured for this scenario

In this section, we describe the key configurations in Rational Team Concert that are used for this act of the scenario.

Configuring users, projects and team areas

The Rational Quality Manager repository is deployed by the corporate leads team, and all testers use the web interface to work with and store assets in the centralized test repository. Log in to Rational Quality Manager as a user with administrative permission, such as ADMIN/ADMIN. The user interface has a menu for the administrator as shown in Figure 11-69.

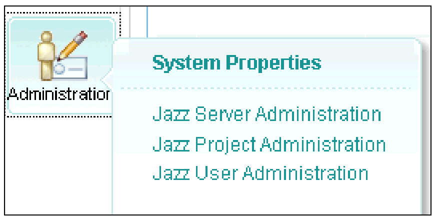


Figure 11-69 The Administration menu

To modify user accounts, click **Jazz User Administration**. User accounts were created for each of the actors in the scenario. When creating an Rational Quality Manager user account, you choose the type of client license to give each user. Tammy and Tanuj were given Rational Quality Manager - Tester licenses, which gives them read access to all capabilities and write access to all capabilities unless otherwise restricted by role-based process permissions. In addition, Tammy was given the Repository Permission of JazzAdmin, while Tanuj was given the Jazz User permission.

Marco, Diedrie, Patricia, Bob, and Al were given Rational Quality Manager - Viewer licenses to give them read access to all capabilities unless otherwise restricted by role-based process permissions. In addition, they were given Jazz User Repository Permission.

User accounts in Rational Quality Manager can be personalized in the same manner as described for Rational Team concert in “Configuring project and team areas” on page 305.

To configure the project area, click **Jazz Project Administration**. A team area, named *Account_Opening*, was created for the test team. A team area called *Lab Managers* was created for users that manage the test lab. This configuration assumes a separation between the testers and the lab administrators.

Restriction: Rational Quality Manager supports one, default project area with multiple team areas. The Administration user interface does provide the ability to create new project areas. However, these areas are not supported. Only the default project area with multiple team areas is supported.

In practice, additional team areas can be created to separate the test team’s functions. For example, if a test team focuses on a particular area of the solution, a team area can be created for them.

The team members were configured by using the roles that are available with the default Rational Quality Manager process. Tammy is assigned the test lead role, while Tanuj is assigned the tester role. The scenario configuration did not require any configuration of additional work item types or changes to the default work item state workflows.

Configuring the Rational RequisitePro integration

Rational Quality Manager integrates with Rational RequisitePro v7.1 and requires the Rational RequisitePro client for Web (RequisiteWeb) to implement the integration. Be sure to select the Web Components option when installing Rational RequisitePro.

After you install and configure Rational RequisitePro, you can use it to manage your requirements and import those requirements into Rational Quality Manager.

See the *IBM Rational RequisitePro Installation and Upgrade Guide* for information about configuring RequisiteWeb. This guide is installed with the product by default <reqpro_install_dir>\RequisitePro\doc\books\rp_install.pdf.

For information about using Rational RequisitePro, see the information center that comes with the product and the Rational Requirements Composer Information Center at the following address:

`https://publib.boulder.ibm.com/infocenter/rpcmpose/v7r1m0/index.jsp`

In summary, to integrate with Rational RequisitePro:

1. Install Rational RequisitePro 7.1 and configure a RequisiteWeb server.

> **Restriction:** To obtain this version of Rational RequisitePro, you must register for the RequisitePro 7.1 beta program.

2. Create a Rational RequisitePro 7.1 project, and add one or more users to the project.
3. Add requirements to the project.
4. Import the requirements into a test plan in Rational Quality Manager.
5. Associate each requirement with a test case.

When using Rational Quality Manager, you can establish which Rational RequisitePro host to use by modifying the system properties (Figure 11-70):

1. Log in as a user who has administrative privileges.
2. Click **Admin** → **System Properties**.
3. On the System Properties tab (Figure 11-70):
   a. From the list box on the left, click **RequisitePro Host Properties**.
   b. Click the green **Add Connection** (plus sign) icon.
   c. Type the host name and port number of the Rational RequisitePro server.
   d. Click **Save Connections** (floppy disk) icon.

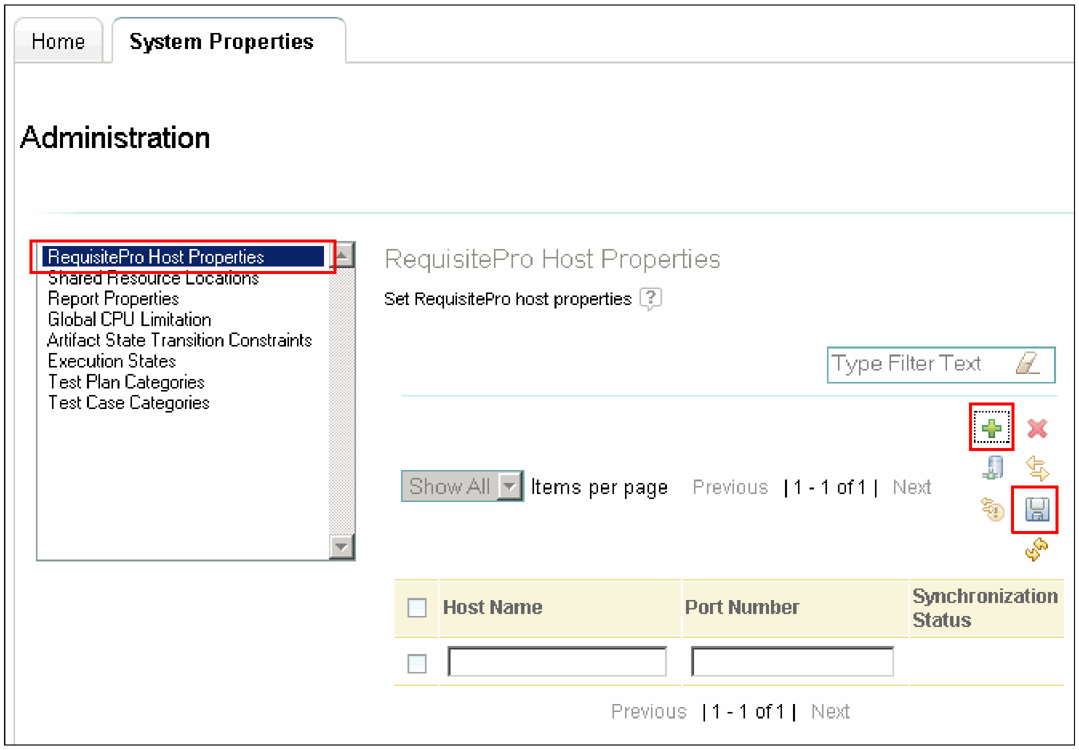


Figure 11-70 Adding a Rational RequisitePro host connection

Configuring automated functional tests

Adapters for automated test tools are provided for Rational Quality Manager. A Rational Quality Manager test script can reference files that are created by Rational Functional Tester or Rational Robot. After you create the functional test script, it becomes a test asset in Rational Quality Manager and can be managed like any other script. After you create the script, you must associate it with a test case. You can then generate an execution work item for that test case and execute it, which is identical to the way that you use scripts of other types.

To incorporate functional tests into your workflow, you must meet the following requirements:

- The Rational Quality Manager Web client and server must be running.
- A functional test product (either Rational Functional Tester or Rational Robot) and the corresponding adapter for Rational Quality Manager must be installed on the same machine.
- The adapter must be running.

It is not necessary for the functional test product to be running when an execution work item referencing a functional test script is executed.

Adapter installation

The installation files (RFTRQMAdapter.zip and RobotRQMAdapter.zip) for the adapters are located in the adapters subfolder in a typical Quality Manager installation. To install an adapter:

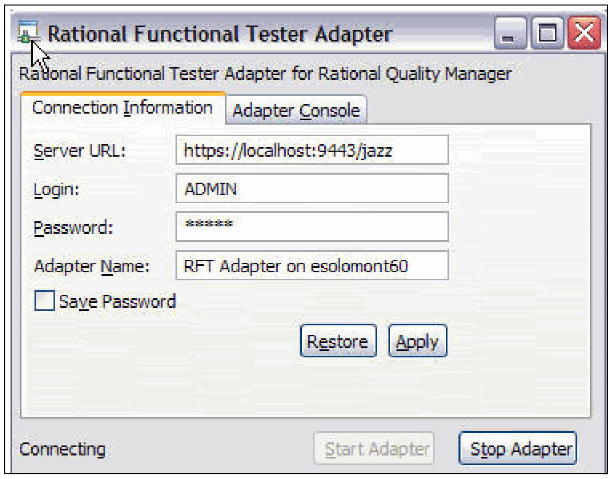
1. Download the compressed file to a machine where the functional test product is installed.
2. Extract the contents of the compressed file.
3. Edit the configuration file as needed.

Starting the adapter

To start the adapter:

1. Click the **startadapter.bat** batch file that is installed by default into C:\Program Files\IBM\SDP\FunctionalTester\RQMAdapter.
2. On the Connection Information tab (Figure 11-71 on page 475):
   a. For Server URL, type the URL for the Rational Quality Manager server.
   b. For Login ID, type a valid ID for the Rational Quality Manager server.
   c. Type the appropriate password for the Login ID.
   d. For Adapter Name, type a unique Name, which is any name that you want to give your adapter to display in the Rational Quality Manager Web UI to identify this instance of the adapter.
   e. Select **Save Password** if you do not want to re-enter the password each time that you start the adapter.
   f. Click **Start Adapter**.

   After the connection is made, the word “Connected” is displayed at the bottom of the Rational Functional Tester Adapter window.



Rational Functional Tester Adapter for Rational Quality Manager

Server URL: https://localhost:9443/jazz
Login: ADMIN
Password: *****
Adapter Name: RFT Adapter on esolomont60
Save Password
Connecting

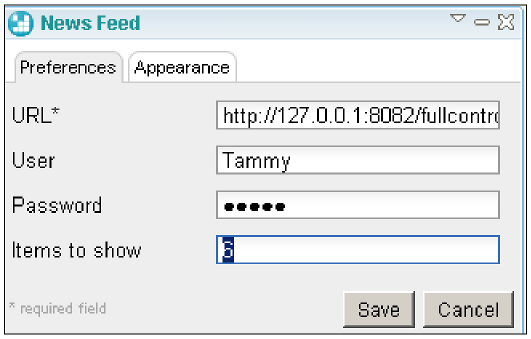
Figure 11-71 Rational Functional Tester Adapter

3. Click the **Adapter Console** tab to view the adapter status.

Configuring the Rational Build Forge news feed

The Rational Quality Manager dashboard has viewlets for presenting feeds. In this section, we explain how to configure the news feed for this scenario:

1. In Rational Build Forge, select **Administration** → **Messages**.
2. In the Message window, click the **RSS** button.
3. Copy the URL from the browser address.
4. In Rational Quality Manager, go to the dashboard.
5. Click the **Add Viewlet** button.
6. Choose the News RSS feed and add it to the dashboard.
7. Click the **Edit** link in the text of the feed viewlet.
8. On the Preferences tab of the News Feed window (Figure 11-72):
   a. In the URL field, paste the URL.
   b. Provide a user name and password for the Rational Build Forge account.
   c. Click **Save**.



URL* http://127.0.0.1:8082/fullcontr
User Tammy
Password •••••
Items to show 6
* required field

Figure 11-72 Adding a Rational Build Forge RSS feed to Rational Quality Manager

To view the news:

- When you see a URL in the news viewlet, click the URL to open the Rational Build Forge messages window.
- Click the **plus sign** in the lower right corner of the viewlet to see a read only view of the headlines.

Configuring automations for lab management

The Rational Test Lab Manager Automations are run by using Rational Build Forge behind the scenes. Each of these automations is a Rational Build Forge job that can be called from the Rational Quality Manager user interface. The steps for configuring Test Lab Manager automations are as follows:

1. Install Rational Build Forge so that you can manage its lab asset information in Rational Quality Manager.
2. Configure Rational Build Forge to work with Rational Quality Manager by modifying its buildforge.conf file and updating its port information if it is installed on the same machine as Rational Quality Manager.
3. Configure Rational Quality Manager to work with Rational Build Forge by updating its integration_config.xml file to include the host name or the IP address of the Rational Build Forge server.
4. Create machines in Rational Quality Manager with the software type “Build Forge agent” installed. Assets defined in the manner are synchronized with Rational Build Forge.

See the Integrating section of the Rational Quality Manager online help for details about these configurations.

After the servers are configured, you build automations by using the Rational Build Forge projects feature. These automations are similar to the build automation that we explain in Chapter 9, “Rational Build Forge for enterprise integration build” on page 341. Only in this instance, automation projects are created for deploying the builds, instead of conducting the build. The same automation concepts apply to creating deployment automations, such as creating libraries, using environments, building projects, defining parallel and sequential steps for an automation, and running jobs.

Some examples of commands that are used in deployment automations are stopping a server, deploying the build from a staging location, starting a server. The examples that follow were taken from the team’s use of Rational Test Lab Manager to deploy the product to their own servers.

Stop a server by navigating to a directory and calling a batch file, as shown in Example 11-1.

Example 11-1 Stopping a server prior to deploying an application

```
cd "C:\JazzBeta2\jazz\server"
server.shutdown.bat
```

Call a deployment script, as shown in Example 11-2, to set the directory to the location of the script and then call the script with parameters set.

Example 11-2 Calling a deployment script

```
cd "C:\RTLMDeploy"
RTLMDeploy_auto.jar -u C:\JazzBeta2 -w C:\wget -auto -n
```

Start a server by changing to the appropriate directory and running a batch file to start the server as shown in Example 11-3.

Example 11-3 Changing the directory and starting the server

```
cd C:\JazzBeta2\jazz\server
cmd /C server.startup.bat
```

Configuring Rational AppScan Tester Edition

Consult the Rational AppScan Tester Edition for Rational Quality Manager help to configure Rational AppScan:

- *AppScan Tester Edition 5.5 Quick Start Guide*, GI11-9120

  http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=GI11-9120-00

- *IBM Rational AppScan Tester Edition Administration Guide 7.7*, SC23-9453

  http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SC23-9453-00

- *IBM Rational AppScan Tester Edition Troubleshooting Guide 7.7*, GC23-9454

  http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=GC23-9454-00

- *IBM Rational AppScan Tester Edition User Guide 7.7*, SC23-9452

  http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=SC23-9452-00

Configuring ClearQuest Connectors

We explain how to configure the ClearQuest Connector in Appendix B, "Configuring interoperability" on page 565.

11.9 Problem determination and known workarounds

Installing Rational ClearQuest 7.1 and Rational RequisitePro 7.1 on the same server can create port conflicts for the Web user interfaces. Running Rational Build Forge, Rational Quality Manager, or Rational Team Concert on the same server can create port conflicts with the Web user interfaces. In this case, create a port numbering strategy and apply it to any products that are installed on the same server.

The Rational Quality Manager integration with Rational RequisitePro uses Requisite Web. Therefore, be sure to have the Rational RequisitePro Web server running on your requirements machine before attempting to import requirements to Rational Quality Manager.

When using integrations with other products, ensure that any required adapters or connectors are running.

When using the ClearQuest Connector, make sure to start the Rational Quality Manager server followed by the ClearQuest Connector.

Part F

Act 5: Delivering the solution

Act 5 is the final act of the storyboard. The team has entered into the end game of the iteration. The leads ensure that all work is complete with expected quality, and the iteration is delivered. In Chapter 12, “The team delivers the solution” on page 481, we provide information about solution delivery as it relates to the scenario. Then in Chapter 13, “The Rational ALM solution for solution delivery” on page 495, we provide detailed information about the Rational products that are used to support this act of the story.

Role-based guide: To understand how the content in this part applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

12

The team delivers the solution

The team has finally reached the end of the iteration. In this chapter, we provide an overview of “Act 5: Delivering the solution” along with a reference scenario for how it can be applied by an enterprise team.

This chapter includes the following sections:

In this chapter, we also include information about how this scenario relates to the previous scenarios and how it can impact delivery scenarios outside the scope of this book.

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

12.1 Introduction to software delivery

The biggest question for software teams at the end of the iteration is knowing whether their work is done, or rather when *release requirements have been completed with sufficient quality*. This is where the previous decisions by the team regarding which requirements to implement and which tests to create and run help the team determine if they are done. The Collaborative Application Lifecycle Management (CALM) solution needs to support this tracking, and resulting decisions help the team achieve the required speed and quality.

As the teams approach this acceptance bar, or exit criteria, it is essential to make the team control the change rate of the project. This is accomplished by enforcing a tighter stabilization, or end-game, process to the iteration and by ensuring team accountability by approvals. The review and approval process ensures that only prioritized changes are delivered and that all functions in the team take ownership of the release quality.

Organizations that adopt agile development practices expect their development teams to increase the delivery rate by effectively providing working solutions by each iteration. The agile principles state, "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software." The requirements for increased delivery rate require that both the build and the release workflows adopted by the project are repeatable, predictable, and implemented with a high degree of automation to achieve efficiency.

In this act of the scenario, we follow the team as the Account Opening project moves into end game and completes the release for handoff.

12.1.1 The changing of software delivery

As discussed in 8.1.1, "The changing enterprise build management market" on page 316, the increased business demands on software delivery, and the adoption of agile principles, are moving development teams to shorter release cycles. More and more this moves applications to the state of "perpetual beta," with needs for shorter delivery iterations and tighter integration of project and release management into the complete end-to-end life-cycle solution:

- The release workflow must seamlessly connect to the output of the build and quality workflows to achieve efficient release management and avoid unnecessary duplication of work, assets, and automation.
- Software regulations and compliance require teams to efficiently manage Application Lifecycle Management (ALM) artifacts and traceability to application changes and approvals, without extensive manual labor, to keep up with more frequent delivery iterations.
- Release management must seamlessly integrate with deployment tools to efficiently deliver solutions to release repositories or deploy them to operational IT environments.
- Development organizations must retain the knowledge of how products and applications are built and delivered. They must also maintain comprehensive process data in a secure centralized knowledge base to enable flexible sourcing of development resources and to minimize business vulnerability.

IBM Rational ALM release management capabilities are seamlessly extended from the development, test, and build environments that support the CALM development cycles. This effectively bridges to the publishing and deployment cycles of the ALM life cycle (Figure 12-1).

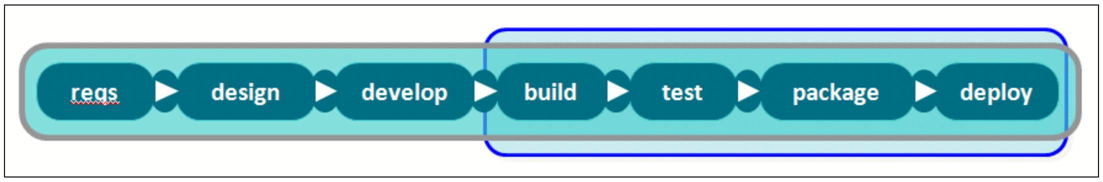


Figure 12-1 Build and release cycles integrating seamlessly into the ALM cycles

Build and test automation framework

The build automation framework in the enterprise build management blueprint ensures that the build, quality, and release workflows can take an adaptable, common, modularized, and reusable approach. This approach enables the quality and release management processes to meet the different and changing requirements on the release workflows cross applications, solutions, and teams.

Packaging

Software organizations tie together the transition of applications between building, packaging, and deployment of the release. Often deployments are managed in an “over the wall” fashion. To realize a build and release management solution that meets the expectations of agile and business-driven development, the transition workflows must be streamlined. The extraction and packaging of the application component from the source control repositories must be automated as should the movement of the packages across solution staging repositories, teams, organizations, machines, and groups. The deployment can be an intermediate drop zone or final deployment environment.

Delivery and deployment

Typically the delivery process consists of some basic steps across may projects. Modularizing packaging and deployment steps enables the reuse and agility of reconfiguring the delivery process. Applying role-based security to the modularized packaging and deployment steps can be used to enable sharing and commonality of a deployment script. In this case, the development team can deploy to a test environment, but only specific users or roles can deploy to staging or production environments.

12.2 Reference scenario for solution delivery

In this section, we provide an overview of the steps that are the development team takes at the end of the final iteration in the Construction phase. Details about the activities of the Transition phase are not covered in a step-by-step scenario. However, in this chapter, we provide information about using Rational Asset Manager to manage the final build as a reusable asset.

The scenario in the chapter continues to build on “Act 4: Managing quality” in this book. In 8.2, “A reference scenario for enterprise build management” on page 325, Rebecca was running the integration build that contained the changes to UI Branding delivered by Diedrie in the CreditCheck component team.

In “Act 5: Delivering the solution”, we explain how the development team switches to an “end-game” process and the team leads start to assess their exit criteria for the iteration.

When the team agrees that the iteration exit criteria has been met, Patricia approves the release. While the teams conduct retrospective to continuously improve their practices, Rebecca publishes the iteration release and submits it to the reuse repository in Rational Asset Manager (Figure 12-2).

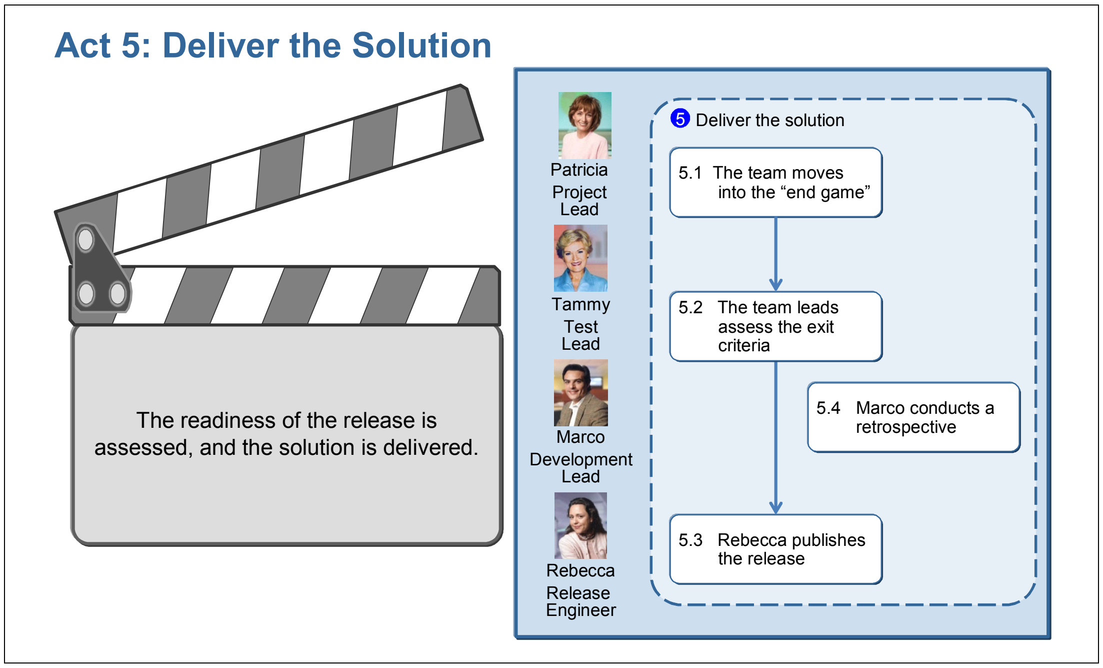

Act 5: Deliver the Solution
The readiness of the release is assessed, and the solution is delivered.
Patricia Project Lead
Tammy Test Lead
Marco Development Lead
Rebecca Release Engineer
5 Deliver the solution
5.1 The team moves into the "end game"
5.2 The team leads assess the exit criteria
5.4 Marco conducts a retrospective
5.3 Rebecca publishes the release

Figure 12-2 Act 5: The team completes the iteration and delivers the solution

12.2.1 The actors

This scenario includes the several key actors as described in this section.

Patricia is the project leader. She is responsible for coordinating the plans and execution of the project and its team of teams. During the end of the iterations, she validates that the project is ready to close down for release and starts to tighten the process and reporting so that nothing is slipping through the cracks. She moves the team into the end-game process.

Marco is the development lead in the Credit Check component team. He is responsible for the delivery and quality of one of the solution components, and he enforces the end-game process within his agile team. For the iteration release, Marco approves the component delivery after the exit criteria for the iteration is met by his team.

Tammy is the test lead. She has a globally distributed quality team that is responsible for conducting solution testing or investigative testing. She is monitoring the test plan and the quality exit criteria for the iteration release and approves the release from a quality perspective.

Rebecca is the release engineer who oversees the solution integration builds and provides global build support to the project teams. She is also responsible for packaging, publishing, and announcing the iteration release.

12.2.2 The workflow

In "Act 5: Delivering the solution" of the ALM scenario, we capture the steps of the closure and delivery of an iteration. This final act in the scenario begins on the last day of the iteration.

The steps in Figure 12-3 are performed by the project team.

- Patricia move the iteration to the end game.
- Marco enforce the end-game process enactment in his component team and starts reviewing and approving deliveries.
- Patricia, Marco, and Tammy monitor and manage outstanding work to reach their exit criteria.
- Marco approves his component release, and Tammy approves the release quality.
- Patricia approves the application release.
- Rebecca publishes and announces the application release.
- Marco, Tammy, and Patricia conduct retrospectives with their teams

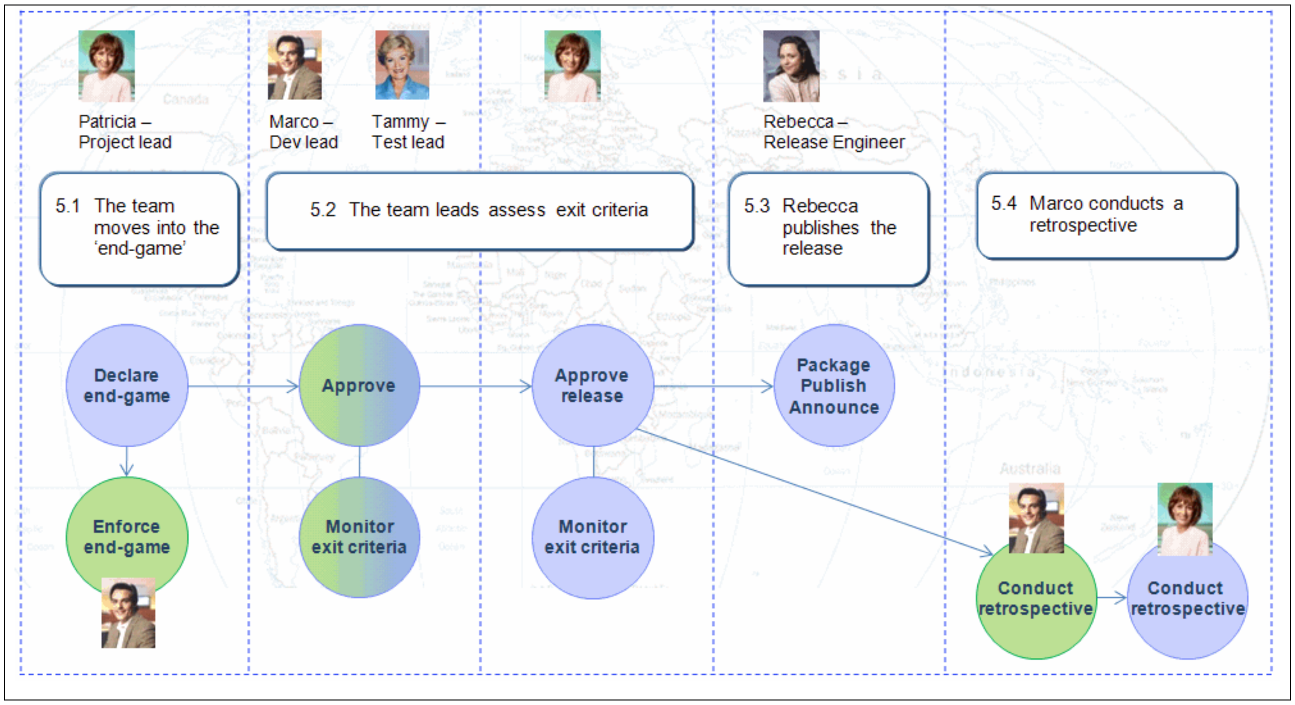


Figure 12-3 Software delivery workflow for this scenario

12.2.3 The team moves to the end game

Synopsis: The end of the iteration is approaching. Patricia and her leadership team have been practicing an end-game process to ensure convergence of change at the end of each iteration. During the end game, deliveries from the teams require approvals by component leads or by project leads in the final Transition phase. Today Patricia moves the project to the end game and requests her leadership team to enforce this process with their teams.

Marco forwards the information about the process change to his team at the next team stand-up meeting and uses the development platform to enforce the tighter end-game process.

The workflow in this scene captures how Patricia and Marco complete the following tasks:

- Assess project health and decide to move the project to the end-game practices
- Enforce the end-game practices
- Use approvals to manage change deliveries

12.2.4 The team leads assess their exit criteria

In this scene Marco, Tammy, and Patricia monitor and manage outstanding work to reach their exit criteria and to approve the release.

Marco monitors the component exit criteria

Synopsis: Marco is focusing his team during the end game to converge the remaining work in the component iteration plan. He works with his team to prioritize low priority requests that must be pushed out of the release. He confirms updates to the iteration plan with Patricia and Bob. He also tracks new defects that are submitted by the component team or by the test team from test failures in component builds or solution integration builds. He works with the owners of prioritized defects to assess impact, confirm importance, validate test status, and approve the deliveries. His team must close all enhancements or defects of blocking or critical severity and close all tasks of high priority to meet their component exit criteria.

Marco is reporting his progress to the exit criteria at Patricia's daily leadership stand-up meetings. Today he can proudly report that his team has reached their objectives and that he has approved the component iteration release.

The workflow in this scene captures how Marco completes the following tasks:

- Monitors the completion of the iteration plan
- Queries for new defects and approves deliveries
- Queries and reports on component exit criteria
- Approves the component iteration release

Tammy monitors the quality exit criteria

Synopsis: Tammy and her test team are monitoring the iteration test plans to validate that the solution release has met its quality goals. She is monitoring the list of quality exit criteria for the iteration release, which includes build validation test stability, code coverage of test cases, functional test failures, and application load performance.

On the final day of the iteration, Tammy confirms that the solution meets the quality exit criteria. She informs Patricia that the quality has met the expectations, all requirements have been tested, and she has approved the release quality.

The workflow in this scene captures how Tammy completes the following tasks:

- Monitors the completion of the test plan
- Queries and reports on the quality exit criteria
- Approves the quality of the iteration release

Patricia approves the release

Synopsis: Patricia confirms that the release requirements that are planned for the iteration have been implemented and there are no outstanding tasks for the team. She also confirms with her leadership team on the approval status across the project.

She agrees that the iteration is complete and approves its release.

The workflow in this scene captures how Patricia completes the following tasks:

- Monitors the completion of the project iteration plan
- Monitors approvals from the project teams
- Approves the iteration release

12.2.5 Rebecca publishes the release

Synopsis: Rebecca receives a notification from Patricia that the iteration release candidate has been approved. The release is now ready to be packaged.

Rebecca creates a new release package for the iteration release. The release code archives are already packaged as part of the build process that automatically pulls all release assets into the release archives. The development team has provided Rebecca with the documentation that should be included in the release. She includes the complementary material including the bill of materials, release notes, documentation, and samples.

Rebecca creates a new public release page on the enterprise IT intranet. She edits the release page and fills in the release information. She provides download links to the release package that she just created. She saves and publishes the release page.

Rebecca uses the Account Opening RSS news feed to announce the availability of the latest iteration release. She links the feed to the release page.

At a later date, the final production release is made as the Account Opening project exits the Transition phase. Rebecca transitions the solution release to production by delivering it to the enterprise reuse repository. She logs into Rational Asset Manager, adds a new version to the Account Opening Solution asset, and submits all application release packages to the repository. She announces the new asset to the Account Opening community in Rational Asset Manager.

The workflow in this scene captures how Rebecca completes the following tasks:

- Creates a release package
- Creates a release page and announces the release
- Submits a new asset version to Rational Asset Manager

12.2.6 Marco conducts a retrospective

Synopsis: Each team in the Account Opening project conducts retrospective to continuously improve the development practices. The retrospectives are owned by each individual team and are facilitated by the team lead.

Marco calls for a retrospective with his component team. The team identifies the timeline for the iteration, any major events during the iteration, and what the team achieved. The team uses the dashboard to review health trends from the iteration. The team also discusses what went well and how the team can do well more often. Marco makes the team discuss alternative behaviors to the tasks that did not go well and what the team can do about it. He structures his notes about behavior, practices, and agreed improvements and submit them to the team repository.

At Patricia's retrospective with the leadership team, Marco reviews the conclusions from his team. Patricia's team concludes on actions and decides to pilot new estimation practices for Marco's team in the next iteration.

The workflow in this scene captures how Marco completes the following tasks:

- Uses the dashboard to support retrospectives
- Submits a retrospective to the team repository

12.3 Considerations in solution delivery

In this section, we discuss additional considerations for solution delivery.

12.3.1 Transitioning to production

Until now, this book has been discussing the activities that occur while the solution is under development during the Construction phase. Some teams adopt multiple phases, such as those defined in the Rational Unified Process (RUP) and Open Unified Process (OpenUP). At some point, however, the team must choose a build to release to their stakeholders.

When the final construction iteration is completed, the team is confident that they have implemented and tested all of the requirements with sufficient quality. The solution is now stable enough for the next level of testing to occur. During transition, a wide variety of tests can take place, either sequentially or in parallel. The test lab either mimics the production environment or is a scaled down version. Security and compliance scans take place on this configuration to ensure that there are no vulnerabilities. Sophisticated performance tests begin, which can last for days and simulate hundreds or thousands of users. Eventually user acceptance testing is performed, and the solution is deployed into production. At any point in this phase of testing, unacceptable defects can be found that send the team back to the Construction phase, where the fix must be implemented and system tested before moving the solution build back to the transition environment (Figure 12-4 on page 489).

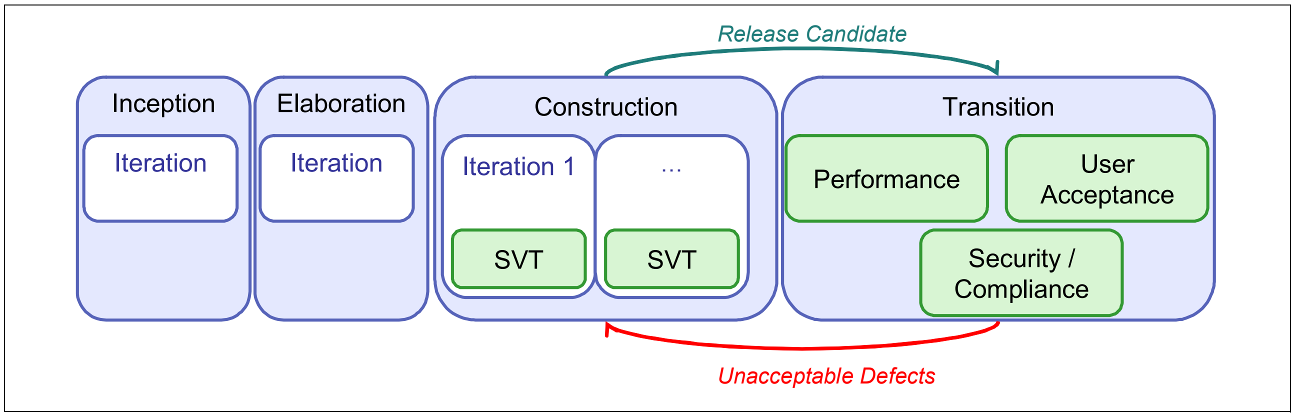


Figure 12-4 Transition to production

Who owns this phase of testing depends on the organization. There's a gradient of ownership that varies from enterprise to enterprise. In some organizations, the test team might own the performance tests, while the operations team owns security, compliance, and user acceptance testing. In other organizations, the operations team might own all of these activities. Ownership is not the key point, however. The important note is that a handoff occurs, and at some point, new people will conduct tests on the solution that the development team knows thoroughly. Sharing the knowledge of known defects and sharing test cases, scripts, and configurations can help the teams test more efficiently. Most importantly, the team must ensure that the same build that was tested by the test team is the one that is deployed into production.

12.3.2 Delivering to operations

Rational Asset Manager v7.1 provides an integration to the Tivoli Change and Configuration Management Database (CCMDB). Organizations can use Rational Asset Manager as a *definitive software library* (DSL). A DSL is a secure location where the definitive, authorized versions of software package configuration items (CIs) are stored and protected. A DSL consists of one or more software libraries or file storage areas, which are referred to as *repositories*. In Information Technology Infrastructure Library (ITIL) v3, this concept has been broadened to be a *definitive media library*, where the repository can store software packages and other forms of assets such as presentations, documents, and video files.

By publishing the release to Rational Asset Manager, Rebecca hands the release over to the IT operations team, where they can then create the deployment scripts for rolling the release into production. By using Rational Asset Manager as a DSL, organizations can more effectively manage their software by knowing which version of the software is deployed into production.

Rational Asset Manager and CCMDB work together in an integrated way to allow IT organizations to link their development assets to deployment software configuration items which are then eventually deployed onto runtime CIs in the IT infrastructure.

Rational Asset Manager manages metadata on assets to support asset management scenarios. The metadata includes the obvious items such as the asset's name, description, version, and state. Other metadata includes the artifact's name, description, version, and reference (or location), as well as custom attributes, categories, and relationships to other assets. To support the integration scenarios, Rational Asset Manager uses the Reusable Asset Specification (RAS) from the Object Management Group (OMG) as the core structure for asset metadata. Additional metadata is required for integrating with tools and other repositories such as Tivoli CCMDB.

RAS provides a structure for unique asset identification and metadata extension. The combination provides the basis for managing references between Rational Asset Manager and Tivoli CCMDB. By using RAS in Rational Asset Manager, the enterprise can configure multiple asset types to meet their particular needs. Examples of asset types include applications, components, services, word templates, and release packages (Figure 12-5).

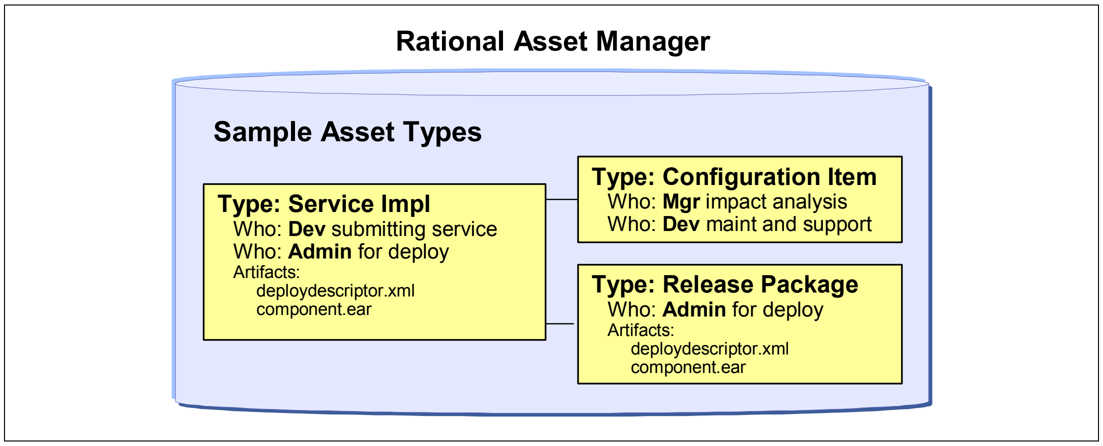


Figure 12-5 Asset types in Rational Asset Manager

They key point here is that regardless of the asset type configuration in Rational Asset Manager, the synchronization can still work with Tivoli CCMDB. In addition to the asset metadata, such as ID, which is used to create a reference between Rational Asset Manager and Tivoli CCMDB, other metadata elements, such as classification, are synchronized between Rational Asset Manager and Tivoli CCMDB. Figure 12-6 illustrates a sample of the asset metadata.

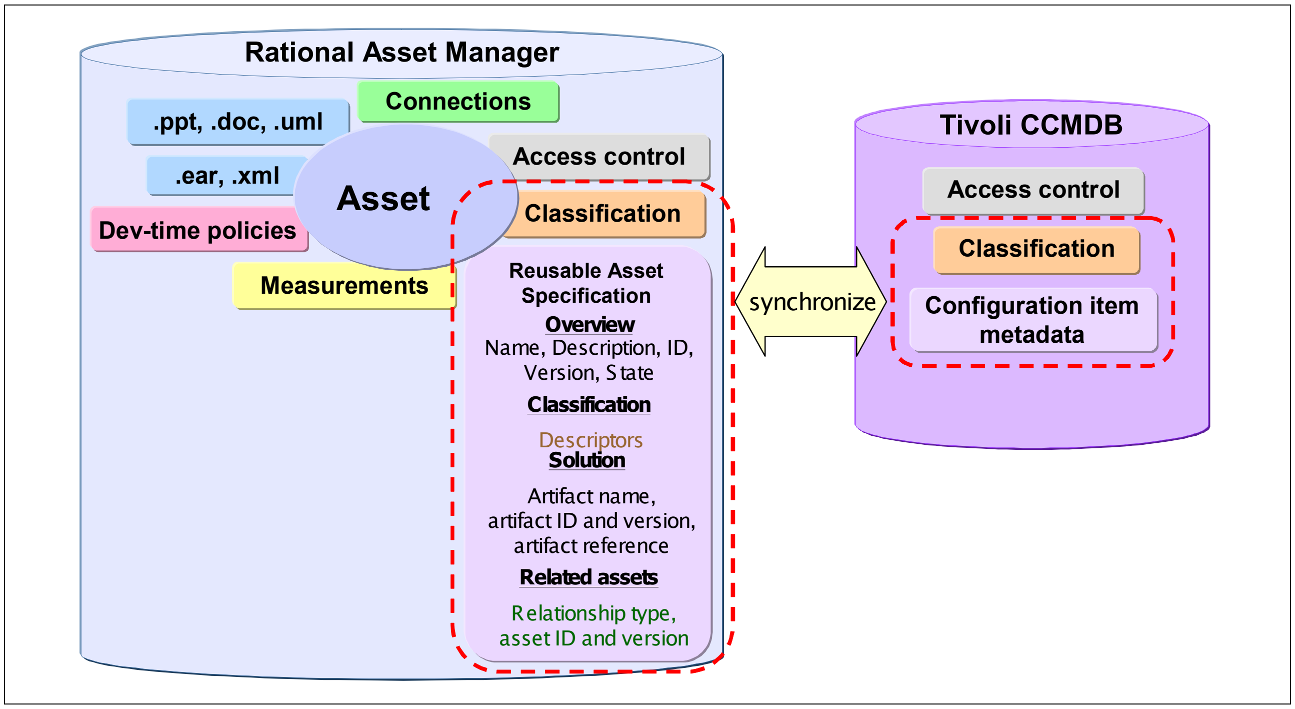


Figure 12-6 Asset metadata in Rational Asset Manager

Synchronizing asset classification and metadata reduces administrative overhead, simplifies developer use, and enables the solution to the challenges introduced earlier.

The user communities and content managed by Rational Asset Manager and Tivoli CCMDB are rather different. Each of the communities for Rational Asset Manager and Tivoli CCMDB has many sources of content and information that they manage. Many assets have nothing to do with configuration items, and not every configuration item should be associated with an asset. However, several artifacts can be of interest to both user communities.

Even with selecting a subset of the assets and configuration items in the repository, a many-to-many connection point is produced that adds to the complexity of tracing relationships and getting the information that IT needs to run its business. Figure 12-7 illustrates this concept.

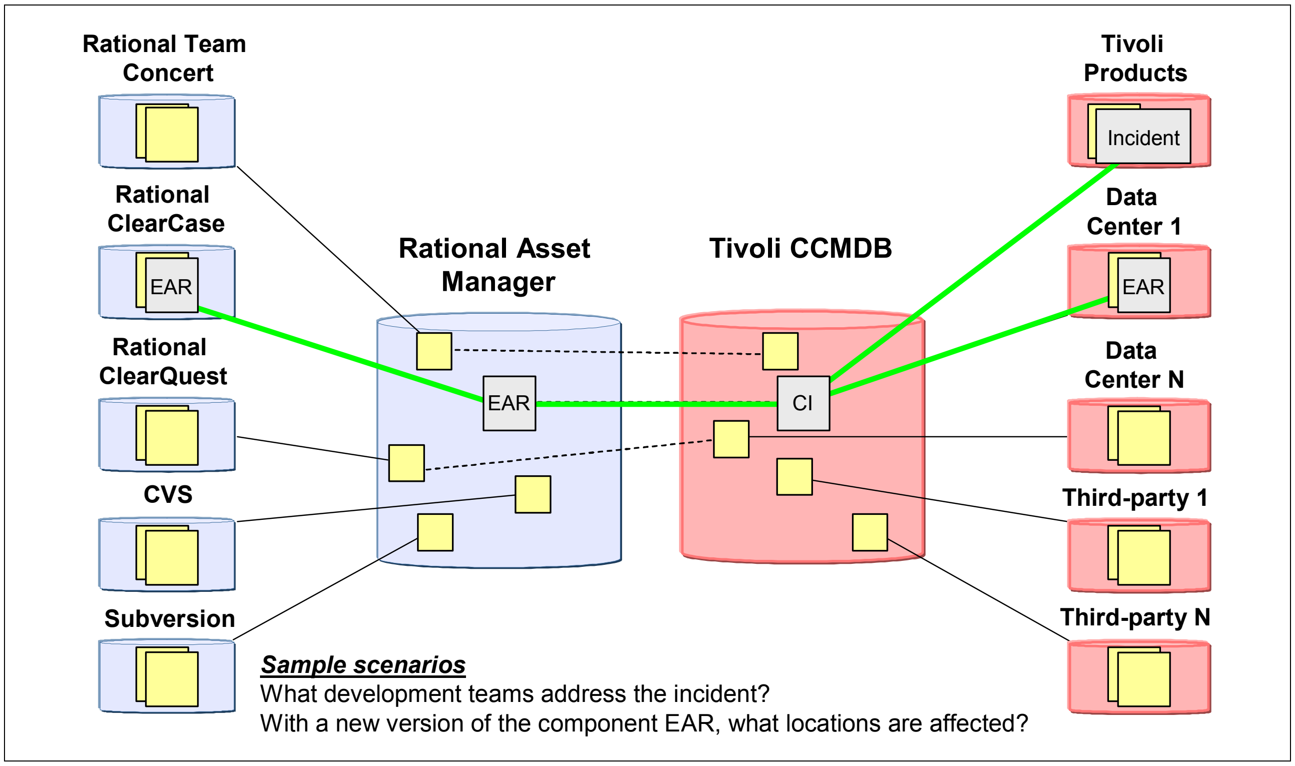


Figure 12-7 Providing traceable asset information to IT operations

There are several levels of integration for these references that are impacted by the volatility of the references that are stored in each of the repositories.

The metadata is federated across the repositories to provide community-relevant views and context. In some cases, this is achieved by physically replicating information, while in other cases, links are used to allow users to navigate from one repository to the other. For any artifact that is replicated between the repositories, the repositories declare whether they have the master copy or the cached copy.

Different communities require different governance and authorization models. Often in a development context, a rather coarse grained workgroup model is used to facilitate collaboration where any member of a workgroup can perform the same operations on an artifact once in the scope of the workgroup. However, governance requirements for CIs often imply a much more fine-grained and restrictive model for who can perform which tasks on specific artifacts. Rather than trying to unify these authorization models, we introduce *bridging roles* in each community that are authorized to make content from one environment available in another environment.

The enterprise must determine what to connect across the repositories. In general, we recommend against connecting all possible dots, as shown in Figure 12-8. It is not reasonable to consider connecting all possible Rational Asset Manager assets for all asset types, to all possible CIs in the CCMDB.

A more reasonable approach is to identify the cross-repository scenarios that are relevant to the enterprise. Then identify the assets and CI relationships that are needed to support those scenarios, selecting a subset of the assets and CI relationships that should be managed (Figure 12-8).

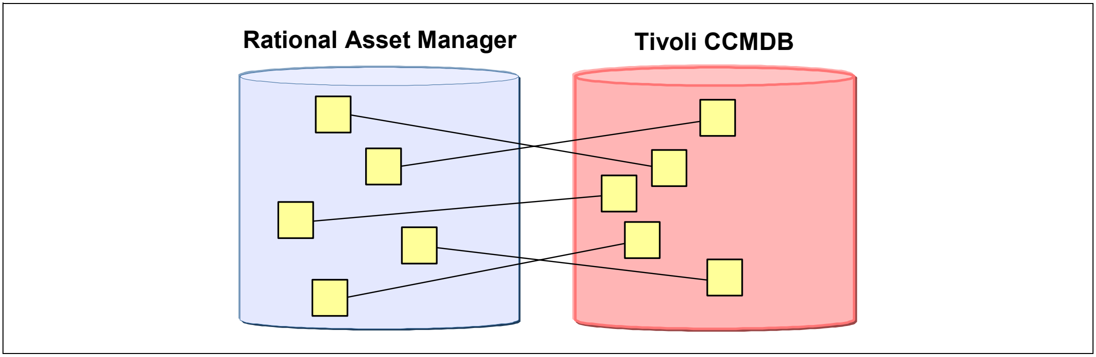


Figure 12-8 Cross-repository tracing scenarios

The Rational Asset Manager and CCMDB integration is focused on establishing the core plumbing to address the challenges that we identified earlier. This integration includes the following major capabilities:

- Determine the appropriate Rational Asset Manager assets and CCMDB CI linkages to create.
- Create the linkages between the Rational Asset Manager assets and CCMDB CIs.
- Keep linkages fresh.
- Discover and navigate linkages across repositories.

Rational Asset Manager v7.1 provides an integration module so that Rational Asset Manager can be configured as a physical DSL repository in Tivoli Release Process Manager. The integration module allows the browsing of assets in Rational Asset Manager from the DSL application of the Tivoli Release Process Manager.

The release manager can browse for software assets in Rational Asset Manager that are ready for deployment and can import those assets as software configuration items into the CCMDB. CIs are created and added to the DSL catalog. The operating status of these CIs can then be controlled in the normal way in the CCMDB and Tivoli Release Process Manager. When the software CIs are ready for production deployment, the operating status can be changed to Ready, for example, when the testing phase is complete and the release manager approves the package for deployment. Software CIs in the Ready status can be added as source CIs for deployment tasks in Tivoli Release Process Manager. Tivoli Release Process Manager then manages the deployment of these CIs to the target resources through a combination of manual and automated steps.

An alternate path is to export this information from Rational Asset Manager by using the discovery library adapter (DLA), creating an International Development Markup Language (IDML) book that can be imported by using IBM Tivoli Application Dependency Discovery into the CCMDB.

The integration between Rational Asset Manager and CCMDB also works in the other direction. You can automatically create assets in Rational Asset Manager from existing CIs in CCMDB. When Rational Asset Manager creates an asset from a CCMDB CI, it is stored as a remote asset. The remote asset contains a link to the CI in the CCMDB, and similarly the user can navigate to the asset in Rational Asset Manager from the CI in CCMDB. Remote assets in the Rational Asset Manager repository retain much of the metadata from the CCMDB CI. They are also given the asset type of “Configuration Item” and the category of “CCMDB,” which can be used as search filters to more quickly search for remote assets in the Rational Asset Manager repository.

13

The Rational ALM solution for solution delivery

In this chapter, we provide detailed "how to" information for "Act 5: Delivering the solution" in the storyboard, where the team uses the Rational ClearQuest Application Lifecycle Management (ALM) solution to manage the release and delivery of the solution.

This chapter discusses the following topics:

- An overview of the product features that are used in the demonstrated scenario
- A step-by-step demonstration of the scenario

Specifically, this chapter includes the following sections:

Role-based guide: To understand how the content in this chapter applies to your role, see the role-based guide in Table 1-1 on page 14. The key for this table is shown in Figure 1-7 on page 13.

13.1 Act 5: Delivering the solution

This chapter includes a step-by-step discussion of how the characters in the story for Act 5 of the storyboard, which is illustrated in Figure 13-1.

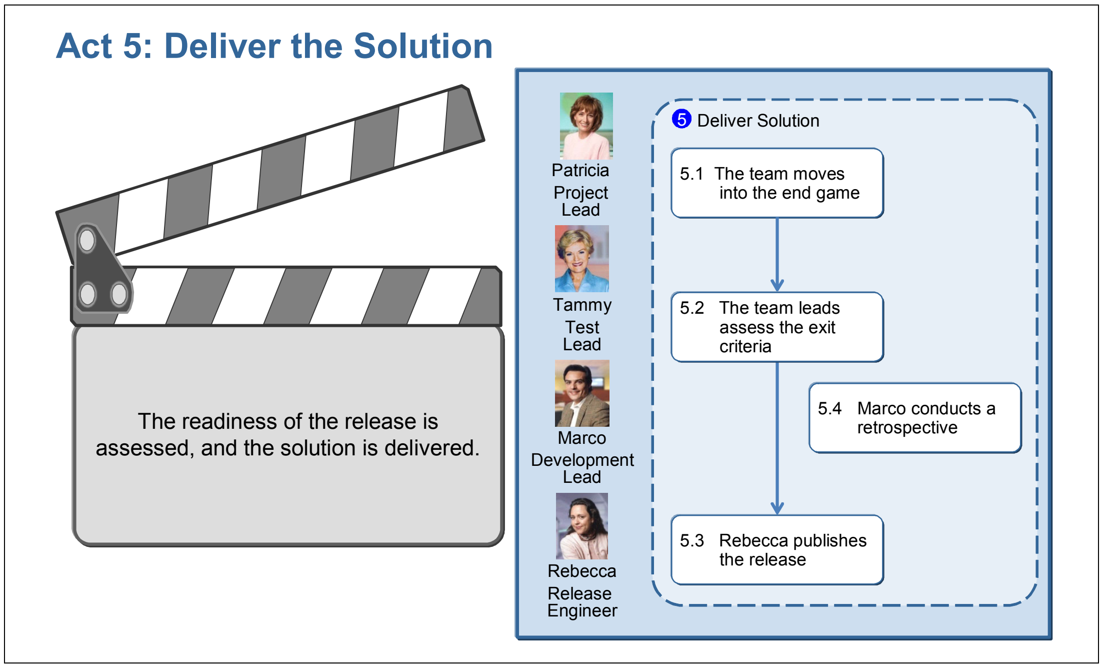

Act 5: Deliver the Solution
The readiness of the release is assessed, and the solution is delivered.
Patricia Project Lead
Tammy Test Lead
Marco Development Lead
Rebecca Release Engineer
5 Deliver Solution
5.1 The team moves into the end game
5.2 The team leads assess the exit criteria
5.4 Marco conducts a retrospective
5.3 Rebecca publishes the release

Figure 13-1 Act 5: The team completes the iteration and delivers the solution

This act, as shown in Figure 13-1, consist of the following scenes:

- Patricia moves the team into the iteration end game.
- The project team leads assess their iteration exit criteria and approve the release.
- Rebecca publishes the release.
- Marco conducts a retrospective with his team.

The following Rational products are used in this act:

- Rational ClearQuest 7.1
- Rational Team Concert 1.0

13.2 The Rational ALM solution and solution delivery

In this section, we explain how the Rational ALM solution supports solution delivery.

13.2.1 Rational ClearQuest and solution delivery

Rational ClearQuest has been configured in this IBM Redbooks publication as the ALM team solution that provides request management, iteration planning and alignment, defect tracking, process and workflow automation, reporting, and life-cycle traceability for better visibility and control of the software development life cycle. In this part of the scenario, we discuss how the ClearQuest ALM solution is used for the phase of closing the iteration and delivering the solution.

One of the key capabilities in the ClearQuest ALM solution is the ability to track the state of planned work to be completed in an iteration or release. It provides the required capabilities for project leads to continuously monitor the trends of the component or functional teams as the project converges to completion.

At completion of a release, each team has to confirm the readiness of their contributions. The ClearQuest ALM solution enables teams to configure approval tracking to manage such confirmations. The ClearQuest ALM solution can also be used to manage the process of continuous improvements, or retrospective, that are used in the Agility-at-Scale method.

13.2.2 Rational Team Concert, Rational Quality Manager, and solution delivery

Both Rational Team Concert and Rational Quality Manager provide key capabilities to the development and test teams in answering the most critical question: “When are done?” With Rational Team Concert, development teams track their progress and health by using the iteration plan. This plan indicates the remaining work and its estimates to complete the iteration. Tests place emphasize the test plan to enable a team to track their progress against the quality objectives of that plan. In both cases, the customizable dashboard is a key component for the teams to track their trends and plans. Dashboards can be created for each project, iteration, view of interest, or team member.

During the end game of an iteration, it is key to focus the team on remaining work while controlling the rate of change that is delivered to the iteration. Rational Team Concert contains key capabilities to enact team processes, such as reviews, to manage change and deliveries during the end game.

13.2.3 Rational Build Forge and Solution Delivery

In 8.1, “Introduction to enterprise build management” on page 316, we describe Rational Build Forge capabilities as a build management solution that provides an adaptive framework for standardizing and automating build tasks. In this chapter, we expand on the capabilities of Rational Build Forge to complete ALM by contributing to the release management and system delivery. Here Rational Build Forge provides the following benefits:

- Automation of package and deploy processes with push-button execution for increased efficiency and reliability
- A complete bill of materials that lists the changes and contents of each build for more accurate testing, problem resolution, and compliance management

- Enterprise reporting for documenting delivery events
- Integration with existing development and deployment technologies so that teams can leverage existing tool investments and integrations

13.3 The Rational ALM solution for solution delivery

In this section we provide detailed information about how the project team uses the Rational ALM solution to deliver their iteration, as illustrated in Figure 13-2.

Synopsis: In Act 5: Delivering the solution, we demonstrate how the team uses the Rational ALM solution to deliver the iteration release. The development team switches to an end-game process to ramp down on the changes, and team leads start to assess their exit criteria for the iteration. The team agrees that the iteration exit criteria has been met and Patricia approves the release. While the teams conduct retrospective, to continuously improve their practices, Rebecca publishes the iteration release.

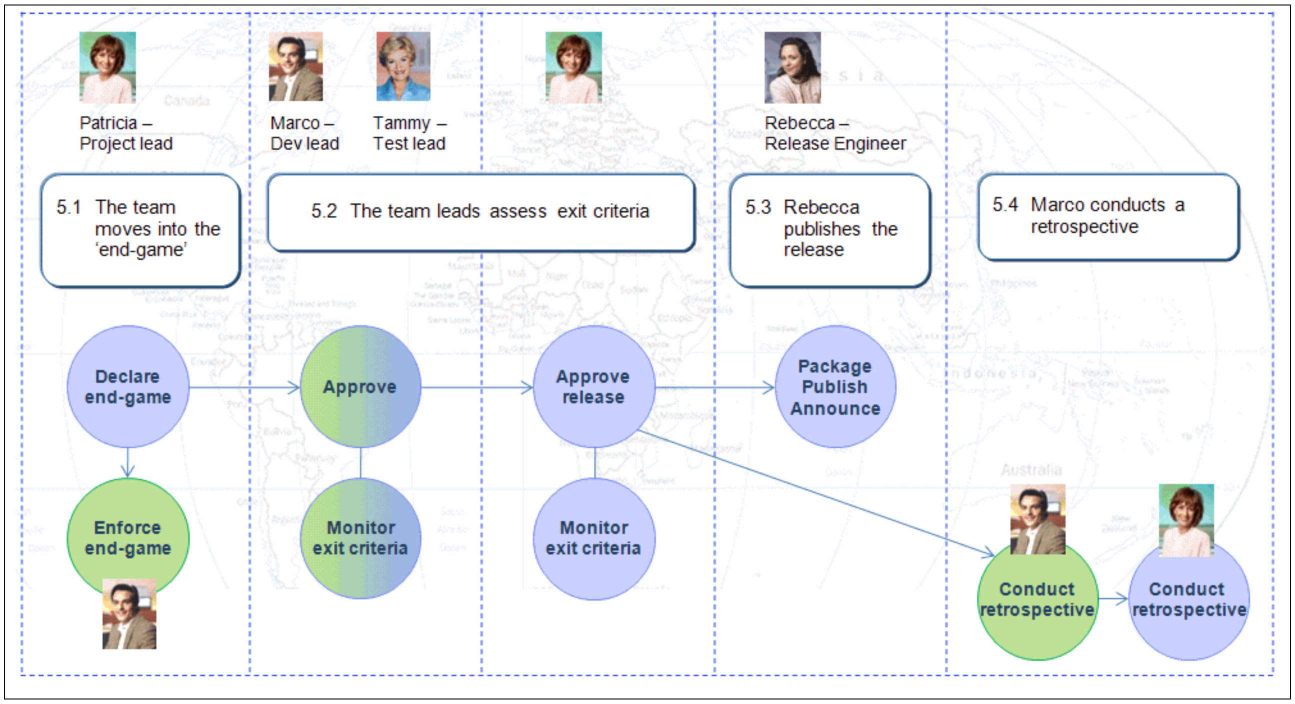


Figure 13-2 Software delivery workflow for this scenario

The solution delivery in Act 5 updates several assets and establishes traceability to support the life-cycle collaboration and tracking. Figure 13-3 show the life-cycle assets that are updated at the end of this act. See 13.4, "Life-cycle collaboration" on page 519, which describe the updates to the artifacts and relationships in detail.

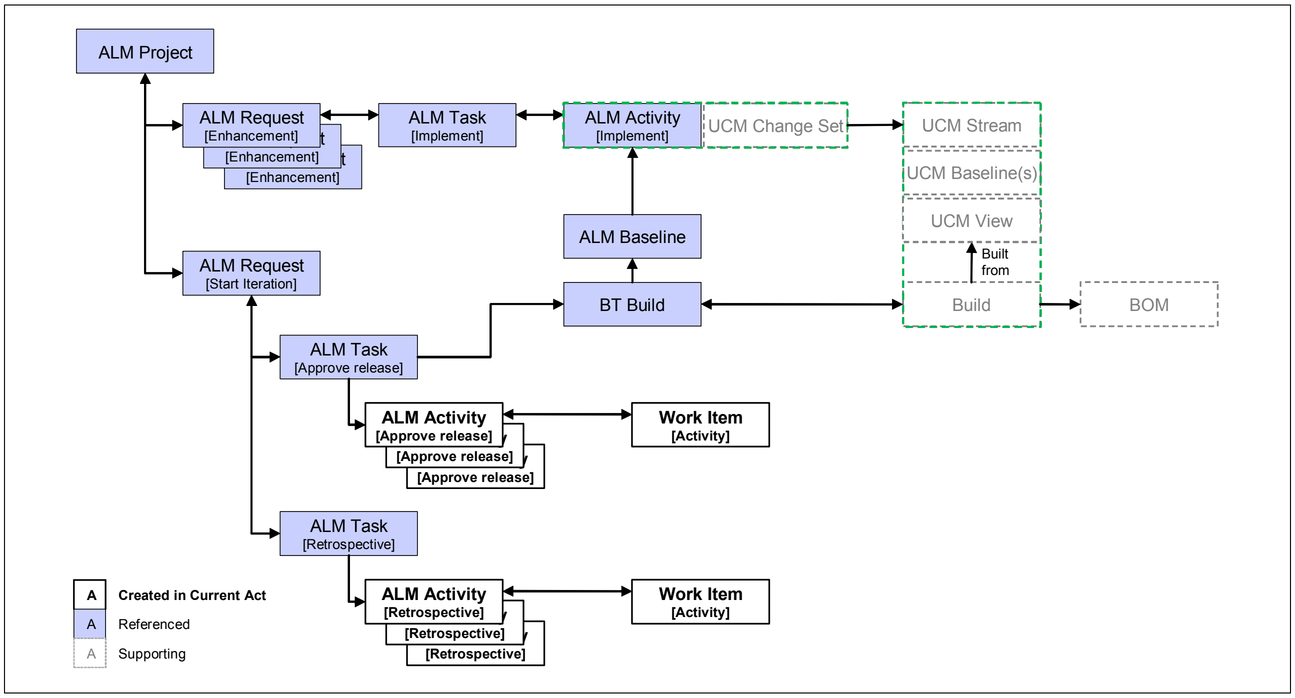


Figure 13-3 Life-cycle collaboration artifacts established in solution delivery

13.3.1 The team moves to the end-game

Synopsis: The end of the iteration is approaching. Patricia and her leadership team have been practicing an end-game process to ensure convergence of change at the end of each iteration. During the end game, deliveries from the teams require approvals by component leads or by project leads in the final Transition phase. Today Patricia moves the project to the end game and requests her leadership team to enforce this process with their teams.

Marco forwards the information about the process change at his next team stand-up meeting and uses the development platform to enforce the tighter end-game process.

The following steps occur in this scene:

- Declare the end game.
- Enforce the end game.
- Review and approve changes and deliveries.

Figure 13-4 show the life-cycle assets and relationship updates in the end game.

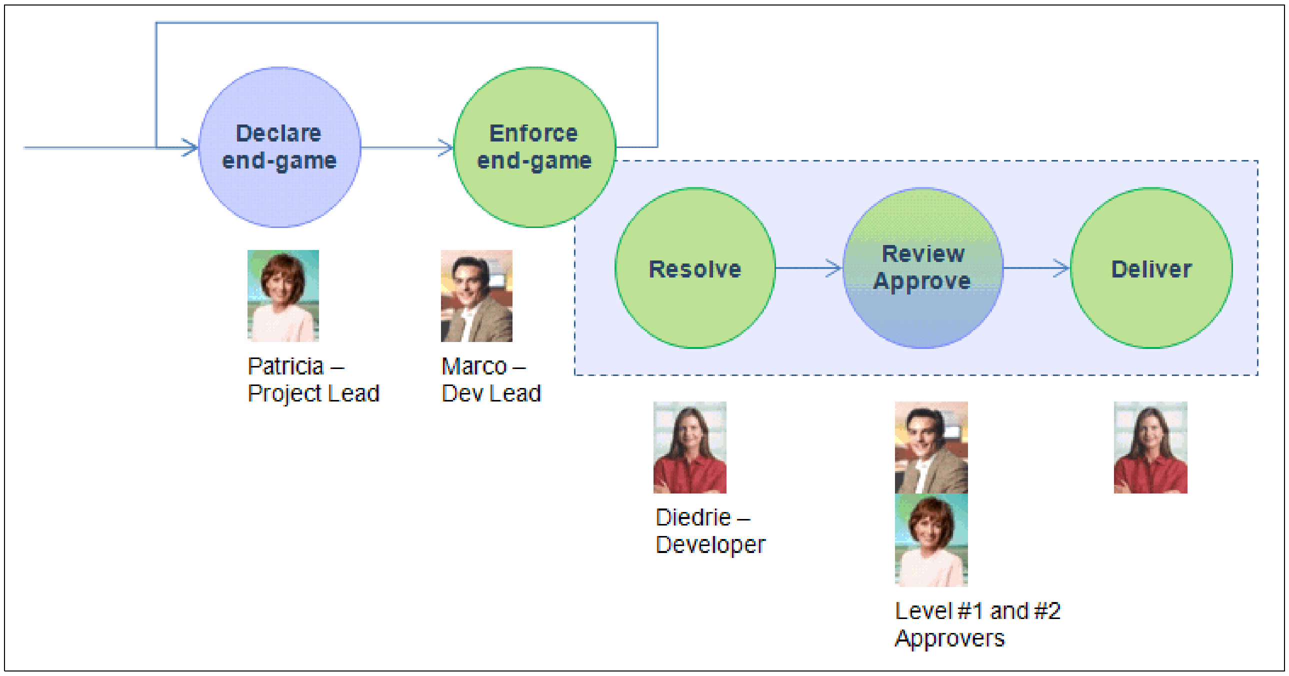


Figure 13-4 The end-game workflow in Act 5

Patricia declare the end game

Goal: The goal is to tighten the development process in order to stabilize the release and ramp down the change rate.

To move the iteration and declare the end-game process, Patricia takes the following actions:

- She discusses her decision at her leadership team stand-up meeting.
- She instructs her team leads to enforce the end game.
- She sends a notification to the project team.

Marco enforces the end game

Goal: The goal is to enforce the end-game process and prevent unapproved changes to be delivered to the integration.

Marco informs the component team about the end-game process at his next team stand-up meeting. He declares to the team that he intends to move to the stabilization phase of the iteration. This phase enforces limitations on delivery permissions to control the change rate.

To enforce the end-game process, Marco performs the following steps in Rational Team Concert:

1. Marco opens the AccountOpening project area and browses to the current iteration in the **Process Iterations** section.
2. He expands the **Construction Iteration C2B** iteration and right-clicks the **Stabilization** phase. Then he selects **Set as Current** as shown in Figure 13-5 on page 501.

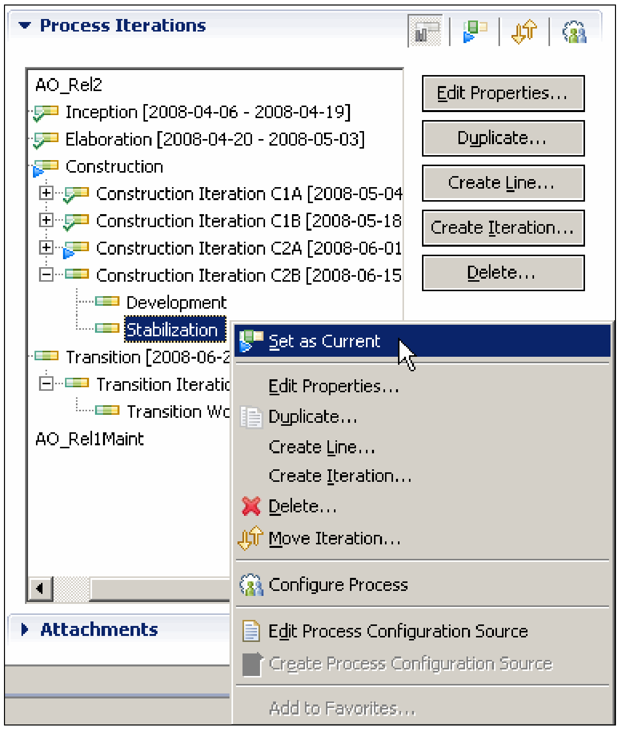


Figure 13-5 Enforcing the end-game by setting the Stabilization phase as current

By selecting Stabilization, Marco enforces the end-game policies to the team. The policies are set for the entire project or per team area. In “Configuring Rational Team Concert for solution delivery” on page 525, we discuss how to configure the Rational Team Concert process to enforce end-game reviews and approvals as part of the delivery process.

Diedrie and Marco resolve, approve, and deliver the changes

In this section, we describe how Marco and Diedrie manage a change delivery with the end-game practices.

Diedrie requests approval to deliver

Diedrie still has critical enhancements and defects to deliver before the final release delivery of the component. During the end game, she must review any work item resolution with Marco prior delivering the changes to the component integration (Figure 13-4 on page 500). This review process is similar to the review discussed in “Diedrie requests a review” on page 278.

To perform the review and gain approval on the delivery, Diedrie performs the following steps in Rational Team Concert:

1. Diedrie open the **Enhancement** work item.
2. She clicks the **Approvals** tab and then clicks **New Approval** to create a new approval for Marco.
3. In the New Approval window, she edits the Subject and types `Deliver`. She sets the due day to today’s date.
4. She clicks **Add approver** and adds Marco and Patricia as the approvers.
5. She clicks **OK** to submit the approval requests.

Marco approves the delivery

Marco receives a notification of the new approval request that was submitted by Diedrie. He performs the following steps to review and approves the delivery:

1. He clicks the event notification to open the work item and reviews the change sets that are attached to the work item.
2. He chats with or talks to Diedrie about the following aspects of the defect:
   – Is this enhancement truly critical for the application and the user?
   – What was the scope of the resolution? Are there any significant architectural implications?
   – How was the enhancement unit tested? Has the test team prepared test cases for the enhancement?
3. He returns to his desk, opens the work item, and clicks the **Approvals** tab.
4. He selects the State column on his entry in the Approvals section and selects **Approved** (Figure 13-6).
5. He saves the work item.

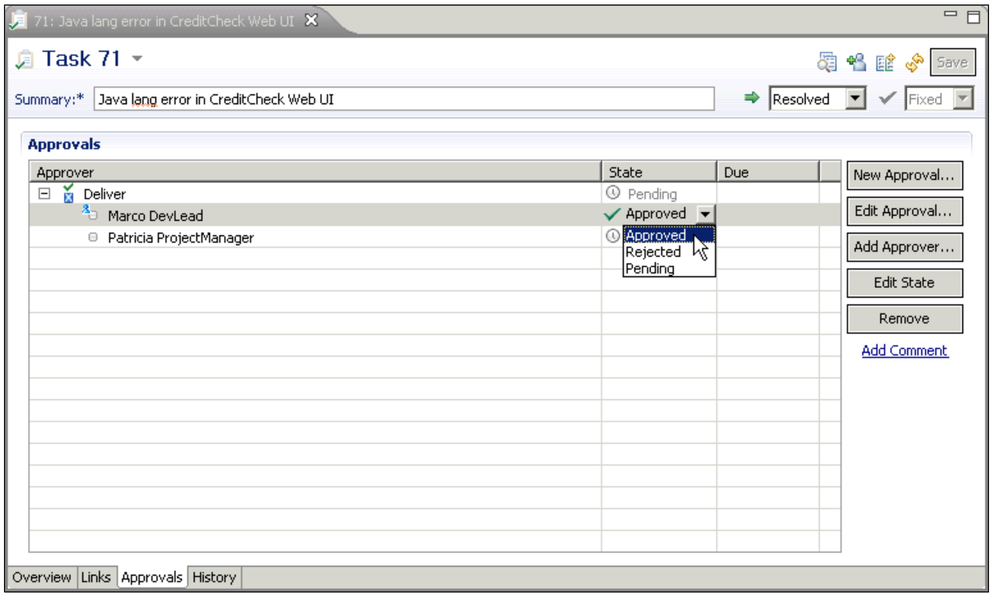


Figure 13-6 Marco giving the required approval to deliver a work item during the end game

Diedrie receives the notification that the enhancement has been approved for delivery. She now proceeds with her regular workflow to deliver her changes to the component integration stream, as discussed in “Diedrie delivers her changes for integration” on page 282.

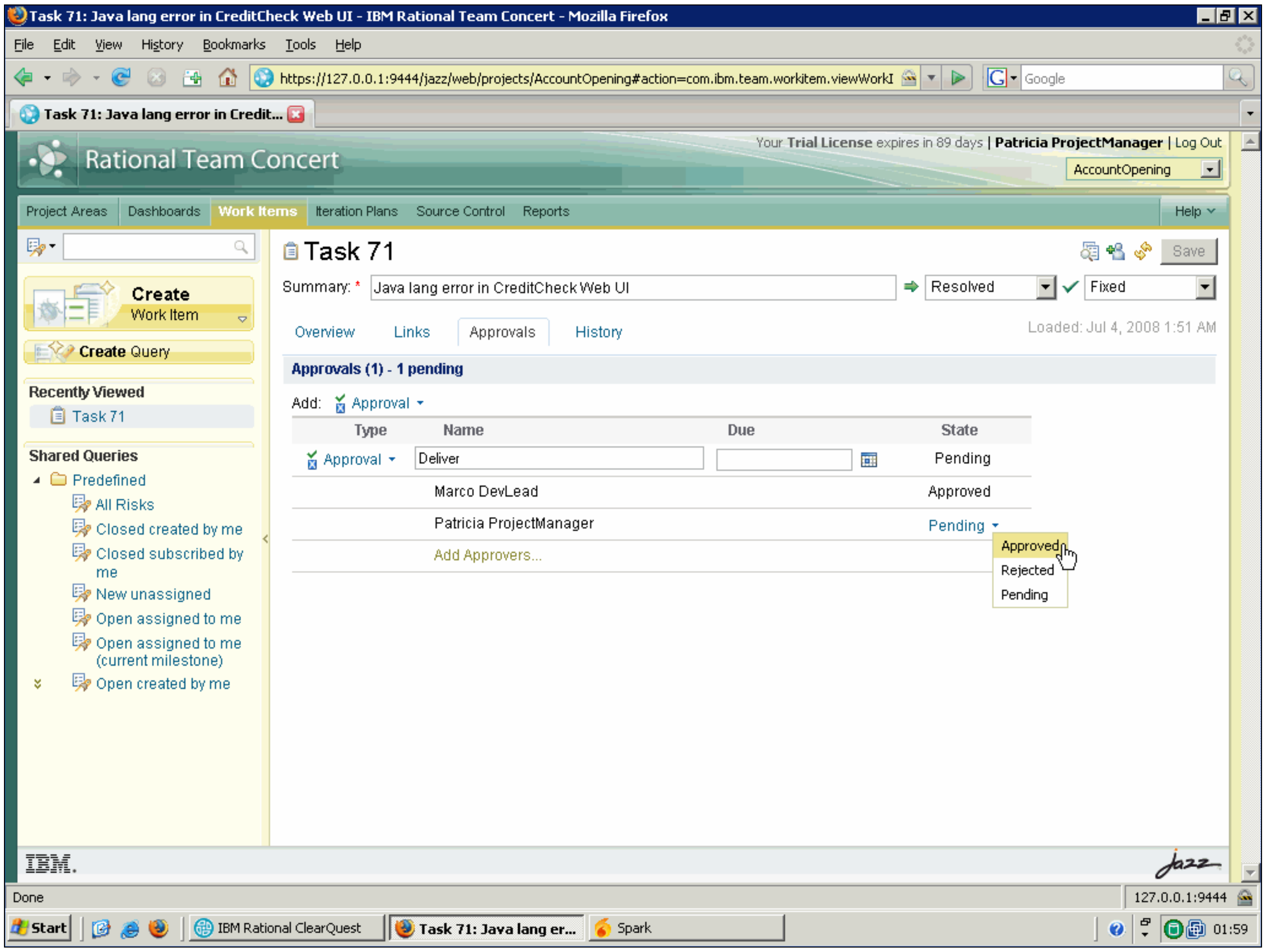
Patricia approves a delivery for the final release

For the end game of the final release, the practices that are used by the team state that the project manager should review and approve all deliveries. Patricia uses the Rational Team Concert Web UI to access and approve work items.

Patricia performs the following steps to review and approve a work item:

1. Because she is as an approver, she receives a notification.
2. She browses the link in the notification and logs into the Rational Team Concert Web UI.

3. On the work item, she clicks the **Approvals** tab. She clicks the State column on her entry in the Approvals section and selects **Approved** (Figure 13-7).


Task 71: Java lang error in CreditCheck Web UI - IBM Rational Team Concert - Mozilla Firefox

https://127.0.0.1:9444/jazz/web/projects/AccountOpening#action=com.ibm.team.workitem.viewWorkI

Your Trial License expires in 89 days | Patricia ProjectManager | Log Out

Task 71

Summary: * Java lang error in CreditCheck Web UI — Resolved — Fixed

Loaded: Jul 4, 2008 1:51 AM

Approvals (1) - 1 pending

Type	Name	Due	State
Approval	Deliver		Pending
	Marco DevLead		Approved
	Patricia ProjectManager		Pending
	Add Approvers...		

Figure 13-7 Patricia giving the required approval to deliver changes for the final release of the application

13.3.2 The team leads assess their exit criteria

In this scene Marco, Tammy, and Patricia monitor and manage outstanding work to reach their exit criteria and to approve the release.

Marco monitors the component exit criteria

Synopsis: Marco is focusing his team during the end game to converge the remaining work in the component iteration plan. He works with his team to prioritize low priority requests that need to be pushed out of the release. He confirms updates to the iteration plan with Patricia and Bob. He also tracks new defects that are submitted by the component team or by the test team from test failures in component builds or solution integration builds. He works with the owners of prioritized defects to assess impact, confirm importance, validate test status, and approve the deliveries. His team must close all enhancements or defects of blocking or critical severity and close all tasks of high priority to meet their component exit criteria.

Marco is reporting his progress to the exit criteria at Patricia’s daily leadership stand-up meetings. Today he can proudly report that his team has reached their objectives and that he has approved the component iteration release.

The workflow in this scene captures how Marco completes the following tasks:

- Monitors the completion of the iteration plan.
- Queries for new defects and approves deliveries.
- Queries and reports on the component exit criteria.
- Approves the component iteration release.

Marco monitors the component exit criteria

In this section, we discuss how Marco and his team are using Rational Team Concert to track the remaining work and assess when the component has reached sufficient functionality and quality to be signed and released.

Marco assesses his iteration plan

Marco is using the component iteration plan to review remaining work and push changes that are of low priority to the next iteration. While this is an continuous process, primarily managed by the individual team members that own the work, Marco focuses his attention and the attention of the team to cleaning up the iteration plan when moving into the end game. He also collaborates with his team members that has an indicated work overload and looks for opportunities on workload balancing, which entails moving work to the next iteration.

Marco validates the remaining work. The following work remains in the plan:

- Defects that are of blocking or critical severity
- Other high priority work item tasks
- Enhancements with agreements on late delivery

He uses the Rational Team Concert collaboration, in context of the work items, to run the discussions. This helps in tracking, and for other stakeholders, in understanding the justification for the reschedule. Marco also reviews, with the owners, the risk items that are identified for the iteration release. Marco uses the iteration plan in Rational Team Concert to resolve many of the validation points. He also uses a handful of queries for this purpose.

To review the iteration plan, Marco performs the following actions:

1. He browses the Team Artifacts view, navigates to **AccountOpening** → **Plans** → **Construction Iteration C2B**, and selects the **CreditCheck C2B** iteration plan.
2. He clicks the **Charts** tab to see a visual view of the work item closure rate for the iteration.
3. He clicks the **Planned Items** tab, and uses the sidebar control to configure the view. He excludes **Resolved items**. For Group by, selects **Folder**, and for Sorts by, selects **Severity** and **Priority**.
4. He works from the bottom up and collaborates with the work-item owners to validate progress, estimates, and opportunities for pushing work to the next iteration.

Marco triages the new defects

Marco is monitoring new incoming defects on the iteration that are found by his component team and by Tammy's test team. New defects are continuously picked up by the team and added to their schedules or pushed to later iterations. Marco is monitoring to make sure that they do not miss any defects. He also collaborates with the defect owners to ensure that the severity and priority set by the submitter are correct.

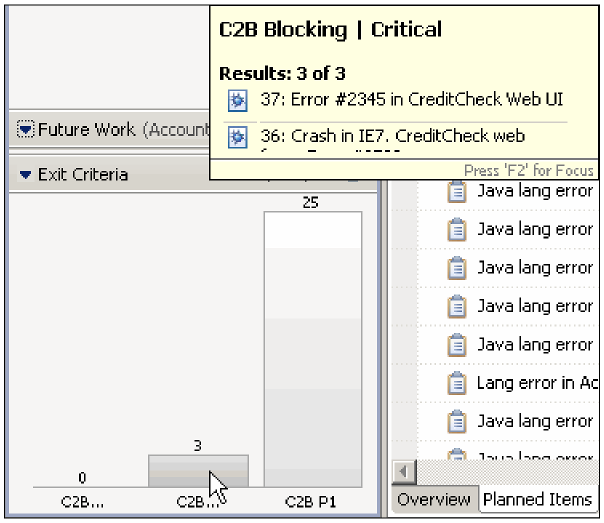
Marco uses Rational Team Concert to monitor the incoming defects:

- He uses the event log in the Team Central view to monitor new and updated work items.
- He uses the predefined Recently created work item query to look for new defects. He optionally drags the query to the Team Central view for continuous monitoring of new incoming work items.
- He uses in-context collaboration to discuss defects with the submitter and the work-item owner.
- He ensures that the defect owner is providing estimates on new work in order to use the iteration plan to assess overload and needs for work rebalancing.

Marco monitors the exit criteria

To monitor exit criteria in the Team Central view, Marco completes the following steps:

1. Marco opens the Team Central view.
2. He clicks the **Menu** button and selects **New Section** → **Queries**.
3. He right-clicks the new Queries section and selects **Rename**. He names the section `Exit Criteria`.
4. He creates a new work item query by selecting **File** → **New** → **Work Item Query**.
5. He names the query `C2B Blocking | Critical`. He then defines the query parameters. For example, Status is New or InProgress, FiledAgainst is CreditCheck category, PlannedFor is Construction Iteration C2B, and Severity is Blocker or Critical.
6. To make the query public to his team, he clicks the **Details** tab. In the Sharing section, he selects **Share** → **Team Area**, and from the Select a Team Area window, selects **CreditCheck**. He saves the query.
7. Marco wants to track the progress and converging of the exit criteria. He drags the new query to the new Team Central Exit Criteria section. The query is run, and the result is represented as a bar chart with a count on the Blocking or Critical work items that remain to be delivered during the end game. By hovering with the cursor over the bar, Rational Team Concert open a window with the query result, including clickable links to the work items (Figure 13-8 on page 506). Marco proceeds to configure other exit criteria queries to be added to the Team Central section.


C2B Blocking | Critical

Results: 3 of 3

37: Error #2345 in CreditCheck Web UI

36: Crash in IE7. CreditCheck web

Figure 13-8 Marco configuring Team Central to track the exit criteria

Marco also ensures that the rest of the team can monitor the exit criteria in the Component Team Dashboard. To configure a dashboard for monitoring the exit criteria, Marco completes the following actions:

1. He opens the Account Opening Project Area and clicks the **Process Configuration** tab.
2. He browses the Configuration section for **Project Configuration** → **Configuration Data** → **Work Item** → **Predefined Queries**.
3. He clicks the **Add** button in the Predefined Queries section to add the new "C2B Blocking | Critical" query.
4. He clicks **Save** to save and deploy the changes.
5. He browses to the Web UI by opening the Team Artifacts view, right-clicking the **AccountOpening** project area, and choosing **Open Web UI for Project**.
6. In the AccountOpening dashboard, he clicks the **Add New** tab. On the new tab, from the command menu, he chooses **Rename**. He names the new tab `Exit criteria`.
7. He clicks **Add Viewlet** and configures the new viewlet by selecting **Development** → **Work Item Queries**. He configures the viewlet to contain the exit criteria queries that are configured in the previous section.

Marco approves the component release

The AccountOpening team has adopted a practice where each component lead or practice lead, such as Marco for the CreditCheck component and Tammy for the test practice, explicitly approves their exit criteria on the application iteration release. This practice helps Patricia to ensure that the responsibilities and ownership are clear and trackable.

Patricia is using Rational ClearQuest and the ClearQuest ALM schema to manage and monitor the required release approvals, as discussed in "Patricia approves the release" on page 508. For release candidates, she uses the ALMTask records to manage the team collaboration and approval process, and the associated ALMActivities that are assigned to each approver, for example, the component leads, to contain the approvals by the team component or practice team. Team interoperability, which is implemented using the ClearQuest Connectors as discussed in "Configuring interoperability" on page 565, is synchronizing the ALMActivities with the Rational Team Concert and Rational Quality Manager platforms. The teams approve their releases within the scope of their regular workflows. Patricia can track the approvals by monitoring the completion of the activities in ClearQuest.

With the final release date approaching, Rebecca is building releases that become candidates for the final release version, and Tammy is completing her test plan to validate the release candidates. Patricia consults her leadership team on their exit criteria, and the team decides to declare the latest build a candidate for release. Patricia submits the approval records for the teams, as discussed in “Patricia approves the release” on page 508. Marco receives a notification of a pending task work item to approve for the release candidate.

To approve the release candidate, Marco performs the following actions:

1. Marco opens the work item that is assigned to him by Patricia.
2. He click the **Approvals** tab, adds a new approval item, adds himself as an approver, and chooses the approved state.
3. He changes the state of the work item to **Completed** and saves the work item.

The updated work items are synchronized with Rational ClearQuest, and Patricia can view the completed approval by Marco.

Tammy monitors the quality exit criteria

Synopsis: Tammy and her test team are monitoring the iteration test plans to validate that the solution release has met its quality goals. She is monitoring the list of quality exit criteria for the iteration release, which includes build validation test stability, code coverage of test cases, functional test failures, and application load performance.

On the final day of the iteration Tammy confirms that the solution meets the quality exit criteria. She informs Patricia that the quality has met the expectations, all requirements have been tested, and she has approved the release quality.

The workflow in this scene captures how Tammy completes the following tasks:

- Monitors the completion of the test plan
- Queries and reports on quality exit criteria
- Approves the quality of the iteration release

Tammy monitors her test plan

Tammy is continuously monitoring her quality metrics. She performs this activity to maintain a real-time view of quality and team health. The steps to continuously monitor quality are described in “Reviewing the test plan” on page 146, and in 11.5.1, “Tammy monitors quality” on page 425.

Tammy and her team use their dashboards to access a live view of the quality and health metrics for the application iteration. As previously shown in Figure 11-54 on page 456, she uses the Trends tab on the dashboard, which is populated with viewlets that provide trend and real-time information, such as the Execution Trend Report, Requirements changes, Requirement Coverage Status, and Defect Arrival Resolution.

Tammy assesses her quality exit criteria

Tammy uses the combination of dashboard viewlets, reports, and plan exit criteria to determine when the iteration is complete. This is a combination of confirming that all planned work is completed, all requirements have been validated, and defects have been fixed to the agreed upon level.

Tammy is monitoring the list of quality exit criteria for the iteration release:

- 100% of the P1 defects fixed by development teams
- 100% of the Requirements covered with test cases
- Achieved quality objectives of all enhancements

Tammy also monitors the trends for the following items:

- Build validation test stability
- Code coverage of executing test cases
- Failures in functional test
- Application load performance

For the steps to configure the queries and reports on quality, see 11.5.5, “Monitoring quality” on page 455.

Tammy approves the iteration quality

The AccountOpening team has adopted a practice where each component lead or practice lead explicitly approves their exit criteria on the application iteration release. Patricia is using Rational ClearQuest and the ClearQuest ALM schema to manage and monitor the required release approvals, as discussed in “Patricia approves the release” on page 508.

To approve the release candidate, Tammy performs the following actions:

1. Tammy opens the work item that is assigned to her by Patricia.
2. She click the Approvals tab, adds a new approval item, adds herself as an approver, and chooses the approved state.
3. She changes the state of the work item to **Completed** and saves the work item.

The updated work items are synchronized with Rational ClearQuest, and Patricia can view the completed approval by Tammy.

Patricia approves the release

Synopsis: Patricia confirms that the release requirements planned for the iteration have been implemented and there are no outstanding tasks for the team. She also confirms with her leadership team on the approval status across the project.

She uses the ALM solution to manage and track the approvals from the component and practice leads on the release candidate. After all approvals are collected, she makes a final assessment of the iteration exit criteria and agrees that the iteration is complete. She approves its release and notifies Rebecca to publish the release.

The workflow in this scene captures how Patricia completes the following tasks:

- Monitors the completion of the project iteration plan
- Monitors the approvals from the project teams
- Approves the iteration release

Patricia assesses the project iteration plan and her exit criteria

Patricia is using the project iteration plan in Rational ClearQuest to review remaining work for the iteration. She also collaborates with her teams’ clean up of the iteration plan and moves work to next iteration.

Patricia validates the remaining work by querying for the following items:

- Reviewing all tasks that are planned for this iteration, but have not been closed
- Defects that are of blocking or critical severity

She uses Rational ClearQuest collaboration, in context of the ALMTask, to run the discussions with the owner. This helps in tracking and for other stakeholders to understand.

To exit the iteration and release the application iteration, she must perform the following tasks:

- Complete all requests that are planned for the iteration.
- Complete all tasks that are planned for the iteration.
- Pass the quality exit criteria.
- Pass all component exit criteria.

Patricia uses the Project Requests and Current Iteration queries to validate the state of the requests and tasks that are planned for the iteration. See 5.5.5, “Patricia confirms the project iteration plan” on page 155, for details about the ClearQuest ALM queries.

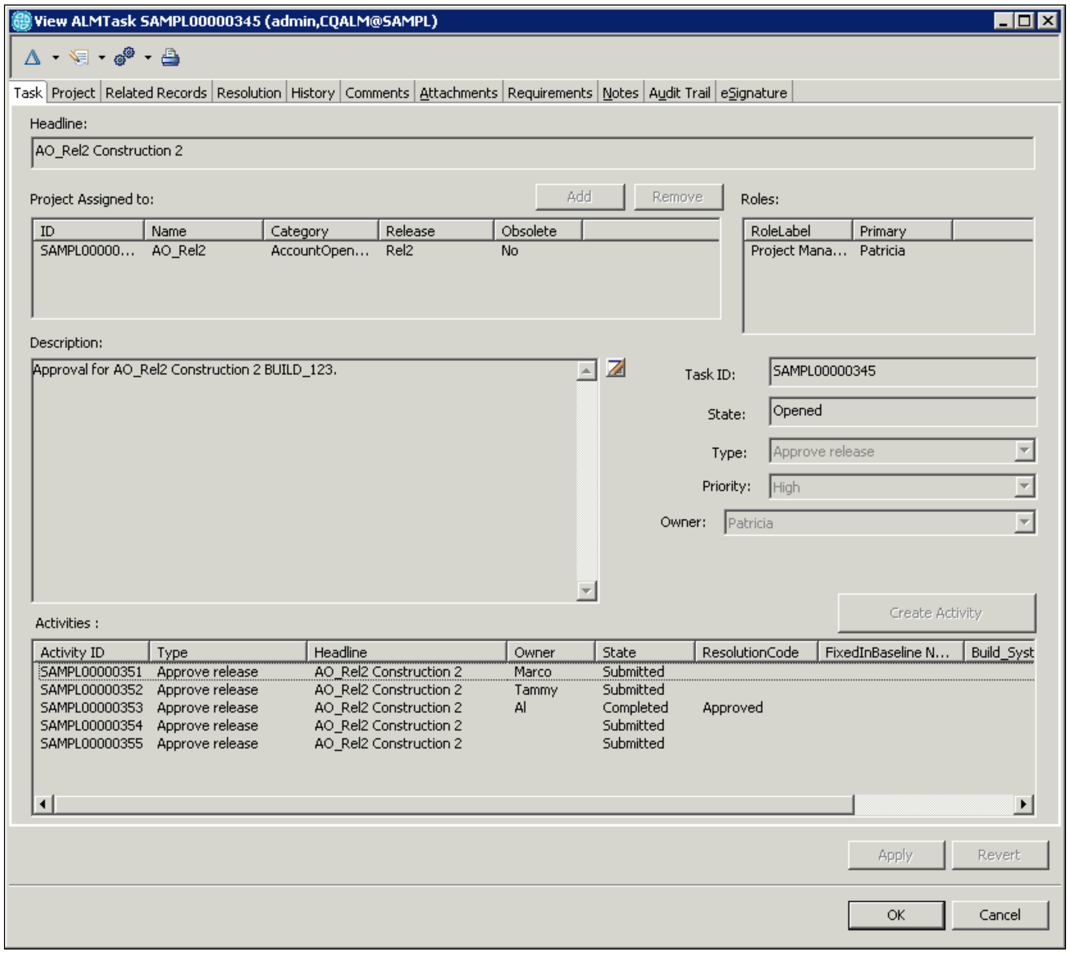
Patricia approves the iteration release

The team is using Rational ClearQuest and the ALM schema to extend and configure the work configuration to include “Approve release” as ALMTask and ALMActivity types on the “Start Iteration” ALM Request type. The steps to configure the ALM schema for approvals are discussed in “Configuring ALM for approvals” on page 521.

To create an approval task, Patricia completes the following actions:

1. Patricia opens the ALMRequest **AO_Rel2 Construction 2** of type Start Iteration.
2. In the Tasks section, she opens the ALMTask of type **Approve release**. She clicks the **Modify** button.
3. She clicks the **Create Activity** button one or more times and then clicks the **Apply** button to save the changes. After saving the changes, the window is updated, and one or more ALMActivities are listed in the Activities section.

4. She opens the first activity, assigns it to Marco, and saves the changes. This activity instructs Marco to approve the component for the "AO_Rel2 Construction 2" iteration. She proceeds with the other activities and assigns them to the other component and practice leads. Patricia can now track the completion of the team approvals. After all team leads complete their activities, Patricia can proceed and make the final approval for the application as shown in Figure 13-9.
5. She opens the **Approve release** ALMTask and updates the state to **Complete**. She proceeds and sets the resolution code to **Approved**. She then clicks **OK** to save the record.
6. Because Rebecca has been added to the notifications list, she receives an e-mail notification that the release has been approved. Rebecca proceeds and begins the work to package and publish the release.


View ALMTask SAMPL00000345 (admin,CQALM@SAMPL)
Task | Project | Related Records | Resolution | History | Comments | Attachments | Requirements | Notes | Audit Trail | eSignature
Add
Remove
Create Activity
Apply
Revert
OK
Cancel

Headline: AO_Rel2 Construction 2

Project Assigned to:

ID	Name	Category	Release	Obsolete
SAMPL00000...	AO_Rel2	AccountOpen...	Rel2	No

Roles:

RoleLabel	Primary
Project Mana...	Patricia

Description: Approval for AO_Rel2 Construction 2 BUILD_123.

Task ID: SAMPL00000345
State: Opened
Type: Approve release
Priority: High
Owner: Patricia

Activities :

Activity ID	Type	Headline	Owner	State	ResolutionCode	FixedInBaseline N...	Build_Syst
SAMPL00000351	Approve release	AO_Rel2 Construction 2	Marco	Submitted			
SAMPL00000352	Approve release	AO_Rel2 Construction 2	Tammy	Submitted			
SAMPL00000353	Approve release	AO_Rel2 Construction 2	Al	Completed	Approved		
SAMPL00000354	Approve release	AO_Rel2 Construction 2		Submitted			
SAMPL00000355	Approve release	AO_Rel2 Construction 2		Submitted			

Figure 13-9 Patricia tracking approvals in Rational ClearQuest

13.3.3 Rebecca publishes the release

Synopsis: Rebecca receives a notification from Patricia that the iteration release candidate has been approved. The release is now ready to be packaged.

Rebecca creates a new release package for the iteration release. The release code archives are already packaged as part of the build process that automatically pulls all release assets into the release archives. The development team has provided Rebecca with the documentation that should be included in the release. She includes the complementary material including the bill of materials, release notes, documentation, and samples.

Rebecca creates a new public release page on the enterprise IT intranet. She edits the release page and completes the release information. She provides download links to the release package that she just created. She saves and publishes the release page.

Rebecca uses the Account Opening RSS news feed to announce the availability of the latest iteration release. She links the feed to the release page.

At a later date, the final production release is made as the Account Opening project exits the Transition phase. Rebecca transitions the solution release to production by delivering it to the enterprise reuse repository. She logs into Rational Asset Manager, adds a new version to the Account Opening Solution asset, and submits all application release packages to the repository. She announces the new asset to the Account Opening community in Rational Asset Manager.

The workflow in this scene captures how Rebecca completes the following tasks:

- Creates a release package
- Creates a release page and announce the release
- Submits a new asset version to Rational Asset Manager

Patricia produces the release notes

Patricia is providing release notes with the release iteration. She uses a project template for the document and appends information about the new capabilities that are included in the release. She mainly receives this information from the ClearQuest ALM solution by using the capabilities of Business Intelligence Reporting Tool (BIRT) reporting, as described in 5.7.2, “Reporting with the Business Intelligence Reporting Tool” on page 178.

To the release notes, she adds the following information as reported from the ALM solution:

- All requests that are delivered in this iteration
- All remaining defects

Patricia also uses reports that are generated in Rational Quality Manager to state iteration quality.

She controls the release notes document in Rational ClearCase where it picked up by the release packaging process that is run by Rebecca.

Rebecca packages and publishes the release

Rebecca packages the release for delivery. She uses her central Rational Build Forge infrastructure to run the release packaging for the project release.

To run an integration, Rebecca performs the following activities:

1. Rebecca executes the Rational Build Forge project for packaging the Account Opening project. This project integrates with the development repositories that are used by the project:
   a. She collects the build identification from Rational ClearQuest.
   b. She collect the distribution archives from Rational ClearCase.
   c. She collects additional release documentation from Rational ClearCase.
   d. She packages the release archives.
   e. She delivers the release archives to the reuse repository.
   f. She announces the availability of the delivery by using Rational Build Forge notification capabilities.
2. Sets the release date and confirms the availability in Rational Asset Manager.
3. She publishes the asset to the IBM Tivoli Change and Configuration Management Database (CCMDB) for use by the operations team.
4. In Rational Build Forge, Rebecca selects the appropriate build project for the end of the iteration (or milestone). The Rational Build Forge Ant scripts generate the release packages and communicate with Rational Asset Manager, submitting the build assets and their relationships. In Rational Asset Manager, Rebecca finds the milestone asset and declares it to be released. Following the asset relationship to the build asset, she selects one or more software images and publishes them to the CCMDB as illustrated in Figure 13-10.

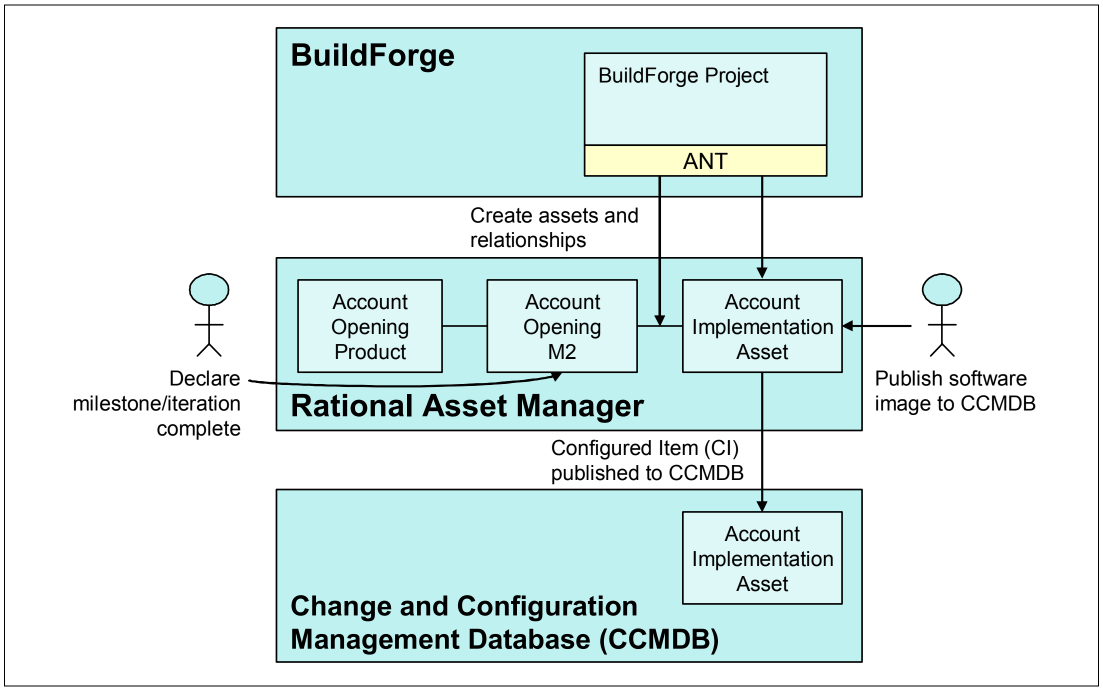
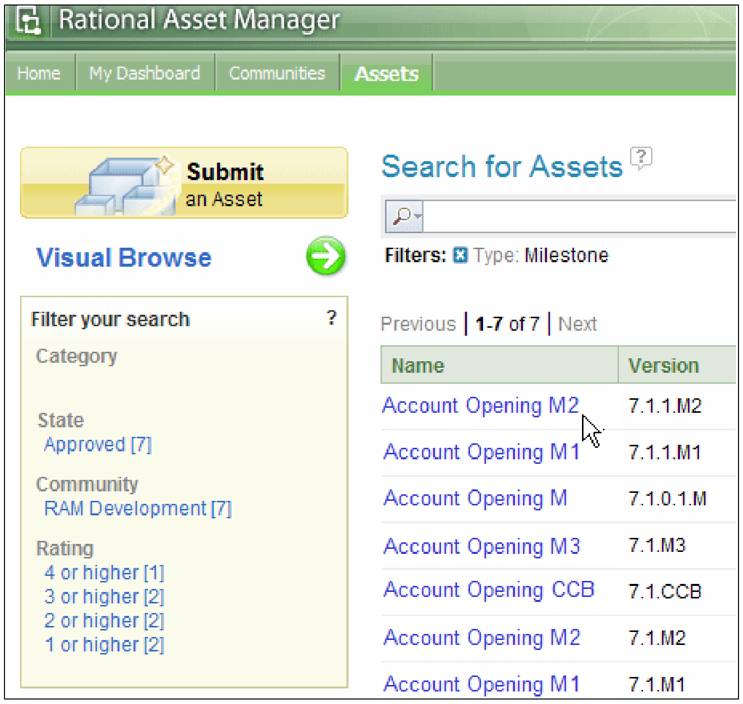

BuildForge
BuildForge Project
ANT
Create assets and relationships
Account Opening Product
Account Opening M2
Account Implementation Asset
Declare milestone/iteration complete
Publish software image to CCMDB
Rational Asset Manager
Configured Item (CI) published to CCMDB
Account Implementation Asset
Change and Configuration Management Database (CCMDB)

Figure 13-10 Overview of Rational Build Forge, Rational Asset Manager, and the CCMDB

5. Rational Build Forge creates the Account Opening software packages and submits them to Rational Asset Manager through the Rational Asset Manager/Ant integration. The Rational Build Forge project creates a build asset in Rational Asset Manager and connects it to the Milestone asset.

6. In Rational Asset Manager, Rebecca searches the milestone asset for the Account Opening project as shown in Figure 13-11. She selects the **Account Opening M2** milestone asset.



Filters: Type: Milestone

Filter your search

Category

State
Approved [7]

Community
RAM Development [7]

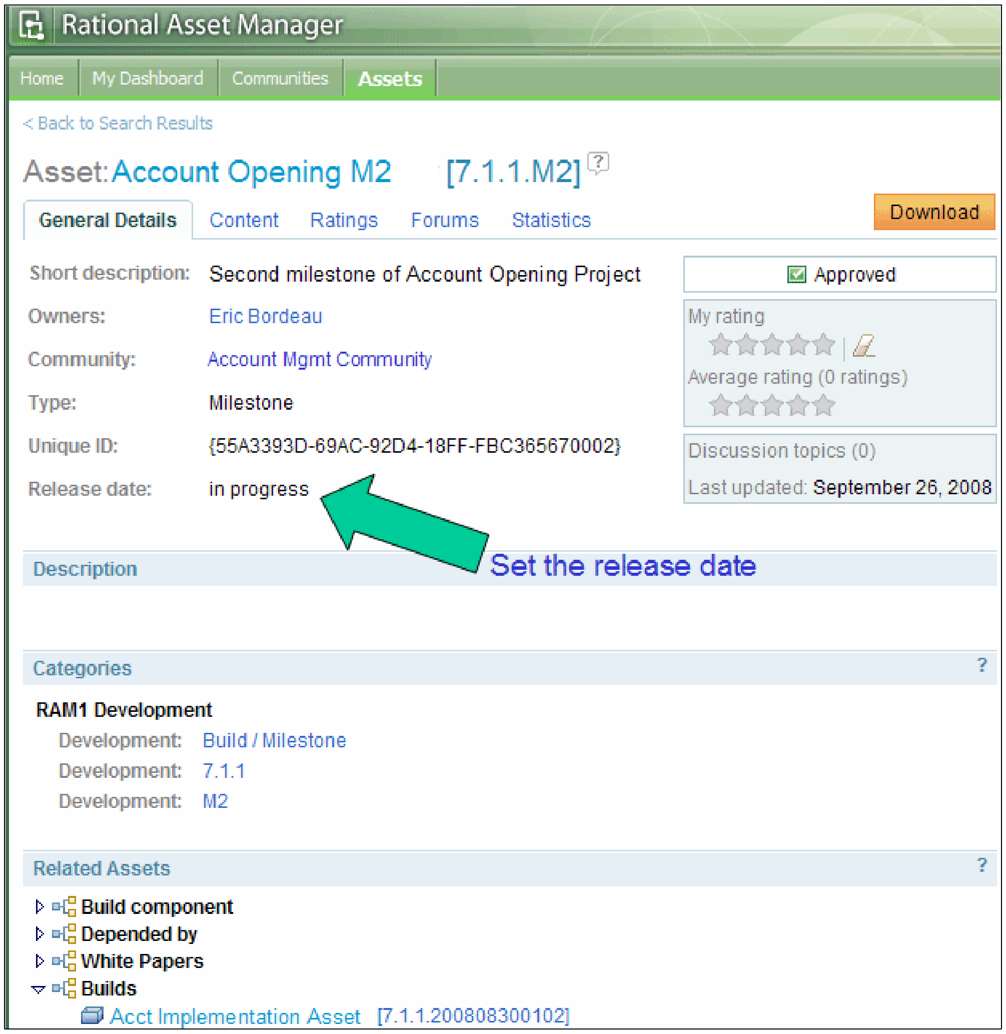
Rating
4 or higher [1]
3 or higher [2]
2 or higher [2]
1 or higher [2]

Previous | 1-7 of 7 | Next

Name	Version
Account Opening M2	7.1.1.M2
Account Opening M1	7.1.1.M1
Account Opening M	7.1.0.1.M
Account Opening M3	7.1.M3
Account Opening CCB	7.1.CCB
Account Opening M2	7.1.M2
Account Opening M1	7.1.M1

Figure 13-11   Searching for assets in Rational Asset Manager

7. Rebecca declares the milestone or iteration to be released by setting a release date on it as shown in Figure 13-12.



Asset: Account Opening M2 [7.1.1.M2]

Short description: Second milestone of Account Opening Project
Owners: Eric Bordeau
Community: Account Mgmt Community
Type: Milestone
Unique ID: {55A3393D-69AC-92D4-18FF-FBC365670002}
Release date: in progress

Approved
My rating
Average rating (0 ratings)
Discussion topics (0)
Last updated: September 26, 2008

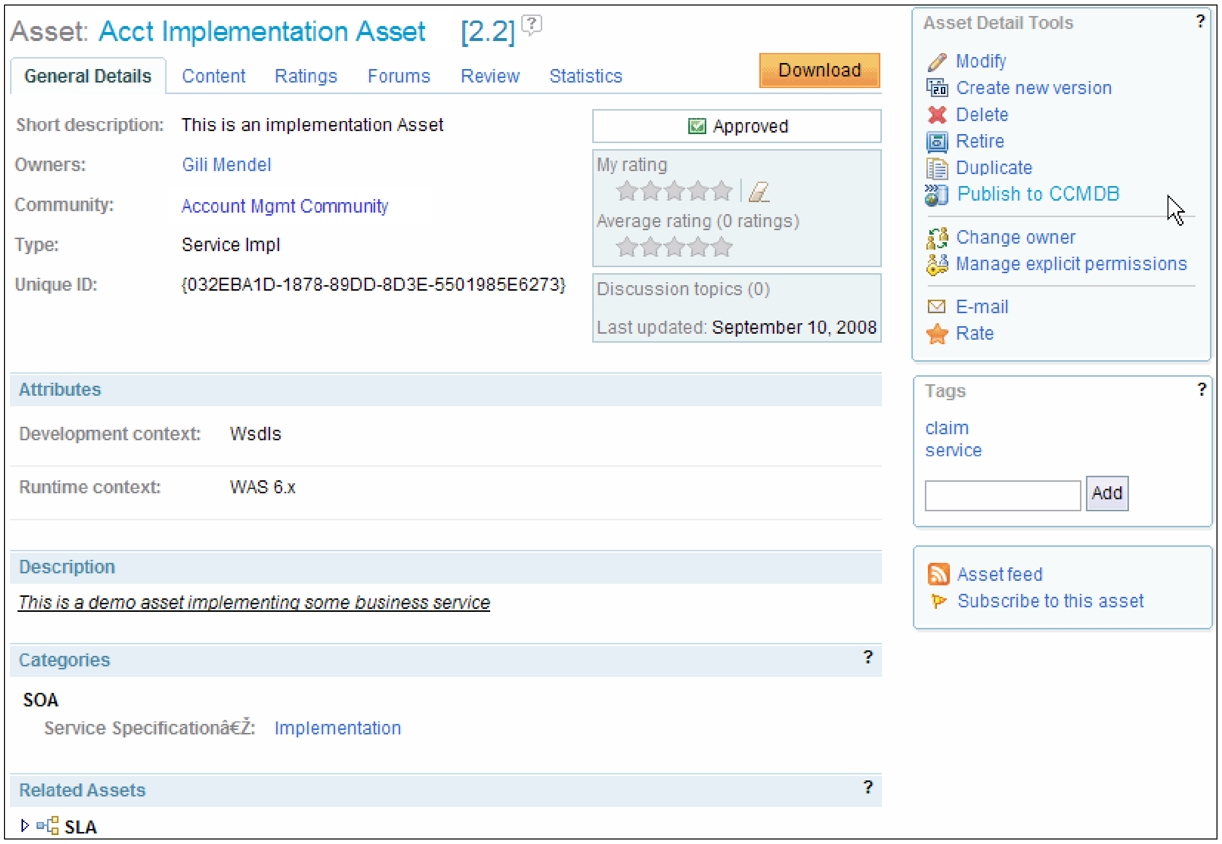
Description

Categories
RAM1 Development
Development: Build / Milestone
Development: 7.1.1
Development: M2

Related Assets
Build component
Depended by
White Papers
Builds
Acct Implementation Asset [7.1.1.200808300102]

Figure 13-12 Setting the release date in Rational Asset Manager

8. By using the Related Assets section (shown in Figure 13-12 on page 514), Rebecca traverses the asset relationship to the Acct Implementation Asset, which was produced from the Rational Build Forge project (Figure 13-13).
9. Rebecca clicks the **Publish to CCMDB** link (shown in Figure 13-13) to publish the asset for the milestone to the CCMDB, which is used by the operations team to manage and track changes to the operations environment. An important aspect of managing the operations environment includes maintaining links to all software that is deployed. By managing the assets in Rational Asset Manager and linking to the CCMDB, the operations team always has access to the software versions that were deployed into production.


Asset: Acct Implementation Asset [2.2]

General Details | Content | Ratings | Forums | Review | Statistics | Download

Short description:	This is an implementation Asset
Owners:	Gili Mendel
Community:	Account Mgmt Community
Type:	Service Impl
Unique ID:	{032EBA1D-1878-89DD-8D3E-5501985E6273}

Approved

My rating

Average rating (0 ratings)

Discussion topics (0)

Last updated: September 10, 2008

Attributes

Development context:	Wsdls
Runtime context:	WAS 6.x

Description

This is a demo asset implementing some business service

Categories ?

SOA

Service Specificationâ€Ž: Implementation

Related Assets ?

SLA

Asset Detail Tools ?

Modify
Create new version
Delete
Retire
Duplicate
Publish to CCMDB
Change owner
Manage explicit permissions
E-mail
Rate

Tags ?

claim
service

Add

Asset feed
Subscribe to this asset

Figure 13-13 Automatically populating Rational Asset Manager with Rational Build Forge

10. After Rational Asset Manager and CCMDB synchronize, the relationships can be graphically viewed as shown in Figure 13-14. The Acct Implementation Asset has a relationship to a CCMDB proxy asset called Acct IMPLEMENTATION ASSET-1012. Rebecca examines the asset details on that proxy asset by selecting **View Asset Details**.

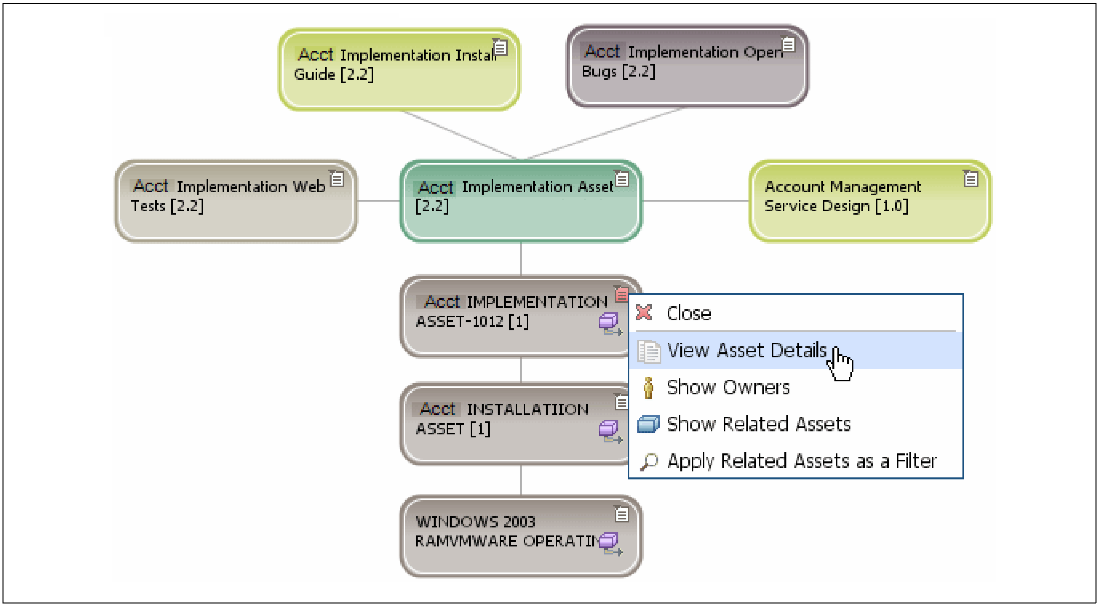
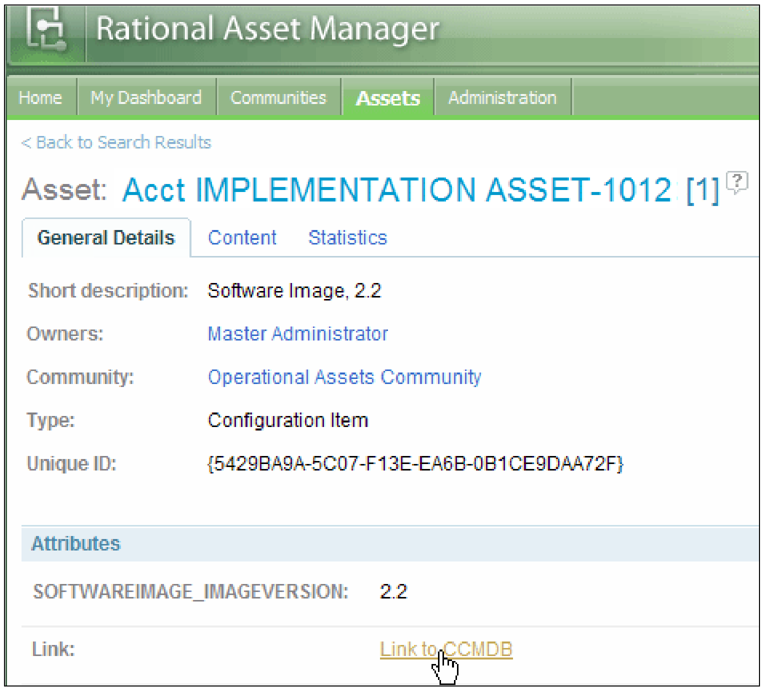

Acct Implementation Install Guide [2.2]
Acct Implementation Open Bugs [2.2]
Acct Implementation Web Tests [2.2]
Acct Implementation Asset [2.2]
Account Management Service Design [1.0]
Acct IMPLEMENTATION ASSET-1012 [1]
Acct INSTALLATIION ASSET [1]
WINDOWS 2003 RAMVMWARE OPERATI
Close
View Asset Details
Show Owners
Show Related Assets
Apply Related Assets as a Filter

Figure 13-14 Relationships synchronized between Rational Asset Manager and CCMDB

11. Rebecca examines the CCMDB proxy asset and clicks the **Link to CCMDB** (Figure 13-15).



Asset: Acct IMPLEMENTATION ASSET-1012 [1]

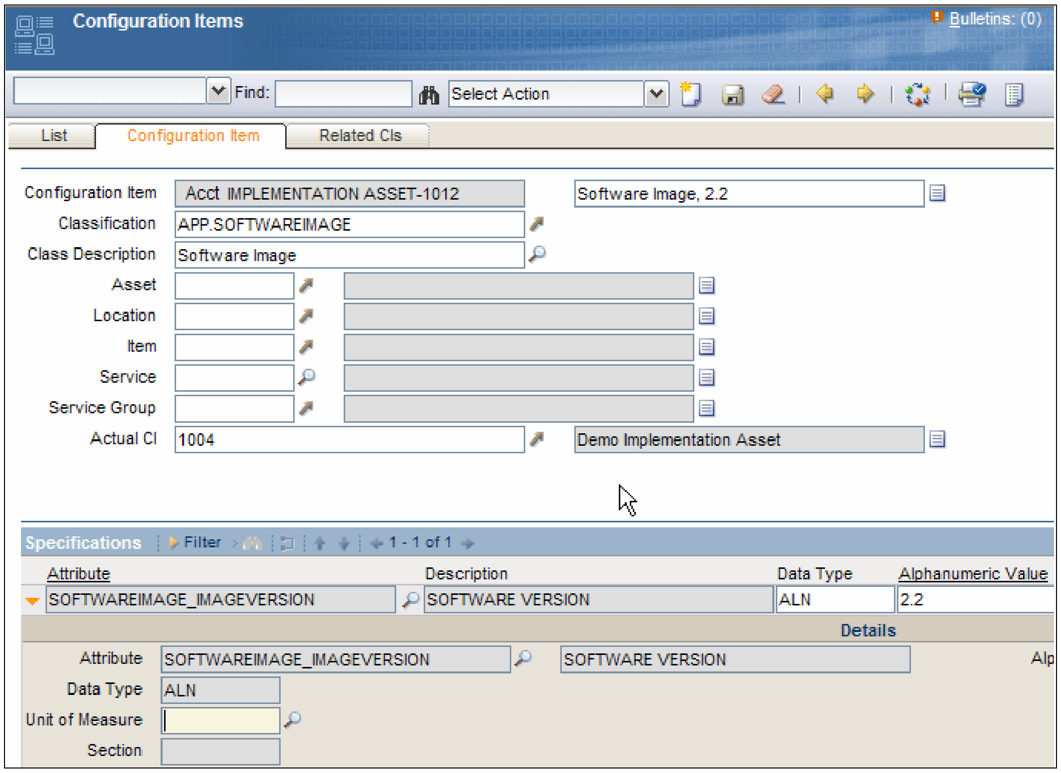
Short description:	Software Image, 2.2
Owners:	Master Administrator
Community:	Operational Assets Community
Type:	Configuration Item
Unique ID:	{5429BA9A-5C07-F13E-EA6B-0B1CE9DAA72F}

Attributes

SOFTWAREIMAGE_IMAGEVERSION:	2.2
Link:	Link to CCMDB

Figure 13-15 Traversing a link to the CCMDB

12. In the Web page opened by the CCMDB that shows the details of the asset that Rebecca just published (Figure 13-16), Rebecca examines the CCMDB configured item of the Acct Implementation Asset, which is now ready to be deployed.



Configuration Item	Acct IMPLEMENTATION ASSET-1012	Software Image, 2.2
Classification	APP.SOFTWAREIMAGE	
Class Description	Software Image	
Asset		
Location		
Item		
Service		
Service Group		
Actual CI	1004	Demo Implementation Asset

Attribute	Description	Data Type	Alphanumeric Value
SOFTWAREIMAGE_IMAGEVERSION	SOFTWARE VERSION	ALN	2.2

Attribute	SOFTWAREIMAGE_IMAGEVERSION	SOFTWARE VERSION
Data Type	ALN	
Unit of Measure		
Section		

Figure 13-16 Deploying the Acct Implementation Asset to CCMDB

Rebecca's work is done, and the final asset is now ready for deployment by the operations team.

13.3.4 Marco conducts a retrospective

Synopsis: Each team in the Account Opening project is conducting retrospective to continuously improve the development practices. The retrospective is owned by the team and is facilitated by the team lead.

Marco calls for a retrospective with his component team. The team identifies the timeline for the iteration, any major events during the iteration, and major achievements by the team. The team discusses what went well and how the team can do well more often. Marco makes the team discuss alternative behaviors to the tasks that did not go well and what the team can do about it. The team uses the dashboard to review health trends from the iteration. Marco structures his notes on team behavior, practices, and agreed improvements. He submits his notes to the team repository.

At Patricia's retrospective with the leadership team, Marco reviews the conclusions from his team. Patricia's team concludes on actions and decides to pilot new estimation practices for Marco's team in the next iteration.

The workflow in this scene captures how Marco completes the following tasks:

- Uses the dashboard to support a retrospective
- Submits a retrospective to the team repository

Retrospectives

Retrospectives are a practice that is performed by agile teams that want to continuously improve their way of working by looking back and learning by experience. This practice is captured as one of the principles of the agile manifesto, which states:[1] “At regular intervals, the team reflects on how to become more effective, and then tunes and adjusts its behavior accordingly.”

Every organization adopts a way to make retrospectives productive within their teams, but a few structural techniques support the collaboration.

- Define and scope the timeframe of the discussion to the latest iteration.
- Identify major achievements by the team.
- Learn what practices went well and which did not go well.
- Agree on the practices to adjust to do well more often and on those to avoid that did not do well.

The Rational ALM solution in retrospectives

In this section, we discuss how to use the Rational ALM solution to support and track team retrospectives.

Rational Team Concert in retrospectives

The health of a project, as supported by the metrics collected from the life-cycle collaboration assets and made visible through the principles around transparency, often give supporting indications to the questions: What went well? What did not?

As discussed in 7.5.2, “Reporting team health with Rational Team Concert” on page 292, the ALM solution provides rich data collection and reporting for health and trend analysis. The ready-to-use reports provide a good starting point for the team to discuss and analyze the team behaviors and practices during retrospectives. As practices are adjusted, the data collection and reporting should also be adjusted to provide continuous learning for future iteration retrospectives.

The team that uses Rational Team Concert might consider configuring a Retrospective Work Item type to manage and persist notes and decisions from the retrospectives. The Eclipse Way process comes with Retrospective as a preconfigured work item type. The steps to configure Rational Team Concert for Retrospectives in other process configurations are discussed in “Configuring work items of type Retrospective” on page 527.

To create a retrospective take the following actions:

1. Create a new work item of type Retrospective. Alternatively, make a new copy from a template work item.
2. Give the name of the iteration as the headline, and set the category (Filed Against) to the team that is conducting the retrospective.
3. Add notes in the agreed content structure in the Documentation or Discussion field as in the following examples:
   - Achievements
   - What worked well
   - What did not work well

[1] The agile manifesto: `http://agilemanifesto.org/principles.html`

   - Adjusted practices
   - Other areas
4. Save the work item changes.

As discussed in the following section, Patricia can manage the retrospectives across the teams by using Rational ClearQuest as an alternative approach. Such an approach implies that Marco gets a work item assigned to him through ClearQuest Connector synchronization.

Rational ClearQuest in retrospectives

The team that uses Rational ClearQuest and the ALM schema might consider extending and configuring the work configuration to include a Conduct Retrospective workflow as ALMTask and ALMActivity types on the Start Iteration ALM Request type. In "Configuring ALM for approvals" on page 521, we explain how to add a retrospective workflow.

Patricia takes the following actions to initiate an iteration retrospective for the project teams:

1. Patricia opens the **ALMRequest Construction 2** of type **Start Iteration**. She then browses the Tasks section for a **Conduct Retrospective** task and opens it.
2. She clicks the **Modify** button to make changes to the task.
3. In the Documentation field, she adds the agreed content structure as discussed in the previous section.
4. She clicks the **Create Activity** button one or more times. She then clicks the **Apply** button to save the changes. After saving the changes, the window is updated, and one or more ALMActivities are listed under the Activities section.
5. She opens the first activity, assigns it to Marco, and saves the changes. This activity instructs Marco to conduct the retrospective for the Construction 2 iteration. She proceeds with the other activities and assigns them to the other component and practice leads.

Patricia can now track the completion of the team retrospectives. After all team leads complete their activities, Patricia can proceed and schedule a retrospective for her leadership team. By using ClearQuest reporting, she can generate a retrospective summary report for the leadership team to discuss. After the iteration retrospective is completed, she changes the state of the Conduct Retrospective ALMTask to Complete.

13.4 Life-cycle collaboration

In this scenario, the actors mainly approve, package, and deliver assets that were already created in the previous acts. Some steps in this act also produce artifacts of more transient types, but are important to the life-cycle collaboration, such as queries, reports, and charts.

We look at the following life-cycle assets (Figure 13-17 on page 520) that are created and used by the characters in this act:

- ALM Type (of type Approval and Retrospective in Rational ClearQuest)
- ALM Activity (Rational ClearQuest)
- ALM Baseline (Rational ClearQuest)
- BT Build (Rational ClearQuest)
- Build (Rational Build Forge)
- Work Item (Rational Team Concert)

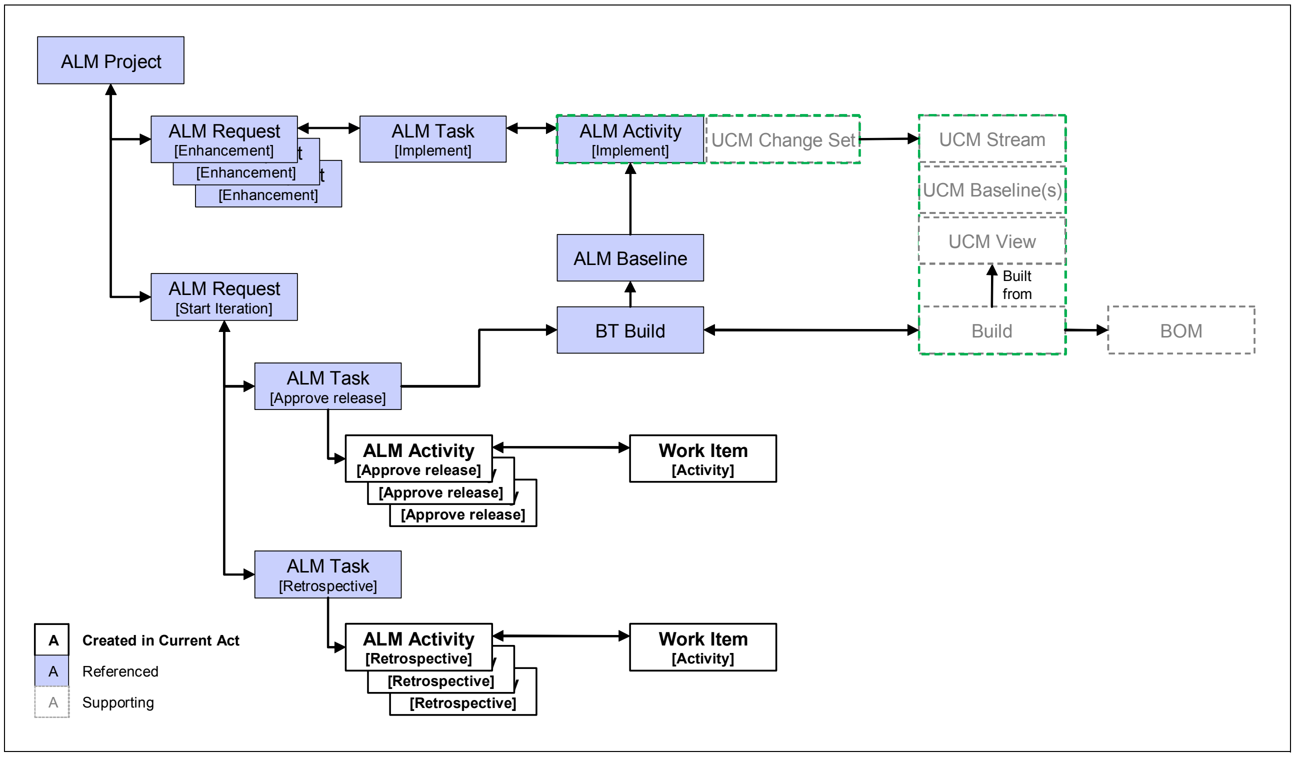


Figure 13-17 Lifecycle assets involved in this solution delivery scenario

13.5 Reference architecture and configuration

In this section, we explain how the products that are used in this act of the storyboard fit into the overall solution architecture and how they are configured.

13.5.1 Fitting into the enterprise ALM solution

The products that are used in this act illustrate one part of an enterprise ALM solution. Figure 13-18 on page 521 highlights the part with the enterprise ALM solution that is discussed in this chapter.

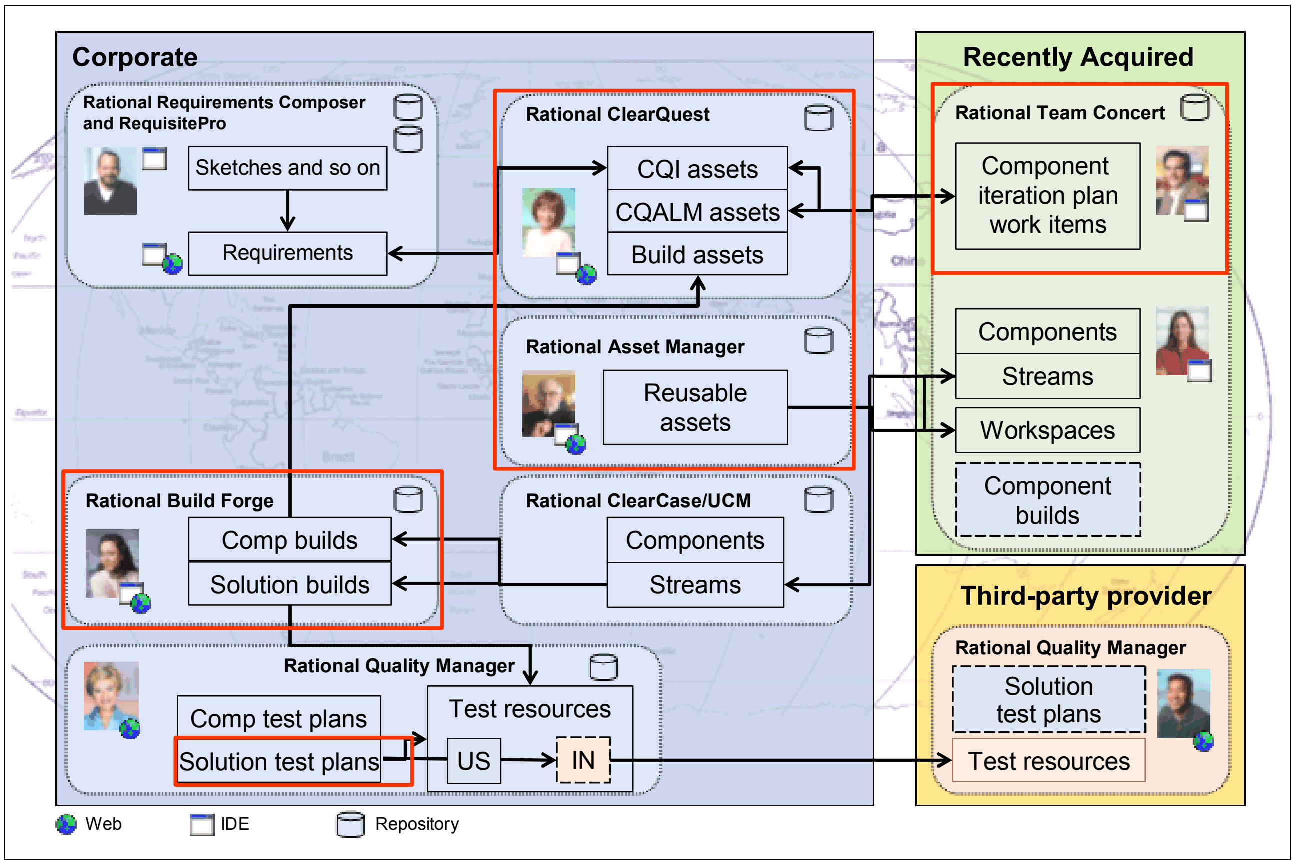


Figure 13-18   Rational ALM solution components contributing to the solution delivery

13.5.2 How the products are configured for this scenario

In this section, we discuss the configurations in Rational ClearQuest to support solution delivery.

Configuring Rational ClearQuest for solution delivery

The ClearQuest ALM schema can be extended with new workflows. By using a work configuration, project managers can establish a customized work management process on a project-by-project basis.

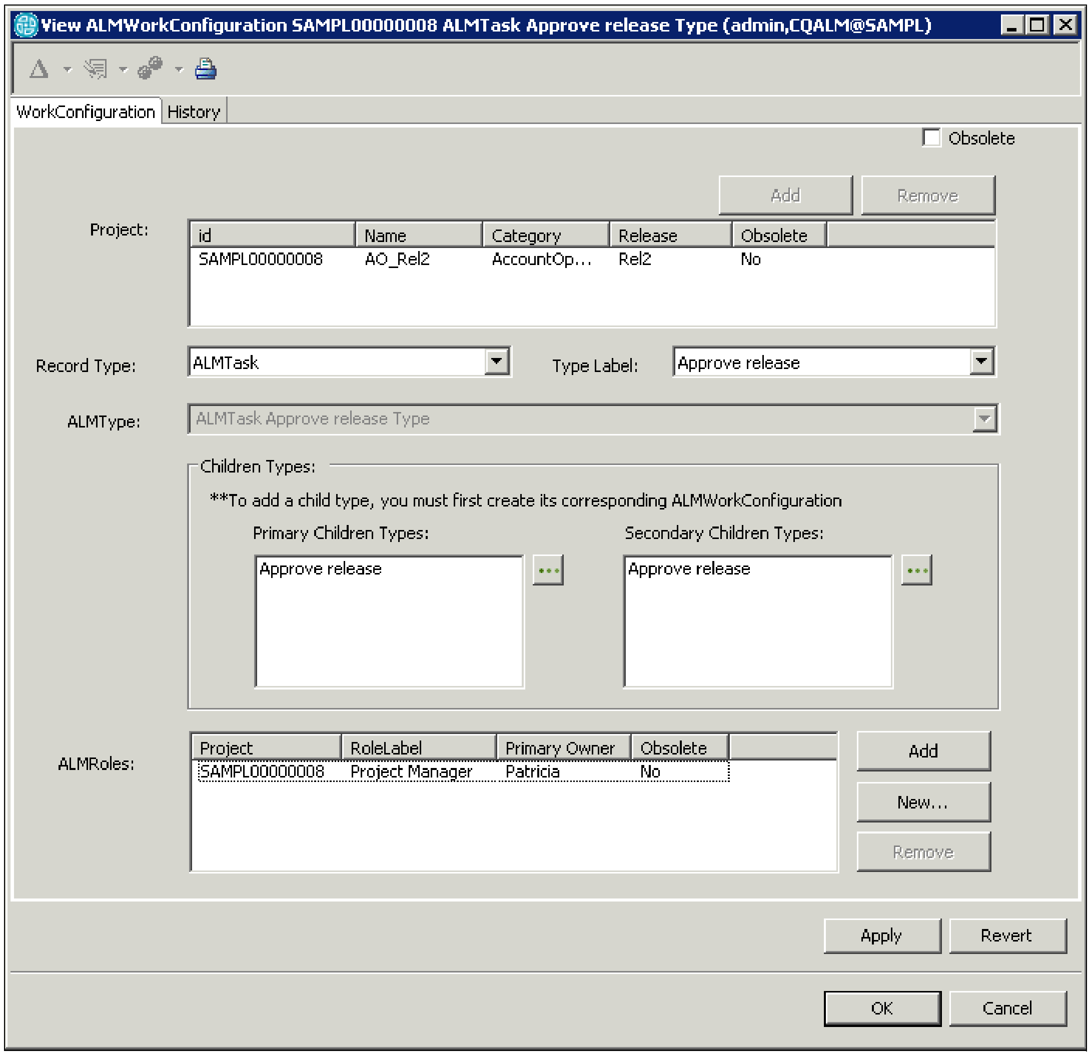
In this chapter, we use an extended workflow to manage approvals and retrospectives by using ALMTasks and ALM Activities. We also use the ClearQuest Connector and interoperability to synchronize these workflows cross the ALM solution platforms.

Configuring ALM for approvals

To define a work configuration for approvals, in Rational ClearQuest:

1. Create a new ALMResolutionCodeLabel record by selecting **File** → **New** → **ALMResolutionCodeLabel**:
   a. Type the name `Approved`.
   b. Click **OK** to save the record.

2. Create a new ALMResolutionCode record by selecting **File** → **New** → **ALMResolutionCode**:
   a. For ALMRecordType, select **ALMActivity**.
   b. For ResolutionCodeLabel, select **Approved**.
   c. Click **OK** to save the record.
3. Create another ALMResolutionCode record, but for ALMRecordType, select **ALMType**. Click **OK** to save the record.
4. Create a new ALMTypeLabel record by selecting **File** → **New** → **ALMTypeRecord**:
   a. Enter the name `Approve release` and type an optional description.
   b. Click **OK** to save the record.
5. Create a new ALMType record by selecting **File** → **New** → **ALMType**:
   a. In the ALMRecordType field, select **ALMType**.
   b. For TypeIndicator, select **Type**.
   c. For Type Label, select **Approve release**.
   d. Enter an optional description and click **OK** to save the record.
6. Create another ALMType, but for the ALMRecordType field, select **ALMActivity**.
7. Create a new ALMWorkConfiguration record by selecting **File** → **New** → **ALMWorkConfiguration**:
   a. Select the project to deploy this workflow to, for example **AO_Rel2**.
   b. For the Record Type field, select **ALMActivity**.
   c. For the Type Label, select **Approve release**.
   d. For the ALMRoles, select **Any Role** in the same project as previously chosen.
   e. Click **OK** to save the record.
8. Create another ALMWorkConfiguration (Figure 13-19 on page 523):
   a. For the Record Type field, select **ALMTask**.
   b. For the Type Label, select **Approve release**.
   c. For the ALMRoles, select **Project Manager**.
   d. In the Primary Children Types section:
      i. Click the **...** button.
      ii. Select the **Approve release** line and double-click to add the selection.
      iii. Click **OK** to close the window.
   e. Repeat the previous step and add **Approve release** to the Secondary Children Type list. Scroll the query result list and open the record with ALMType **ALMRequest Start Iteration Type**.
   f. Click **OK** to save the record.


View ALMWorkConfiguration SAMPL00000008 ALMTask Approve release Type (admin,CQALM@SAMPL)

WorkConfiguration | History

Obsolete

Project:

id	Name	Category	Release	Obsolete
SAMPL00000008	AO_Rel2	AccountOp...	Rel2	No

Record Type: ALMTask

Type Label: Approve release

ALMType: ALMTask Approve release Type

Children Types:

**To add a child type, you must first create its corresponding ALMWorkConfiguration

Primary Children Types: Approve release

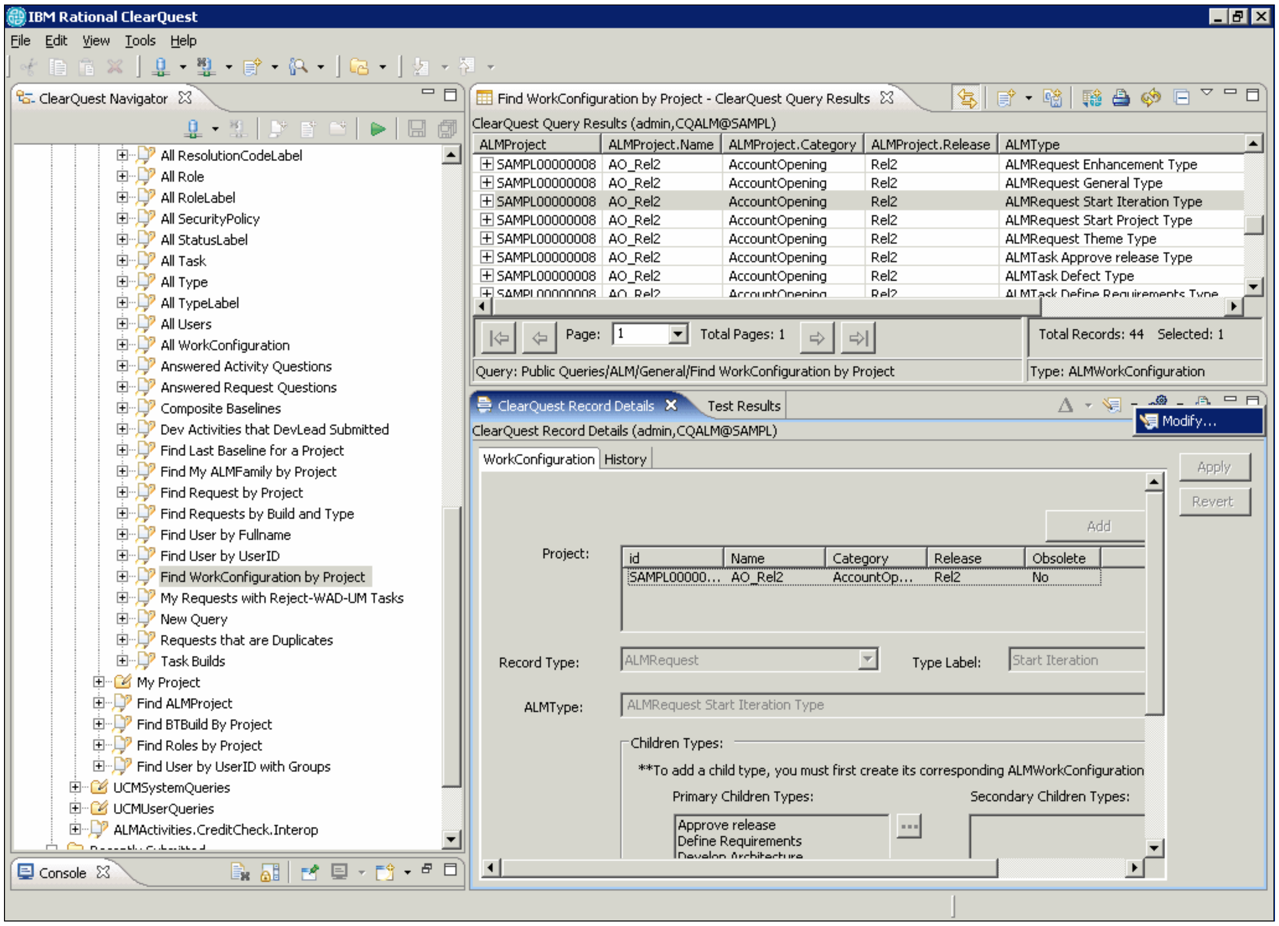
Secondary Children Types: Approve release

ALMRoles:

Project	RoleLabel	Primary Owner	Obsolete
SAMPL00000008	Project Manager	Patricia	No

Figure 13-19 Adding a new Approve release work configuration

9. Run the query **Find Work Configuration by Project** (Figure 13-20 on page 524):
   a. In the Dynamic Filters window, select the project to which you are adding the approval workflow and click **OK** to run the query.
   b. Scroll the search results and locate the **ALMRequest Start Iteration Type** record.
   c. Click **Modify**.
   d. In the Primary Children Types section, add the **Approve release** task. Click **OK** to save the Work Configuration record.


IBM Rational ClearQuest

ClearQuest Navigator

- All ResolutionCodeLabel
- All Role
- All RoleLabel
- All SecurityPolicy
- All StatusLabel
- All Task
- All Type
- All TypeLabel
- All Users
- All WorkConfiguration
- Answered Activity Questions
- Answered Request Questions
- Composite Baselines
- Dev Activities that DevLead Submitted
- Find Last Baseline for a Project
- Find My ALMFamily by Project
- Find Request by Project
- Find Requests by Build and Type
- Find User by Fullname
- Find User by UserID
- Find WorkConfiguration by Project
- My Requests with Reject-WAD-UM Tasks
- New Query
- Requests that are Duplicates
- Task Builds
- My Project
- Find ALMProject
- Find BTBuild By Project
- Find Roles by Project
- Find User by UserID with Groups
- UCMSystemQueries
- UCMUserQueries
- ALMActivities.CreditCheck.Interop

Find WorkConfiguration by Project - ClearQuest Query Results

ClearQuest Query Results (admin,CQALM@SAMPL)

ALMProject	ALMProject.Name	ALMProject.Category	ALMProject.Release	ALMType
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMRequest Enhancement Type
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMRequest General Type
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMRequest Start Iteration Type
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMRequest Start Project Type
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMRequest Theme Type
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMTask Approve release Type
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMTask Defect Type
SAMPL00000008	AO_Rel2	AccountOpening	Rel2	ALMTask Define Requirements Type

Page: 1 Total Pages: 1 Total Records: 44 Selected: 1

Query: Public Queries/ALM/General/Find WorkConfiguration by Project Type: ALMWorkConfiguration

ClearQuest Record Details Test Results

ClearQuest Record Details (admin,CQALM@SAMPL)

WorkConfiguration History

Project:

id	Name	Category	Release	Obsolete
SAMPL00000...	AO_Rel2	AccountOp...	Rel2	No

Record Type: ALMRequest Type Label: Start Iteration

ALMType: ALMRequest Start Iteration Type

Children Types:

**To add a child type, you must first create its corresponding ALMWorkConfiguration

Primary Children Types: Secondary Children Types:

Approve release
Define Requirements
Develop Architecture

Figure 13-20 Adding the new Approve release work configuration to the Start Iteration workflow

The new workflow is now ready to be used. When starting a new iteration, by creating a Start Iteration, the new approval task is added and pending for the completion of the iteration. We discuss the approval workflow in "Patricia approves the release" on page 508.

Configuring ALM for retrospectives

A work configuration that implements a workflow for retrospectives can be configured in Rational ClearQuest using the ALM schema. We provide an example of detailed step descriptions for work configuration creation in "Configuring ALM for approvals" on page 521.

To configure a retrospective work configuration in ClearQuest:

1. Create a new ALMTypeLabel record named `Retrospective`.
2. Create a new ALMType record with the following characteristics:
   - Type is ALMTask.
   - TypeIndicator is Type.
   - TypeLabel is Retrospective.
3. Create a new ALMType record with the following characteristics:
   - Type is ALMActivity.
   - TypeIndicator is Type.
   - TypeLabel is Retrospective.

4. Create a new ALMWorkConfiguration record with the following characteristics:
   - Type is ALMActivity.
   - TypeLabel is Retrospective.
   - ALMRoles is AllRole.
5. Create a new ALMWorkConfiguration record with the following characteristics:
   - Type is ALMTask.
   - TypeLabel is Retrospective.
   - ALMRoles is AllRole.

   Add Retrospective and Primary and Secondary Children Types.
6. Run the query Find Work Configuration by Project. Locate the ALMRequest Start Iteration Type record. Select the **Modify** command.
7. Run the query Find Work Configuration by Project:
   a. In the Dynamic Filters window, select the project to which you are adding the approval workflow and click **OK** to run the query.
   b. Scroll the search result and locate the ALMRequest Start Iteration Type record.
   c. Edit the Primary Children Types section, and add the Retrospective task.
   d. Click **OK** to save the Work Configuration record.

For the retrospective workflow, no resolution code is required. To resolve a retrospective, simply use the code Completed.

The ClearQuest Connector integrates the Retrospective workflow across Rational ClearQuest and Rational Team Concert. New retrospectives are assigned to teams by using Rational Team Concert and Rational Quality Manager. Notes and conclusions from completed retrospectives synchronize to Rational ClearQuest for consolidation by Patricia and her leadership team as discussed in “Rational ClearQuest in retrospectives” on page 519.

Configuring Rational Team Concert for solution delivery

In this section, we discuss the configurations in Rational Team Concert to support solution delivery.

Configuring the process for approvals

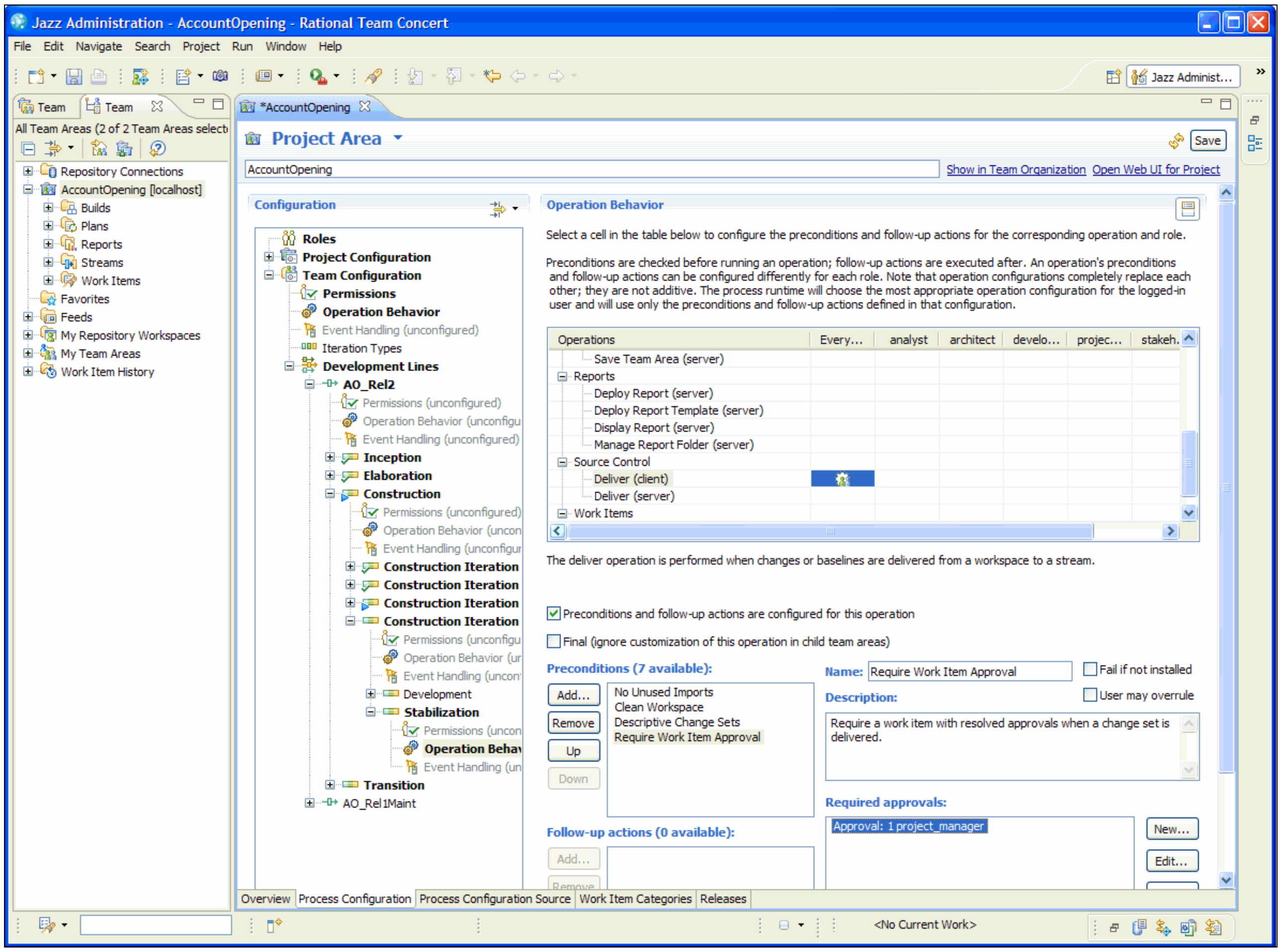
Teams that use Rational Team Concert 1.0 can use the process specification for the project to enforce the end-game policies to the team. The process specification rules that are enacted for the end game can be configured to require reviews and approvals. As the team enters the end game, it turns on process rules that require approvals for fixes to be delivered. Because every code delivery must be associated with a work item, it seamlessly binds the work item, the attached approvals, and the delivery together.

For each iteration in the end game, the level of required approvals and reviews can be increased as required to tighten the change process. In an early iteration, the end game can require a review, while later phases can involve two required code reviews, a team lead approval, and two project lead approvals. The review and approval processes make the team understood the seriousness of evaluating the risk and reward of any code change.

To configure the process specification for approvals:

1. In Rational Team Concert, open the Project Area, for example, AccountOpening, and click the **Process Configuration** tab.
2. Expand the **Configuration** section and locate the iteration where you want to add process enforcement rules, for example **Team Configuration** → **Development Lines** → **Main Development** → **Construction** → **Construction Iteration C2** → **Stabilization** → **Operation Behavior**. See Figure 13-21.

> **Note:** The path to the Operation Behavior element depends on the structure of the iteration that is used in your project.



AccountOpening

Select a cell in the table below to configure the preconditions and follow-up actions for the corresponding operation and role.

Preconditions are checked before running an operation; follow-up actions are executed after. An operation's preconditions and follow-up actions can be configured differently for each role. Note that operation configurations completely replace each other; they are not additive. The process runtime will choose the most appropriate operation configuration for the logged-in user and will use only the preconditions and follow-up actions defined in that configuration.

Operations	Every...	analyst	architect	develo...	projec...	stakeh.
Save Team Area (server)						
Reports						
Deploy Report (server)						
Deploy Report Template (server)						
Display Report (server)						
Manage Report Folder (server)						
Source Control						
Deliver (client)	✓					
Deliver (server)						
Work Items						

The deliver operation is performed when changes or baselines are delivered from a workspace to a stream.

[✓] Preconditions and follow-up actions are configured for this operation

[ ] Final (ignore customization of this operation in child team areas)

Preconditions (7 available):
Add... Remove Up Down

No Unused Imports
Clean Workspace
Descriptive Change Sets
Require Work Item Approval

Name: Require Work Item Approval [ ] Fail if not installed

Description: [ ] User may overrule

Require a work item with resolved approvals when a change set is delivered.

Required approvals:

Approval: 1 project_manager

New... Edit...

Follow-up actions (0 available):
Add... Remove

Figure 13-21 Adding Operational Behavior to the Stabilization phase to enforce end-game approvals

3. In the Operation Behavior section, scroll the list of Operations and locate the section **Source Control** → **Deliver (client)**. Select the cell in the **Everyone** column.
4. In the Preconditions, click **Add**. In the Add Preconditions window (shown at left in Figure 13-22 on page 527), select **Require work item approval** from the list and click **OK**.
5. Make sure that the new **Require work item approval** item is selected in the Preconditions section. Then click **Add** in the Required Approvals section.

6. In the New Required Approval window (shown at right in Figure 13-21), for Type, select **Approval** and select and increment the approvals that are required per role for a work item delivery, for example a single project manager.
7. Optional: Add another required approval. For Type, select **Review** that is to be done by one or more developers or architects.

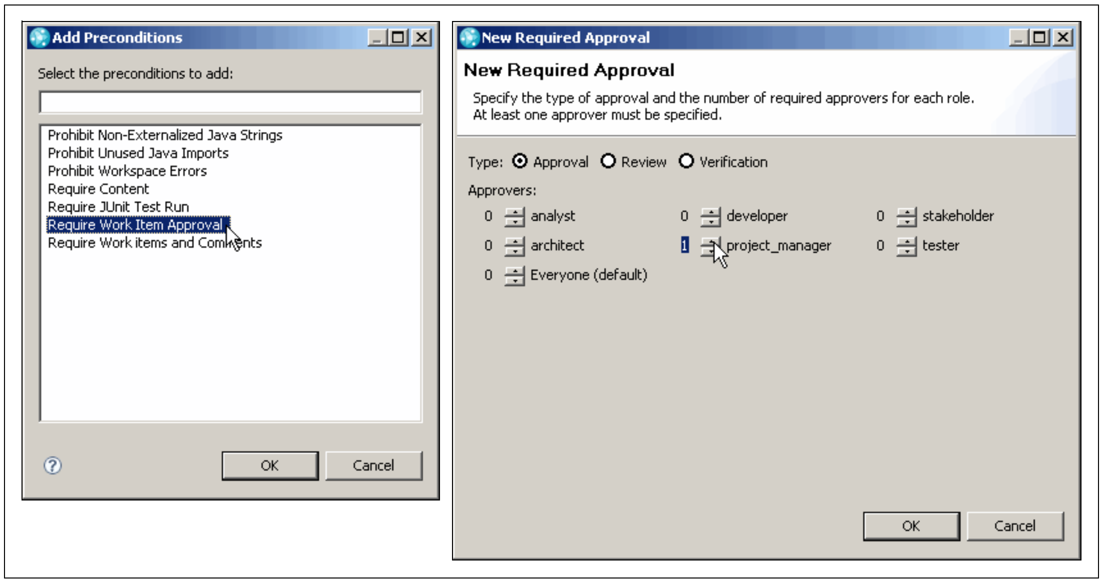

Add Preconditions
Select the preconditions to add:
Prohibit Non-Externalized Java Strings
Prohibit Unused Java Imports
Prohibit Workspace Errors
Require Content
Require JUnit Test Run
Require Work Item Approval
Require Work items and Comments
OK
Cancel
New Required Approval
New Required Approval
Specify the type of approval and the number of required approvers for each role.
At least one approver must be specified.
Type: Approval Review Verification
Approvers:
0 analyst
0 developer
0 stakeholder
0 architect
1 project_manager
0 tester
0 Everyone (default)
OK
Cancel

Figure 13-22 Adding work item approvals

8. Click **Save** in the Project Area to save the process changes and deploy the new operational behavior to the team.

Note: The new operational behavior is only active during the selected iteration phase. To deploy the approval process to other iterations and phases, repeat the steps in this task to each required iteration.

Configuring work items of type Retrospective

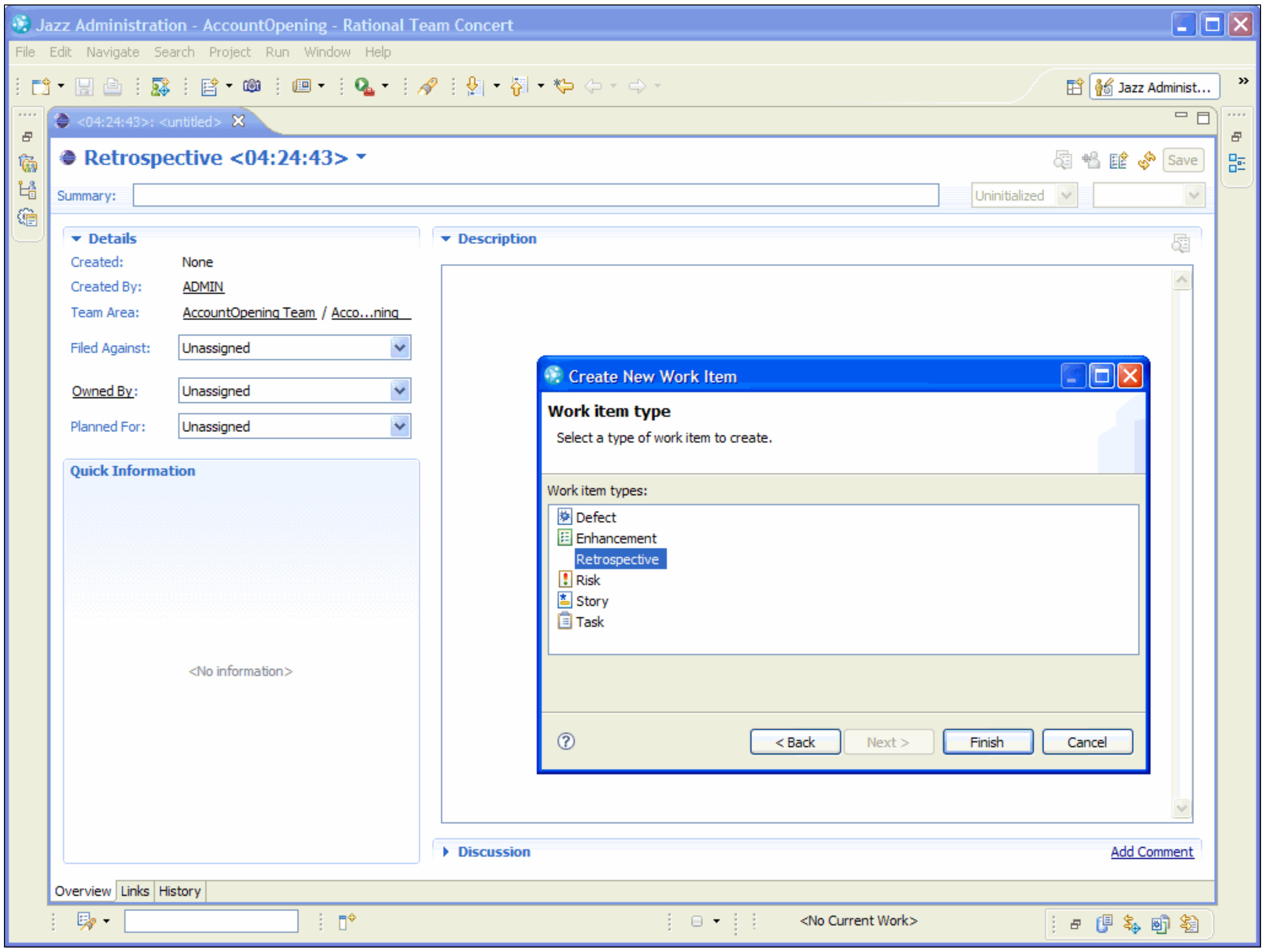
Teams that use Rational Team Concert 1.0 can benefit from the ready-to-use process configuration for retrospectives. This configuration is available as part of the Eclipse Way process template. If your team is using the Eclipse Way process, no further configuration is required. If you are planning to use another process, such as the Open Unified Process (OpenUP), which is used by the story in this book, you must complete the steps as explained in this section to refactor the process into your project area or your customized process template.

To locate the process specifications to refactor into your process:

1. In Rational Team Concert, open the **Team Artifacts** view. Expand **Repository Connections**, right-click your repository connection, for example, ADMIN@localhost, and select **Administer** → **Process Templates**.
2. In the Process Templates view, open the Eclipse Way Process and select the **Process Configuration Source** tab.
3. Select **Edit** → **Find/Replace** or press Ctrl+F.
4. In the Find field, type `retrospective`. Click **Find** to go to the first and subsequent source occurrences.

5. In the Team Artifact view, open your target project area and click the **Process Configuration Source** tab.
6. Depending of the level of retrospective process functionality that you need, you can complete one or more of the following steps. Copy the complete XML sections from the process template into your process definition.
   - In the **process-specification → project-configuration → data → configuration-data** section for `com.ibm.team.workitem.configuration.workItemTypes`, copy the source definition of the Retrospective type section. Paste the XLM section into the corresponding location in your target process definition.
   - In the **process-specification → project-configuration → data → configuration-data** section for `com.ibm.team.workitem.configuration.workflow`, copy the source definition of the `com.ibm.team.workitem.retrospectiveWorkflow` workflowDefinition section. Paste the XLM section into the corresponding location in your target process definition.
   - In the **process-specification → project-configuration → data → configuration-data** section for `com.ibm.team.workitem.configuration.workflowBinding`, copy the source definition of the `com.ibm.team.workitem.workItemType.retrospective` query section. Paste the XLM section into the corresponding location in your target process definition.
   - In the **process-specification → project-configuration → data → configuration-data** section for `com.ibm.team.workitem.editor.configuration.presentations`, copy the source definition of the `com.ibm.team.workitem.editor.retrospective` editor section. Paste the XLM section into the corresponding location in your target process definition.
   - In the **process-specification → project-configuration → data → configuration-data** section for `com.ibm.team.workitem.editor.configuration.presentations`, copy the `com.ibm.team.workitem.tab.retroOverview` tab section and the `com.ibm.team.workitem.section.RetroDetails` section.
   - In the **process-specification → project-configuration → data → configuration-data** section for `com.ibm.team.workitem.editor.configuration.workitemTypeEditorIdBinding`, copy the source definition of the `com.ibm.team.workitem.editor.retrospective` workitemTypeEditorIdBinding section. Paste the XLM section into the corresponding location in your target process definition.
   - Optionally, in the **process-specification → project-configuration → data → configuration-data** section for `com.ibm.team.workitem.configuration.queries`, copy the source definition of the “All Retrospectives” workItemCategoryBinding section. Paste the XLM section into the corresponding location in your target process definition.
7. Test the process configuration extension by creating a new work item.

The type Retrospective is now displayed in the list and in the editor as shown in Figure 13-23.


Jazz Administration - AccountOpening - Rational Team Concert
File Edit Navigate Search Project Run Window Help
Jazz Administ...
<04:24:43>: <untitled>
Save
Overview Links History
<No Current Work>

Retrospective <04:24:43>

Summary:

Uninitialized

Details

Created: None

Created By: ADMIN

Team Area: AccountOpening Team / Acco...ning

Filed Against: Unassigned

Owned By: Unassigned

Planned For: Unassigned

Quick Information

<No information>

Description

Discussion

Add Comment

Create New Work Item

Work item type

Select a type of work item to create.

Work item types:

- Defect
- Enhancement
- Retrospective
- Risk
- Story
- Task

< Back Next > Finish Cancel

Figure 13-23 The Retrospective editor in Rational Team Concert

Part G

Measuring team success in Application Lifecycle Management

In this part, we discuss techniques for measuring success in software development projects. If the purpose of Collaborative Application Lifecycle Management (CALM) is to streamline a project team's ability to deliver a release of software, then strategies for measuring success are imperative to continual improvement.

This part includes Chapter 14, "Delivering and measuring success in Application Lifecycle Management" on page 533. In this chapter, we discuss the use of process definition and enactment, metrics, measures, dashboards, and retrospectives as techniques for measuring success.

14 Delivering and measuring success in Application Lifecycle Management

We summarize and close this book with a discussion about measuring success in software development projects.

This chapter includes the following sections:

14.1 Introduction to measuring success

The fundamental goal of Application Lifecycle Management (ALM) is to increase the delivery of business value to customers. The primary endpoint for measuring success is to answer the fundamental, universal question for everyone who is involved with the project and organization: Is this all really working?

Software delivery processes play an important role in the long-term, repeatable success of an organization. In ALM, the software process that is instituted encompasses all phases and team member roles in a project. However, processes that are never truly instituted or embraced provide no value to the organization. In fact, they can confuse project teams and distort expected results. Processes in software delivery must be capable of being elaborated or customized to fit your project, not vice versa. Organizations have varying levels of process formality in ALM. It is important for the software delivery environment to be capable of flexibly supporting their varying needs.

Agile software development and Collaborate Application Lifecycle Management (CALM) share a common thread. In order to be agile, they must be highly collaborative, and in order to capitalize on the benefits of collaboration, they must be flexible and agile. In this world, responding to change is more important than following a rigid plan. For more information, see "Manifesto for Agile Software Development," on the Web at the following address:

`http://agilemanifesto.org`

Everyone shares in the responsibility of responding to change in CALM. In order to be successful, the entire team must have individual and team level visibility, along with the appropriate permissions to respond to changes on the project.

In CALM, reports are not used simply to communicate status, but they are active tools the team uses for achieving project success. Teams leverage measurements and reporting as temporary, in-context diagnostic tools to focus their teams on resolving problem areas of the project and guide local process improvements. Metrics and measurements provide project teams with a proverbial compass that enables them to navigate and adjust to changing project needs.

Instituting successful methods and processes, measurements and metrics, and project navigation approaches into your organization does not have to be a project itself. Your software delivery environment should enable you to institute your process while providing project teams with the flexibility they need to remain agile and adjust to project changes. Additionally, your software delivery platform must automate the data collection and report generation for you so that you can stay focused on what the data is telling you, and not how to collect and transform it.

Dashboards and reports must provide measurements that are real time and focused specifically on the areas of the project in which you are interested.

14.2 Process understanding and implementation: Improving project success with predictability and repeatability

Process refers to the collection of practices, rules, guidelines, and conventions that are used to organize a team's work. A team's process is the sum total of the that ways the team has decided (or evolved) to do things. An organizations software delivery process is usually defined in two ways: a process description and process implementation. The process description is the documented process, where the process specification implements the documented process in the tools and software delivery environment that the organization uses.

Small teams tend to have little documented process and a lightweight implementation of the process. These types of small teams are usually located in a single location and organization and change infrequently. They know how to work together successfully from experience and have a common set of previous project experiences to draw upon. As the teams grow and change more frequently, the process and implementation are defined more formally in order to enable new and distributed team members. For example, the team might start to document common team processes such as the workflow for different change requests or the branching and promotion rules for their software configuration management. Larger or more complex organizations and projects might find that they need to document and implement their process more formally in order to be successful in software delivery.

As processes become more formalized in an organization, project teams might also find that they become restrictive and nonconducive to projects. Therefore, it is important that processes provide the project manager and leads with control to customize the process definition and implementation to fit the needs of their project. This enables the organization to continue to enforce necessary process mandates, such as regulatory compliance requirements, and still give project owners the ability to adjust to changing project requirements.

14.2.1 Process specifications: Implementing your process in software delivery

When you think of software delivery processes in the literal sense, many people envision documentation rather than practice. That is because process is frequently documented in detail first and then provided as a reference for the project team to institute. Unfortunately, many organizations are unable to successfully transition the documented process to an adopted practice that the software delivery organization uses. It is common to see companies that say they are using a particular best practice or process, but find that what is being done in practice is different or highly variant from the defined process or best practice.

Process enactment is necessary for process adoption that increases the agility and performance of your software delivery teams. This means that process is not simply documentation but a specification that is used by the software delivery environment to institute the process. As such, process adoption becomes a more natural and normal course of software delivery. It eliminates the overhead required to govern and police necessary process changes. Process enactment also improves the ability for project managers to plan and estimate the project because the enacted process provides a more predictable basis for measurement.

Processes can be defined for an organization, but they must be *specified* for a project. Process definitions take the form of process templates, and those standards are customized and specified for each project.

Process enactment with Rational Team Concert

In IBM Rational Team Concert, every project has an associated team process. The team process governs all activities, artifacts, artifact relationships, and operations that are pursued inside the associated project. A process is defined as a process specification and an associated iteration structure. When starting a new project, a process template is used to select the process that the project will use. The process and its behavior can then be further customized (Figure 14-1).

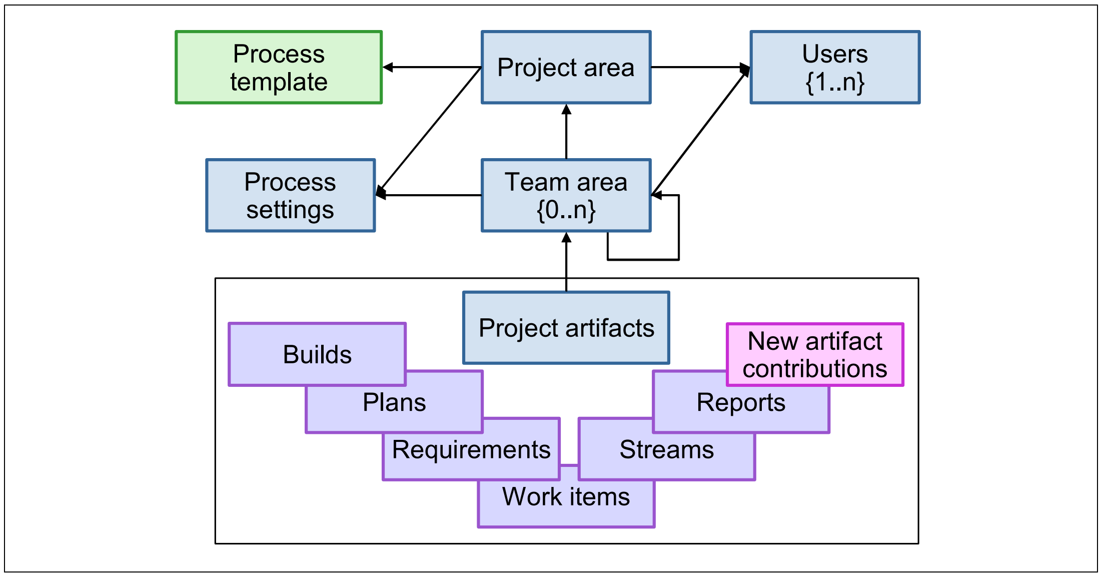
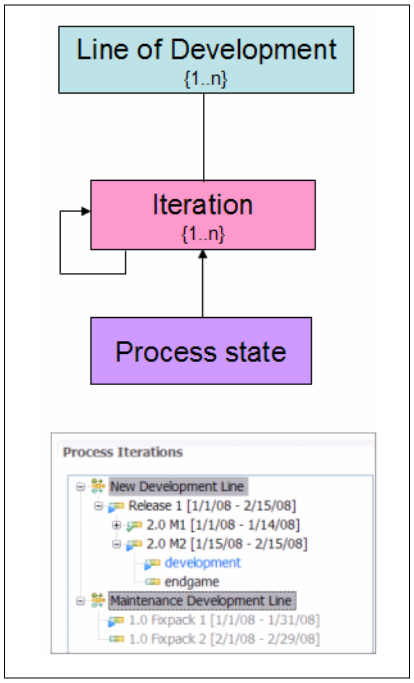


Figure 14-1 Defining a process specification by customizing a process template for the project

The iteration structure defines the existing development lines and the break down of the development lines into iterations. It also defines which iteration is the current iteration for each development line, as illustrated in Figure 14-2.



```
New Development Line
    Release 1 [1/1/08 - 2/15/08]
        2.0 M1 [1/1/08 - 1/14/08]
        2.0 M2 [1/15/08 - 2/15/08]
            development
            endgame
Maintenance Development Line
    1.0 Fixpack 1 [1/1/08 - 1/31/08]
    1.0 Fixpack 2 [2/1/08 - 2/29/08]
```

Figure 14-2 The process adapting to the changing needs of the project as work progresses

The majority of modern software development processes and methods today follow an iterative software development approach. Rational Team Concert provides a number of ready-to-use process templates that implement these industry processes. For example, Rational Team Concert includes Open Unified Process (OpenUP), Eclipse Way, and an example process that are ready to use. Rational Team Concert also makes it easy to customize or redesign these processes to fit your organization's way of doing things by using the process templates as a starting point. The process template library makes it easy to create and share new or customized processes. For example, the Scrum process implementation is available in IBM developerWorks at the following address:

http://www.ibm.com/developerworks/rational/library/08/0701_ellingsworth/index.html?ca=drs-

Your process controls can be customized for each project iteration. Different lines of development might have different active controls based on the process rules that are defined for that iteration as illustrated in Figure 14-3.

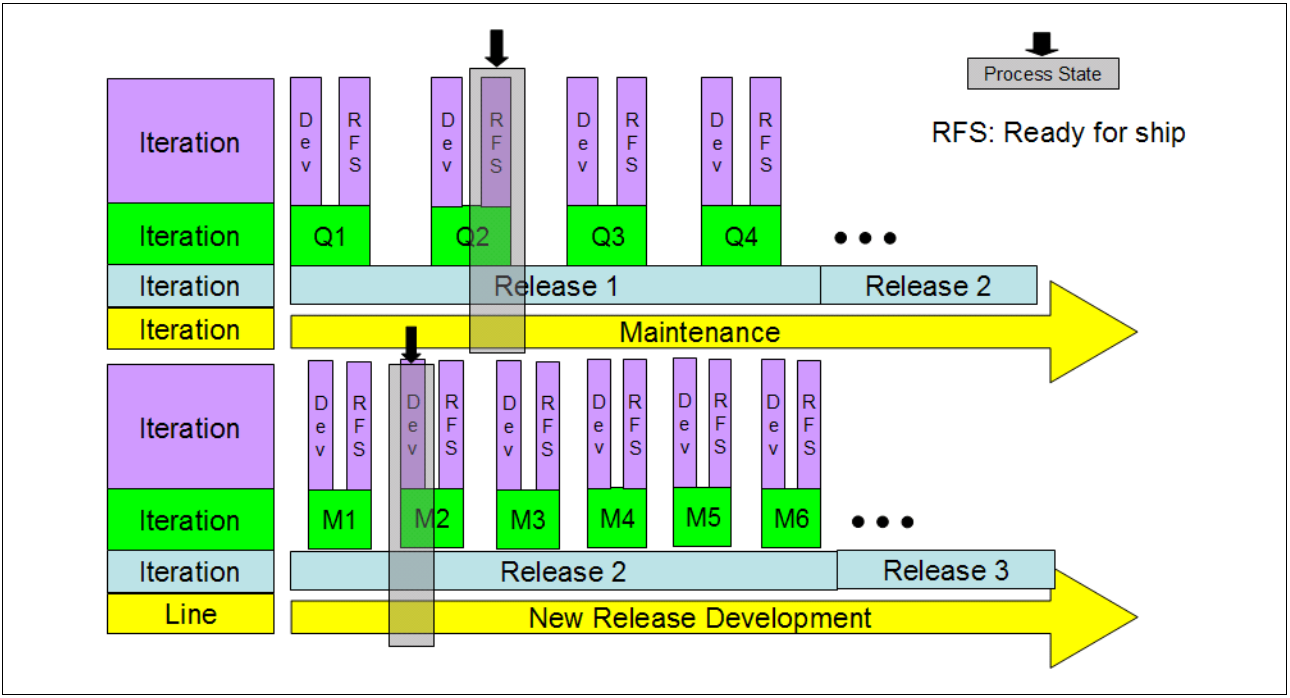


Figure 14-3 The process for two separate lines of development with varying process control based on their needs

The process specification describes the roles team that members can play and the process rules that apply to each role across the project iterations. This specification allows you to govern the project development process while providing the team with the flexibility that they need to deliver their work (Figure 14-4 on page 538). For example, as your project progresses to completion and enters the final iteration before shipment, you can issue a "code freeze" and you might want greater control over work that is performed on the product.

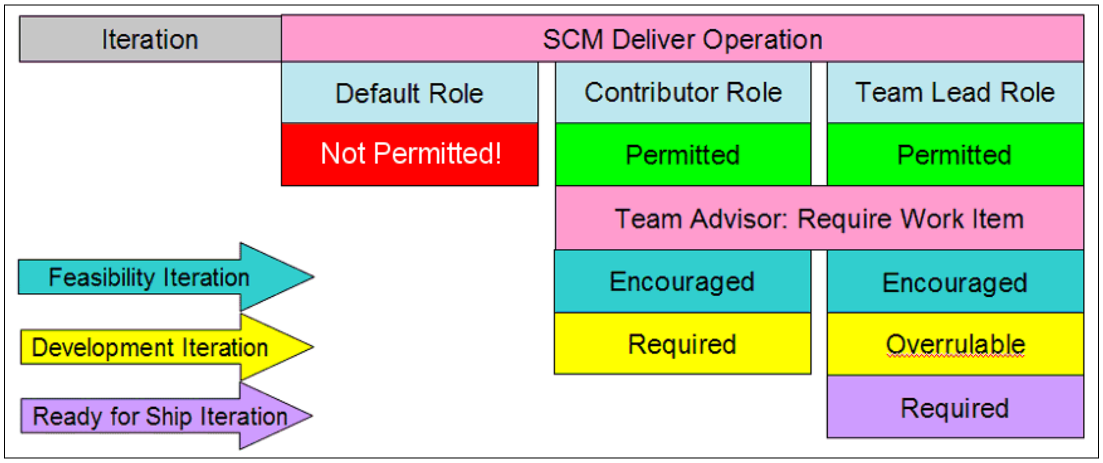


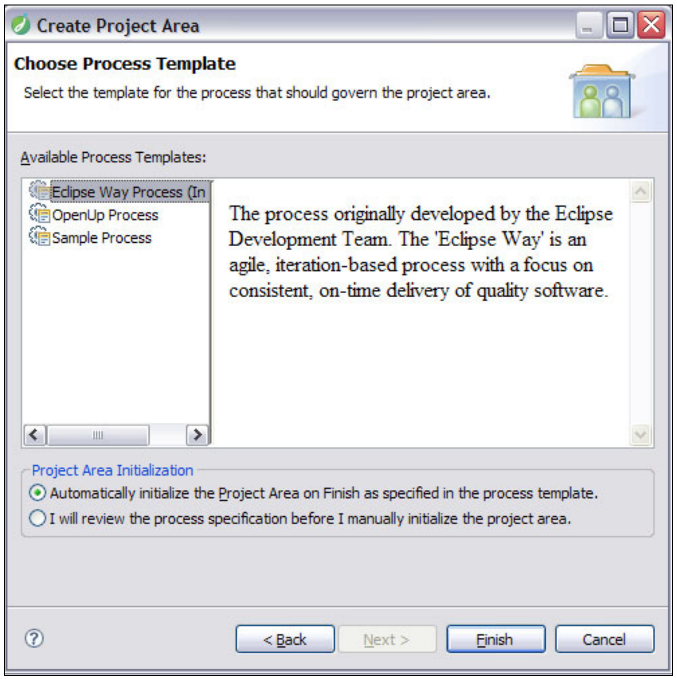
Figure 14-4 Specifying process rules based on the users role by iteration in Rational Team Concert

The process specification also defines which of the process rules can further be customized.

Example: Enactment of the OpenUP with Rational Team Concert

When you create your project work area, Rational Team Concert prompts you to specify the process template that you will use. You can create a new process template from scratch, or you can select a prepackaged or previously created process template. The OpenUP/Basic process is one of the ready-to-use process templates that comes with Rational Team Concert. OpenUP/Basic preserves the essential characteristics of the Rational Unified Process (RUP), which includes iterative development, use cases, and scenarios that drive development, risk management, and an architecture-centric approach. OpenUp/Basic is one of the processes that is defined by the Eclipse Process Framework project.

You can create a new process template from scratch, or you can select a prepackaged or previously created process template (Figure 14-5 on page 538).



Select the template for the process that should govern the project area.

Available Process Templates:

Eclipse Way Process (In
OpenUp Process
Sample Process

The process originally developed by the Eclipse Development Team. The 'Eclipse Way' is an agile, iteration-based process with a focus on consistent, on-time delivery of quality software.

Project Area Initialization

[✓ Automatically initialize the Project Area on Finish as specified in the process template.]

I will review the process specification before I manually initialize the project area.

Figure 14-5 Prepackaged process templates - Eclipse Way and OpenUP in Rational Team Concert

When you create users for your project, you define a *process role* for them. Recall that with Rational Team Concert, you can configure your process behavior based on the user role (Figure 14-6). Every user in Rational Team Concert can have one or more user roles.

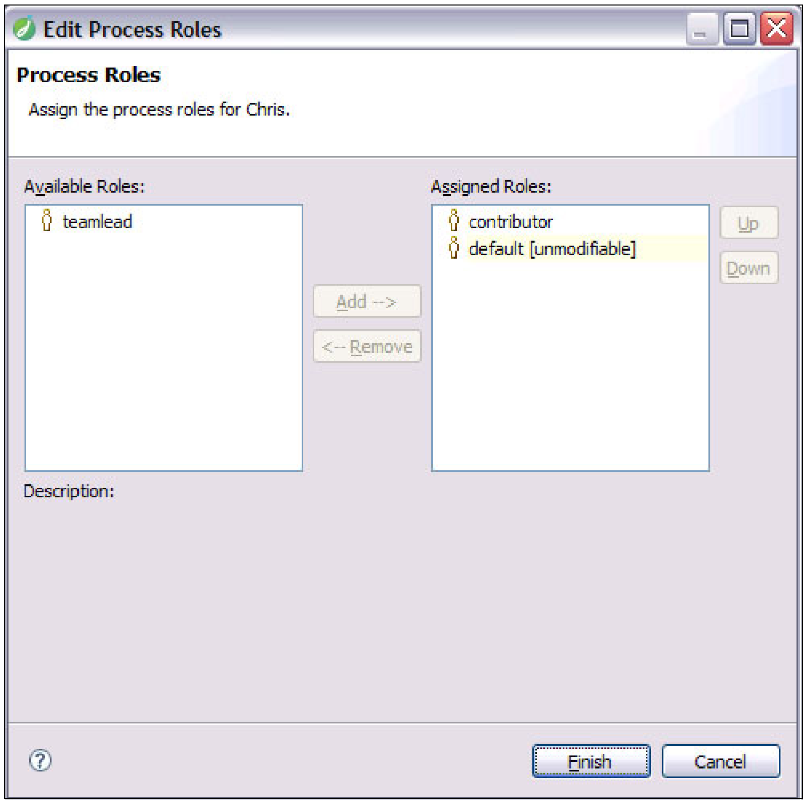
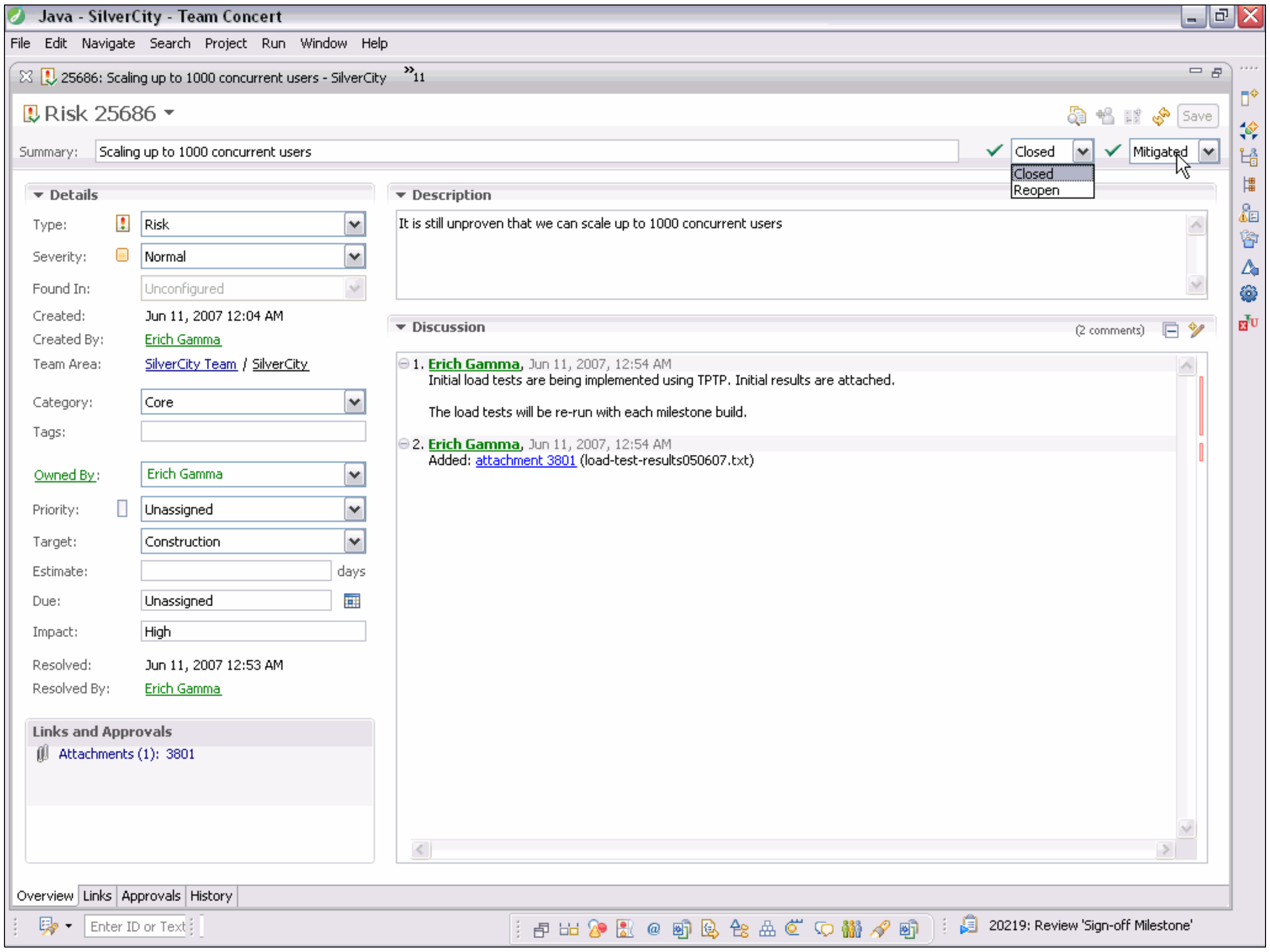


Figure 14-6 Process roles to modify the actions a user can take based on the projects selected process

The process template includes everything that you need to institute the process on your project. For example, the risk work item workflow and description detailed in OpenUP (Figure 14-7) are enacted by the process specification for OpenUP in Rational Team Concert.


Java - SilverCity - Team Concert
File Edit Navigate Search Project Run Window Help
25686: Scaling up to 1000 concurrent users - SilverCity
Risk 25686
Save
Closed
Reopen
Overview Links Approvals History
Enter ID or Text
20219: Review 'Sign-off Milestone'

Summary: Scaling up to 1000 concurrent users | Closed | Mitigated

Details

Type:	Risk
Severity:	Normal
Found In:	Unconfigured
Created:	Jun 11, 2007 12:04 AM
Created By:	Erich Gamma
Team Area:	SilverCity Team / SilverCity
Category:	Core
Tags:	
Owned By:	Erich Gamma
Priority:	Unassigned
Target:	Construction
Estimate:	days
Due:	Unassigned
Impact:	High
Resolved:	Jun 11, 2007 12:53 AM
Resolved By:	Erich Gamma

Links and Approvals

Attachments (1): 3801

Description

It is still unproven that we can scale up to 1000 concurrent users

Discussion (2 comments)

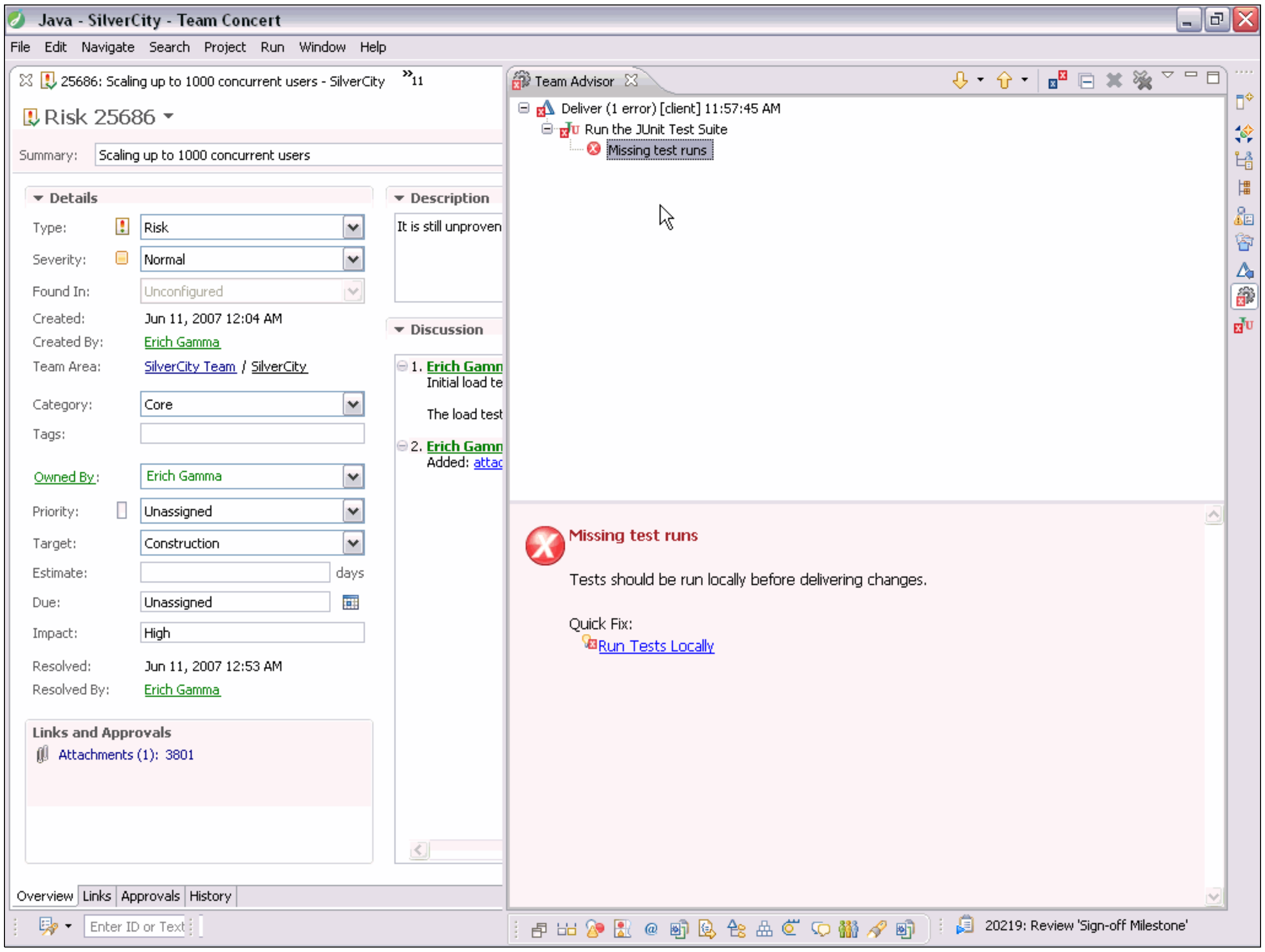
1. Erich Gamma, Jun 11, 2007, 12:54 AM
   Initial load tests are being implemented using TPTP. Initial results are attached.

   The load tests will be re-run with each milestone build.
2. Erich Gamma, Jun 11, 2007, 12:54 AM
   Added: attachment 3801 (load-test-results050607.txt)

Figure 14-7 Implementing the risk work item workflow process in Rational Team Concert

Rational Team Concert goes one step further by enabling you to enforce the project rules that you have defined for your project. In doing so, it helps to prevent team members from accidentally overstepping defined project rules and to improve the overall health and success of the project.

For example, you might institute a rule that all project team members must run JUnit tests before they deliver their changes to the team work area. When a project team member tries to deliver their changes without first successfully unit testing their code with JUnit, the team advisor steps in and ensures that the process is followed. Additionally, Rational Team Concert provides a quick fix to the problem by providing a link to run the local JUnit tests (Figure 14-8).


Java - SilverCity - Team Concert
File Edit Navigate Search Project Run Window Help
Overview Links Approvals History
Enter ID or Text

25686: Scaling up to 1000 concurrent users - SilverCity

Risk 25686

Summary: Scaling up to 1000 concurrent users

Details

Type:	Risk
Severity:	Normal
Found In:	Unconfigured
Created:	Jun 11, 2007 12:04 AM
Created By:	Erich Gamma
Team Area:	SilverCity Team / SilverCity
Category:	Core
Tags:	
Owned By:	Erich Gamma
Priority:	Unassigned
Target:	Construction
Estimate:	days
Due:	Unassigned
Impact:	High
Resolved:	Jun 11, 2007 12:53 AM
Resolved By:	Erich Gamma

Links and Approvals
Attachments (1): 3801

Description
It is still unproven

Discussion
1. Erich Gamm
Initial load te
The load test
2. Erich Gamm
Added: attac

Team Advisor
Deliver (1 error) [client] 11:57:45 AM
Run the JUnit Test Suite
Missing test runs

Missing test runs

Tests should be run locally before delivering changes.

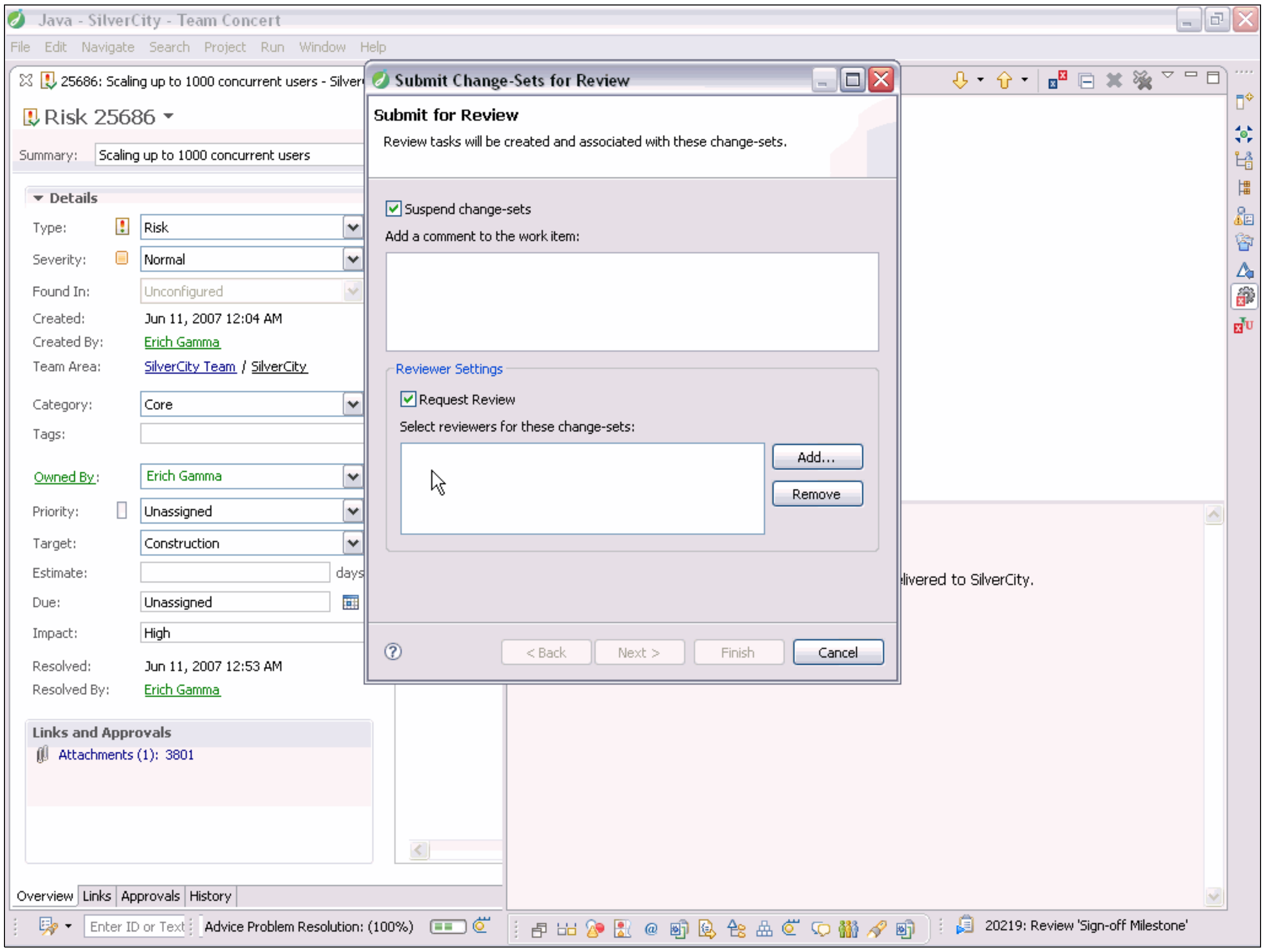
Quick Fix:
Run Tests Locally

20219: Review 'Sign-off Milestone'

Figure 14-8 Team Advisor in Rational Team Concert preventing a process rule from being broken

Clearly, all projects are not the same, and you must have flexibility to institute the rules that matter based on your project needs. Additionally, as your project progresses or your needs change, your process also must change with you.

For example, you might find the need for rapid sharing of project changes with little to no approval or review in the beginning of your project and later. As your project or milestone nears completion, you might find that you need new changes reviewed before they are delivered to the team area (Figure 14-9).


Java - SilverCity - Team Concert
File Edit Navigate Search Project Run Window Help
25686: Scaling up to 1000 concurrent users - Silver
Risk 25686
Details
Links and Approvals
Overview Links Approvals History
Submit Change-Sets for Review
Add...
Remove
< Back Next > Finish Cancel

Summary: Scaling up to 1000 concurrent users

Type:	Risk
Severity:	Normal
Found In:	Unconfigured
Created:	Jun 11, 2007 12:04 AM
Created By:	Erich Gamma
Team Area:	SilverCity Team / SilverCity
Category:	Core
Tags:	
Owned By:	Erich Gamma
Priority:	Unassigned
Target:	Construction
Estimate:	days
Due:	Unassigned
Impact:	High
Resolved:	Jun 11, 2007 12:53 AM
Resolved By:	Erich Gamma

Attachments (1): 3801

Submit for Review

Review tasks will be created and associated with these change-sets.

☑ Suspend change-sets

Add a comment to the work item:

Reviewer Settings

☑ Request Review

Select reviewers for these change-sets:

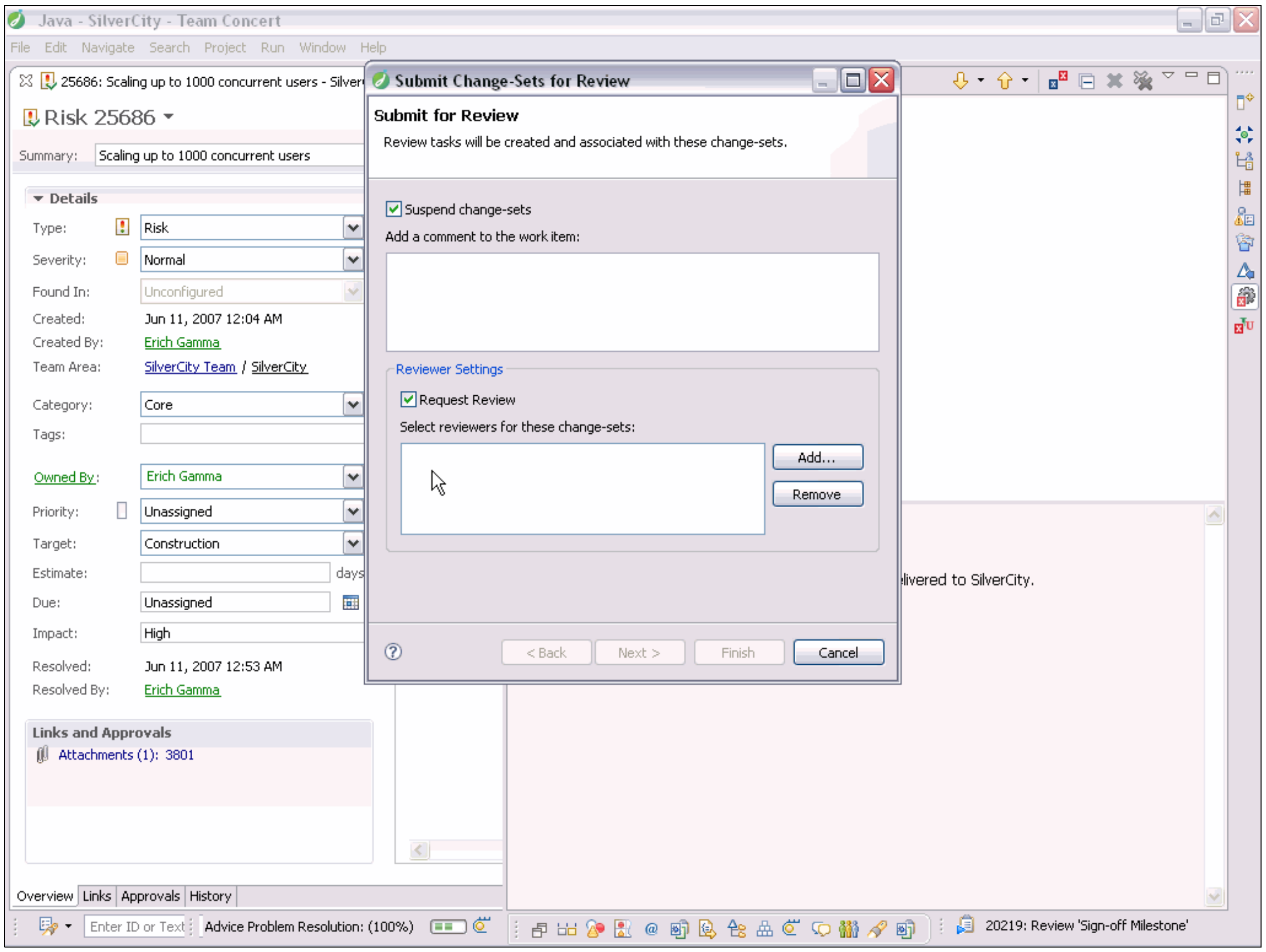
…livered to SilverCity.

Enter ID or Text | Advice Problem Resolution: (100%) | 20219: Review 'Sign-off Milestone'

Figure 14-9 Team Advisor in Rational Team Concert preventing delivery of a change set without prior approval

When you enforce a process, it is critically important that you do not block or slow the progress of project work unnecessarily. The difference between process enactment and simple process enforcement is that *process enactment* provides you with contextual guidance to resolve a process misstep. Process enactment in Rational Team Concert does not force team members into a rigid mold, but rather it advises them about process missteps and provides advice about potential resolutions.

With the enforcement of the change set approval before promotion requirement, Rational Team Concert provides a direct link to submit the change set for review (Figure 14-10). It aligns the team member with the process rules that are in effect and provides them with guidance when there is a misstep.


Java - SilverCity - Team Concert
File Edit Navigate Search Project Run Window Help

25686: Scaling up to 1000 concurrent users - Silver

Risk 25686

Summary: Scaling up to 1000 concurrent users

Details

Type:	Risk
Severity:	Normal
Found In:	Unconfigured
Created:	Jun 11, 2007 12:04 AM
Created By:	Erich Gamma
Team Area:	SilverCity Team / SilverCity
Category:	Core
Tags:	
Owned By:	Erich Gamma
Priority:	Unassigned
Target:	Construction
Estimate:	days
Due:	Unassigned
Impact:	High
Resolved:	Jun 11, 2007 12:53 AM
Resolved By:	Erich Gamma

Links and Approvals
Attachments (1): 3801

Submit Change-Sets for Review

Submit for Review
Review tasks will be created and associated with these change-sets.

☑ Suspend change-sets
Add a comment to the work item:

Reviewer Settings
☑ Request Review
Select reviewers for these change-sets:

…livered to SilverCity.

Advice Problem Resolution: (100%)

20219: Review 'Sign-off Milestone'

Figure 14-10 Direct links from Rational Team Concert to resolve process mis-steps and realign with the process

All aspects of a software development and delivery process are not executable. Process enactment must be supported with textual descriptions in order to establish team understanding of the process.

14.2.2 Process descriptions: Team understanding of the enacted process

The process description supplements the project team with the process details and best practices that are instituted on the project. The project team members use the process description to better understand the process and explore areas where they are unclear as necessary. The process description also describes process elements that are not instituted as rules or cannot be directly implemented in the software delivery environment.

Depending on the scope of the project, the process description can simply outline important team considerations or describe the process in detail relating it to process standards such as Capability Maturity Model Index (CMMI) or regulatory compliance mandates such as Sarbanes Oxley. For small teams or simple projects, the process description can be as simple as a project home page that describes important and necessary process or workflow procedures for new and changing team members. For larger teams or complex projects, the process can be represented formally, outlining the best practices and standards that the organization has wholly adopted and are expected to be implemented on every project.

Process descriptions must be modular and consumable in pieces. Having smaller chunks makes them more reusable (Figure 14-11) and easier to customize for individual projects. For example, the OpenUP is organized at a top level by disciplines, work products, roles, and the life-cycle process (Figure 14-12). These form the base hierarchy and are further organized into folders or categories, making the process descriptions easy to navigate and use. Further, the relationships between the process elements are maintained. For example, you can navigate from a work product, such as a Requirements Management Plan, to the roles that work with the work product such as the Business Analyst. This helps everyone understand how their work is interrelated and used by other team members. Organizational expectations should be that the defined process fits to the project, not vice versa.

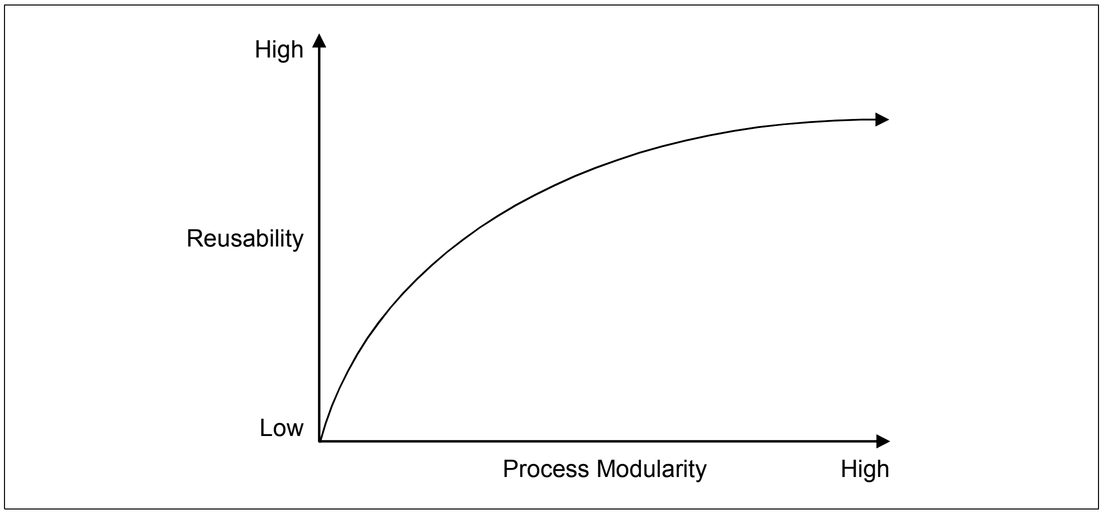


Figure 14-11 Modular processes increasing process reusability in organizations

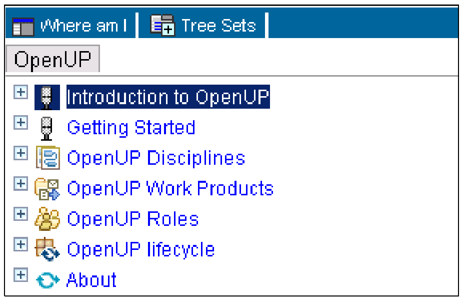
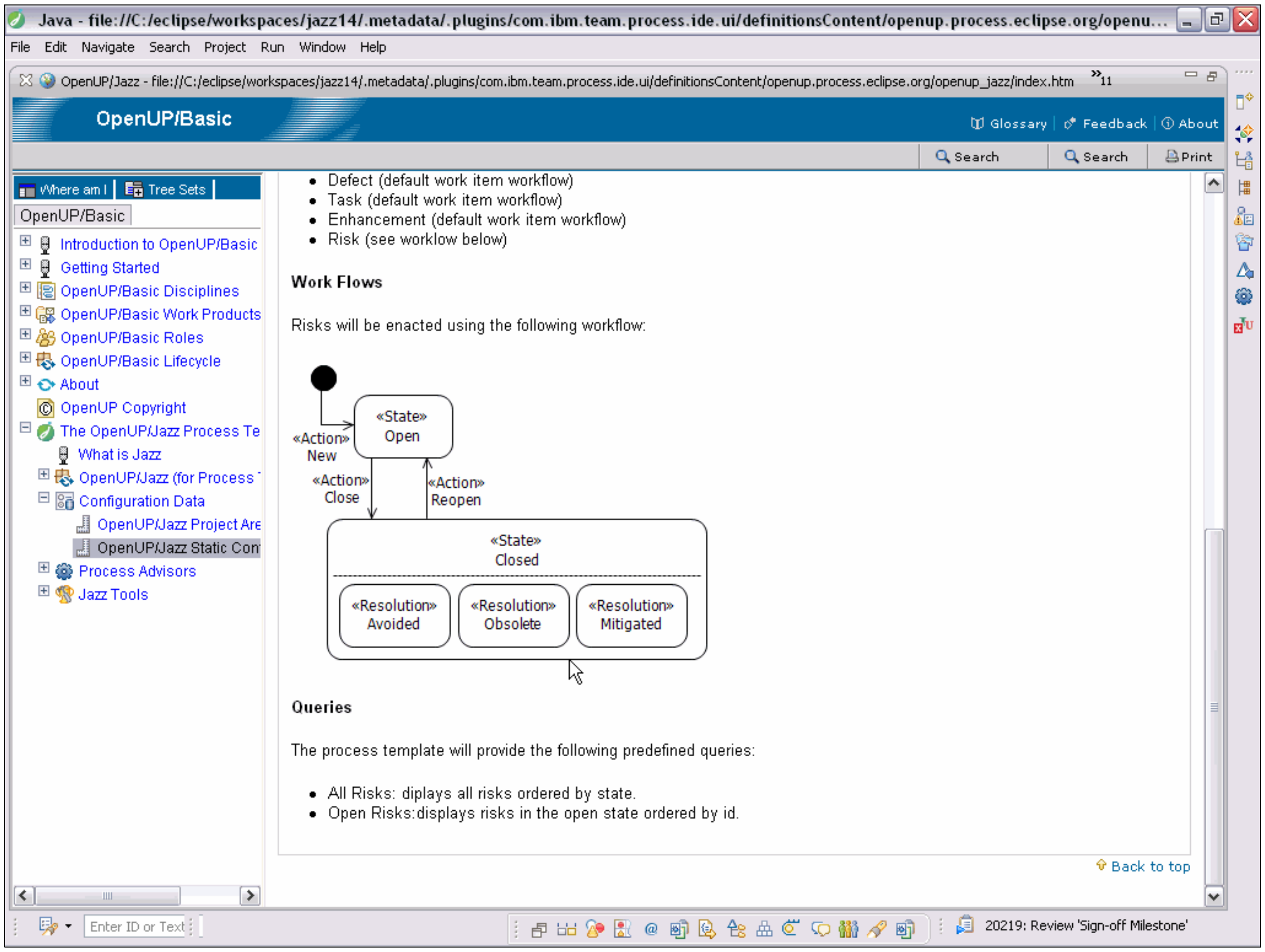


Figure 14-12 Organization of processes and best practices into easily separable chunks by using the OpenUP and the Eclipse process framework

Your process description should go hand in hand with your process implementation. The greatest risk to a lack of process adoption is too much overhead or effort that is required by a project team to follow it. Making the implementation of the process simple and a normal part of software delivery makes the adoption of the process consumable by project teams without threatening individual project goals.

For example, the OpenUP description details the workflow for the Risk Work Item. This process description is implemented by the OpenUP template in Rational Team Concerts in the form of the Risk Work Item and the associated workflow. As a result, Rational Team

Concert makes the association and connection between the described and defined process in the process description and the implemented process enacted by the Rational team.


Java - file://C:/eclipse/workspaces/jazz14/.metadata/.plugins/com.ibm.team.process.ide.ui/definitionsContent/openup.process.eclipse.org/openu...
File Edit Navigate Search Project Run Window Help
OpenUP/Jazz - file://C:/eclipse/workspaces/jazz14/.metadata/.plugins/com.ibm.team.process.ide.ui/definitionsContent/openup.process.eclipse.org/openup_jazz/index.htm
OpenUP/Basic
Glossary | Feedback | About
Search | Search | Print
Where am I | Tree Sets
OpenUP/Basic
Introduction to OpenUP/Basic
Getting Started
OpenUP/Basic Disciplines
OpenUP/Basic Work Products
OpenUP/Basic Roles
OpenUP/Basic Lifecycle
About
OpenUP Copyright
The OpenUP/Jazz Process Te
What is Jazz
OpenUP/Jazz (for Process
Configuration Data
OpenUP/Jazz Project Are
OpenUP/Jazz Static Con
Process Advisors
Jazz Tools
«State» Open
«Action» New
«Action» Close
«Action» Reopen
«State» Closed
«Resolution» Avoided
«Resolution» Obsolete
«Resolution» Mitigated
Back to top
Enter ID or Text
20219: Review 'Sign-off Milestone'

- Defect (default work item workflow)
- Task (default work item workflow)
- Enhancement (default work item workflow)
- Risk (see worklow below)

Work Flows

Risks will be enacted using the following workflow:

Queries

The process template will provide the following predefined queries:

- All Risks: diplays all risks ordered by state.
- Open Risks:displays risks in the open state ordered by id.

Figure 14-13 The OpenUP/Basic Web-based process description supplementing the project team with the details of the implemented process

14.3 Using metrics and measurements effectively to drive team success

Metrics are often used incorrectly in organizations to solely measure individual contribution and then focus attention on weak contributors to resolve project deficiencies. Although using discrete metrics to measure individual contribution is noble, it usually is not an effective driver of overall project success and teamwork.

Metrics must be aligned with the theme and goals of the project. They should be visible to the entire team and presented in a way that everyone on the team can interpret and act upon them. Successful agile project managers focus on regulation and enablement rather than delegation. The best metrics are established at the team level and encourage the team to collaborate and work together on project work rather than siloing the project into individual units that are measured independently and then aggregated to measure project success. Over segmentation encourages individualism and leads to unhealthy behavior that divides the project team.

Metrics and measurements have a life cycle, and project teams must regularly consider the ongoing value of a metric or measurement. Completed projects can help your organization determine which metrics provide the greatest benefit. These projects should be reused, and those that are not benefiting the organization should be reevaluated.

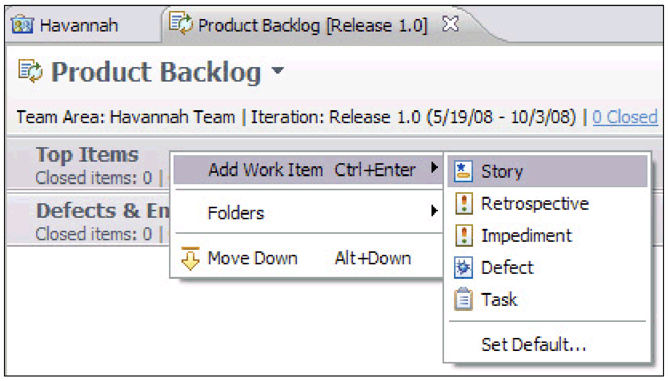
14.3.1 Selecting the right metrics

Selecting and implementing metrics is an important precursor activity to a healthy project. Metrics establish the basis for project visibility and allow the project teams to diagnose and resolve project challenges early and quickly. The most common and important metric that all projects must deal with is the sizing and estimating of project work. These metrics and measurements establish the basis for many other metrics that are used to assert the health and direction of the project. For stakeholders and customers, these metrics establish "what" will be delivered and "when" they will get it. Although the project manager is usually the person who is accountable for project estimation, the responsibility belongs to the entire project team. Agile estimation provides one technique for project estimation.

14.3.2 Agile estimation

Agile measurement and estimation focus on measuring the business value that is delivered to the customer. The basis for agile measurement is working software that is delivered to the customer. Since different teams and team members can complete the same work in varying amounts of time, agile estimation abstracts from time-based measures for effort to points. Understanding agile estimation, even if you do not use it, is important because it addresses the project dynamics of sizing, velocity, and effort that all projects will have to deal with.

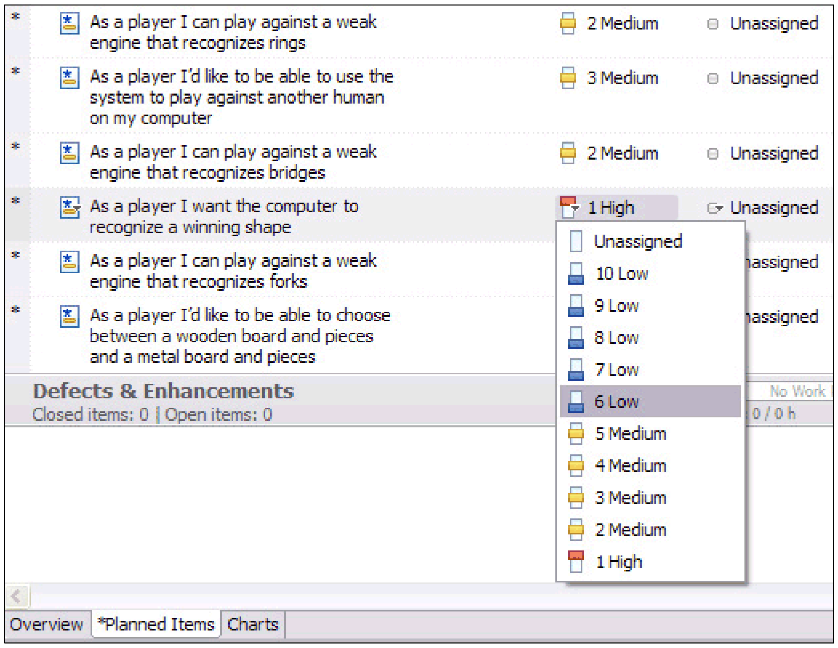
In agile planning and estimation, the project or product is defined as a collection of high level capabilities in the form of stories. These stories are collected in priority order in a product backlog. A product backlog can be likened to an iteration plan that contains all the user stories for the project/product (Figure 14-14).



Team Area: Havannah Team | Iteration: Release 1.0 (5/19/08 - 10/3/08) | 0 Closed

Figure 14-14 A work item type for capturing user stories in Rational Team Concert

Defining the user stories is a customer-centric activity, and your users or customers should be highly involved in this process. As a general rule, define stories that are small enough to be consumable and estimatable, for example, approximately 8 to 16 hours of work per story. After the complete set of stories is defined, it is are ranked in priority order in the product backlog (Figure 14-15).


As a player I can play against a weak engine that recognizes rings — 2 Medium — Unassigned

As a player I'd like to be able to use the system to play against another human on my computer — 3 Medium — Unassigned

As a player I can play against a weak engine that recognizes bridges — 2 Medium — Unassigned

As a player I want the computer to recognize a winning shape — 1 High — Unassigned

As a player I can play against a weak engine that recognizes forks

As a player I'd like to be able to choose between a wooden board and pieces and a metal board and pieces

Defects & Enhancements
Closed items: 0 | Open items: 0

Figure 14-15 Prioritizing user stories with your customers after they are defined

After the stories are ranked, the development team responsible for delivering the stories must meet to size the stories.

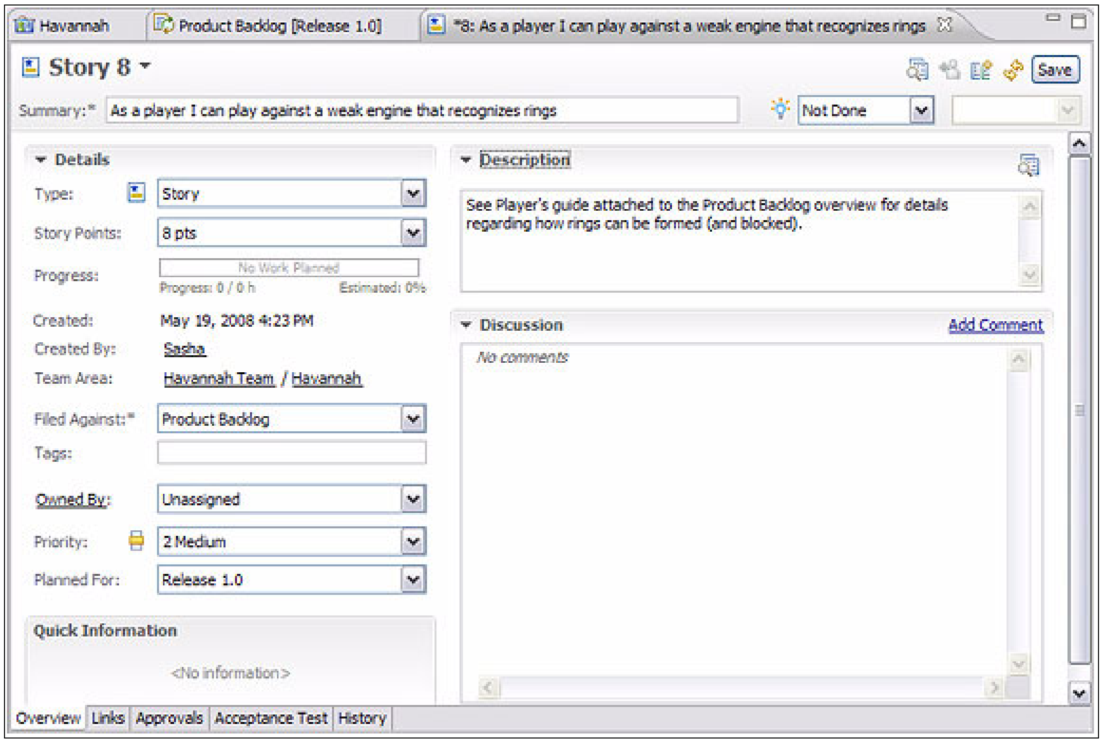
Agile sizing: Using points to estimate the amount of project work

Points in agile estimation are the measure for the amount of effort to complete a story. The agile point measurement system provides the team with the ability to overcome variances in individual capability and productivity to complete work and collectively agree on the *relative* size of work in relation to other stories with which they have been tasked.

For example, if asked how long it takes to implement a particular story, Marco might respond by saying it takes 3 hours, where Diedrie might indicate that it takes 8 hours. Both of these estimations are correct but are based on their individual capability and capacity.

If instead the team is asked to rank the size of the story relative to other stories, they can come to an agreement on the *relative size*. For example, there might be 10 other stories that must be completed. By reviewing these stories, they can agree on which stories requires the smallest amount of work and give this a point measure, such as 10. Then they can rank the other items that are relative to the size of the first. For example, they might determine that a given story they have been asked to estimate is about three times as big as the smallest story, giving it a point estimate of 30.

Estimates for size should be bound to the project team because a different project team might have a different notion for the size of a particular story based on their collective capability and experiences. Additionally, the more team members that will implement the story you involve for a given body of work in the estimation effort, the greater the accuracy of the size estimates. Particularly, at least the development leads representing different functional areas required for story implementation should be involved. Ideally, this should not be limited to the development team because software delivery involves more than writing the code. Other team members such as quality assurance, project management, release engineering, systems or service management, legal, marketing, and business subject matter experts (SMEs) should be involved. Figure 14-16 shows an example of estimating stories in Rational Team Concert.


Havannah
Product Backlog [Release 1.0]
*8: As a player I can play against a weak engine that recognizes rings
Story 8
Save

Summary:* As a player I can play against a weak engine that recognizes rings

Not Done

Details

Type: Story

Story Points: 8 pts

Progress: No Work Planned
Progress: 0 / 0 h Estimated: 0%

Created: May 19, 2008 4:23 PM

Created By: Sasha

Team Area: Havannah Team / Havannah

Filed Against:* Product Backlog

Tags:

Owned By: Unassigned

Priority: 2 Medium

Planned For: Release 1.0

Quick Information

<No information>

Description

See Player's guide attached to the Product Backlog overview for details regarding how rings can be formed (and blocked).

Discussion

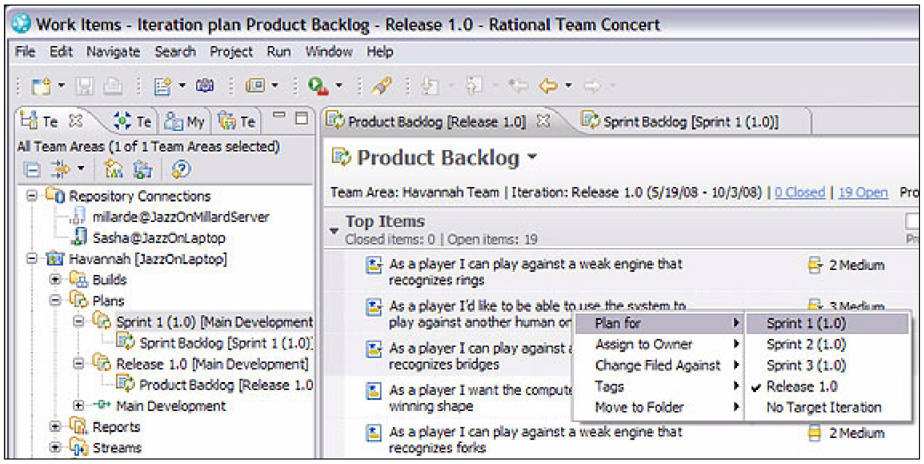
Add Comment

No comments

Figure 14-16 Estimating stories in Rational Team Concert

Estimating effort

After a prioritized, estimated, elaborated product backlog of stories is established, the project team pulls stories from the product backlog into the *sprint backlog*. The sprint backlog is the iteration plan for the sprint. In general, a sprint can be likened to an iteration. User stories are pulled from the product backlog and planned for the sprint by adding them to the sprint backlog (Figure 14-17).


Work Items - Iteration plan Product Backlog - Release 1.0 - Rational Team Concert
File Edit Navigate Search Project Run Window Help
Te
Te
My
Te
Product Backlog [Release 1.0]
Sprint Backlog [Sprint 1 (1.0)]
All Team Areas (1 of 1 Team Areas selected)
Repository Connections
millarde@JazzOnMillardServer
Sasha@JazzOnLaptop
Havannah [JazzOnLaptop]
Builds
Plans
Sprint 1 (1.0) [Main Development]
Sprint Backlog [Sprint 1 (1.0)]
Release 1.0 [Main Development]
Product Backlog [Release 1.0]
Main Development
Reports
Streams
Product Backlog
Plan for
Assign to Owner
Change Filed Against
Tags
Move to Folder
Sprint 1 (1.0)
Sprint 2 (1.0)
Sprint 3 (1.0)
✓ Release 1.0
No Target Iteration

Team Area: Havannah Team | Iteration: Release 1.0 (5/19/08 - 10/3/08) | 0 Closed | 19 Open

Top Items
Closed items: 0 | Open items: 19

As a player I can play against a weak engine that recognizes rings — 2 Medium

As a player I'd like to be able to use the system to play against another human or — 3 Medium

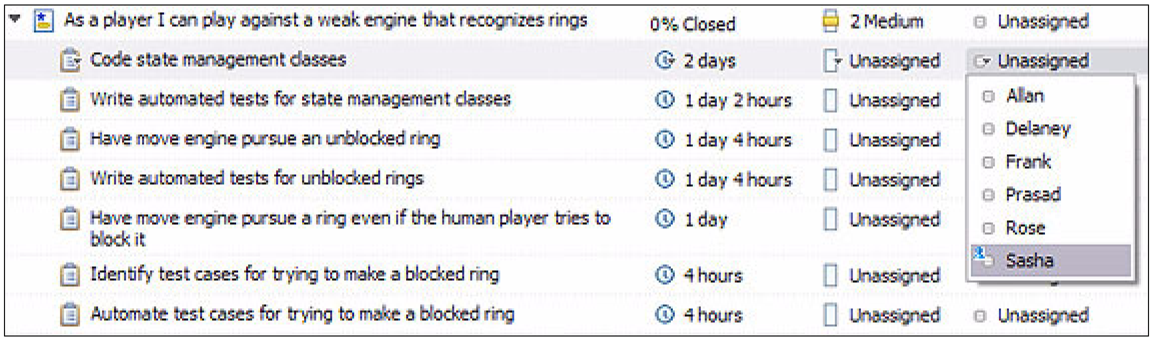
As a player I can play against a recognizes bridges

As a player I want the compute winning shape

As a player I can play against a weak engine that recognizes forks — 2 Medium

Figure 14-17 Stories taken from the product backlog and added to the sprint backlog

After the stories are added to the current sprint, they are then further elaborated with the tasks that are required to implement them. The tasks are initially estimated and assigned to team members (Figure 14-18).



As a player I can play against a weak engine that recognizes rings	0% Closed	2 Medium	Unassigned
Code state management classes	2 days	Unassigned	Unassigned
Write automated tests for state management classes	1 day 2 hours	Unassigned	
Have move engine pursue an unblocked ring	1 day 4 hours	Unassigned	
Write automated tests for unblocked rings	1 day 4 hours	Unassigned	
Have move engine pursue a ring even if the human player tries to block it	1 day	Unassigned	
Identify test cases for trying to make a blocked ring	4 hours	Unassigned	
Automate test cases for trying to make a blocked ring	4 hours	Unassigned	Unassigned

Figure 14-18 Task estimates for stories established for each story by the assigned team member

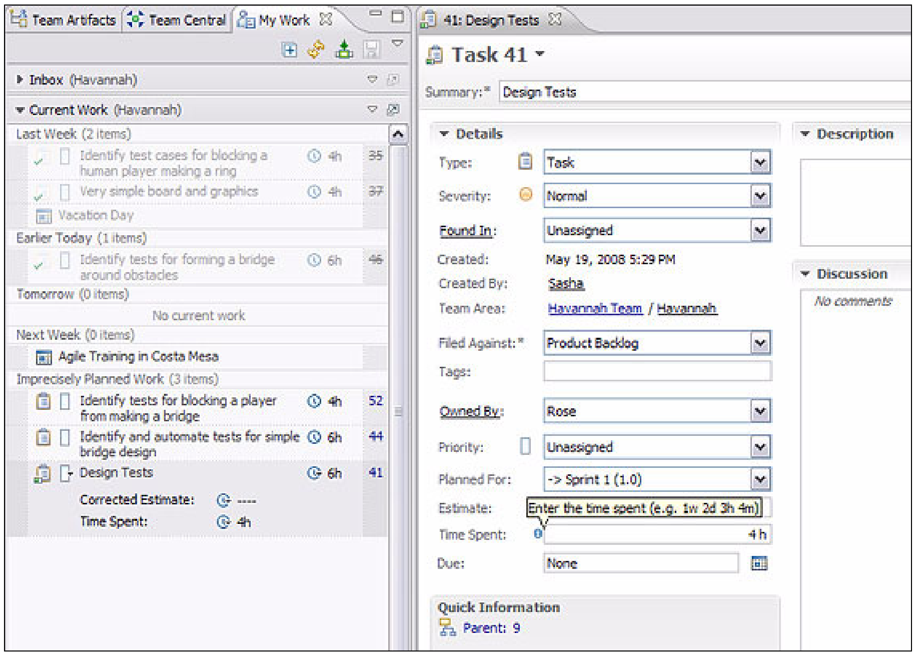
Agile estimation of effort translates the point estimations that the team has defined for stories to a time-based unit such as days or hours. These time estimates remain imprecise until they are assigned to a team member and the team member actively manages the time estimate. The estimating activity is primarily done individually because each person's estimates for effort will be different.

Estimation of effort happens after individual team members have either been assigned work or sign up to take on a new work item. The reason estimation happens after work assignment is because time estimates for completing work items vary depending on the team member that is assigned the work as in the following examples:

- Diedre (Work item: Fix logo on home page, 3 points) Work Estimate: 1 hour
- Marco (Work item: Resolve performance issue in logging service, 30 points) Work Estimate: 24 actual hours

Traditional estimating and sizing are imprecise because they draw conclusions on the general amount of effort that is required to complete a task rather than their specific teams' effort to complete a task. Agile estimation promotes an agile team by allowing team members to more freely work and adjust to the changing project dynamics rather than be locked into a rigid, difficult-to-change project plan. Agile estimation makes project scheduling more accurate, and as a consequence, more dynamic (Figure 14-19 on page 550).

Agile estimation allows team members to truly take ownership of a work item. Each team member provides the effort estimate because this person is naturally the best one to gauge it and their estimate matters most. Team members should be encouraged to regularly review their work item estimates and adjust them judiciously. When you make an estimation, which is a responsibility that the whole team shares, the credibility and accuracy of the iteration plan increases significantly.


Team Artifacts | Team Central | My Work

Inbox (Havannah)

Current Work (Havannah)

Last Week (2 items)

Identify test cases for blocking a human player making a ring	4h	35
Very simple board and graphics	4h	37
Vacation Day		

Earlier Today (1 items)

Identify tests for forming a bridge around obstacles	6h	46

Tomorrow (0 items)

No current work

Next Week (0 items)

Agile Training in Costa Mesa

Imprecisely Planned Work (3 items)

Identify tests for blocking a player from making a bridge	4h	52
Identify and automate tests for simple bridge design	6h	44
Design Tests	6h	41

Corrected Estimate: ----
Time Spent: 4h

41: Design Tests

Task 41

Summary:* Design Tests

Details

Type:	Task
Severity:	Normal
Found In:	Unassigned
Created:	May 19, 2008 5:29 PM
Created By:	Sasha
Team Area:	Havannah Team / Havannah
Filed Against:*	Product Backlog
Tags:	
Owned By:	Rose
Priority:	Unassigned
Planned For:	-> Sprint 1 (1.0)
Estimate:	Enter the time spent (e.g. 1w 2d 3h 4m)
Time Spent:	4 h
Due:	None

Quick Information
Parent: 9

Description

Discussion
No comments

Figure 14-19 Agile estimating, planning for the team to actively manage work estimates in the project

Project velocity: Aligning team capacity to the project work

Project velocity measures the amount of work that a project team can complete within an iteration. Project velocity is a key measurement for project planning because it helps a project team understand their capacity and plan or schedule future work accordingly.

For example, a project team might plan to deliver 100 points in the first iteration, but they only deliver 80 points. In this case, their velocity was less than anticipated, and they might adjust down the points that are planned for the next iteration to match their real capacity, for example 90. In this case, they believe that they can do a little better than the first iteration because the team has become more cohesive.

Project velocity uses the team's historical performance as a basis for measurement. For example, in a project with three sprints (iterations) completed from a total of 6 planned sprints, the team can use the number of points that they were able to deliver in past sprints as a basis for the amount of story points that they can realistically deliver in the upcoming sprint.

Project velocity helps a team understand their real capacity so that they can set expectations appropriately with stakeholders and other subteams that might depend on their contributions. Project velocity is often expressed as a ratio with delivered points or planned points. The velocity ratio allows for more accurate planning across subteams. Remember that point estimations for work are relative and bound to a particular project team.

Estimating capacity based on points across teams does not work. For example, let us say that Marco's development team sizes a set of work items to be 300 points, and another offshore team sizes the work items to be 900 points for the next iteration. They both deliver exactly the work items that they planned for the iteration. Clearly, measuring by points only might incorrectly show the offshore team as more productive.

Additionally, if there was a work item that needed to be completed for the next iteration, which team is most likely to be capable of taking it if their next iteration was already planned? What if Marco's team had a project velocity of 2 and the offshore team had a velocity of .8? Clearly, Marco's team may be best suited to take on the additional work and still deliver the work that is planned for the next iteration.

Figure 14-20 shows an example of a Team Velocity report.

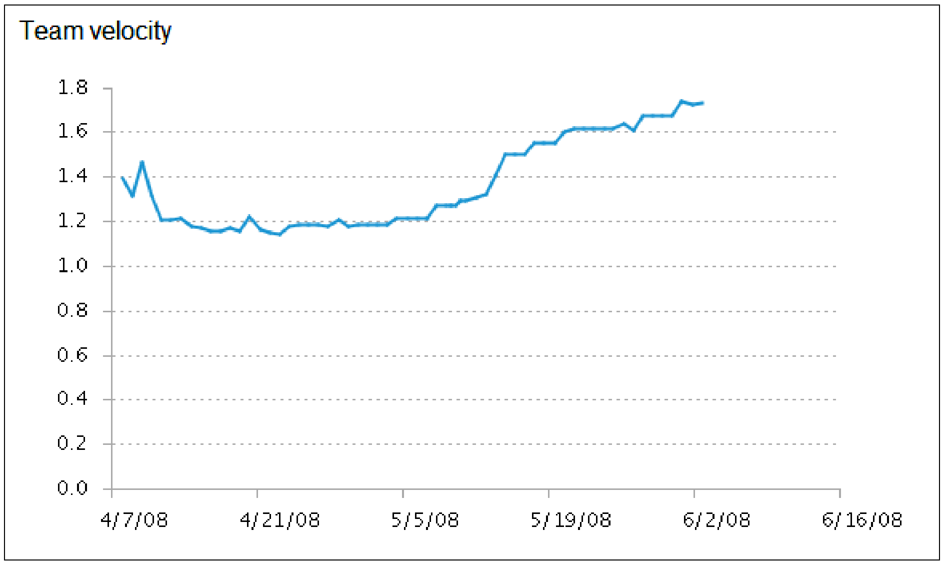


Figure 14-20 The Team Velocity report showing how the team is performing against their planned performance

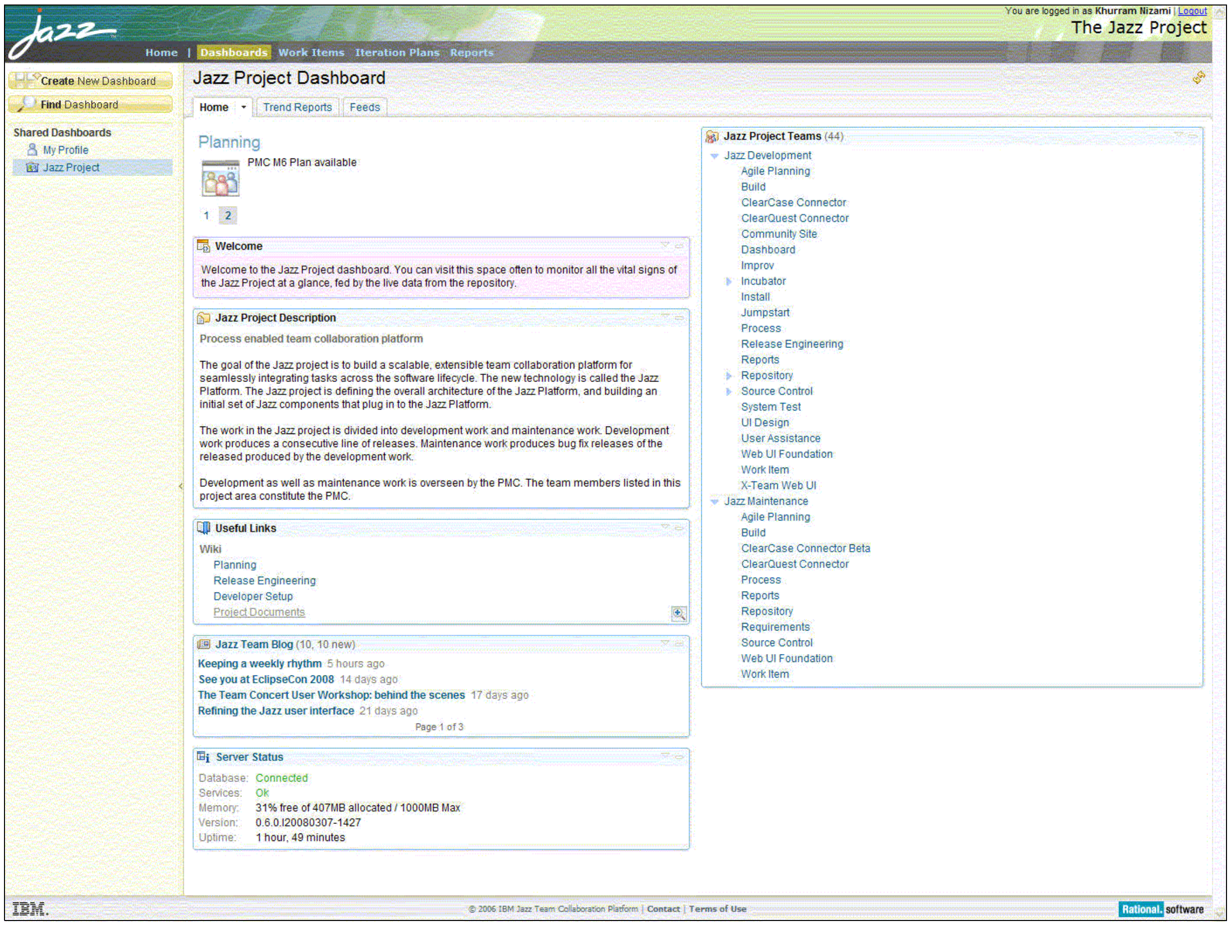
14.4 Using dashboards for decision making

Dashboards are an effective way of establishing transparency and visibility throughout the project by communicating the current health of various parts of the project. Projects should have at least one common shared dashboard that the entire team uses to understand and act on overall project health. Projects should also have personal dashboards that team members can use to focus on the project area that they own.

Projects should look at their instituted metrics if more effort is being spent on collecting measurements rather than acting on them. Additionally, metric quality is far more important than metric quantity on a project. The purpose and value of each metric should be visible and understood by the entire project team.

Rational Team Concert provides the ability to create *shared dashboards* that provide the entire project with measures fed by live data. Shared dashboards should be created at the beginning of the project by the project manager and updated throughout the project. The home page for the dashboard should include a collaborative element such as a team blog where others can comment on observations from trend reports.

Figure 14-21 on page 552 shows the shared Jazz project dashboard. Note that it provides an overview of the project, vital links, and a team blog where team members can interact. It also includes a list of the Jazz Project Sub Teams hyperlinked to their dashboard. The team dashboard is a great place for communicating news such as team members who are joining or departing.



Jazz
You are logged in as Khurram Nizami | Logout
The Jazz Project
Home | Dashboards Work Items Iteration Plans Reports
Create New Dashboard
Find Dashboard
Shared Dashboards
My Profile
Jazz Project
Home
Trend Reports
Feeds
IBM.
© 2006 IBM Jazz Team Collaboration Platform | Contact | Terms of Use
Rational. software

Jazz Project Dashboard

Planning

PMC M6 Plan available

1 2

Welcome

Welcome to the Jazz Project dashboard. You can visit this space often to monitor all the vital signs of the Jazz Project at a glance, fed by the live data from the repository.

Jazz Project Description

Process enabled team collaboration platform

The goal of the Jazz project is to build a scalable, extensible team collaboration platform for seamlessly integrating tasks across the software lifecycle. The new technology is called the Jazz Platform. The Jazz project is defining the overall architecture of the Jazz Platform, and building an initial set of Jazz components that plug in to the Jazz Platform.

The work in the Jazz project is divided into development work and maintenance work. Development work produces a consecutive line of releases. Maintenance work produces bug fix releases of the released produced by the development work.

Development as well as maintenance work is overseen by the PMC. The team members listed in this project area constitute the PMC.

Useful Links

Wiki
- Planning
- Release Engineering
- Developer Setup
- Project Documents

Jazz Team Blog (10, 10 new)

Keeping a weekly rhythm 5 hours ago
See you at EclipseCon 2008 14 days ago
The Team Concert User Workshop: behind the scenes 17 days ago
Refining the Jazz user interface 21 days ago
Page 1 of 3

Server Status

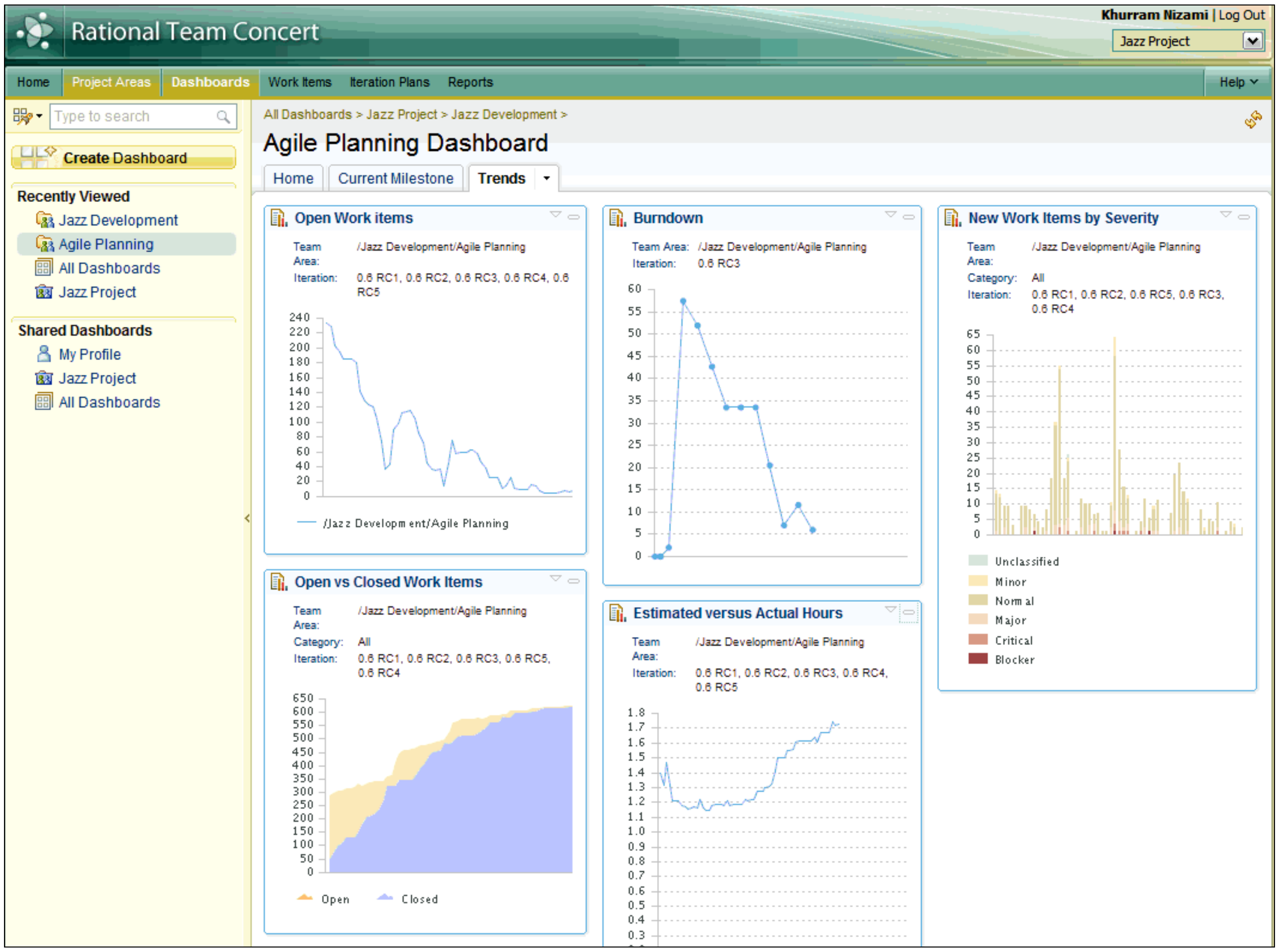
Database: Connected
Services: Ok
Memory: 31% free of 407MB allocated / 1000MB Max
Version: 0.6.0.I20080307-1427
Uptime: 1 hour, 49 minutes

Jazz Project Teams (44)

- Jazz Development
  - Agile Planning
  - Build
  - ClearCase Connector
  - ClearQuest Connector
  - Community Site
  - Dashboard
  - Improv
  - Incubator
  - Install
  - Jumpstart
  - Process
  - Release Engineering
  - Reports
  - Repository
  - Source Control
  - System Test
  - UI Design
  - User Assistance
  - Web UI Foundation
  - Work Item
  - X-Team Web UI
- Jazz Maintenance
  - Agile Planning
  - Build
  - ClearCase Connector Beta
  - ClearQuest Connector
  - Process
  - Reports
  - Repository
  - Requirements
  - Source Control
  - Web UI Foundation
  - Work Item

Figure 14-21 Rational Team Concert Dashboard

Measurements in dashboards should be visible to the entire team in a meaningful way and empower team members to act on them (Figure 14-22).



Open Work items — Team Area: /Jazz Development/Agile Planning; Iteration: 0.6 RC1, 0.6 RC2, 0.6 RC3, 0.6 RC4, 0.6 RC5

Burndown — Team Area: /Jazz Development/Agile Planning; Iteration: 0.6 RC3

New Work Items by Severity — Team Area: /Jazz Development/Agile Planning; Category: All; Iteration: 0.6 RC1, 0.6 RC2, 0.6 RC5, 0.6 RC3, 0.6 RC4

Open vs Closed Work Items — Team Area: /Jazz Development/Agile Planning; Category: All; Iteration: 0.6 RC1, 0.6 RC2, 0.6 RC3, 0.6 RC5, 0.6 RC4

Estimated versus Actual Hours — Team Area: /Jazz Development/Agile Planning; Iteration: 0.6 RC1, 0.6 RC2, 0.6 RC3, 0.6 RC4, 0.6 RC5

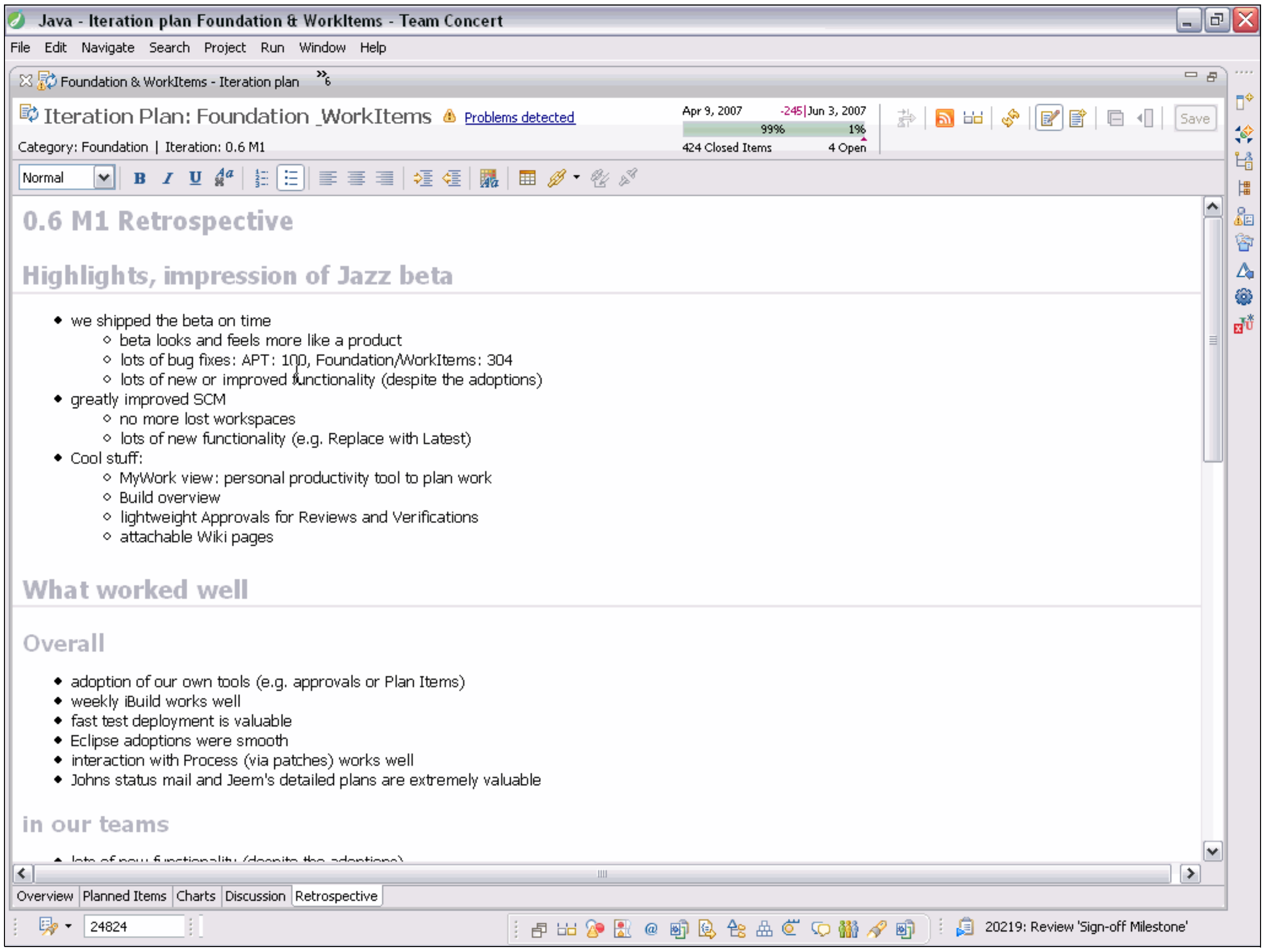
Figure 14-22 A dashboard displaying key project health reports that everyone on the team understands

14.5 Using retrospectives to capture lessons learned and make adjustments

Project managers must recognize that everything on a project cannot (and should not) be quantitatively assessed. Many important qualities that are experienced throughout the project are best communicated by natural language rather than mathematical observations or calculations. Project milestones, such as a completed iteration, should also include an open project team discussion of lessons learned and “senses” by the project team. One technique for doing so is to conduct retrospectives.

Retrospectives give the team a chance to look back at a completed iteration and reflect on what worked well and what did not work so well. Retrospectives allow teams to absorb and express quantitative results of the project in a more natural form. They allow the team to get a “sense” for the project. These lessons learned can help them decide what changes to the process, if any, are necessary (Figure 14-23).

During a retrospective meeting, it is important to facilitate a constructive session rather than a synopsis of past events. Focus the team on what actions can be taken to better handle similar challenges proactively in the future.


Java - Iteration plan Foundation & WorkItems - Team Concert
File Edit Navigate Search Project Run Window Help
Foundation & WorkItems - Iteration plan
Iteration Plan: Foundation _WorkItems
Problems detected
Apr 9, 2007
-245 Jun 3, 2007
99%
1%
424 Closed Items
4 Open
Save
Category: Foundation | Iteration: 0.6 M1
Normal

0.6 M1 Retrospective

Highlights, impression of Jazz beta

- we shipped the beta on time
  - beta looks and feels more like a product
  - lots of bug fixes: APT: 100, Foundation/WorkItems: 304
  - lots of new or improved functionality (despite the adoptions)
- greatly improved SCM
  - no more lost workspaces
  - lots of new functionality (e.g. Replace with Latest)
- Cool stuff:
  - MyWork view: personal productivity tool to plan work
  - Build overview
  - lightweight Approvals for Reviews and Verifications
  - attachable Wiki pages

What worked well

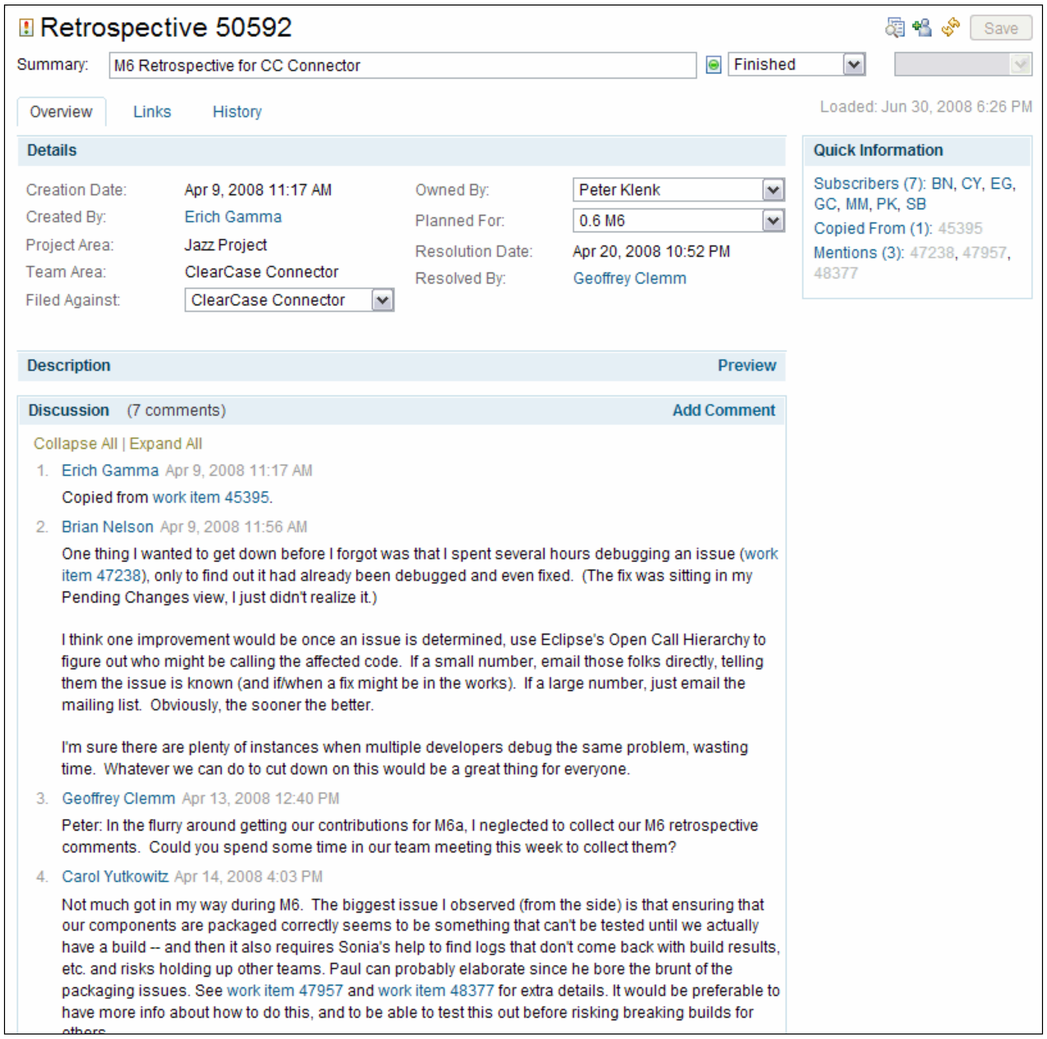
Overall

- adoption of our own tools (e.g. approvals or Plan Items)
- weekly iBuild works well
- fast test deployment is valuable
- Eclipse adoptions were smooth
- interaction with Process (via patches) works well
- Johns status mail and Jeem's detailed plans are extremely valuable

in our teams

Figure 14-23 Capturing team conclusions and ways forward for project milestones by using retrospectives

Retrospectives can be captured in the iteration plan or in a work item or change request (Figure 14-24). Capturing retrospectives in a work item allows the team to use the product collaboration features. For example, with Rational Team Concert, the discussion and subscription work item features can be used to enable project team members to easily contribute to the retrospective. This enables a geographically distributed team to provide their input without the challenges of coordinating a real-time meeting. Team leads and team members can gather their thoughts and provide quality input on the retrospective and have a record of the entire team's feedback.


Retrospective 50592

Summary: M6 Retrospective for CC Connector — Finished

Loaded: Jun 30, 2008 6:26 PM

Details

Creation Date:	Apr 9, 2008 11:17 AM	Owned By:	Peter Klenk
Created By:	Erich Gamma	Planned For:	0.6 M6
Project Area:	Jazz Project	Resolution Date:	Apr 20, 2008 10:52 PM
Team Area:	ClearCase Connector	Resolved By:	Geoffrey Clemm
Filed Against:	ClearCase Connector		

Quick Information

Subscribers (7): BN, CY, EG, GC, MM, PK, SB
Copied From (1): 45395
Mentions (3): 47238, 47957, 48377

Description

Discussion (7 comments)

Collapse All | Expand All

1. Erich Gamma Apr 9, 2008 11:17 AM
   Copied from work item 45395.
2. Brian Nelson Apr 9, 2008 11:56 AM
   One thing I wanted to get down before I forgot was that I spent several hours debugging an issue (work item 47238), only to find out it had already been debugged and even fixed. (The fix was sitting in my Pending Changes view, I just didn't realize it.)

   I think one improvement would be once an issue is determined, use Eclipse's Open Call Hierarchy to figure out who might be calling the affected code. If a small number, email those folks directly, telling them the issue is known (and if/when a fix might be in the works). If a large number, just email the mailing list. Obviously, the sooner the better.

   I'm sure there are plenty of instances when multiple developers debug the same problem, wasting time. Whatever we can do to cut down on this would be a great thing for everyone.
3. Geoffrey Clemm Apr 13, 2008 12:40 PM
   Peter: In the flurry around getting our contributions for M6a, I neglected to collect our M6 retrospective comments. Could you spend some time in our team meeting this week to collect them?
4. Carol Yutkowitz Apr 14, 2008 4:03 PM
   Not much got in my way during M6. The biggest issue I observed (from the side) is that ensuring that our components are packaged correctly seems to be something that can't be tested until we actually have a build -- and then it also requires Sonia's help to find logs that don't come back with build results, etc. and risks holding up other teams. Paul can probably elaborate since he bore the brunt of the packaging issues. See work item 47957 and work item 48377 for extra details. It would be preferable to have more info about how to do this, and to be able to test this out before risking breaking builds for

Figure 14-24 Retrospectives captured in work items or change requests to facilitate collaboration

A

Principles for Collaborative Application Lifecycle Management

In this appendix, we provide additional considerations for Collaborative Application Lifecycle Management (CALM). In particular, we provide a set of philosophical and technical principles.

This appendix includes the following sections:

Philosophical principles

In this section, we present a set of philosophical principles to consider when evaluating or designing a CALM solution.

Development is not an island unto itself

Development provides a service to the business. Technologists often like to create technology because there is an interesting new problem to solve. However, today IT is managed more like a business, and the software it produces directly contributes to the bottom line. Therefore, software must be created in context of the business goals and objectives. This information needs to seamlessly flow from the business stakeholders to the development teams that are providing the software solutions. Information, such as current versus simulated business processes, and key performance indicators become critical pieces of information for the development team.

At the same time, the enterprise architecture drives project decisions in context of a larger architectural vision such as adhering to a set of architectural principles or an architectural framework. Furthermore, many software solutions are deployed into the IT infrastructure where they must be managed by the operations team. Visibility into the service level agreements, capacity, existing assets, and monitoring strategies influences how the solution is developed and tested.

To add to the mix, software development is global in nature, with offshoring and outsourcing becoming common realities for many organizations. Teams are comprised of people from around the world, from within the enterprise, and with a core set of trusted partners, each contributing to some aspect of the solution.

Therefore, the development team is not an island unto itself. Rather, the development team innovates within the context of business goals, enterprise architectures, and operational constraints while collaborating with people in multiple organizations and geographies. With this perspective in mind, the role of the development team is to provide the service of producing software for the rest of the business.

Software solutions are the product of many conversations

Often in software development, teams try to decompose solutions down to data, whether it is objects in an object model, test cases in a test suite, or a collection of requirements in the requirements management system. The reality, however, is that teams arrive at conclusions after having many conversations with peers and stakeholders. Intellectual property comes from human beings, and these people are in multiple organizations and geographies, using a wide array of tools to produce artifacts.

A CALM solution must support people regardlesss of who they are, where they are, the conversations they have, and the artifacts that they create as a result of these conversations. Lifecycle management involves managing networks (or clouds) of people, artifacts, and their shared bookmarks. Project teams set up wikis and blogs to share and communicate information. Communities of people form around core ideas. Information and ideas are shared liberally until they form a final state.

By linking people and the artifacts they produce, a lifecyle management solution can help team members to easily find answers to some of the following questions:

- Who do I collaborate with most often?
- What other artifacts are linked to mine?

- How can I more easily share the documents that others keep requesting?
- Who else has an artifact of interest to the work that I am attempting to complete?
- Who else is interested in this topic?
- Are there any experts I can consult?

The following basic rules involve these networks of people and artifacts:

- People collaborate to share information and exchange ideas in the context of their current project, using a wide array of techniques such as shared bookmarks, blogs, wikis, communities, webcasts, and demo videos.
- People produce artifacts by using a wide array of tools. In the end, the tool itself is not important. The artifact that contains the data (and versions thereof) is what is managed.
- Some people and artifacts are extremely useful, an others are not. Ratings help others on the team understand the importance and value of the artifacts that are produced by the community. They can also indicate the people that provide helpful information.
- In the context of a conversation, a user might reference, share, or want to find artifacts that are related to the discussion. Providing URLs, shared bookmarks, or both facilitates the sharing.
- A change to one artifact impacts other artifacts and, therefore, the people who use them. Team members must be aware of the impact of these changes.

  For example, a change to a requirement might trickle all the way through design, implementation, build scripts, test cases, and scripts, causing massive rework by the entire team. If any member of the team is not aware of such a change, the project might fail to meet the business expectation. Additionally, allowing the owner of an artifact to understand the potential impact to other team members before making the change can help them manage their changes more effectively.

The relationships between people and artifacts are fundamental to developing deployable software.

Solutions are rarely sunset; they are refined and maintained for years

Development tools vendors often design products to support teams that create “green field” implementations. The reality is that almost 75% of the IT budget is spent on managing existing applications. A CALM solution at its heart must support existing investments in software solutions. By this, it must be extremely easy to recreate a past release of the solution or to discover which version is currently running in production.

Therefore, all artifacts should be treated as investments. By this we mean, that all artifacts that are created during the development of a solution should be managed and versioned in context of that solution. This goes well beyond the traditional definition of managing source code, to encompass managing the relationships between requirements, designs, models, source code, unit tests, build and deployment scripts, test plans, configurations, test cases, and results. It also involves capturing the initial business goals and objectives, along with capturing information about the operations infrastructure and transferring knowledge to the operations team for managing and monitoring.

These artifacts need to stay in place but also move forward. To recreate past releases, and to adhere to regulatory requirements, the state of all of the artifacts at the end of the release must be captured and maintained. The baseline that is captured must include more than just source code. It must also include all other artifacts that contribute to the definition and history of the release, such as the requirements, test cases, test results, and so forth. To that end, capturing a baseline of *all* resources in the project (team members, groups, roles in addition to the artifacts) might be equally important.

At the same time, these artifacts must be reused and modified to apply to the next release of the solution. For example, the set of requirements for one release must be validated again by using the same test cases in the next release to ensure that there are no regressions. Team members and the roles they play, along with the processes they use, might also move forward.

Additionally, some artifacts can be valuable enough to package into an asset that can be reused by anyone with access to it. The ability to package and promote assets, and reuse existing assets, increases the value of the artifacts, while increasing the efficiency of the IT team. As more solutions become assembled from parts of other solutions, this need for asset reuse is only expected to increase.

Many cycles are ripe for automation and recommendation

By definition, CALM involves managing a cycle. However, it is more interesting to note that there cycles within cycles for a single software solution. The solution moves through more than just the cycle of strategy, design, transition, and operation. There are cycles where the business priorities are reviewed against the current project portfolio to ensure alignment. Priorities from operations drive maintenance cycles and hot fixes to existing applications. Project schedules drive development software projects, which involves cycles of planning, executing, and evaluating. Implementation involves continuous cycles of coding, unit testing, delivering, building, deploying, and testing against the requirements. The test results indicate the level of quality, which then drives additional development cycles within iterations. A defect has a cycle.

Within all of these cycles, repeated patterns of activity can be automated. For example, agile methods advocate that teams conduct continuous builds to drive out build errors and increase the quality of the builds. This is an example of automation where the developers deliver their code, which kicks off a build to ensure that their changes do not break the team build. In addition, this build process includes running test scripts, code scans, or both, which provide an indicator of build quality.

With a little imagination and a focus on the cycles that are inherent in software development, many of the manual and mundane processes within the software development life cycle can be automated given the proper relationships between artifacts.

Simplicity first

It is easy to grow from something simple, but it is difficult to simplify something that is complex. Existing Application Lifecycle Management (ALM) solutions grow by taking existing discipline-specific tools and creating loose integrations between them. To look at it another way, each of these discipline-specific tools are already complex within their discipline. To attempt integrations simply results in increased complexity.

When team members focus on a single discipline, it is extremely easy to delve deeper into that domain. However, in doing so, rather than thinking of the simplest case, teams risk building mountains of nuance until the solution is no longer usable by the people who need it the most. Rather than thinking in terms of the individual disciplines, team members must think about the entire team and how they interact and rely on each other’s work.

The team can set its sights on best practices for software development teams. These team-based best practices lead to better understand the relationships between people and the artifacts they produce, thus allowing the team to provide a high-value innovative solution, rather than an overly complex system full of subtle nuance. The reality is that every enterprise approaches software development differently. Attempting to handle every case is an enormous and risk-filled challenge. However, by adopting core best practices as guidelines, the team can stay focused on achieving the goal of delivering a software release.

Technical principles

In this section, we provide a set of technical principles to consider when evaluating or designing a CALM solution.

Focus on the team's ability to produce a release of software

The traditional view has been to focus on a single role or domain. For example, teams focus on requirements for analysts, test management for testers, and source code control for developers. While this view has worked in the past, changes in the software market and in IT demand a change in focus.

The new focus involves a team of people working on a release, where the data produced by one individual acting in one role is needed and referenced by data produced by another individual acting in another role. Business proposals are evaluated in light of the overall portfolio. Projects are approved and funded. Business and system analysts collaborate to agree on a set of software requirements. These requirements drive the design, implementation and testing efforts. The results of the build are deployed onto test machines and eventually into production. All of this happens in the context of a release.

In addition, regulatory compliance initiatives require that organizations produce reports stating exactly what changed in the release of a solution, how it was tested and with what result, and where it is deployed. To answer these questions, all resources that are produced by the team must be managed in context of the release that they affect.

These activities must be planned, completed, and governed a coherent whole.

Use multiple repositories

While it makes sense to consolidate repositories where possible, it is also true that important project resources always exist in other repositories. The reality is that IBM clients have invested in solutions that house much of their data. Data about problems in production exist in the service desk and the IBM Tivoli Change and Configuration Management Database (CCMDB). Project funding decisions are managed by a portfolio management system, requirements, tests, and code. Defect data is likely to remain in a slew of different repositories, including our own repositories and those of third parties.

Therefore, you adopt a strategy that allows and expects data to be housed in multiple repositories. In doing so, you can adopt a strategy to open your own repositories to interoperate and share data freely without requiring multiple clients and user logins.

Processes link roles and access multiple repositories

It is relatively easy to define a state machine for a single resource. A test case has these states. A defect has a different set of states. However, process automation becomes more interesting when the process spans multiple roles and resources.

For example, the request to implement a new requirement can involve the following roles:

- An analyst does some form of analysis or modeling, such as modeling a business process.
- An architect examines the impact to the system and designs the approach.
- Developers implement the code that satisfies the requirement.
- A test team evaluates the available servers, configures them with the necessary software, creates a test plan, writes test cases, automates tests, and eventually runs tests, evaluates the results and logs the defects.
- The project manager determines how long it will take and ensures that all of the work is completed.

As shown in this example, a single requirement has the potential to span all roles on a development team. You can imagine a process that defines the implementation of a requirement with a multitude of subtasks.

Process definitions are not new. The Rational Unified Process (RUP) and the Open Unified Process (OpenUP) are industry leading process definitions. The game changes, however, when the process is inherently part of the team’s daily work. That is to say that the process guides the behavior of the team. Project plans and work assignments are created in the context of predefined process definitions. Both Rational Team Concert and Rational ClearQuest provide implementations of OpenUP. Project teams can choose and customize the process definitions for their project. As the team works in context of that project, the process definition guides their action.

Automate repetitive tasks

Every day development teams are tasked with manually performing mundane tasks. They have been doing many of these tasks for so long that they no longer seek ways to improve them. However, when they step back at look at some of the work that they must complete, development teams sometimes discover new approaches to completing the tasks.

For example, the act of preparing a machine for testing is something that testers face quite often. Tracking which machines are available and which software is installed on them can be a more complex task than one might expect. Testers can spend hours installing the operating system, middleware, test harnesses, and most recent build of the application. Yet in recent years, there have been advances for simplifying this task. IBM Tivoli Provisioning Manager has the ability to prepare a machine from the bare metal up. Virtual machines have advanced to the point where they are a normal part of the testing strategy. Rational Quality Manager provides functionality to enable test teams to begin managing their lab, thereby automating some of the mundane tasks that testers face most often.

Every time development teams encounter a mundane task instead of simply accepting it, they have the opportunity to step back and ask: “Is there a way to automate this?”

Link people and the assets they access

Development teams can link people and their assets in the following ways:

- Subscription and change broadcast services

  Syndicate content in repositories and allow users to subscribe to feeds. A multitude of information that users might want to obtain is available through feeds. Broadcast changes to resources in the repositories, and let the users decide to which information they want to subscribe.

- Team awareness

  Individual team members must learn to effectively change context as they move from one project to another, or from release to release. Individual team members need awareness about the team they are joining and its culture, policies, and best practices. They also need to know how to configure their environment for the project, how to share changes, and how the team as a whole is progressing. In essence, individuals need awareness on how they can fit in and contribute to the team's success.

- Rating systems and tags

  Use rating systems and tags to help you find the information the matters. Many management systems rely on a set of attribute/value pairs for classifying content. However, those attribute/value pairs are limited by the system or the people who create them. Each individual on the team might think of the content differently or want to build their own relationships between artifacts that are not applicable to the rest of the team.

  Bringing tagging techniques into software development repositories enables team members to tag content and surf tag clouds to discover new content. During the course of a software release and over the span of many releases, enormous amounts of data are produced. In this sea of data, it can become challenging to determine the inherent value to the current task. By allowing users to rate the resources in the repository, team members can create a shared view of the perceived quality or value of that resource.

B

Configuring interoperability

In this appendix, we provide additional details a bout configuring interoperability in the Rational Application Lifecycle Management solution (ALM) by using Rational ClearQuest, Rational Team Concert, and Rational Quality Manager. In particular, we examine the configuration of the Rational ClearQuest ALM schema, the ClearQuest Connector Gateway, and the ClearQuest Connector Synchronization rules.

The use of ALM interoperability integrates work assignment and iteration planning workflows across a team on the ClearQuest ALM solution and teams by using the Jazz platform tools or Rational Team Concert and Rational Quality Manager.

This appendix includes the following sections:

Configuring the ClearQuest ALM schema

In the following sections, we describe the steps to access and deploy the ClearQuest ALM schema.

ClearQuest and the ALM schema

Beginning with release 7.1.0.0, Rational ClearQuest provides a ready-to-use schema named "ALM" along with two new packages to help implement ALM in projects. The solution can be used as a new schema as provided by Rational ClearQuest or by adding the required packages to an existing schema.

The Rational ClearQuest schema that used in this IBM Redbooks publication was created by using the ALM schema that is provided in Rational ClearQuest and by using the OpenUP configuration that is provided as a sample with the schema. The ALM schema already has the main packages applied, which are necessary for the ALM scenario.

Packages for download: The ALM Packages for Rational ClearQuest are available for Rational ClearQuest 7.0.1 users to download for free from IBM. The packages can be accessed from the Web at the following address:

http://www.ibm.com/services/forms/preLogin.do?lang=en_US&source=swg-ratcq

Registration is required to access the packages. The download includes the ALM Packages for Rational ClearQuest, instructions for applying the packages, a sample database, and three tutorials for use with the sample database. The ClearQuest ALM solution can be used with both ClearQuest versions 7.0.1 and 7.1.

Using the ALM schema and sample database

The story and examples in this Redbooks publication use the ready-to-use schema named "ALM" that is provided with version 7.1.0.0 of Rational ClearQuest. The ALM schema extends the capabilities in Rational ClearQuest with ALM. Rational ClearQuest v7.1.0.0 include a preconfigured schema and sample data that greatly simplifies the configuration of the ALM schema and ALM interoperability.

The ClearQuest ALM solution provides rich capabilities for configuring the ALM process and workflows that are used by organizations, teams and projects. The ClearQuest ALM solution also provides ready-to-use process configurations in the form of samples. The story and examples in this Redbooks publication use the Open Unifies Process (OpenUP) work configuration that is provided with the ClearQuest ALM solution.

The ClearQuest ALM solution defines a configured process and workflows by using project definition records and ALM system-wide settings records. These record types provide the power to modify the ALM solution without impacting the underlying schema. The project definition records define the templates for projects, roles, and role-based workflows.

Furthermore, the system-wide settings can define categories of projects and label types that can help set policies for standardization, organization, and governance of projects. The Label record types provide the definition of names which appear in the user interface, most often in the form of drop-down lists on records. Some examples of Label types are Work Type, Role, Resolution Code, and Category Type. These settings allow for reuse and consistent classification across multiple projects, and can adapt to the enterprise. The ALM system-wide settings records are provided with the ALM schema sample database.

Creating a schema repository and sample database

To create schema repository including the ALM functionality:

1. Start the Rational ClearQuest Maintenance Tool and select **Schema Repository** → **Create**.
2. Select the **Vendor** database option. Click **Next**.

   More information: For detailed guidance about how to install the Rational ClearQuest repository on supported vendor database solutions, see the ClearQuest Information Center at the following address:

   `https://publib.boulder.ibm.com/infocenter/cqhelp/v7r1m0`

3. Select the Data Code Page to be used by ClearQuest.
4. On the Sample Database page, choose the ALM schema and select the option to create a sample database.

   Note: Use the default name **SAMPL** for the new ALM database base.

   Click **Next**.
5. Select the Vendor database option for the sample database. Click **Finish**. The new schema and sample databases are created.

You have now configured the schema, applied the required ALM packages, created a sample database, and imported the system-wide settings that are required for the ClearQuest ALM solution. If you successfully completed these steps, proceed to “Adding packages for interoperability to the ALM schema” on page 569.

In the following section, we explain how to configure a new or existing schema with ALM packages and system-wide settings data.

Adding packages for the ALM schema

To configure a new or an existing schema for ALM, you must apply the following two packages to the schema:

- ALMProject 1.0 (or later).
- ALMWork 1.0 (or later)

Installing the ALM packages

To install the ALM packages:

1. Download the ALM packages compressed file from IBM as explained in the note box on page 566.
2. Extract the contents of the compressed file into the \Program Files\Rational\ClearQuest\Packages directory.
3. Open a command prompt and register the ALMProject and ALMWork 1.0 packages by using the following commands

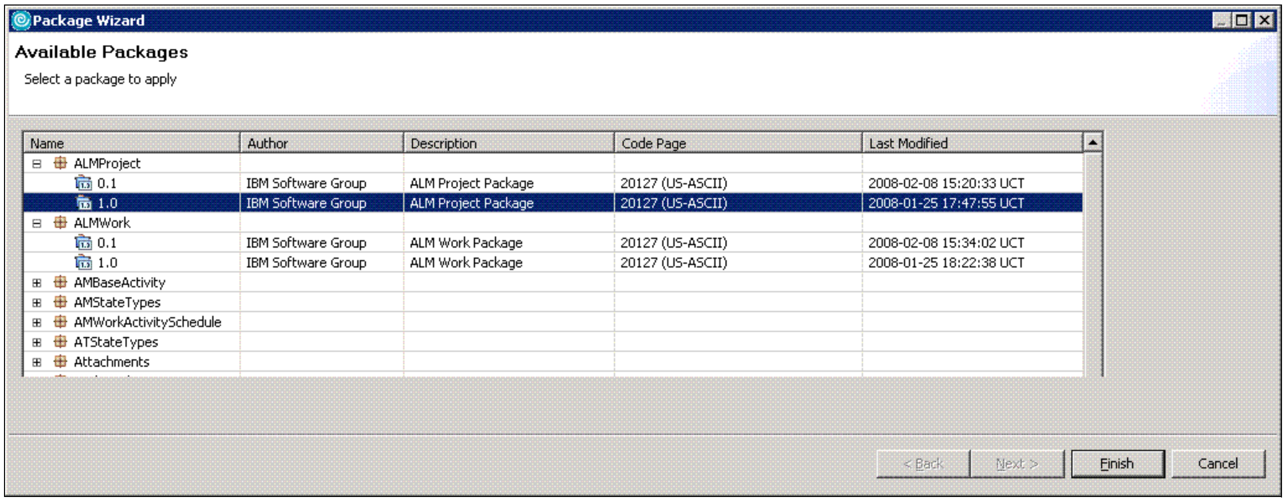
```
packageutil registerpackage ALMWork 1.0 "C:\Program
Files\Rational\ClearQuest\packages\ALMWork\1.0"

packageutil registerpackage ALMProject 1.0 "C:\Program
Files\Rational\ClearQuest\packages\ALMProject\1.0"
```

Path: The path that used in the previous commands might be different depending on the Rational ClearQuest installation directory.

Adding a package

To apply the ALM packages to a schema:

1. Open the ClearQuest Designer.
2. Browse the ClearQuest Schema Repository Explorer view for the schema to be modified, for example, ALM. Open the schema by logging in as an administrator.
3. Select the latest schema version (Figure B-1), and choose **Revision Control** → **Check Out**.



Name	Author	Description	Code Page	Last Modified
⊟ ALMProject				
0.1	IBM Software Group	ALM Project Package	20127 (US-ASCII)	2008-02-08 15:20:33 UCT
1.0	IBM Software Group	ALM Project Package	20127 (US-ASCII)	2008-01-25 17:47:55 UCT
⊟ ALMWork				
0.1	IBM Software Group	ALM Work Package	20127 (US-ASCII)	2008-02-08 15:34:02 UCT
1.0	IBM Software Group	ALM Work Package	20127 (US-ASCII)	2008-01-25 18:22:38 UCT
⊞ AMBaseActivity				
⊞ AMStateTypes				
⊞ AMWorkActivitySchedule				
⊞ ATStateTypes				
⊞ Attachments				

Figure B-1 Applying packages ALMWork and ALMProject

4. Select **Packages** → **Apply Package**.
5. In the Package Wizard, browse and select for the package version to apply. Click **Next**.
6. On the Apply Packages to Record Types page, optionally select the records types to which to apply the package. The ALM packages do not need to be applied to any record types. Click **Finish**.
7. Select **Team** → **Check In** to validate and check in the changes to the schema.

The ALMProject and ALMWork packages are now ready to be used by Rational ClearQuest.

Installing the ClearQuest ALM system-wide settings

The required system-wide settings are delivered as a sample database for the ALM schema. The records values can be imported by using the sample data files and the ClearQuest Import Tool.

To install the system-wide settings:

1. Browse for the sample directory in the ClearQuest installation directory, for example, C:\Program Files\Rational\ClearQuest\sample_db_files.
2. Open the schemas.ini file and scroll to the [ALM] section.

Note: The exact order of the files is given in the schemas.ini file and completes the successful import of all CSV files in the [ALM] section. Each line in the INI file represents one import of a record type and the file to import from. For example, the first line of `sampledata1=records,ALMAdmin,sample_alm_admin.csv` imports ALMAdmin records from the sample_alm_admin.csv file.

3. Select **Start** → **Rational ClearQuest Import Tool**.
4. In the import tool, select the schema connection to use for the import. Log into the database as administrator.
5. On the first page, select the record type to be imported. The record name, for example, ALMAdmin, is given in the INI file.
6. On the second page, browse to the CSV file to import from, for example \sample_db_files\sample_alm_admin.csv. The file name is given in the CSV file. Also, provide a path to a new temporary log file, for example `c:\temp\import.log`.
7. On the third page, click **Next**.
8. In the window that opens that prompts you for a field name to store unique identifiers, click **No**.
9. On the next page, click **Next**.
10. Click **Finish** to start the import.
11. Monitor the import summary error log.

Configuring ClearQuest ALM schema for interoperability

In this section, we explain how to configure the ClearQuest ALM schema for interoperability.

Adding packages for interoperability to the ALM schema

In addition to the packages that are required for ALM, additional packages must be applied to support interoperability between Rational ClearQuest, Rational Team Concert, and Rational Quality Manager. Add the following packages to the ALM schema that is to be used:

- Notes 5.1

  This package is optional, but its use is recommended for collaboration purposes. The Notes package enables interoperability with discussions in Rational Team Concert and Rational Quality Manager.

- JazzInterop 1.0 (or later)

  This package contains the required functionality for integration with the ClearQuest Connector Gateway and Rational Team Concert or Rational Quality Manager.

Follow the steps in “Adding a package” on page 568 to apply the packages to the ALM schema.

Restriction: There are limitations regarding the usage of the Notes package in ClearQuest MultiSite configurations. Consult the Rational ClearQuest documentation.

Applying the JazzInterop package

The JazzInterop package enables record synchronization between Rational Team Concert and Rational Quality Manager work items and Rational ClearQuest records. The JazzInterop package is required to be associated with all record types that are to be used in synchronization rules definitions.

ClearQuest ALM interoperability requires the JazzInterop package to be associated with the following record types:

- ALMActivity
- ALMTask
- ALMResolutionCodeLabel
- ALMTypeLabel
- ALMProject
- ALMCategory

Applying the Notes package

The Notes package enabling interoperability of in-context discussions between team members who are using ClearQuest ALM records and other team members who are using the synchronized Work Items in Rational Team Concert or Rational Quality Manager.

The Notes package should be associated with the following record types:

- ALMActivity
- ALMTask

Associating a package to record types

To associate a package:

1. Open the ClearQuest Designer.
2. Browse the ClearQuest Schema Repository Explorer view for the schema to be modified, for example, **ALM**. Open the schema by logging in as an administrator.
3. Select the latest schema version and select **Team** → **Check Out**.
4. Select **Packages** → **Setup Record Types for Package**.

5. In the Apply Package to Record Types window (Figure B-2), from the packages list, select a package, for example, **JazzInterop**, and select the desired record types from the list. Click **Finish**.

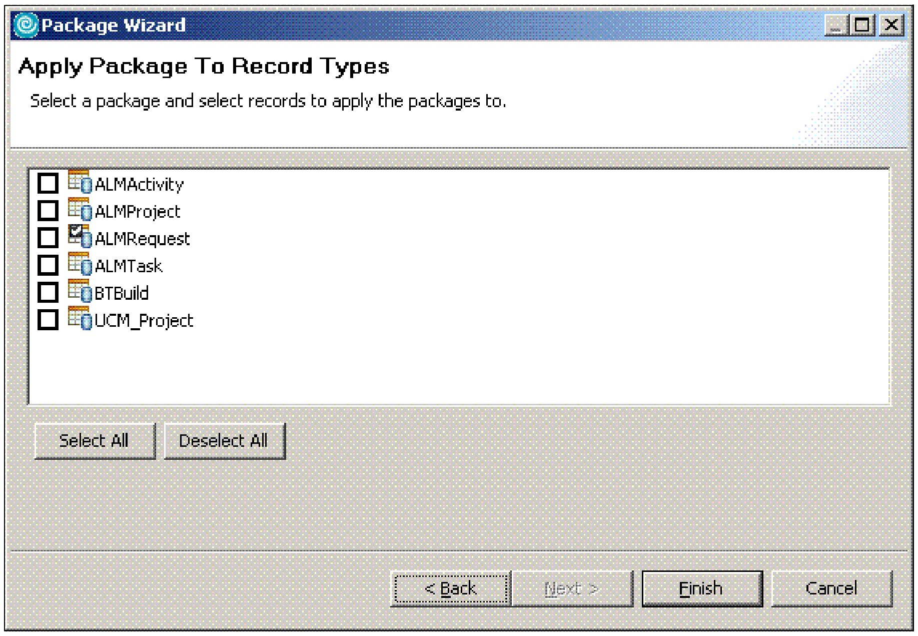


Figure B-2 Applying the JazzInterop package to ALM record types

6. Select **Team** → **Check In** to validate and check in the changes to the schema.

Configuring ClearQuest ALM system-wide settings for interoperability

The ClearQuest ALM solution comes with a set of system-wide settings that enable process, workflow, and terminology configurations without impacting the underlying schema. Tool administrators or project leads can set policies for standardization, organization, and governance of projects.

One of the configuration properties in the system-wide settings is Label record types. Label record types allows for the definition of the names that are displayed in the user interface, most often in the form of lists on records. Some examples of Label records types are Work Types, Roles, Resolution Codes, Category Types, Severities, and Priorities. These settings allow for consistent classification across multiple projects or the enterprise. ALM interoperability should be configured to allow for consistent classification across the integrated ALM solution.

When using the ClearQuest Connector to synchronize an ALM record and Work Item, field enumeration values, such as Priorities and ResolutionCodes, are required to be consistent across the tools. While the ClearQuest Connector provides mapping table functionality, such mappings are not supported for fields of reference type. A one-to-one (1:1) literal match is required to synchronize these fields.

The configuration that is documented in this section achieves consistent classification by extending the ClearQuest ALM system-wide settings. The following records are required to be added:

- ALMTypeLabel and ALMType for Work Item priorities
- ALMResolutionCodeLabels and ALMResolutionCodes for Work Item resolutions

It is also possible to extend the process definitions in Rational Team Concert and Rational Quality Manager. This alternate configuration is described in “Extended Jazz configurations” on page 614.

Creating new system-wide records for interoperability

To add system-wide records in ClearQuest ALM for interoperability work:

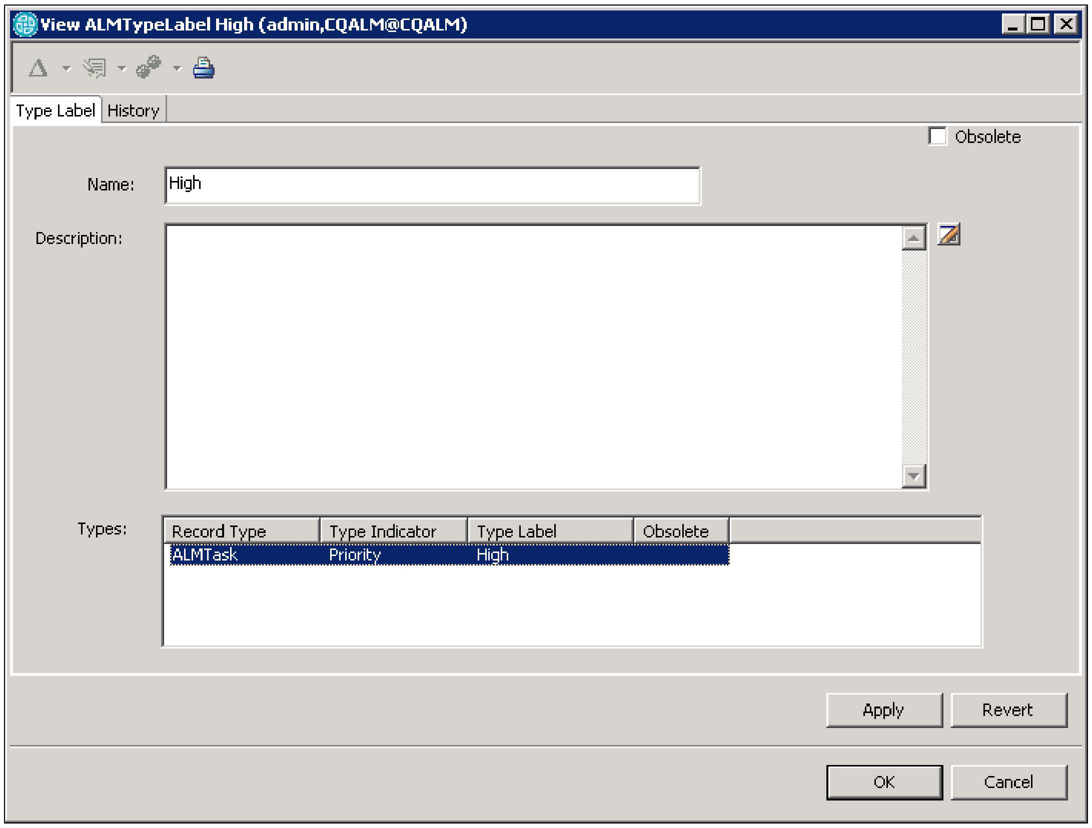
1. Open ClearQuest and log in to the ALM schema that is used as *admin*.
2. Select **File** → **New** and choose the ALM record type to be added, for example **ALMTypeLabel**, **ALMType**, **ALMResolutionCodeLabel**, or **ALMResolutionCode**.
3. Enter the values for the priority or resolution code to define the new system-wide setting.
4. Click **OK**.

Adding priorities

ALM records of type ALMTypeLabel are used to define the enumeration values for the ALMTask Property field. A set of new priority values is required to match the Rational Team Concert and Rational Quality Manager priorities (Figure B-3).

The following ALMTypeLabel types should be added:

- High
- Medium
- Low
- Unassigned


View ALMTypeLabel High (admin,CQALM@CQALM)

Type Label | History

Obsolete: ☐

Name: High

Description:

Types:

Record Type	Type Indicator	Type Label	Obsolete
ALMTask	Priority	High	

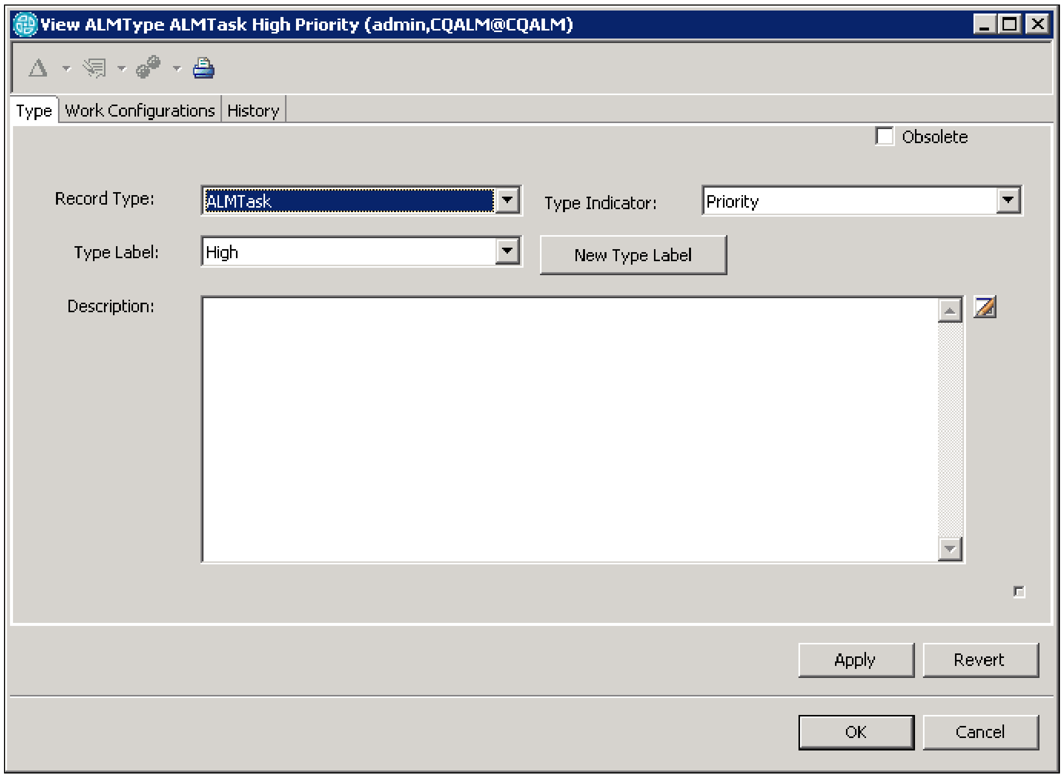
Figure B-3 Extending the system-wide settings with a new priority label for high priorities

Note: The Types list in the Figure B-3 is empty upon creation of the label. The list contains references to type instances that reference the label, for example, the High Priority type created in the following task.

ALM records of type ALMType are used to define the actual enumeration instance taking the name from the ALMTypeLabel. New ALMType records are required to be created for the High, Medium, Low, and Unassigned priorities (Figure B-4).

The following new ALMType records should be added with the following fields:

- High
  - ALMRecordType: **ALMTask**
  - TypeLabel: **High**
  - TypeIndicator: **Priority**
- Medium
  - ALMRecordType: **ALMTask**
  - TypeLabel: **Medium**
  - TypeIndicator: **Priority**
- Low
  - ALMRecordType: **ALMTask**
  - TypeLabel: **Low**
  - TypeIndicator: **Priority**
- Unassigned
  - ALMRecordType: **ALMTask**
  - TypeLabel: **Unassigned**
  - TypeIndicator: **Priority**


View ALMType ALMTask High Priority (admin,CQALM@CQALM)

Type | Work Configurations | History

☐ Obsolete

Record Type: ALMTask
Type Indicator: Priority
Type Label: High
Description:

Figure B-4 System-wide settings extended with a new high priority value

Removing original priorities

The original priority enumeration values cannot be used in combination with the ClearQuest interoperability configuration. For new ALM deployments, remove them from the database.

The following ALMTypeLabel and ALMTypes records can be removed:

- P1 - Urgent (removed or replaced with High)
- P2 - Important (removed or replaced with Medium)
- P4 - Low (removed or replaced with Low)
- P3 - Moderate (removed)

Important: ALMTypes records must be deleted prior to the ALMTypeLabel records.

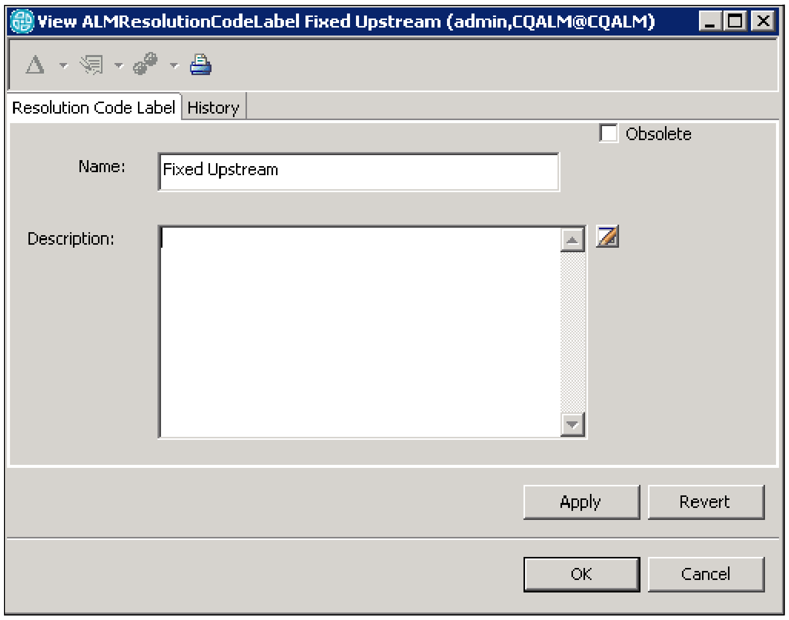
Adding resolution codes

The ClearQuest ALM solution required a resolution code to be given when resolving ALM activities, tasks, and requests. In Rational Team Concert, a resolution code is also provided when resolving a Work Item. To synchronize resolved Work Items to ClearQuest ALM records, the resolution codes in ClearQuest ALM must be extended to match the resolution codes for the Work Items.

The following resolution codes should be added:

- New ALMResolutionCodeLabel records with the following names:
  - Fixed Upstream
  - Invalid
  - Remind
  - Won’t Fix
  - Works for me
  - Later
  - Unresolved

  Figure B-5 shows a new resolution code label added.



Name: Fixed Upstream

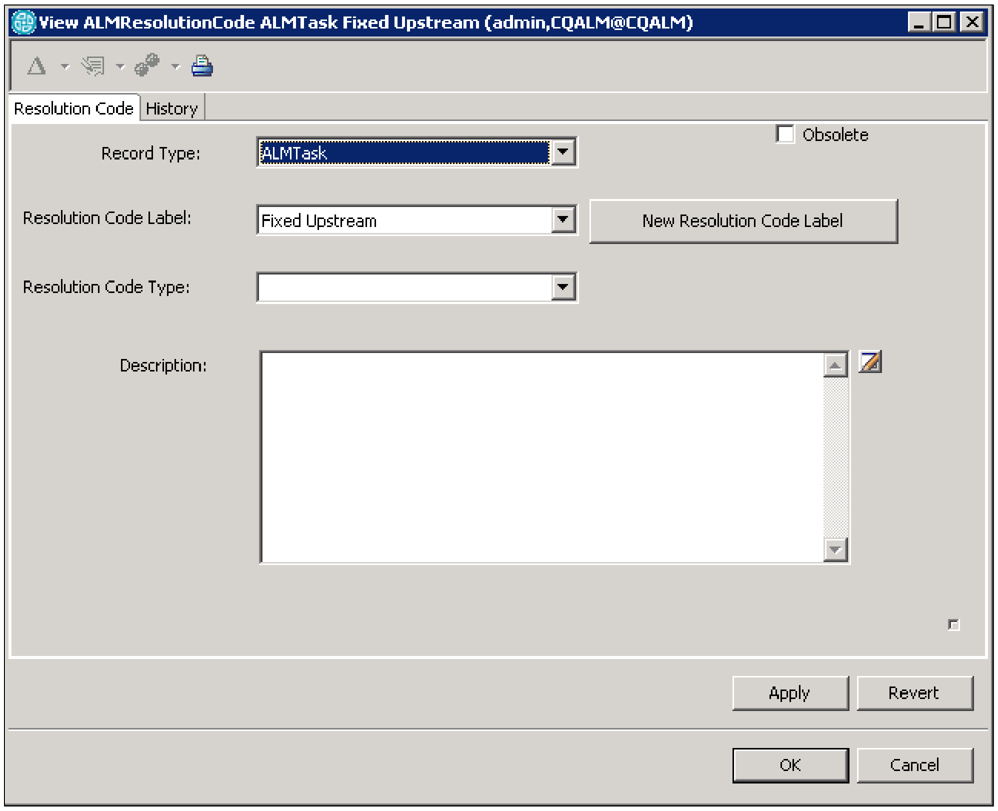
Figure B-5 Extending the system-wide settings with a new Resolution Code Label

- New ALMResolutionCode records for ALMTasks with the following names and settings:
  - Fixed Upstream
    - ALMRecordType: ALMTask
    - ResolutionCodeLabel: Fixed Upstream
  - Invalid
    - ALMRecordType: ALMTask
    - ResolutionCodeLabel: Invalid
  - Remind
    - ALMRecordType: ALMTask
    - ResolutionCodeLabel: Remind
  - Won't Fix
    - ALMRecordType: ALMTask
    - ResolutionCodeLabel: Won't Fix
  - Works for me
    - ALMRecordType: ALMTask
    - ResolutionCodeLabel: Works for me
  - Later
    - ALMRecordType: ALMTask
    - ResolutionCodeLabel: Later
  - Unresolved
    - ALMRecordType: ALMTask
    - ResolutionCodeLabel: Unresolved
- New ALMResolutionCode records for ALMActivities with the following names and settings:
  - Fixed Upstream
    - ALMRecordType: ALMActivity
    - ResolutionCodeLabel: Fixed Upstream
  - Invalid
    - ALMRecordType: ALMActivity
    - ResolutionCodeLabel: Invalid
  - Remind
    - ALMRecordType: ALMActivity
    - ResolutionCodeLabel: Remind
  - Won't Fix
    - ALMRecordType: ALMActivity
    - ResolutionCodeLabel: Won't Fix
  - Works for me
    - ALMRecordType: ALMActivity
    - ResolutionCodeLabel: Works for me
  - Later
    - ALMRecordType: ALMActivity
    - ResolutionCodeLabel: Later

- Unresolved
  - ALMRecordType: ALMActivity
  - ResolutionCodeLabel: Unresolved

Note: Each ALM record type must define its individual Resolution Codes values. ClearQuest interoperability synchronizes both ALMTasks and ALMActivities to Jazz Work Items. The extended Resolution Codes values are required for both ALM types.

Figure B-6 shows a new resolution code added for ALMTasks.


View ALMResolutionCode ALMTask Fixed Upstream (admin,CQALM@CQALM)

Resolution Code | History

Record Type: ALMTask

Obsolete: ☐

Resolution Code Label: Fixed Upstream

Resolution Code Type:

Description:

Figure B-6 System-wide settings extended with a new Fixed Upstream resolution value

Adding optional resolution codes

Resolution codes can optionally be used in approval and sizing workflows in ClearQuest ALM. The sizing scenario is briefly discussed in 4.3.2, “Sizing requests” on page 113. The approval scenario is discussed in “Patricia approves the release” on page 508.

To extend the system-wide settings to support approvals and sizing, add the following resolution code labels and resolution codes to both ALMTask and ALMActivitiy records:

- Approved
- Containable
- Not Containable

Note: For approvals, use the resolution codes of Approved and Rejected. The Rejected enumeration does not need to be added because it is already present in the list of enumerations.

Configuring users

Interoperability between Rational ClearQuest, Rational Team Concert, and Rational Quality Manager requires that users who are assigned work or are participating in discussions have user records in these repositories. Users in Rational ClearQuest must be added manually. Users in Rational Team Concert and Rational Quality Manager can be created using a synchronization rule for the ClearQuest Connector.

The ALMTask and ALMActivity records, participating in synchronization with Work Items, are selected by the ClearQuest Gateway by using specified selection queries. The filter parameters for the selection queries often use ClearQuest groups to identify records to be synchronized. For example, a ClearQuest group can be used to identify a team that is using Rational Team Concert or Rational Quality Manager as their prime development or test environment. When adding Rational Team Concert users to Rational ClearQuest, consider creating a group that can be used for such user identification. Note that all users in this group must be mastered at the same site where the ClearQuest Gateway is located if MultiSite is used.

The ClearQuest ALM solution uses role-based workflows. New users are assigned project roles by using the project record.

Adding ClearQuest users

To add a user to ClearQuest:

1. Open the ClearQuest User Administration tool and log in as an administrator.
2. To add a group, select **Group Action** → **Add Group**. Type the name of the group and click **OK**.
3. To add a User, select **User Action** → **Add User**. Enter the user information. Make sure to type the e-mail address because that field is used to link Rational ClearQuest and Rational Team Concert users. Assign the user to the appropriate groups. Click **OK**.
4. To add the users to the database, select **DB Action** → **Upgrade**. Select the databases to upgrade and click **OK**.
5. Close the ClearQuest User Administration tool.

Configuring ALM roles

To add users to the project roles:

1. Open the current project, for example by running the **Public Queries** → **ALM** → **General** → **All Project**. Open the project record.
2. Switch to the **Team Members** tab.
3. From the Roles and Members list, double-click a name to open a role.
4. In the opened ALMRole record, switch to the **Members** tab.
5. Click the **Modify** button to make the record editable.
6. Click the **Add** button and search for new users (project team members) to add to the role.
7. Click **Apply** to save the changes.

Tip: Adding a user name of Unassigned creates significant flexibility in binding the role and primary owner in the work configurations. The Unassigned user is just a placeholder that replaces the empty string value for Owners that is not allowed in the ClearQuest ALM solution.

Configuring filter queries

The ClearQuest Connector uses Rational ClearQuest queries to filter and select the records to synchronize with work items in the Rational Team Concert and Rational Quality Manager repositories. These filter queries are stored in a structured hierarchy of query folders in the Public Queries or Personal Queries folder in Rational ClearQuest. The query folder names in the folder hierarchy determine which Rational Team Concert and Rational Quality Manager project areas the work items are to be synchronized.

To configure the filter queries:

1. Consider which ALM projects, categories, and releases should be synchronized with which Jazz repositories and project areas. This is input is required for the next steps.
2. Create a hierarchy of query folders in Rational ClearQuest with folder names that match the Jazz project areas of the repositories to synchronize.
3. Create one or more filter queries for each ClearQuest ALM record type to be synchronized. Repeat this step for all project area query folders.
4. Add the root path to the hierarchy of query folders in Rational ClearQuest in the cqconnector.properties file.

Determining which ALM assets to synchronize

The first step when configuring the selection query is to determine which records must be synchronized and which repositories to synchronize with. In most cases, this highly depends on how the projects and teams are organized and on the ALM tools used by the project teams.

As a general practice, consider the following points:

- Query by record type. Only record types that are setup for synchronization can be included, see “Applying the JazzInterop package” on page 570. For the ClearQuest ALM schema, ALMTask and ALMActivity should be synchronized.
- Filter by ALMProject. This limits the synchronization to work only in current projects.
- Filter by the ALMCategory. This limits the synchronization to all work, regardless of the project, to a specific part of the application.
- Filter by Groups or Users. This limits the synchronization only to work that is assigned to selected users or specific teams.
- Filter by ALMType. This limits the synchronization to a specific discipline or workflow.

For the example used in this book, we took the following considerations for the design of the filter queries:

- The CreditCheck component team is using Rational Team Concert and must synchronize ClearQuest ALMTasks and ALMActivities in order to integrate project iteration planning and work assignment workflows. The records should be assigned to a team member and planned for the current project.
- The Account Opening test team is using Rational Quality Manager and must synchronize ClearQuest ALMTasks and ALMActivities in order to test planning and assignment workflows. The records should be assigned to a team member and planned for the current project.
- The change management triage workflow, performed by the project lead, ensures that tasks are assigned to the appropriate teams and team members.

- Both teams have user groups, AO_Regional and AO_Test, defined in Rational ClearQuest. Basing filter queries on these groups makes the maintenance simpler and more stabile to team member changes.

> **Note:** The scenario in this book does not separate the test management repositories that are used by the enterprise test team and the external third-party solution testing team. By using filter queries based on ClearQuest groups, projects and categories can be effectively used to synchronize the desired subset of ALM assets to a third-party contributor.

Based on the filter requirements, the following filter queries were configured. Also, see Figure B-7 on page 580.

- Filter queries for the CreditCheck component team:
  - Owner.groups = AO_Regional
  - (optional) Project.Name = AO_Rel2
  - (optional) Project.Category = CreditCheck
- Filter queries for the Account Opening test team:
  - Owner.groups = AO_Testers and AO_3rd_Party
  - Optional: Project.Name = AO_Rel2

> **Tip:** The use of record IDs, rather than name fields, provides a more robust configuration that is resilient to name changes.

Configuring the hierarchy of query folders

To create the hierarchy of query folders in the Public Queries:

1. Open ClearQuest and log in as an administrator.
2. In the ClearQuest Navigator view, right-click **Public Queries** and select **New Folder**.
3. Right-click the **New Folder** and select **Rename**. Name the folder `CQ Connector`.
4. Select the new ClearQuest Connector folder and create two new subfolders. Name the folders `AccountOpening` and `Quality Manager`.

> **Important:** The names of the subfolders must match the names of the project areas in the Rational Team Concert and Rational Quality Manager repositories. Create one subfolder for each individual project area.

Configuring the filter queries

To add a new selection query in Rational ClearQuest:

1. Right-click **Public Queries** and select **CQ Connector** → **AccountOpening**. Choose **New Query**.
2. Type a name for the query, for example `CreditCheck.ALMTask`. Select the **ALMTask** type. Click **Next**.
3. Select filters for the query:
   a. Click the plus icon next to the Owner field and drag the **Groups** field subitem to the filter pane.
   b. Click and drag the **Project** field to the filter pane.
   c. Click the plus icon next to the Project field and drag the **Category** field subitem to the filter pane.

d. Right-click **Project** and select **Use Project**.

e. Right-click **Project.Category** and select **Group Project with Project.Category**.

f. Right-click the top most filter and select **Or**. Then click **Next**.

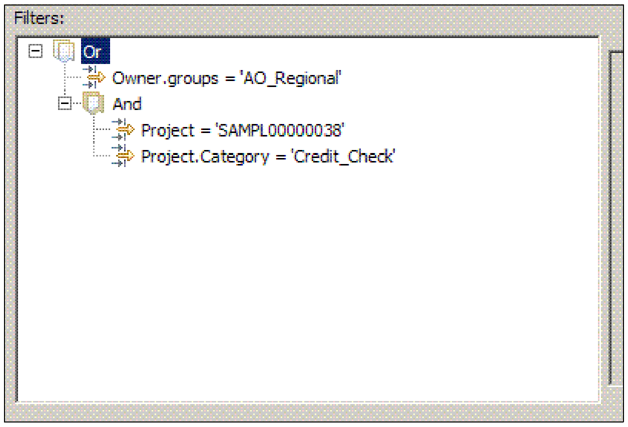
g. Click the plus icon next to the **Or** grouping to expand the filter list. Then click the **Owner.groups** filter (Figure B-7).

h. In the Define Filters pane, click the **Values** button.

i. Select the appropriate groups with members of the team that will use Rational Team Concert and click **OK**.

j. Repeat steps h and i for the two additional filters of *Project* and *Project.Category* and click **Next**.

k. Optional: Select any display fields, although they are ignored by the connector. Click **Finish**.


Filters:

```
Or
  Owner.groups = 'AO_Regional'
  And
    Project = 'SAMPL00000038'
    Project.Category = 'Credit_Check'
```

Figure B-7 Rational ClearQuest query for Rational Team Concert Interop Synchronization

4. Repeat the first three steps to create a new query named *CreditCheck.ALMActivity* in the AccountOpening query folder. Base the query on the ALMActivity record type, and use the same query filter definition.

5. Proceed and create *Test.ALMTask* and *Test.ALMActivity* filter queries in the Quality Manager query folder. For the Owner.groups filter, select the values **AO_Testers** and **AO_3rd_Party** as group names.

Tip: Before proceeding, run your new query to validate that the return result set contains all, but only, the ALM Activity records that you want to synchronize with Rational Team Concert.

Updating the root path to the filter queries

The root path to the hierarchy of the query folders in Rational ClearQuest must be specified in the cqconnector.properties file:

1. Open the cqconnector.properties file in the IBM\ClearQuestConnector\gateway directory.
2. Edit or add the following cq.queryTreeRoot statement:

   ```
   cq.queryTreeRoot=Public Queries/CQConnector
   ```

3. Save the cqconnector.properties file.

Configuring the ClearQuest Gateway

The ClearQuest Gateway is a dedicated server that performs synchronization between the Rational ClearQuest, Rational Team Concert, and Rational Quality Manager repositories. Because the ClearQuest Gateway connects to the Rational ClearQuest repository as a regular ClearQuest client, Rational ClearQuest must be installed and licensed on the server that is running the ClearQuest Gateway. The ClearQuest Gateway and the Rational ClearQuest database must also be co-located and LAN connected.

Configuring of the ClearQuest Gateway involves the following tasks:

- Installing the ClearQuest Gateway server
- Setting the CQ_Home environment variable
- Creating and configuring the cqconnector users
- Configuring the cqconnector.properties file
- Starting the ClearQuest Gateway

For further documentation and instructions about installing the ClearQuest Gateway, see the following Web site or the Rational Team Concert help system provided with the product:

`http://jazz.net`

Installing the ClearQuest Gateway

The ClearQuest Connectors are installed from the Rational Team Concert Launchpad. To install ClearQuest Connectors:

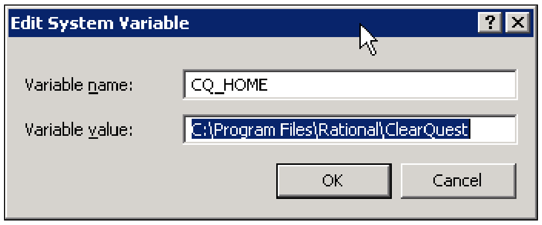
1. Run the Rational Team Concert Launchpad.
2. Under the Install Optional Products section, select **Rational Team Concert - ClearQuest Connector**.
3. Follow the instructions in the IBM Installation Manager.

Setting the CQ_Home environment variable

The ClearQuest Gateway depends on libraries that are provided in the Rational ClearQuest client installation directory and uses the CQ_Home Windows environment variable to locate the files.

To configure the CQ_Home Windows environment variable:

1. Open **Start** → **Control Panel** → **System**.
2. Click the **Advanced** tab.
3. Click the **Environment Variables** button.
4. In the System Variables section, click **New**.
5. Create a new variable named `CQ_HOME` with the path to the Rational ClearQuest installation directory, for example, `C:\Program Files\Rational\ClearQuest` (Figure B-8).
6. Click **OK** to close the window.



Variable name: CQ_HOME

Variable value: C:\Program Files\Rational\ClearQuest

Figure B-8 CQ_Home Windows environment variable

Creating a cqconnector user in Rational ClearQuest

The ClearQuest Gateway logs into Rational ClearQuest to access record information. A recommended practice is to create a dedicated user for the ClearQuest Gateway. Follow the steps in “Adding ClearQuest users” on page 577 to add a new Rational ClearQuest user named “cqconnector” for the ClearQuest Gateway.

Note that the cqconnector requires that you set SQL Editor privileges. Also note that the cqgateway user must be given sufficient ALM security privileges to access the required records and fields. For a simple setup, assign the cqgateway to the ALM admin role.

Creating a cqconnector user in Rational Team Concert

The ClearQuest Gateway logs into Rational Team Concert to access and update repository information. A recommended practice is to create a dedicated user for the ClearQuest Gateway to use when logging into Rational Team Concert. Follow the steps in “Configuring a cqconnector user in Rational Team Concert” on page 584 to add a new Rational Team Concert user named *cqconnector*.

Note that all changes to items that result from changes to corresponding ClearQuest records are saved in the context of the cqconnector user account. Recording all changes under the same user account has disadvantages in work item histories and regarding security. A recommended practice is to use the External Modifier property in the synchronization rules to avoid these limitations. The synchronization rules in “Configuring and deploying synchronization rules” on page 585 are configured by using this setting.

Optional: Creating a cqconnector user in Rational Quality Manager

The ClearQuest Gateway logs into Rational Quality Manager to access record information. A recommended practice is to create a dedicated user for the ClearQuest Gateway. Follow the steps in “Configuring the cqconnector user in Rational Quality Manager” on page 585 to add a new user named *cqconnector*.

Note the discussion in the previous section regarding the External Modifier synchronization rule property.

Optional: Creating optional Rational Team Concert and Rational Query Manager users in the ClearQuest Gateway

The outgoing synchronization processes in Rational Team Concert and Rational Quality Manager logs into the ClearQuest Gateway for synchronization. By default, the Rational Team Concert and Rational Quality Manager servers use the ADMIN account. Optionally, you can create one or more users in the ClearQuest Gateway for the Rational Team Concert and Rational Quality Manager server logins.

To create additional users in ClearQuest Gateway:

1. Open the **tomcat-users.xml** file from the IBM\ClearQuestConnector\gateway\tomcat\conf directory.
2. Add or edit the user definitions as follows:

   ```
   <user username="rtc" password="rct" roles="CQConnector" />
   <user username="rqm" password="rqm" roles="CQConnector" />
   ```

3. Save the file and restart the ClearQuest Connector server.

Attention: Secure passwords should be provided when configuring new ClearQuest Gateway users.

Configuring cqconnector.properties file

To configure the cqconnector.properties file:

1. Locate the IBM\ClearQuestConnector\gateway\cqconnector.properties file.
2. Open the file by using a text editor.
3. Edit the file and provide information regarding the Rational ClearQuest environment as shown in Example B-1. The example uses the following settings:
   - The `CQALM` is the dbset name, and `SAMPL` is the logical database name.
   - The ClearQuest Gateway uses the default `cqconnector` user account to log in and synchronize ClearQuest records.
   - The ClearQuest Gateway uses the default `cqconnector` password. A recommended practice is to replace this default password with a secure password.
   - The ClearQuest Gateway connects to both the Rational Team Concert and Rational Quality Manager repositories. The URIs are separated with a semicolon (;) character. Make sure to give the correct IP addresses and port numbers for the repository servers. In this example, all servers are running on the local host. Note that port numbers must be unique per server.
   - The query uses the queries in the Public Queries/CQ Connector query.

   Example: B-1 cqconnector.properties file

   ```
   com.ibm.rational.interop.pollingPeriod=10
   cq.dbSetDbName=CQALM/SAMPL
   cq.userid=cqconnector
   cq.password=cqconnector
   com.ibm.team.uris=https://cqconnector:cqconnector@localhost:9443/jazz;https://c
   qconnector:cqconnector@localhost:9444/jazz
   cq.queryTreeRoot=Public Queries/CQConnector
   ```

4. Save the changes.

Starting the ClearQuest Gateway

To start and stop the ClearQuest Gateway server, use the **`server.startup.bat`** and **`server.shutdown.bat`** commands in the \IBM\ClearQuestConnector\gateway folder.

Important: The ClearQuest Gateway requires access to the connected repositories when started. Make sure that the Rational Team Concert and Rational Quality Manager repository servers are started prior to starting the ClearQuest Gateway.

In addition, you are required to restart the ClearQuest Gateway after you modify the ALM schema, upgrade the ClearQuest database to a new schema version, or add and remove filter queries.

Configuring Jazz repositories for interoperability

In this section, we provide details about how to configure the Jazz repositories for interoperability. The configuration of the Jazz repositories is based on the assumption, taken for this book, that the OpenUP template is used in Rational Team Concert and in Rational ClearQuest.

Configuring users and licenses

The ClearQuest Gateway, Rational Team Concert, and Rational Quality Manager require user accounts to access repository information. In this section, we describe how to configure the cqconnector users.

Licenses to administer or develop external connections

The ClearQuest Connector functionality is licensed with the Standard Edition of Rational Team Concert v 1.0. To work with external connections, a developer or tester license is required.

To manage licenses:

1. Assign developer or tester licenses to any users that need to administer or develop external connections.
2. Assign a ClearQuest Connector license to the user account that is used by the ClearQuest Gateway, that is the cqconnector user account.

Configuring a cqconnector user in Rational Team Concert

The ClearQuest Gateway logs into Rational Team Concert to access repository information. A recommended practice is to create a dedicated user for the ClearQuest Gateway.

To create a new cqconnector user:

1. Open a browser window and go to the Jazz server Admin Web UI:

   ```
   https://localhost:9443/jazz/admin
   ```

2. Log in as the ADMIN administrator user.
3. In the Admin Web UI, click the **User Management** button on the navigation bar. Go to the Active Users section.
4. To create a new cqconnector user, click the **Create User** button and fill in the cqconnector account details as follows (Figure B-9 on page 585):
   – For UserId, type `cqconnector`.
   – For E-mail address, type `cqconnector`.
   – For repository permissions, select **Jazz users**.
   – For Client Access Licenses, select **ClearQuest Connector** or **Rational Team Concert - Developer** license.
5. Save the new user.
6. Open the tomcat-users.xml file from the IBM\JazzTeamServer\server\tomcat\conf directory and validate that cqconnector is defined. If this user is not defined, add a new user definition:

   ```
   <user username="cqconnector" password="cqconnector" fullName="cqconnector"
   roles="JazzUsers"/>
   ```

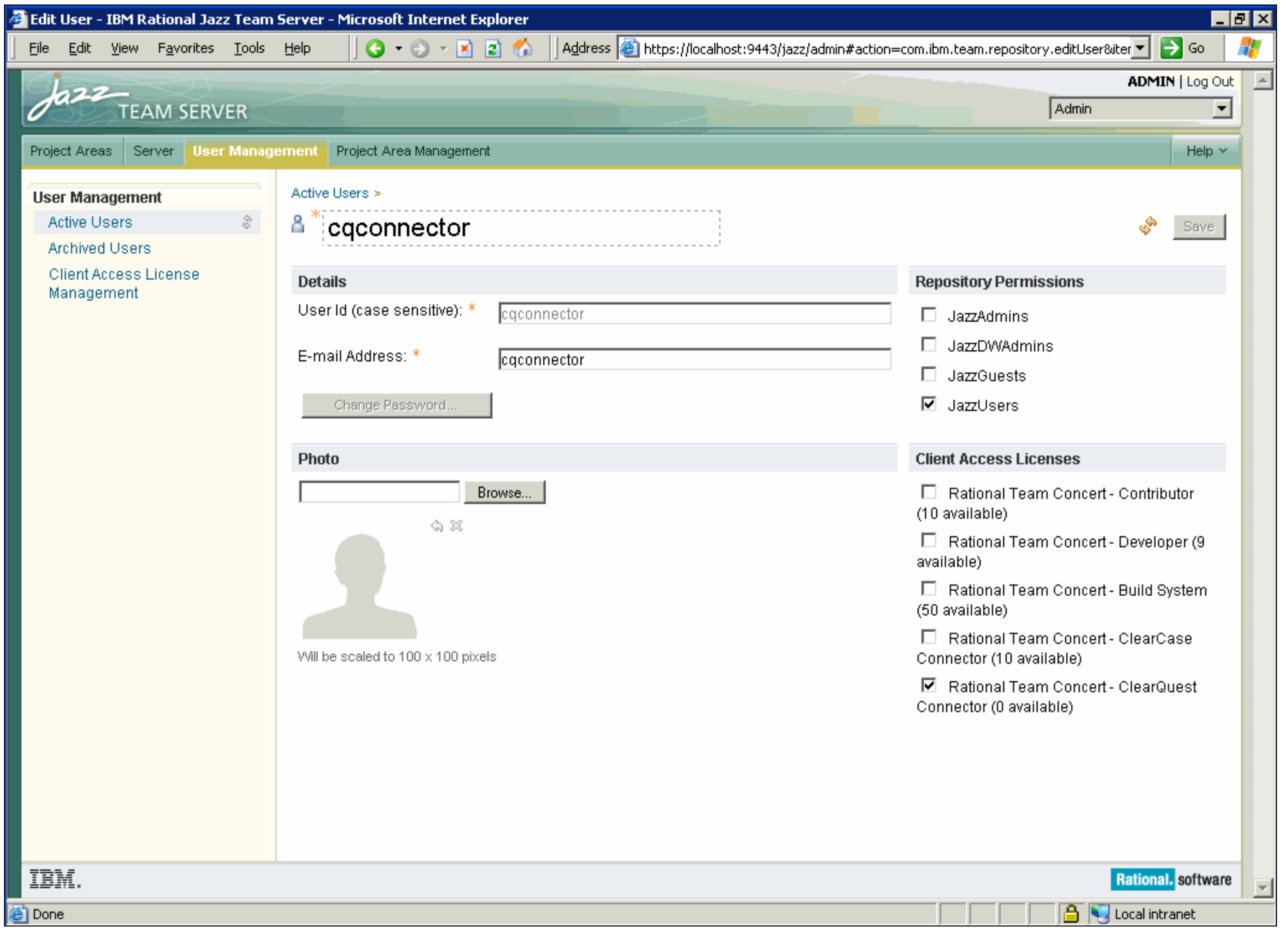
   Note: Secure passwords should be provided when configuring the new cqconnector user.

7. Save the file
8. Open a Command Prompt window.

9. Run the **`repotools.bat`** command from the IBM\JazzTeamServer\server directory to encrypt passwords:

```
repotools -convertTomcatUsers tomcatUsersPath=tomcat/conf/tomcat-users.xml
```

10. Rename the file from tomcat-users.xml.converted to `tomcat-users.xml`.


Edit User - IBM Rational Jazz Team Server - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address https://localhost:9443/jazz/admin#action=com.ibm.team.repository.editUser&iter
Go
TEAM SERVER
ADMIN | Log Out
Admin
Project Areas
Server
User Management
Project Area Management
Help
User Management
Active Users
Archived Users
Client Access License Management
Active Users >
Save
Change Password...
Photo
Browse...
Will be scaled to 100 x 100 pixels
IBM.
Rational. software
Done
Local intranet

cqconnector

Details	
User Id (case sensitive): *	cqconnector
E-mail Address: *	cqconnector

Repository Permissions

- [ ] JazzAdmins
- [ ] JazzDWAdmins
- [ ] JazzGuests
- [✓] JazzUsers

Client Access Licenses

- [ ] Rational Team Concert - Contributor (10 available)
- [ ] Rational Team Concert - Developer (9 available)
- [ ] Rational Team Concert - Build System (50 available)
- [ ] Rational Team Concert - ClearCase Connector (10 available)
- [✓] Rational Team Concert - ClearQuest Connector (0 available)

Figure B-9 Adding the cqconnector user to Rational Team Concert

Configuring the cqconnector user in Rational Quality Manager

The ClearQuest Gateway logs into Rational Quality Manager to access record information. A recommended practice is to create a dedicated user for the ClearQuest Gateway.

To create a new cqconnector user, follow the steps in “Configuring a cqconnector user in Rational Team Concert” on page 584.

Note: When assigning client access licenses for the ‘cqconnector’, use a ClearQuest Connector license or a Rational Quality Manager - Tester license.

Configuring and deploying synchronization rules

The details of the synchronization of the Rational ClearQuest, Rational Team Concert, and Rational Quality Manager repositories are specified in the ClearQuest Connector synchronization rules. These rules specify how ClearQuest records and Jazz Work Items are paired, which fields to synchronize, and any specific field value mapping to use. The rules also specify the type of extended transformation that is provided by synchronization managers on the ClearQuest Gateway.

The main synchronization in this example is made with ALMTasks and ALMActivities synchronized with Work Items. New ALMTasks and ALMActivities that are created as part of the iteration planning are created in the connected Jazz repositories. Note that new Work Items created in Rational Team Concert or Rational Quality Manager are not synchronized to Rational ClearQuest. Such items are considered local to the component or practice team.

The synchronization uses a simple mapping of types. By default, ALMTasks are synchronized as Work Items of type Task. ALMActivities are synchronized as Work Items of type Defect. A synchronization rule is defined for each record. Synchronization of the record fields requires additional helper rules. A more generic type of mapping for ALMActivities to a Activity Work Items type is possible as an alternate configuration.

Configuring and deploying synchronization rules entails the following tasks:

- Creating new repository connections
- Creating new or import existing synchronization rules
- Configuring the Jazz server interoperability properties
- Enabling interoperability permissions

The Rational Team Concert client is required to configure new repository connections to Jazz repositories and to configure synchronization rules. Administrators of Rational Team Concert and Rational Quality Manager servers require the Rational Team Concert client to perform the configuration steps.

For further documentation and instructions about creating synchronization rules, see the following Web site or the Rational Team Concert and Rational Quality Manager help system provided with the product:

`http://jazz.net`

Creating a repository connection

To create a new ClearQuest Gateway connection:

1. Open Rational Team Concert and log in as administrator.
2. In the Team Artifacts view, select the repository connection from the Repository Connections item.

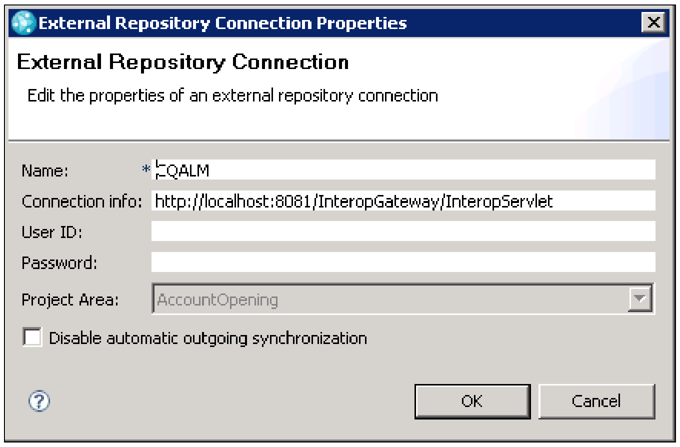
   Note: If no repository connection has been established, use the New command on the Repository Connections item to create a new connection to a Rational Team Concert or Rational Quality Manager repository.

3. Select **Repository Connections** → **Administer** → **Synchronization Rules**.
4. In the Synchronization Rules view, select the **External Repository Connection**. Choose **New** → **External Repository Connection**.
5. Enter the details for the connection:
   - For Name, type `CQALM`.
   - For Connection for, type `http://`*`localhost`*`:8081/InteropGateway`.

   localhost: Replace *localhost* with the name or IP address of the server running the ClearQuest Connector Gateway.

6. Click **OK** to save the changes.
7. Repeat steps 4 on page 586 through 6 to create external repository connections to additional repositories. In the example in this book, we configured connections to both the Rational Team Concert and Rational Quality Manager repositories. Give each connection a unique name.

Figure B-10 shows an example of creating an external repository connection.


External Repository Connection Properties
External Repository Connection
Edit the properties of an external repository connection

Name: * CQALM
Connection info: http://localhost:8081/InteropGateway/InteropServlet
User ID:
Password:
Project Area: AccountOpening
☐ Disable automatic outgoing synchronization

Figure B-10 Creating an external repository connection

Importing existing synchronization rules

The following synchronization rules are provided in the example in this book for interoperability between Rational ClearQuest and Rational Team Concert:

- com.ibm.rational.clearquest.CQALM.ALMTask

  This rule synchronizes ClearQuest ALMTasks and Jazz repository Work Items.
- com.ibm.rational.clearquest.CQALM.ALMActivity

  This rule synchronizes ClearQuest ALMActivities and Jazz repository Work Items.
- com.ibm.rational.clearquest.CQALM.ALMTypeLabel

  This rule is a helper rule to synchronize the Priority reference field.
- com.ibm.rational.clearquest.CQALM.ALMResolutionCodeLabel

  This rule is a helper rule to synchronize the ResolutionCode reference field.
- com.ibm.rational.clearquest.CQALM.Attachments

  This rule manages the creation and synchronization of attachments in Rational ClearQuest and Rational Team Concert.
- com.ibm.rational.clearquest.CQALM.users

  This rule manages the creation of new contributors in Rational Team Concert from user records in Rational ClearQuest.
- com.ibm.rational.clearquest.CQALM.ALMProject

  This rule is a helper rule to synchronize the Project reference field.
- com.ibm.rational.clearquest.CQALM.ALMCategory

  This rule is a helper rule to synchronize the Project.Category reference field.

To import an existing synchronization rule:

1. In the Synchronization Rules view, select the Project Area name, for example, **AccountOpening**, and select **Import Synchronization Rules**.
2. Select the **Use existing external repository connection** option and click **Finish**.
3. Browse the file system for synchronization rules to import. Select one or more synchronization rules and click **Open** to import.
4. Expand the project area folder and open the imported synchronization rules in the Synchronization Rules view.
5. Optional: Edit the Name field to match the ClearQuest name of your dbset, for example `com.ibm.rational.clearquest.`*`myALM`*`.ALMTask`.
6. Click the **Save** button to save the changes.

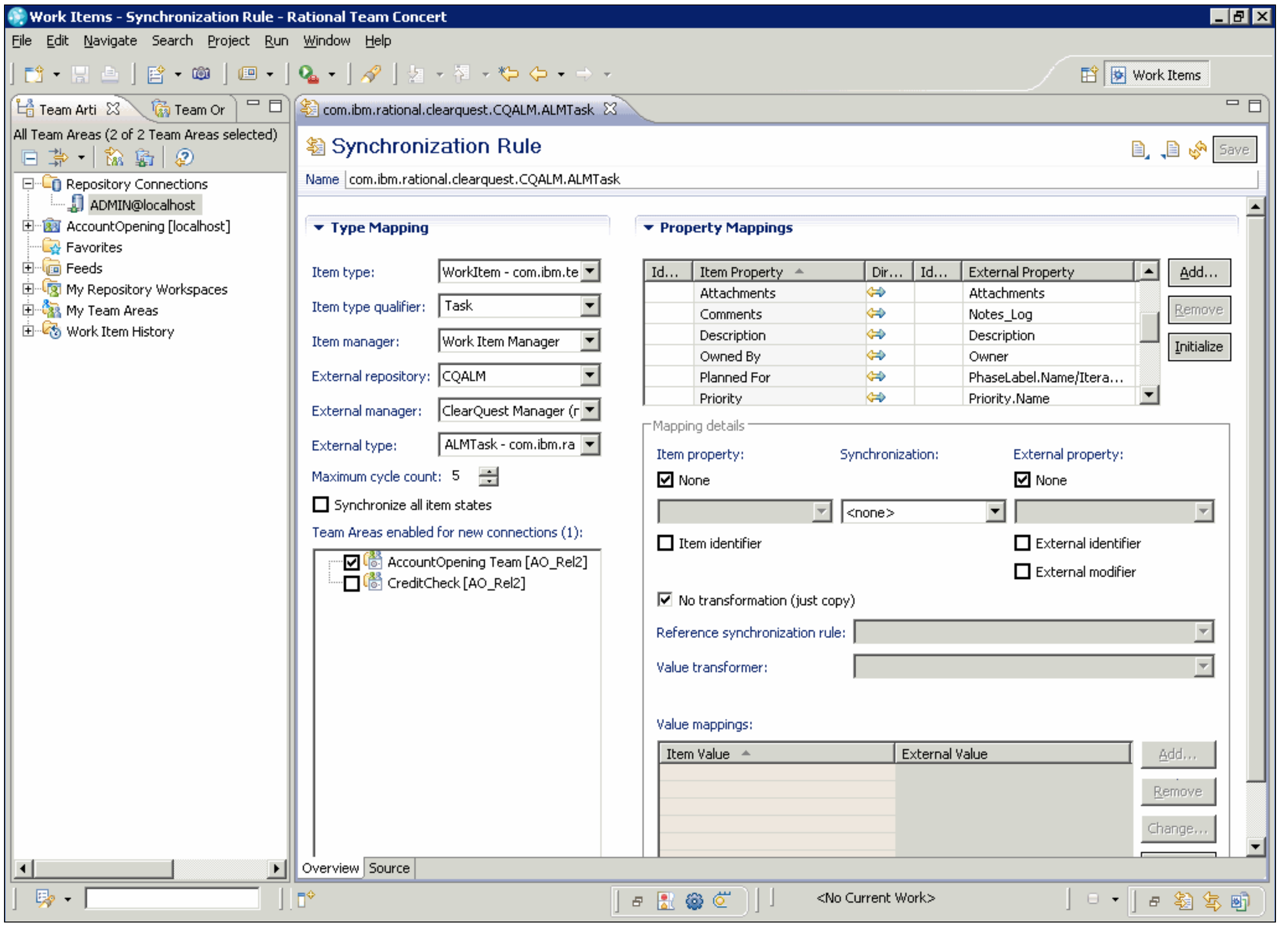
Attention: Only new synchronization rules can be imported. If the synchronization rule name already exists for your external connection, then import is not allowed. Use care when deleting and re-importing synchronization rules because this might corrupt your repository data. Deleting a synchronization rule deletes all of the connector information for all items that used that synchronization rule and all connected items become disconnected. For practices on how to update an existing synchronization rule, see "Importing existing synchronization rules" on page 587.

Create a new synchronization rule

To create a new synchronization rule:

1. In the Synchronization Rules view, select the Project Area name, for example **AccountOpening**, and select **New** → **Synchronization Rule**.
2. Type a name for the new synchronization rule. Use a pattern to indicate the name of the record type to be synchronized, for example `com.ibm.rational.clearquest.CQALM.ALMTask`.
3. Define the mapping for the synchronization rule by using the Synchronization Rules editor.
4. Click the **Save** button to save the new synchronization rule.

Figure B-10 shows the synchronization rule editor in Rational Team Concert.


Work Items - Synchronization Rule - Rational Team Concert
File Edit Navigate Search Project Run Window Help
Work Items
Team Arti
Team Or
com.ibm.rational.clearquest.CQALM.ALMTask
Save
Add...
Remove
Initialize
Overview
Source
<No Current Work>

All Team Areas (2 of 2 Team Areas selected)

- Repository Connections
  - ADMIN@localhost
- AccountOpening [localhost]
- Favorites
- Feeds
- My Repository Workspaces
- My Team Areas
- Work Item History

Synchronization Rule

Name com.ibm.rational.clearquest.CQALM.ALMTask

Type Mapping

Item type:	WorkItem - com.ibm.te
Item type qualifier:	Task
Item manager:	Work Item Manager
External repository:	CQALM
External manager:	ClearQuest Manager (r
External type:	ALMTask - com.ibm.ra
Maximum cycle count:	5

☐ Synchronize all item states

Team Areas enabled for new connections (1):

- ☑ AccountOpening Team [AO_Rel2]
- ☐ CreditCheck [AO_Rel2]

Property Mappings

Id...	Item Property	Dir...	Id...	External Property
	Attachments	⇔		Attachments
	Comments	⇔		Notes_Log
	Description	⇔		Description
	Owned By	⇔		Owner
	Planned For	⇔		PhaseLabel.Name/Itera...
	Priority	⇔		Priority.Name

Mapping details

Item property: ☑ None ☐ Item identifier

Synchronization: <none>

External property: ☑ None ☐ External identifier ☐ External modifier

☑ No transformation (just copy)

Reference synchronization rule:

Value transformer:

Value mappings:

Item Value	External Value

Figure B-11 The synchronization rule editor in Rational Team Concert

com.ibm.rational.clearquest.CQALM.ALMTypeLabel

The com.ibm.rational.clearquest.CQALM.ALMTypeLabel helper synchronization rule synchronizes Priority reference fields. Table B-1 lists the synchronization rule properties for ALMTypeLabel.

Table B-1 ALMTypeLabel synchronization rule properties

Synchronization rule property	Property value
Item type	<none>
Item type qualifier	
Item manager	<none>
External repository	CQALM
External manager	ClearQuest Manager (non-user records)
External type	ALMTypeLabel - com.ibm.rational.clearquest

Table B-2 lists the synchronization rule property mappings for ALMTypeLabel.

Table B-2 ALMTypeLabel synchronization rule property mappings

Item ID	Item property	I/O	Ext ID	Element property
			X	Name

com.ibm.rational.clearquest.CQALM.ALMResolutionCodeLabel

The com.ibm.rational.clearquest.CQALM.ALMResolutionCodeLabel helper synchronization rule synchronizes ResolutionCodeLabel reference fields. Table B-3 lists the synchronization rule properties for ALMResolutionCodeLabel.

Table B-3 ALMResolutionCodeLabel synchronization rule properties

Synchronization rule property	Property value
Item type	<none>
Item type qualifier	
Item manager	<none>
External repository	CQALM
External manager	ClearQuest Manager (non-user records)
External type	ALMResolutionCodeLabel - com.ibm.rational.clearquest

Table B-4 lists the synchronization rule property mappings for ALMResolutionCodeLabel.

Table B-4 ALMResolutionCodeLabel synchronization rule property mappings

ID	Item property	I/O	ID	Element property
			X	Name

com.ibm.rational.clearquest.CQALM.users

The com.ibm.rational.clearquest.CQALM.users synchronization rule synchronizes user records in Rational ClearQuest with contributors in Rational Team Concert. Table B-5 lists the synchronization rule properties for CQALM.users.

Table B-5 CQALM.users synchronization rule properties

Synchronization rule property	Property value
Item type	Contributor
Item type qualifier	
Item manager	Contributor manager
External repository	CQALM
External manager	ClearQuest Manager (user records)
External type	users - com.ibm.rational.clearquest.CQALM

Table B-6 lists the synchronization rule property mappings for CQALM.users.

Table B-6 CQALM.users synchronization rule property mappings

ID	Item property	I/O	ID	Element property
	EmailAddress	in		email
	Name	in		fullname
X	UserId	in	X	login_name

com.ibm.rational.clearquest.CQALM.Attachment

The com.ibm.rational.clearquest.CQALM.Attachment synchronization rule synchronizes record attachments in Rational ClearQuest records with Work Item attachments in Rational Team Concert. Attachments can be applied to record types by using the ClearQuest Attachments package. The ALM schema has attachments that are already configured. The Attachments synchronization rule is invoked from other rules that mapping ClearQuest records with Work Item types, for example ALMActivity.

Table B-7 lists the synchronization rule properties for CQALM.Attachments.

Table B-7 CQALM.Attachments synchronization rule properties

Synchronization rule property	Property value
Item type	Attachment
Item type qualifier	
Item manager	Work Item Attachments Manager
External repository	CQALM
External manager	ClearQuest Manager (non-user records)
External type	attachments - com.ibm.rational.clearquest.CQALM

Table B-8 lists the synchronization rule property mappings for CQALM.Attachments.

Table B-8 CQALM.Attachment synchronization rule property mappings

ID	Item property	I/O	ID	Element property
	Content	In/Out		Content
	Name	In		file-name

com.ibm.rational.clearquest.CQALM.ALMActivity

The com.ibm.rational.clearquest.CQALM.ALMActivity synchronization rule synchronizes ALMActivities records in ClearQuest with Work Items in Rational Team Concert. Table B-9 lists the synchronization rule properties for ALMActivity.

Table B-9 ALMActivity synchronization rule properties

Synchronization rule property	Property value
Item type	WorkItem - com.ibm.team.workitem
Item type qualifier	Defect [a]
Item manager	Work Item Manager
External repository	CQALM
External manager	ClearQuest Manager (non-user records)
External type	ALMActivity - com.ibm.rational.clearquest.CQALM

a. An alternate configuration can be used to map ALMActivities to Work Items of type Activity. See "Adding an Activity Work Item type" on page 617.

Important: When multiple Rational ClearQuest record types are mapped to Work Items, the Item type qualifier synchronization property is required to be unequally set for each synchronization rule. For ALMActivity synchronizations, the Item type qualifier must be set to Defect. For additional configuration considerations using an Activity type, see "Adding an Activity Work Item type" on page 617.

Table B-10 lists the synchronization rule property mappings for ALMActivity.

Table B-10 com.ibm.rational.clearquest.CQALM.ALMActivity synchronization rule property mappings

ID	Item property	I/O	ID	Element property
				Priority
				ResolutionCode
				Project
				Category
				Type
				modifiedBy
	Attachments	In/Out		Attachments
	Comments	In/Out		Notes_Log
	Description	In/Out		Description
	Filed Against	In		Project.Category.Name
	Owned By	In/Out		Owner
	Parent	In		Task
	Related	In		ActivitiesRelated
	Resolution	Out		ResolutionSummary

ID	Item property	I/O	ID	Element property
	Resolution	Out		ResolutionCode.Name
	Status	Out		State
	Tags [a]	In		Type.Name
	Summary	In/Out		Headline
	Type [b]	In		

a. An alternate configuration can be used to map the ALMActivity type to a Work Item Subtype property.
b. An alternate configuration can be used to map ALMActivities to Work Items of type Activity.

Priority element property mapping settings

The priority element is a reference field synchronization property that is used for delegating the synchronization of Priority.Name to the ALMTypeLabel synchronization rule. Table B-11 lists the element property mapping settings for Priority.

Table B-11 Priority synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMTypeLabel
Value transformer	<none>

ResolutionCode element property mapping settings

The ResolutionCode element is a reference field synchronization property that is used for delegating the synchronization of ResolutionCode.Name to the ALMResolutionCodeLabel synchronization rule. Table B-12 lists the synchronization rule properties for ResolutionCode.

Table B-12 ResolutionCode synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMResolutionCodeLabel
Value transformer	<none>

Project element property mapping settings

The Project element is a reference field synchronization property that is used for delegating the synchronization of Project.Category.Name to the ALMProject synchronization rule. Table B-13 lists the synchronization rule properties for Project.

Table B-13 Project synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMProject
Value transformer	<none>

Category element property mapping settings

The Category element is a reference field synchronization property that is used for delegating the synchronization of Project.Category.Name to the ALMCategory synchronization rule. Table B-14 lists the element property mapping settings for Category.

Table B-14 Category synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMCategory
Value transformer	<none>

Type element property mapping settings

The Type element is a reference field synchronization property that is used for delegating the synchronization of Type.Name to the ALMTypeLabel synchronization rule. Table B-15 lists the element property mapping settings for Type.

Table B-15 Type synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMTypeLabel
Value transformer	<none>

modifiedBy element property mapping settings

The modifiedBy synchronization rule setting makes the work items to be made in the context of the corresponding Rational ClearQuest user and not the cqconnector user that is used by the ClearQuest Gateway. Table B-16 lists the element property mapping settings for modifiedBy.

Table B-16 modifiedBy synchronization rule properties

Synchronization rule property	Property value
External modifier	X
Reference synchronization rule	
Value transformer	<none>

Attachments item property mapping settings

The Attachments item is a field synchronization property that is used to delegate the synchronization of Attachments by using the Work Item Attachments Transformer. Table B-17 lists the property mapping settings for Attachments.

Table B-17 Attachments synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.Attachment
Value transformer	Work Item Attachments Transformer

Comments item property mapping settings

The Comments item is a field synchronization property that is used to delegate the synchronization of comments and notes by using the Work Item Comments Transformer. Table B-18 lists the property mapping settings for Comments.

Attention: The Comments property requires that the optional Notes 5.1 package has been added to the ALM schema and applied to the ALMActivity record type. We explain the steps to apply the Notes package in "Applying the Notes package" on page 570. If the Notes package is not applied, do not add this mapping.

Table B-18 Comments synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.users
Value transformer	Work Item Comments Transformer

Description item property mapping settings

The Description item is a field synchronization property that is used for the synchronization of Comments. Table B-19 lists the property mapping settings for Description.

Table B-19 Description property mapping settings

Synchronization rule property	Property value
No transformation (just copy)	X
Reference synchronization rule	<none>
Value transformer	<none>

Filed Against item property mapping settings

The Filed Against item is a field synchronization property that is used to delegate the synchronization of FiledAgainst and Category by using the Work Item Category Transformer. Table B-20 lists the property mapping settings for Filed Against.

Table B-20 Filed Against property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	<none>
Value transformer	Work Item Category Transformer

OwnedBy item property mapping settings

The OwnedBy item is a field synchronization property that is used to delegate the synchronization of Owners by using the users synchronization rule. Table B-21 lists the property mapping settings for OwnedBy.

Table B-21 OwnedBy property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.users
Value transformer	<none>

Parent item property mapping settings

The Parent item is a field synchronization property that is used to delegate the synchronization of parent Tasks by using the ALMTask synchronization rule. Table B-22 lists the property mapping settings for Parent.

Table B-22 Parent property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMTask
Value transformer	<none>

Related item property mapping settings

The Related item is a field synchronization property that is used to delegate the synchronization of related Activities by using the ALMActivity synchronization rule. Table B-20 lists the property mapping settings for Related.

Table B-23 Related item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMActivity
Value transformer	<none>

Resolution - ResolutionSummary property mapping settings

The Resolution - ResolutionSummary is a field synchronization property that is used for synchronization of the resolution summary field. Table B-24 lists the property mapping settings for Resolution - ResolutionSummary.

Table B-24 Resolution item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	<none>

Resolution - ResolutionCode.Name property mapping settings

The Resolution - ResolutionCode.Name is a reference field synchronization property that uses the ResoutionCode synchronization property and the Customer Attribute Transformer to resolve the reference. Table B-25 lists the property mapping settings for Resolution - ResolutionCode.Name.

Table B-25 Resolution item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Connect Field To Custom Attribute Transformer

Status item property mapping settings

The Status item is a field synchronization property that is used for synchronization of the resolution status field. The mapping table is required to map the state models in the Jazz platform and ClearQuest. Table B-26 lists the property mapping settings for Status.

Table B-26 Status item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	<none>

Table B-27 lists the property value mapping for Status.

Table B-27 Status property value mappings

Item value	External value
Closed	Completed
In Progress	Activated
New	
Reopened	
Resolved	
Triaged	
Unconfirmed	
Verified	

Tags item property mapping settings

The Tags items is a field synchronization property that is used for synchronization of the tags field. Table B-28 lists the property mapping settings for Tags.

Table B-28 Tags item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Connect Field To Custom Attribute Transformer

Note: An alternate configuration can be used to map the ALMActivity type to a Work Item Subtype property.

Summary item property mapping settings

The Summary item is a field synchronization property that is used for synchronization of the summary field. Table B-29 lists the property mapping settings for Summary.

Table B-29 Summary item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	<none>

Type item property mapping settings

The Type item is a field synchronization property that is used for synchronization of the type field. Table B-30 lists the property mapping settings for Type.

Table B-30 Type item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	<none>

Table B-31 lists the property value mappings for Type.

Table B-31 Type item property value mappings

Item value	External value
defect	

Note: An alternate configuration can be used to map ALMActivities to Work Items of type Activity.

com.ibm.rational.clearquest.CQALM.ALMTask

The com.ibm.rational.clearquest.CQALM.ALMTask synchronization rule synchronizes ALMTask records in ClearQuest with Work Items in Rational Team Concert. Table B-32 lists the synchronization rule properties for ALMTask.

Table B-32 ALMTask synchronization rule properties

Synchronization rule property	Property value
Item type	WorkItem - com.ibm.team.workitem
Item type qualifier	Task
Item manager	Work Item Manager
External repository	CQALM <your schema connection>
External manager	ClearQuest Manager (non-user records)
External type	ALMTask - com.ibm.rational.clearquest.CQALM

Important: When multiple ClearQuest record types are mapped to Work Items, the Item type qualifier synchronization property must be unique for each synchronization rule. For ALMTask synchronizations, the Item type qualifier must be set to Task. For additional configuration considerations, see "Adding a new Type category" on page 621.

Table B-33 lists the property mappings for ALMTask.

Table B-33 com.ibm.rational.clearquest.CQALM.ALMTask synchronization rule property mappings

ID	Item property	I/O	ID	Element property
				Category
				Project
				Type
				Priority
				ResolutionCode
				modifiedBy
	Attachments	In/Out		Attachments
	Comments	In/Out		Notes_Log
	Description	In/Out		Description
	Filed Against	In		Project.Category.Name
	Owned By	In/Out		Owner
	Priority	In/Out		Priority.Name
	Related	In		TasksRelated
	Resolution	Out		ResolutionCode.Name
	Resolution	Out		ResolutionSummary
	Status	Out		State
	Tags [a]	In		Type.Name
	Summary	In/Out		Headline
	Type	In		

a. An alternate configuration can be used to map the ALMType type to a Work Item Subtype property.

Category element property mapping settings

Table B-34 lists the synchronization rule properties for Category.

Table B-34 Category synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMCategory
Value transformer	<none>

Project element property mapping settings

Table B-35 lists the synchronization rule properties for Project.

Table B-35 Project synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMProject
Value transformer	<none>

Type element property mapping settings

Table B-36 lists the synchronization rule properties for Type.

Table B-36 Type synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMTypeLabel
Value transformer	<none>

Priority element property mapping settings

Table B-37 lists the synchronization rule properties for Priority.

Table B-37 Priority synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMTypeLabel
Value transformer	<none>

ResolutionCode element property mapping settings

Table B-38 lists the synchronization rule properties for ResolutionCode.

Table B-38 ResolutionCode synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMResolutionCodeLabel
Value transformer	<none>

modifiedBy element property mapping settings

The modifiedBy element synchronization rule setting makes the work items to be made in the context of the corresponding ClearQuest user and not the cqconnector user that is used by the ClearQuest Gateway. Table B-39 lists the synchronization rule properties for modifiedBy.

Table B-39 modifiedBy synchronization rule properties

Synchronization rule property	Property value
External modifier	X
Reference synchronization rule	
Value transformer	<none>

Attachments item property mapping settings

Table B-40 lists the synchronization rule properties for Attachments.

Table B-40 Attachments synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.Attachment
Value transformer	Work Item Attachments Transformer

Comments item property mapping settings

Table B-41 lists the synchronization rule properties for Comments.

Important: The Comments property requires that the optional Notes 5.1 package has been added to the ALM schema and applied to the ALMActivity record type. The steps to apply the Notes package is described in “Applying the Notes package” on page 570. If the Notes package is not applied, do not add this mapping.

Table B-41 Comments synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.users
Value transformer	Work Item Comments Transformer

Description item property mapping settings

Table B-42 lists the property mapping settings for Description.

Table B-42 Description property mapping settings

Synchronization rule property	Property value
No transformation (just copy)	X
Reference synchronization rule	<none>
Value transformer	<none>

Filed Against item property mapping settings

Table B-43 lists the property mapping settings for Filed Against.

Table B-43 Filed Against property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Work Item Category Transformer

OwnedBy item property mapping settings

Table B-44 lists the property mapping settings for OwnedBy.

Table B-44 OwnedBy property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.users
Value transformer	<none>

Priority item property mapping settings

Table B-45 lists the property mapping settings for Priority.

Table B-45 Priority property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Connect Field To Custom Attribute Transformer

Related item property mapping settings

Table B-46 lists the property mapping settings for Related.

Table B-46 Related item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	com.ibm.rational.clearquest.CQALM.ALMTask
Value transformer	<none>

Resolution - ResolutionCode.Name property mapping settings

Table B-47 lists the property mapping settings for Resolution - ResolutionCode.Name.

Table B-47 Resolution property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Connect Field To Custom Attribute Transformer

Resolution - ResolutionSummary property mapping settings

Table B-48 lists the property mapping settings for Resolution - ResolutionSummary.

Table B-48 Resolution property mapping settings

Synchronization rule property	Property value
No transformation (just copy)	X
Reference synchronization rule	
Value transformer	<none>

Status item property mapping settings

Table B-49 lists the property mapping settings for Status.

Table B-49 Status item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	<none>

Table B-50 lists the property value settings for Status.

Table B-50 Status property value mappings

Item value	External value
Closed	Completed
In Progress	Activated
New	
Reopened	
Resolved	
Triaged	
Unconfirmed	
Verified	

Tags item property mapping settings

Table B-51 lists the property mapping settings for Tags.

Table B-51 Tags item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Connect Field To Custom Attribute Transformer

Note: An alternate configuration can be used to map the ALMTask type to a Work Item Subtype property.

Summary item property mapping settings

Table B-52 lists the property mapping settings for Summary.

Table B-52 Summary item property mapping settings

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	<none>

Type item property mapping settings

Table B-53 lists the property mapping settings for Type.

Table B-53 Type item property mapping settings

Item Value	External Value
task	

Practices for using ALM interoperability

In this section, we provide practices for using the ClearQuest Connector for ALM interoperability.

Starting ClearQuest Gateway

The ClearQuest Gateway must be started to enable ALM interoperability. To start the gateway:

1. Browse the file system on the server that has the ClearQuest Gateway. Locate the IBM\ClearQuestConnector\gateway directory.
2. Run the **`server.startup.bat`** script.
3. Monitor the startup and synchronization process logged to the Tomcat command prompt window.

Tip: Additional log traces from the ClearQuest Gateway can be configured by editing the log4j.properties file in the \jazz\connectors\gateway directory. Uncomment the following lines:

```
log4j.logger.com.ibm.rational.interop.level1=DEBUG
log4j.logger.com.ibm.rational.interop.level2=DEBUG
```

Starting Jazz server synchronization

The synchronization service on the Jazz server must be started to enable ALM interoperability. To start the service:

1. Open a browser window and go to the Jazz server Admin Web UI:

   ```
   https://localhost:9443/jazz/admin
   ```

2. Log in as the ADMIN administrator user.
3. Select **Configuration** → **Advanced**.
4. On the Advanced Properties page, scroll to the **com.ibm.team.interop.service.internal.InteropService** section.
5. For the Connector - Outgoing Sync Enabled property, update it to `True`. By default, the synchronization is disabled by the value False.
6. Optional: Scroll to the **com.ibm.team.interop.service.internal. OutgoingSyncScheduledTask** property and update the value for the synchronization interval Outgoing Sync Task Fixed Delay to a new value. By default, the synchronization interval is 300 seconds.
7. Scroll to the top of the page and click **Save** to persist the changes.
8. Monitor the synchronization process that is logged to the Tomcat command prompt window.

Tip: Additional log traces from the Jazz server can be configured by editing the log4j.properties file in the \JazzTeamServer\server directory. Add the following line:

```
log4j.logger.com.ibm.team.interop=DEBUG
```

Preventing outgoing synchronization for new work items

The ALM interoperability solution that documented in this book does not support new work items that are created in Rational Team Concert or Rational Quality Manager to be synchronized to Rational ClearQuest. Such items are considered local to the component or practice team and cause errors to be reported by the ALMTask and ALMActivity synchronization rules.

Synchronization rules can be configured to disable outgoing synchronization for new work items. This setting is available in the synchronization rules editor. By using the *Team areas enabled for new connections(1)* controls, a synchronization rule can be configured to only allow outgoing synchronization for selected team areas. In Rational Team Concert 1.0, at least one team area must be enabled for outgoing synchronization.

Disabling outgoing synchronization for new work items has the following synchronization behavior:

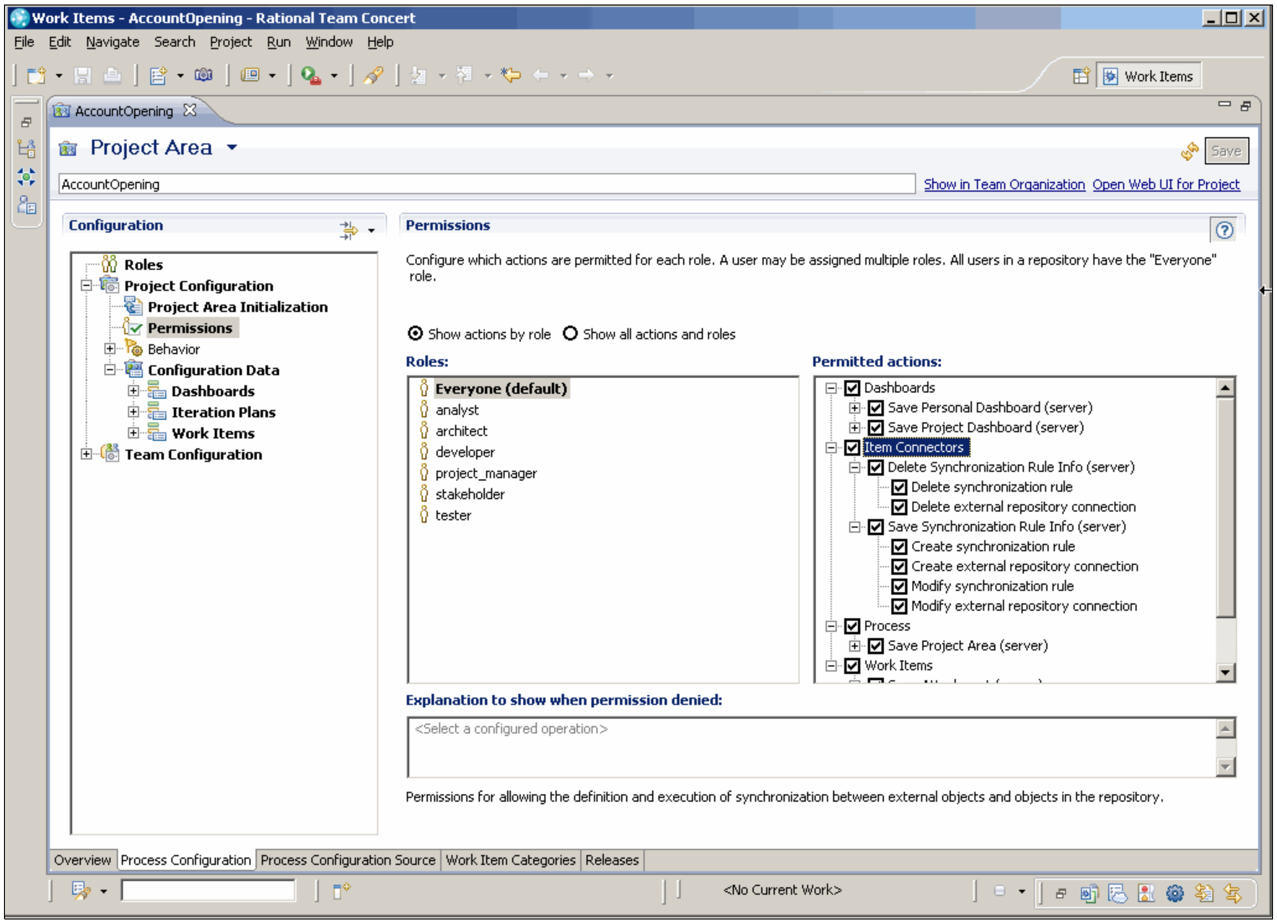
- Continues synchronization out changes for work items that were previously created from Rational ClearQuest
- Does not attempt outgoing synchronization for new work items that are created first in Rational Team Concert

To turn off synchronization for all used team areas, create a new team area and give it a descriptive name, for example, cqconnector. Disable outgoing synchronization for all team areas, with the exception on this new cqconnector team area. Optionally, archive the new team area to avoid it from being displayed in browsers and pick lists.

Setting permissions

The process configuration in Rational Team Concert and Rational Quality Manager specifies the actions that are permitted for each role. A user can perform all actions granted to any of their assigned roles. All users in a repository have the “Everyone” role.

When configuring ALM interoperability, validate that the user roles that have responsibilities to administer remote connections, and synchronization rules have the appropriate permissions enabled. See Figure B-12.


Work Items - AccountOpening - Rational Team Concert

AccountOpening

Project Area

AccountOpening

Show in Team Organization Open Web UI for Project

Configuration

- Roles
- Project Configuration
  - Project Area Initialization
  - Permissions
  - Behavior
  - Configuration Data
    - Dashboards
    - Iteration Plans
    - Work Items
- Team Configuration

Permissions

Configure which actions are permitted for each role. A user may be assigned multiple roles. All users in a repository have the "Everyone" role.

[✓ Show actions by role] Show all actions and roles

Roles:

- Everyone (default)
- analyst
- architect
- developer
- project_manager
- stakeholder
- tester

Permitted actions:

- ✓ Dashboards
  - ✓ Save Personal Dashboard (server)
  - ✓ Save Project Dashboard (server)
- ✓ Item Connectors
  - ✓ Delete Synchronization Rule Info (server)
    - ✓ Delete synchronization rule
    - ✓ Delete external repository connection
  - ✓ Save Synchronization Rule Info (server)
    - ✓ Create synchronization rule
    - ✓ Create external repository connection
    - ✓ Modify synchronization rule
    - ✓ Modify external repository connection
- ✓ Process
  - ✓ Save Project Area (server)
- ✓ Work Items

Explanation to show when permission denied:

<Select a configured operation>

Permissions for allowing the definition and execution of synchronization between external objects and objects in the repository.

Overview | Process Configuration | Process Configuration Source | Work Item Categories | Releases

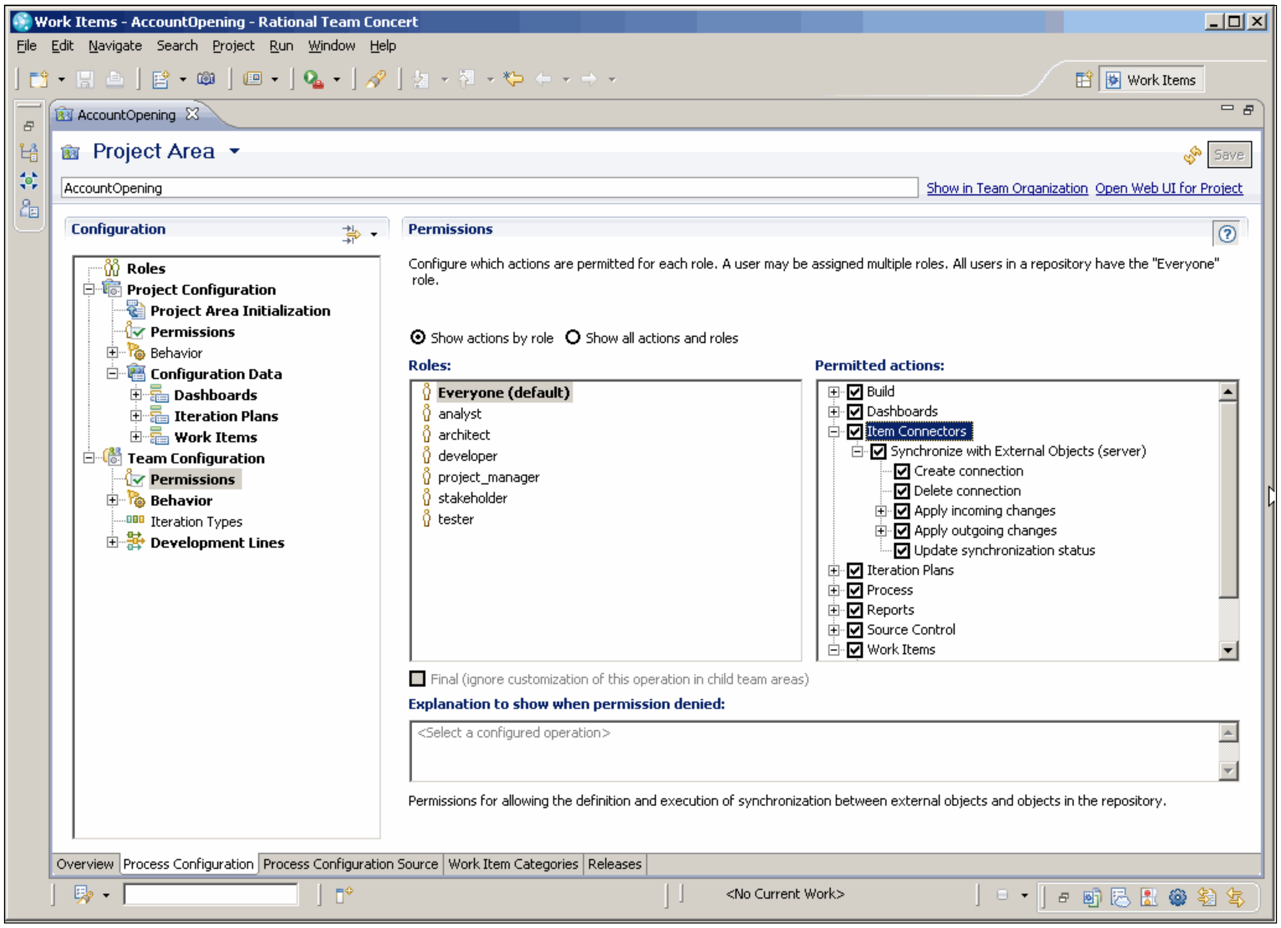
<No Current Work>

Figure B-12 Enabling permissions to manage synchronization rules

To enable project-wide permissions to manage synchronization rules:

1. Select the **Process Configuration** tab in the AccountOpening Project Area and expand **Project Configuration** → **Permissions**.
2. In the Permissions view, select **Everyone**. Validate the Item Connectors, and select the appropriate permissions for the project-wide settings.
3. Select other team roles and set the appropriate Item Connectors permissions.

4. Expand **Team Configuration** → **Permissions**.
5. Repeat steps 2 on page 606 and 3 on page 606 to modify the permissions for team areas to establish connections and synchronization. See Figure B-13.


Work Items - AccountOpening - Rational Team Concert
File Edit Navigate Search Project Run Window Help
Work Items
AccountOpening
Project Area
Save
Show in Team Organization
Open Web UI for Project
Overview
Process Configuration
Process Configuration Source
Work Item Categories
Releases
<No Current Work>

AccountOpening

Configuration

- Roles
- Project Configuration
  - Project Area Initialization
  - Permissions
  - Behavior
  - Configuration Data
    - Dashboards
    - Iteration Plans
    - Work Items
- Team Configuration
  - Permissions
  - Behavior
  - Iteration Types
  - Development Lines

Permissions

Configure which actions are permitted for each role. A user may be assigned multiple roles. All users in a repository have the "Everyone" role.

[✓ Show actions by role] Show all actions and roles

Roles:

- Everyone (default)
- analyst
- architect
- developer
- project_manager
- stakeholder
- tester

Permitted actions:

- ☑ Build
- ☑ Dashboards
- ☑ Item Connectors
  - ☑ Synchronize with External Objects (server)
    - ☑ Create connection
    - ☑ Delete connection
    - ☑ Apply incoming changes
    - ☑ Apply outgoing changes
    - ☑ Update synchronization status
- ☑ Iteration Plans
- ☑ Process
- ☑ Reports
- ☑ Source Control
- ☑ Work Items

☐ Final (ignore customization of this operation in child team areas)

Explanation to show when permission denied:

<Select a configured operation>

Permissions for allowing the definition and execution of synchronization between external objects and objects in the repository.

Figure B-13 Enabling permissions to manage connections and perform synchronizations

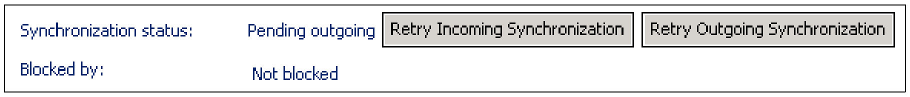
6. Click **Save** to save and deploy the new permissions.

Viewing the synchronization status

The status for work items that are mapped to ClearQuest records can be viewed by using the link that is provided with the work item. To view the synchronization status:

1. Open the work item and select the **Links** tab.
2. In the Links section, select the item under External connection named **Synchronization status for** and click **Open**.
3. View the Synchronization Status page for the following detailed information:
   – Synchronization state and status
   – Work item and ClearQuest record field data
   – Errors from last synchronization

Be aware that the ClearQuest Connector must complete both incoming and outgoing synchronization to establish a stabile state. Modifying records during the synchronization often results in a conflict state. Do not modify items that are in a state of *Pending outgoing*. See Figure B-14.


Synchronization status: Pending outgoing

Blocked by: Not blocked

Retry Incoming Synchronization
Retry Outgoing Synchronization

Figure B-14 Synchronization Status indicating 'Pending output'

Viewing synchronization errors

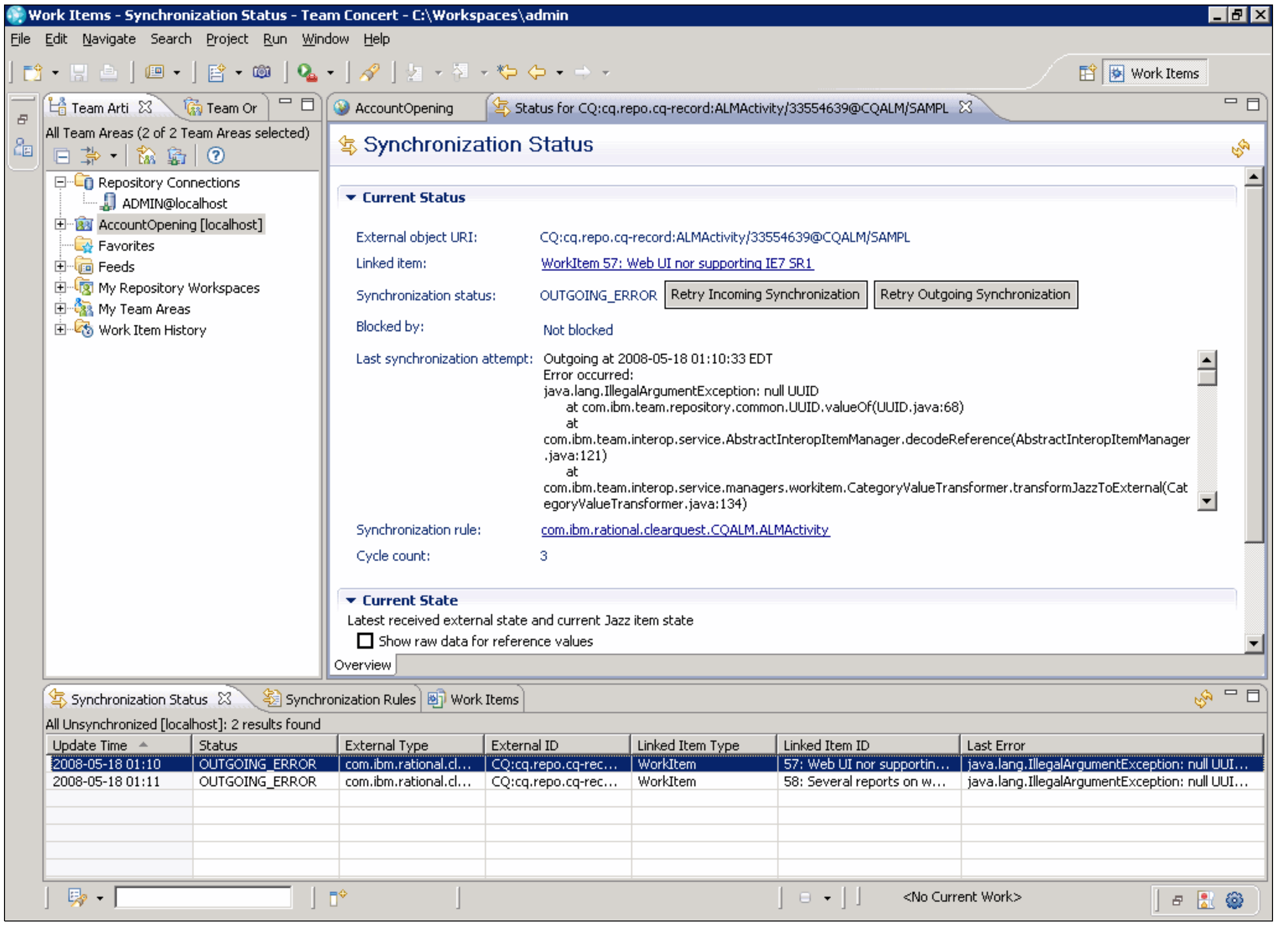
The ClearQuest Connector reports errors when synchronization cannot be completed. Items that cannot be synchronized are added to the Synchronization Status view. There might be multiple reasons, such as the following examples, for such errors:

- Errors when connecting to Rational ClearQuest or Rational Team Concert
- Configuration errors in the synchronization rules
- Undefined values or mappings detected at synchronization

Each item in the Synchronization Status view represents a failure in synchronizing an item pair. To view all synchronization errors:

1. Open the Synchronization Rules view and select the team area.
2. Choose **Show All Unsynchronized**.
3. Browse the Synchronization Status view and open each error item.
4. View the Java style stack trace to diagnose the synchronization issue.
5. If the error is understood and corrections are possible, modify the ClearQuest record, the work item, or the synchronization rule.

6. To re-run synchronization, click the **Retry Incoming Synchronization** or **Retry Outgoing Synchronization** button, for INCOMMING_ERROR or OUTGOING_ERROR respectively. See Figure B-15.



Work Items - Synchronization Status - Team Concert - C:\Workspaces\admin

Status for CQ:cq.repo.cq-record:ALMActivity/33554639@CQALM/SAMPL

Synchronization Status

Current Status

External object URI:	CQ:cq.repo.cq-record:ALMActivity/33554639@CQALM/SAMPL
Linked item:	WorkItem 57: Web UI nor supporting IE7 SR1
Synchronization status:	OUTGOING_ERROR
Blocked by:	Not blocked
Last synchronization attempt:	see below
Synchronization rule:	com.ibm.rational.clearquest.CQALM.ALMActivity
Cycle count:	3

```
Outgoing at 2008-05-18 01:10:33 EDT
Error occurred:
java.lang.IllegalArgumentException: null UUID
    at com.ibm.team.repository.common.UUID.valueOf(UUID.java:68)
    at
com.ibm.team.interop.service.AbstractInteropItemManager.decodeReference(AbstractInteropItemManager
.java:121)
    at
com.ibm.team.interop.service.managers.workitem.CategoryValueTransformer.transformJazzToExternal(Cat
egoryValueTransformer.java:134)
```

Current State

Latest received external state and current Jazz item state

Show raw data for reference values

All Unsynchronized [localhost]: 2 results found

Update Time	Status	External Type	External ID	Linked Item Type	Linked Item ID	Last Error
2008-05-18 01:10	OUTGOING_ERROR	com.ibm.rational.cl...	CQ:cq.repo.cq-rec...	WorkItem	57: Web UI nor supportin...	java.lang.IllegalArgumentException: null UUI...
2008-05-18 01:11	OUTGOING_ERROR	com.ibm.rational.cl...	CQ:cq.repo.cq-rec...	WorkItem	58: Several reports on w...	java.lang.IllegalArgumentException: null UUI...

Figure B-15 ClearQuest Connector Synchronization Status view

After the synchronization succeeds, the item is removed from the Synchronization Status view.

Tip: Synchronization can be disabled for items by select **Stop Synchronizing**. Use this function only after considering the implications of breaking the synchronization between the Rational ClearQuest and Rational Team Concert items.

Importing changes to existing synchronization rules

The deployment of synchronization rules must be maintained because the Rational ClearQuest schema or the Rational Team Concert and Rational Quality Manager process specifications are refined over time due to continuous process improvement by the project team. Some considerations should be taken when maintaining synchronization rules.

Deleting a synchronization rule deletes all of the connector information for all items that used the synchronization rule. Essentially, all connected items become disconnected. You are warned about this when you select the delete operation for the synchronization rule. The entire synchronization history is removed, and there is no longer any memory that those items were ever connected to anything. Then, if a new synchronization rule is created or imported that is configured to select previously synchronized items, it acts as though synchronization is being done for the first time. The Jazz server discovers that there are work items that are configured for synchronization, but those items are not connected to any external objects.

To update a new version of a synchronization rule, practice using the Import button in the upper right area of the synchronization rule editor (Figure B-16).

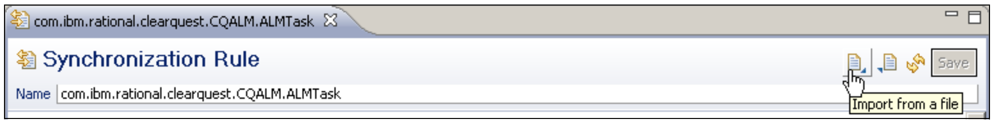

com.ibm.rational.clearquest.CQALM.ALMTask
Synchronization Rule
Name
com.ibm.rational.clearquest.CQALM.ALMTask
Save
Import from a file

Figure B-16 The Import button to update an existing synchronization rule

Extended ALM interoperability configuration

In this section, we describe the extended configurations for ALM interoperability.

Extended ClearQuest ALM configurations

In this section, we describe alternate configurations to Rational ClearQuest for ALM interoperability.

Extending the ALM schema for iteration planning

The example ALM interoperability configuration in "Configuring and deploying synchronization rules" on 585 does not include synchronization of Iteration plans across the ALM solution. When using the ClearQuest ALM schema version 1.0, such synchronization requires modifications of the PERL hook code.

To implement synchronization of iteration plans between Rational ClearQuest and Rational Team Concert, the following configurations are suggested:

- Modify the ALM schema
  - Add new RTCPlannedFor field to the ALMTask record type
  - Add new RTCEstimate and RTCDueDate fields to the ALMTask and ALMActivity record types
  - Add new PERL hook code
- Modify the com.ibm.rational.clearquest.CQALM.ALMTask synchronization rule
- Modify the com.ibm.rational.clearquest.CQALM.ALMActivity synchronization rule

Adding a new field to the ClearQuest schema

To add the RTCPlannedFor field to the ALMTask record type:

1. Open the ClearQuest Designer.
2. Browse the ClearQuest Schema Repository Explorer view for the ALM schema to be modified. Open the schema by logging in as an administrator.
3. Select the latest schema version and select **Revision Control** → **Check Out**.
4. Add new fields to ALMActivity. Select **Record Types** → **ALMActivity** → **Fields** and repeat the steps to add the following fields:
   - Field name: `RTCEstimate` with a Type of **Short String**
   - Field name: `RTCDueDate` with a Type of **Date**

5. Add new fields to ALMTask:
   a. Select **Record Types** → **ALMTask** → **Fields**, right-click and select **New Field**.
   b. In the New Field window, enter the following values:
      - Field name: `RTCPlannedFor`
      - Type: **Short String**
   c. Repeat the steps to add the following fields:
      - Field name: `RTCEstimate` with a Type of **Short String**
      - Field name: `RTCDueDate` with a Type of **Date**
6. Add a Perl Value Change hook for RTCPlannedFor:
   a. With **Record Types** → **ALMTask** → **Fields** selected, locate the ValueChanged cell for the RTCPlannedFor field. Select the cell and choose **Scripts** → **Perl**.
   b. Enter the script code as shown in Example B-2 into the script editor.

Example: B-2 rtcplannedfor_ValueChanged hook

```
sub rtcplannedfor_ValueChanged {
   my($fieldname) = @_;
   # $fieldname as string scalar
   # record type name is ALMTask
   # field name is RTCPlannedFor
   $session->OutputDebugString("\nEntering RTCPlannedFor value changed
hook\n");
   my $RTCPlannedFor = $entity->GetFieldValue("$fieldname")->GetValue();
   $session->OutputDebugString("\nRTCPlannedFor value is $RTCPlannedFor\n");
   my @PhaseIterLBL = split(/\//, $RTCPlannedFor);
   if ($#PhaseIterLBL == 1) {
      $session->OutputDebugString("\nSetting PhaseLabel: $PhaseIterLBL[0]
and
         IterationLabel: $PhaseIterLBL[1] on ALMTask\n");
      my $failure=$entity->SetFieldValue("PhaseLabel","$PhaseIterLBL[0]");
      my
$failure=$entity->SetFieldValue("IterationLabel","$PhaseIterLBL[1]");
   }
}
```

7. Add a new base action for ALMTask record type:
   a. Select **Record Types** → **ALMTask** → **States and Actions** → **Actions**, right-click and select **New Action**.
   b. In the New Action window, for Name, type `SetRTCPlannedFor`, and for Type, select **BASE**. Click **Finish**.
   c. In the Record Actions editor, click the **SetRTCPlannedFor** or **Validation** cell and select **SCRIPTS** → **Add** to add a Perl script.

d. In the script editor, enter `&ALMTask_SetRTCPlannedFor;` as highlighted in Example B-3. Example B-3 shows how the hook code looks after you add the single statement.

Example: B-3 ALMTask_Validation hook

```
sub almtask_Validation {
    my($actionname, $actiontype) = @_;
    my $result;
    # $actionname as string scalar
    # $actiontype as long scalar
    # $result as string scalar
    # action is SetRTCPlannedFor
    # record type name is ALMTask
   &ALMTask_SetRTCPlannedFor;
   return $result;
}
```

8. Add a new global script:

   a. Select **Global Scripts** → **Perl**, right-click and select **New Script**.

   b. Enter the script name as `ALMTask_SetRTCPlannedFor`.

   c. Right-click and select **Open Script**.

   d. Enter the script code from Example B-4 into the script editor.

Example: B-4 ALMTask_SetRTCPlannedFor script

```
sub ALMTask_SetRTCPlannedFor {
   my $PhaseLBL = $entity->GetFieldValue("PhaseLabel")->GetValue();
   my $IterationLBL = $entity->GetFieldValue("IterationLabel")->GetValue();
   if ($PhaseLBL && $IterationLBL) {
      my
$failure=$entity->SetFieldValue("RTCPlannedFor","$PhaseLBL/$IterationLBL");
   }
}
```

   e. Change the scripting language by selecting **ALM** → **Version**, right-clicking, and selecting **Properties**. In the Properties view, select **Windows** or **UNIX** scripting and choose **PERL**.

Note: Users who use Basic on Windows instead of Perl should provide a similar hook in Basic.

9. Select **File** → **Save** to save the changes.

10. Select **ALM** → **Version**, right-click and select **Revision Control** → **Check-In**.

Modifying the ALMTask synchronization rule

The com.ibm.rational.clearquest.CQALM.ALMTask synchronization rule must be updated to include mapping of the RTCPlannedFor or Planned For, RTCEstimate or Estimate, and RTCDueDate or DueDate fields and properties in the ClearQuest and Jazz repositories.

To update the synchronization rule:

1. Open the **com.ibm.rational.clearquest.CQALM.ALMTask** synchronization rule.
2. In the Property Mappings section, click the **Add** button.

3. In the Define Property Mapping window, set the values as provided in Table B-54.

Table B-54 com.ibm.rational.clearquest.CQALM.ALMTask synchronization rule property mappings

ID	Item property	I/O	ID	Element property
	Planned For	In/Out		RTCPlannedFor
	Estimate	Out		RTCEstimate
	Due Date	Out		RTCDueDate

4. Click **OK** to add the new mapping.
5. Complete the Mapping details section by using the values that are provided in Table B-55 and Table B-56.

Table B-55 Planned For synchronization rule properties

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	<none>

Table B-56 Planned For property value mappings

Item value	External value
Inception Iteration I1	Inception/I1
Elaboration Iteration E1	Elaboration/E1
Construction Iteration C1	Construction/C1
Transition Iteration T1	Transition/T1

Note: The mapping values used in Table B-56 are valid when using OpenUP. If other processes are used in the ClearQuest ALM solution, or in Rational Team Concert and Rational Quality Manager, the values must be edited.

6. Click **Save** to save and deploy the updated synchronization rule.

Modifying the ALMTask and ALMActivity forms

To use the added fields for DueDates and Estimates on the ALMTask and ALMActivity records, these fields must be added to the record type forms. We suggest adding them to the Resolution tab. Note that these forms changes must be reapplied after package upgrades.

Modifying the ALMActivity synchronization rule

The com.ibm.rational.clearquest.CQALM.ALMActivity synchronization rule must be updated to include mapping of the RTCEstimate or Estimate and the RTCDueDate or DueDate fields and properties in the ClearQuest and Jazz repositories.

To update the synchronization rule:

1. Open the **com.ibm.rational.clearquest.CQALM.ALMAcitivty** synchronization rule.
2. Follow the steps in “Modifying the ALMTask synchronization rule” on page 612.

3. Define the property mappings for Estimate and Due Dates. The values are provided in Table B-57 on page 614, Table B-58, and Table B-59.

Table B-57 Property mappings added to the ALMActivity synchronization rule

ID	Item property	I/O	ID	Element property
	Estimate	Out		RTCEstimate
	Due Date	Out		RTCDueDate

Table B-58 Estimate synchronization rule properties - Estimate

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Timestamp/Days Transformer

Table B-59 Estimate synchronization rule properties - DueDate

Synchronization rule property	Property value
Reference synchronization rule	
Value transformer	Timestamp/Days Transformer

4. Click **Save** to save and deploy the updated synchronization rule.

Extended Jazz configurations

In this section, we describe alternate configurations to Rational Team Concert and Rational Quality Manager for ALM interoperability.

Extending the process configurations

In “Configuring ClearQuest ALM system-wide settings for interoperability” on page 571, we discuss that the ClearQuest ALM solution comes with a set of system-wide settings. These settings provide consistent classification across the integrated ALM solution. We also explain how to extend the ALMTypeLabels and ALMResolutionCodeLabels in Rational ClearQuest to match the corresponding labels in Rational Team Concert and Rational Quality Manager.

An alternate configuration is to keep the ready-to-use ALM system-wide settings and align the enumerations that are defined in Rational Team Concert and Rational Quality Manager.

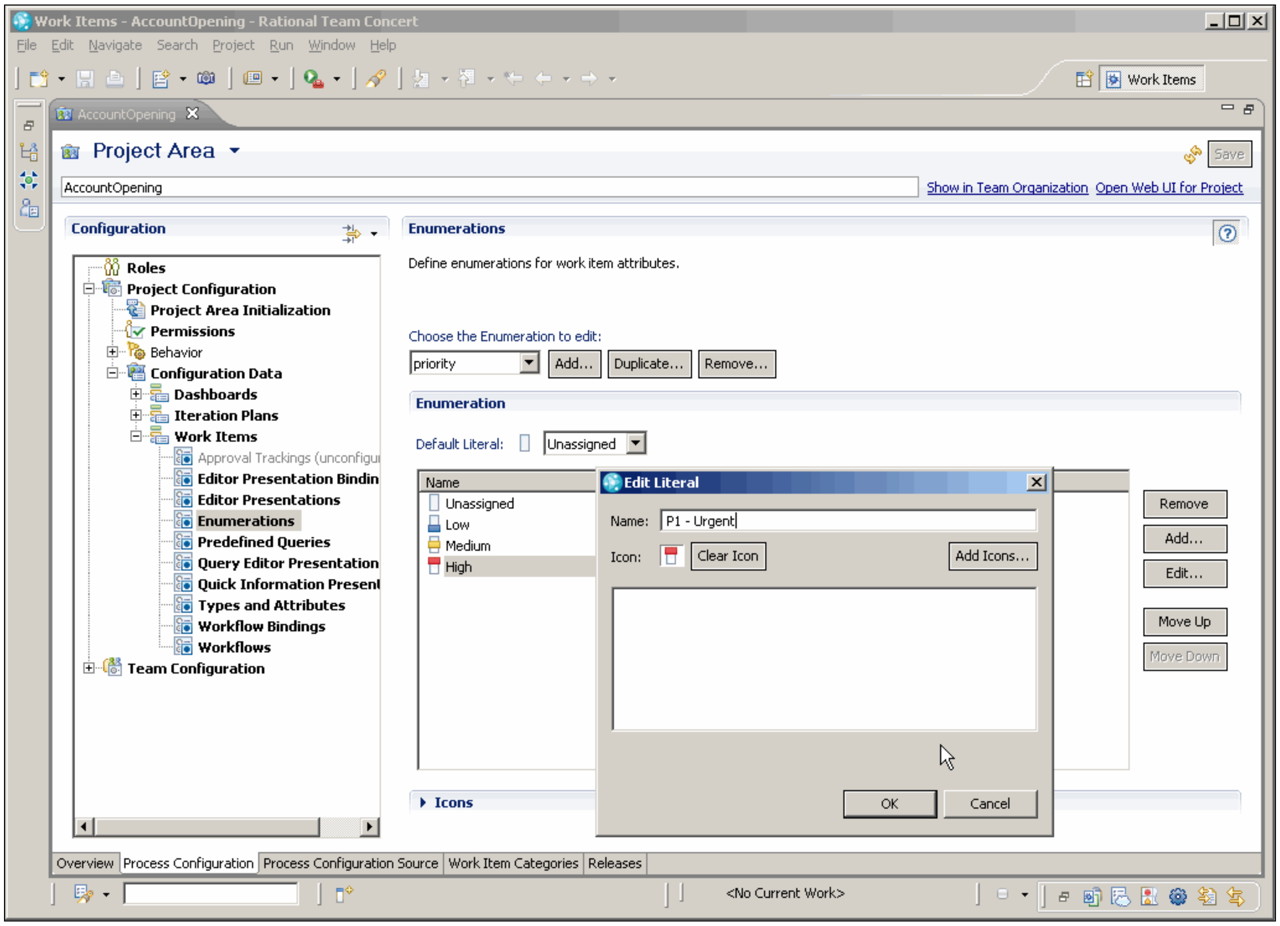
Modifying the priority values

To modify the Priority enumerations in the Team Area Process:

1. Open the AccountOpening Project Area from the Team Artifacts view.
2. Click the **Process Configuration** tab.
3. From the Configuration expand and select **Project Configuration** → **Configuration Data** → **Work Items** → **Enumerations**.
4. In the Enumeration view, select the **Priority** enumerations.

5. In sequence, select each Priority enumeration value, click **Edit**, and rename the priority values as indicated in the following list and shown in Figure B-17:
   – For High, type a name of `P1 - Urgent`.
   – For Medium, type a name of `P2 - Important`.
   – For Unassigned, type a name of `P3 - Moderate`.
   – For Low, type a name of `P4 - Low`.
6. From the Default Literal list, choose **P3 - Moderate**.
7. Click the **Save** button to save and deploy the changes.

Important: The ClearQuest Connector synchronization rules require a literal match between Priority enumerations in Rational ClearQuest, Rational Team Concert, and Rational Quality Manager.



AccountOpening

Define enumerations for work item attributes.

Choose the Enumeration to edit: priority

Default Literal: Unassigned

Unassigned
Low
Medium
High

Name: P1 - Urgent

Icon:

Figure B-17 Modifying Priority labels in Rational Team Concert to define a consistent classification across the integrated ALM solution

ALMResolutionCodeLabels

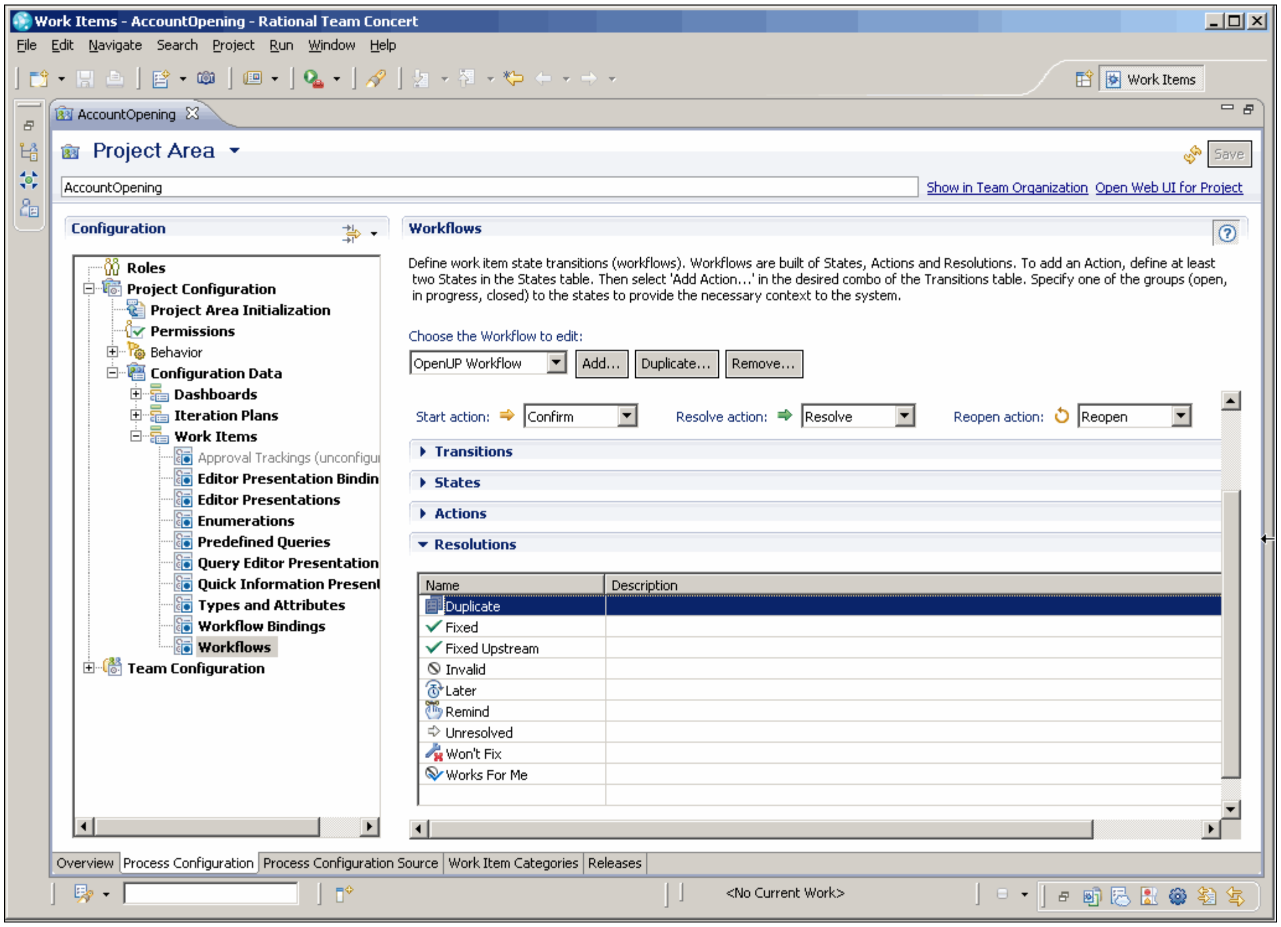
The ClearQuest ALM solution and the Jazz process definitions both contribute with unique resolution codes.

The following resolution codes are provided by the ClearQuest ALM:

- Complete
- Duplicate
- Fixed
- Rejected
- Unreproducible
- Works as Designed

To modify the resolution codes in Rational Team Concert:

1. Click the **Process Configuration** tab in the AccountOpening Project Area (Figure B-18).
2. Expand **Project Configuration** → **Configuration Data** → **Work Items** → **Workflows**.
3. Modify the Resolutions list, by choosing **Remove** and **Edit** to create a match between the available resolution codes in ClearQuest ALMResolutionCodeLables and the Resolutions in the Jazz process configuration.
4. Click **Save** to save and deploy the changes.



AccountOpening

Define work item state transitions (workflows). Workflows are built of States, Actions and Resolutions. To add an Action, define at least two States in the States table. Then select 'Add Action...' in the desired combo of the Transitions table. Specify one of the groups (open, in progress, closed) to the states to provide the necessary context to the system.

Choose the Workflow to edit: OpenUP Workflow

Start action: Confirm
Resolve action: Resolve
Reopen action: Reopen

Name	Description
Duplicate	
Fixed	
Fixed Upstream	
Invalid	
Later	
Remind	
Unresolved	
Won't Fix	
Works For Me	

Figure B-18 Modifying the resolution codes in the process configuration

Extending the work items definitions

The ClearQuest ALM solution provides work items to govern work processes. These work items include Request, Task and Activity record. Each of these work items can take a type label, for example, an Enhancement or a Defect. The type labels are also used to categorize the type of work, for example Define Requirements, Design Architecture, Implementation, or Test.

In Rational Team Concert, the work item types are specified as, for example, Task, Defect, or Enhancement.

ALM interoperability must map both the ClearQuest Task/Activity classification and the subtype classification to the Jazz Work Items types. The default mapping used in this example maps ALMTasks to Work Items of type Task, and ALMActivities to Work Items of type Defect. The default mapping also maps the Task/Activity label types to the Work Item Tags property.

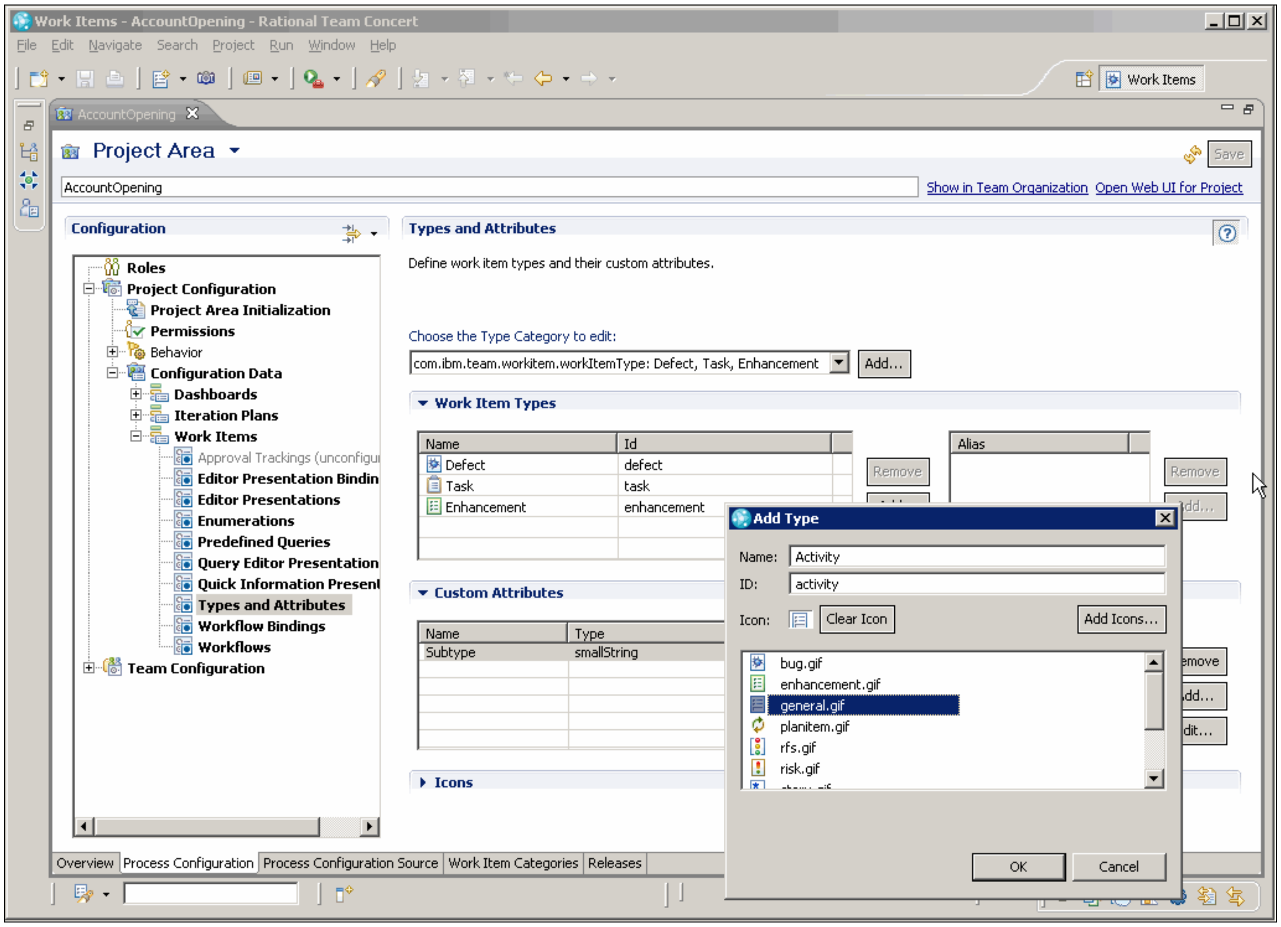
Extending the work item type definitions, and their custom attributes, in Rational Team Concert and Rational Quality Manager can create a more flexible ALM interoperability mapping.

Adding an Activity Work Item type

To add a new Activity Work Item type:

1. Click the **Process Configuration** tab in the AccountOpening Project Area.
2. Expand **Project Configuration** → **Configuration Data** → **Work Items** → **Types and Attributes**.
3. In the Type Category list, select **com.ibm.team.workitem.workItemType**.
4. Click **Add** to create a new Work Item Type.
5. In the Add Type window, for Name, type `activity`. For Icon, type `general.gif`. Click **OK** to add the new type.
6. Click **Save** to save and deploy the changes.

Figure B-19 shows the new Activity types that are added to the process specification.



Define work item types and their custom attributes.

Choose the Type Category to edit: com.ibm.team.workitem.workItemType: Defect, Task, Enhancement

Name	Id
Defect	defect
Task	task
Enhancement	enhancement

Name	Type
Subtype	smallString

Name: Activity
ID: activity
Icon:

bug.gif
enhancement.gif
general.gif
planitem.gif
rfs.gif
risk.gif

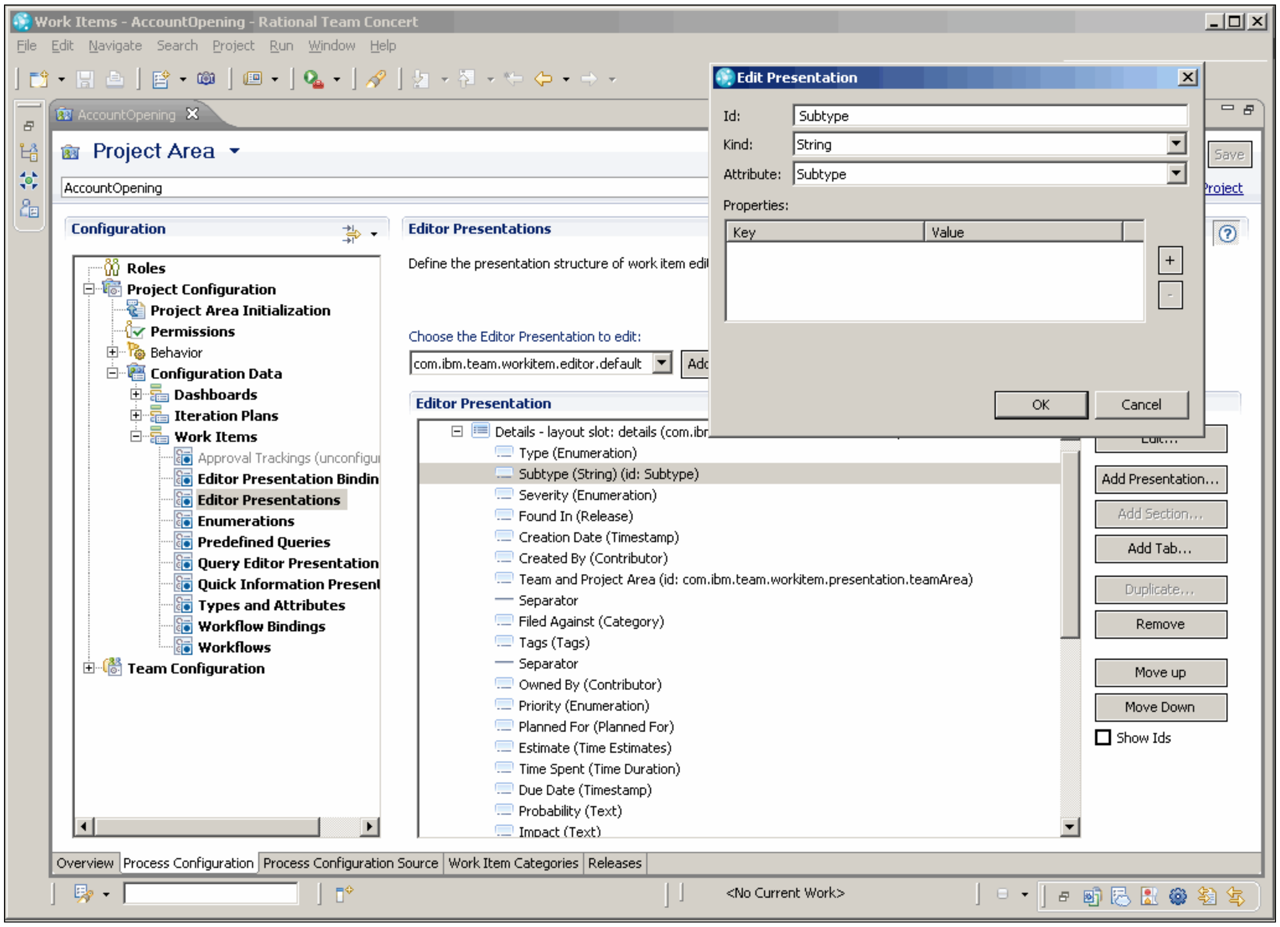
Figure B-19 Adding a new Activity work item type to the process configuration

Adding a Subtype property

To add a new Subtype property to a work item:

1. Click the **Process Configuration** tab in the AccountOpening Project Area.
2. Expand **Project Configuration** → **Configuration Data** → **Work Items** → **Types and Attributes**.
3. Click **Add** to create a new Custom Attribute.
4. In the Add Custom Attribute window, type a name of `Subtype`, and choose the type **smallString**. Click **OK** to add the new attribute.
5. Expand **Project Configuration** → **Configuration Data** → **Work Items** → **Editor Presentations**.
6. In the Editor Presentation view, expand **Overview** → **Details** → **Type (enumeration)**.
7. Click **Add Presentation** to add a new attribute to the Work Item editor.
8. In the Add Presentation window, for Id, type `Subtype`, for Kind, select **String**, and for Attribute, select **Subtype**. Click **OK** to save the editor presentation changes.
9. Click **Save** to save and deploy the changes

Figure B-20 shows the new Subtype attribute that is added to the Work Item editor presentation.

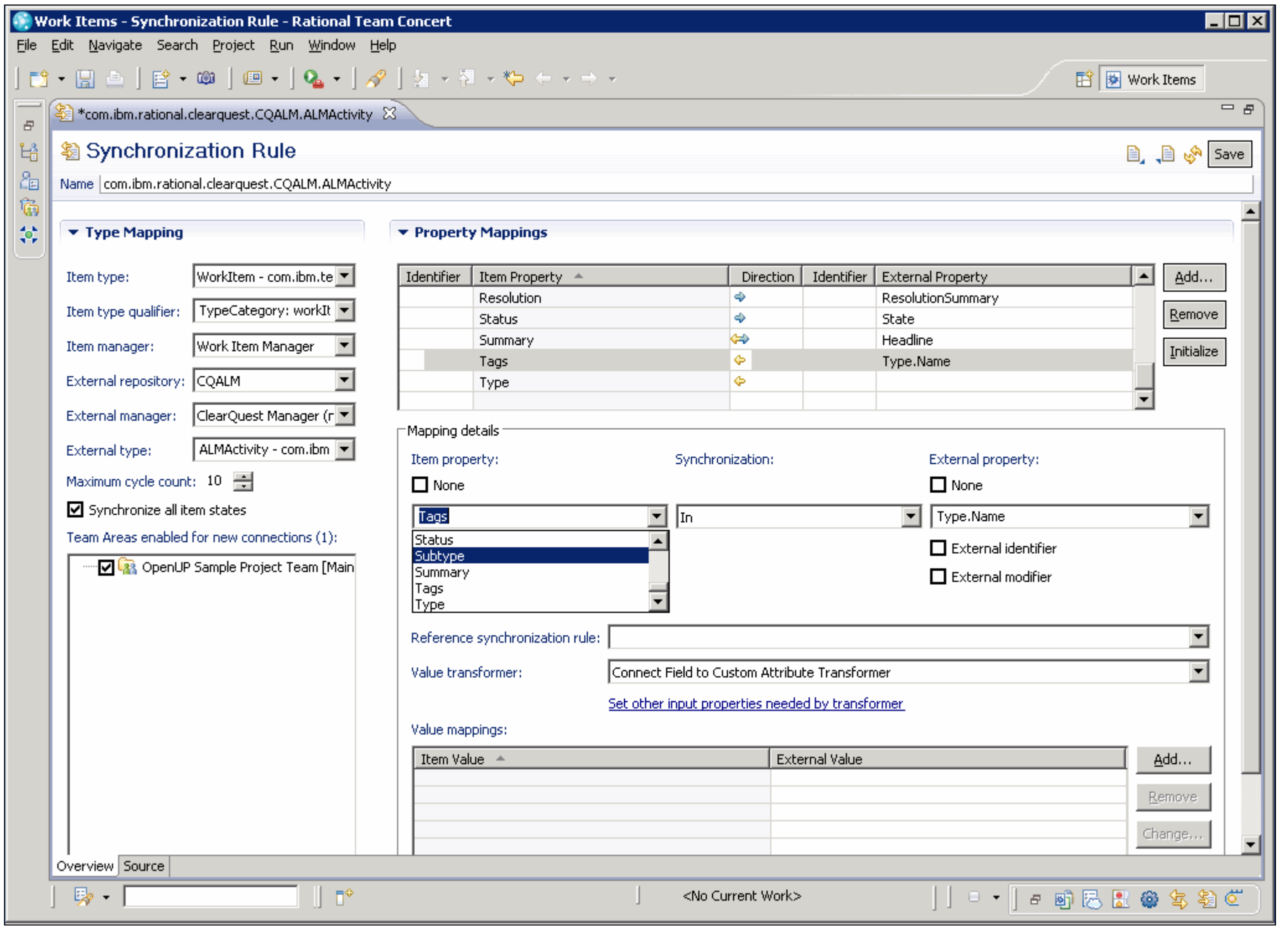


Id: Subtype
Kind: String
Attribute: Subtype
Properties:

Figure B-20 Adding the Subtype attribute to the Work Item editor presentation

Modifying the synchronization rules

The synchronization rules must be updated to include the Activity types and Subtype attribute mappings. Figure B-21 shows the ALMActivity synchronization rules updated.



*com.ibm.rational.clearquest.CQALM.ALMActivity

Synchronization Rule

Name com.ibm.rational.clearquest.CQALM.ALMActivity

Type Mapping

Item type:	WorkItem - com.ibm.te
Item type qualifier:	TypeCategory: workIt
Item manager:	Work Item Manager
External repository:	CQALM
External manager:	ClearQuest Manager (r
External type:	ALMActivity - com.ibm
Maximum cycle count:	10

☑ Synchronize all item states

Team Areas enabled for new connections (1):

☑ OpenUP Sample Project Team [Main

Property Mappings

Identifier	Item Property	Direction	Identifier	External Property
	Resolution	→		ResolutionSummary
	Status	→		State
	Summary	↔		Headline
	Tags	←		Type.Name
	Type	←		

Mapping details

Item property: ☐ None — Tags (Status, Subtype, Summary, Tags, Type)

Synchronization: In

External property: ☐ None — Type.Name

☐ External identifier

☐ External modifier

Reference synchronization rule:

Value transformer: Connect Field to Custom Attribute Transformer

Set other input properties needed by transformer

Value mappings:

Item Value	External Value

<No Current Work>

Figure B-21 Modifying the ALMActivity Synchronization Rule to map the Type.Name property to the new Subtype property

Make the following changes to the com.ibm.rational.clearquest.CQALM.ALMActivity synchronization rule:

1. Open the **com.ibm.rational.clearquest.CQALM.ALMActivity** synchronization rule.
2. Select the mapping of the **Tags** item property and the **Type.Name** external property.
3. Replace the Tags item property with the new **Subtype** item property.
4. Select the mapping of the Type item property.
5. In the Value Mappings list, select **defect** mapping and click **Change**.
6. In the Define Value Mapping window, replace defect with the **activity** value. Click **OK**.
7. Click **Save** to save the changes to the synchronization rule.

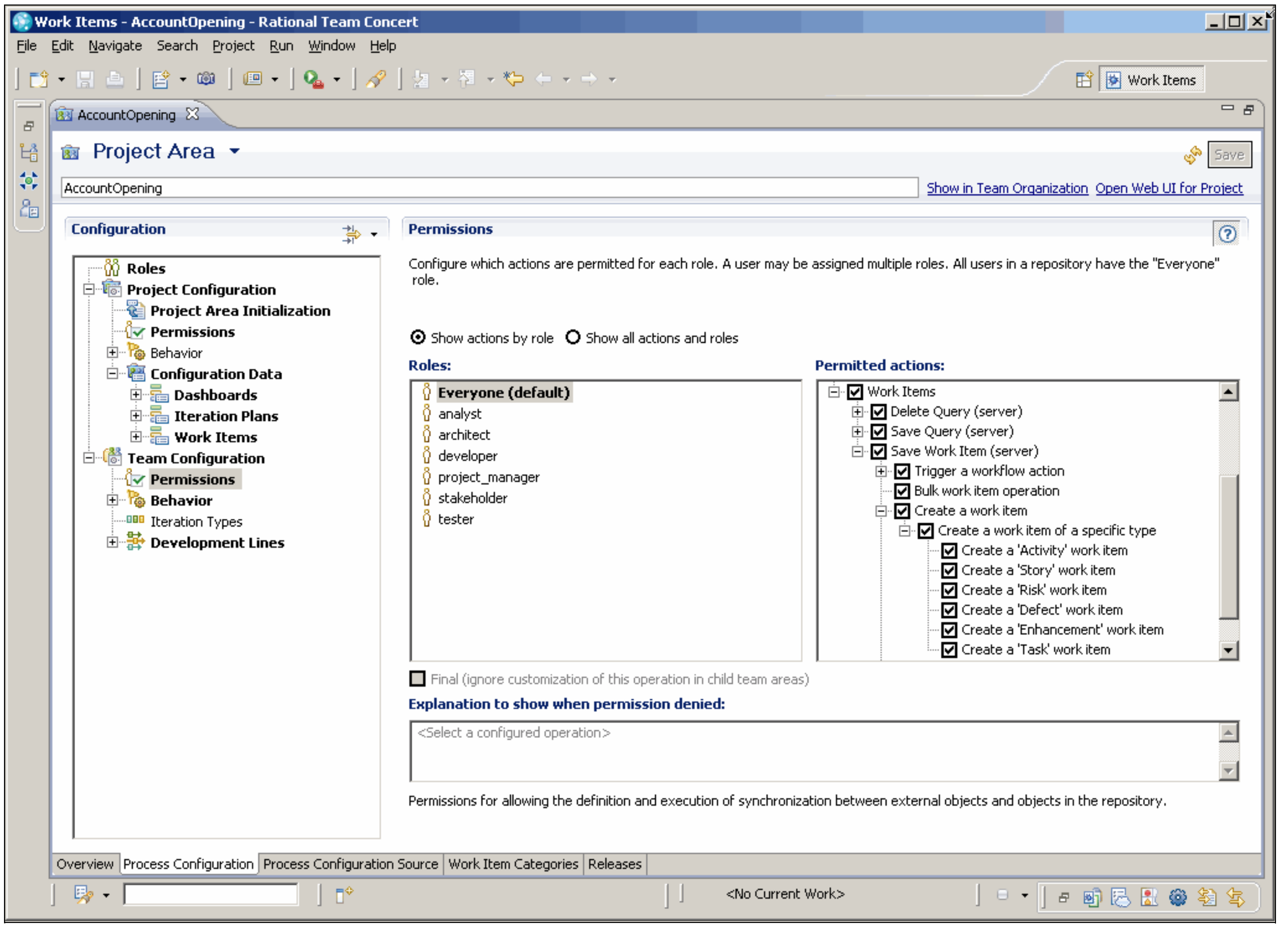
Make the following changes to the com.ibm.rational.clearquest.CQALM.ALMType synchronization rule:

1. Open the **com.ibm.rational.clearquest.CQALM.ALMType** synchronization rule.
2. Select the mapping of the **Tags** item property and the **Type.Name** external property.
3. Replace the Tags item property with the new **Subtype** item property.
4. Click **Save** to save the changes to the synchronization rule.

Adding permission to create and modify new Activity work items

To add permissions to create and modify Activity Work Items do the following:

1. Click the **Process Configuration** tab in the AccountOpening Project Area.
2. Expand **Team Configuration** → **Permissions**.
3. In the Permissions view, select **Everyone**.
4. In the Permitted Actions section, expand **Work Items** → **Save Work Items** → **Create a work item** → **Create a work item of a specific type**. Validate that permissions are granted for a new Activity work item to be created and modified (Figure B-22).
5. Click **Save** to save and deploy the new permissions.



Configure which actions are permitted for each role. A user may be assigned multiple roles. All users in a repository have the "Everyone" role.

[✓ Show actions by role] Show all actions and roles

Roles:
- Everyone (default)
- analyst
- architect
- developer
- project_manager
- stakeholder
- tester

Permitted actions:
- ✓ Work Items
  - ✓ Delete Query (server)
  - ✓ Save Query (server)
  - ✓ Save Work Item (server)
    - ✓ Trigger a workflow action
    - ✓ Bulk work item operation
    - ✓ Create a work item
      - ✓ Create a work item of a specific type
        - ✓ Create a 'Activity' work item
        - ✓ Create a 'Story' work item
        - ✓ Create a 'Risk' work item
        - ✓ Create a 'Defect' work item
        - ✓ Create a 'Enhancement' work item
        - ✓ Create a 'Task' work item

☐ Final (ignore customization of this operation in child team areas)

Explanation to show when permission denied:

<Select a configured operation>

Permissions for allowing the definition and execution of synchronization between external objects and objects in the repository.

Figure B-22 Permissions granted to create new Activity work items

Adding a new Type category

By using the ClearQuest Connector and multiple synchronization rules, it is possible to map multiple ClearQuest record types to specific Work Item types. In the example in this book, we use the following mapping:

- ALMTask records <> Work Items of type Task
- ALMActivity records <> Work Items of type Defect (or to an added Activity type)

It is required that the type category mapping is uniquely specified in the synchronization rule for each mapping. Mapping multiple Rational ClearQuest record types to the same Work Item type, or the same Work Item type category, causes the synchronization rules to fail.

In some cases, it is beneficial to create groups of Work Item types, organized by Work Item Type Categories, and map a Rational ClearQuest record type to such a type category. Note that a single work item type can only be a part of one category.

To create a new Type Category for a Task, while keeping a Defect, Enhancement, and Activity type in the default Work Item category:

1. Open the **Select the AccountOpening Project Area** and click the **Process Configuration** tab.
2. Add a Type category:
   a. Expand **Project Configuration** → **Configuration Data** → **Work Items** → **Types and Attributes**.
   b. In the Work Items Types section, select the **Task** type and click **Remove**.
   c. In the Types and Attributes view, click the Add button to create a new Type Category.
   d. Enter a new Type Category Id, for example `com.ibm.team.workitem.workItemType.task`. Then click **OK**.
   e. With the new type category selected, in the Work Item Types section, click **Add**. Enter the name `Task`, select the **task.gif** icon, and click **OK**.
   f. In the Permitted Actions section, expand **Work Items** → **Save Work Items** → **Create a work item** → **Create a work item of a specific type**. Validate that permissions are granted for a new Activity work item to be created and modified (Figure B-22 on page 621).
   g. Click **Save** to save and deploy the new permissions.
3. Add a workflow for the new Type Category:
   a. Expand **Project Configuration** → **Configuration Data** → **Work Items** → **Workflows**.
   b. Select the **Default** workflow and click the **Duplicate** button.
   c. Type a new workflow ID, for example `taskWorkflow`, and click **OK**.
   d. Edit the name of the new workflow, for example Task Workflow.
4. Add a workflow binding:
   a. Expand **Project Configuration** → **Configuration Data** → **Work Items** → **Workflow Bindings**.
   b. From the Type Category list, select the **com.ibm.team.workitem.workItemType.task** category.
   c. From the Workflow list, select the **taskWorkflow**.
5. Click **Save** to save and deploy the changes to the process configuration.

To use the new type category in a synchronization rule, open the synchronization rule editor and choose the new type category in the Item type qualifier property. Save the changes to the synchronization rule.

C

Rational Build Forge adapter templates

The Rational Build Forge product provides adapter templates for all kinds of applications as listed in Table C-1.

Licensing: The templates for Rational ClearCase and Rational ClearQuest do not require a separate license key, but other application templates are licensed through the Rational Build Forge Adapter Toolkit.

You can find detailed information about adapters in Rational Build Forge in "Rational ClearCase, Rational ClearQuest, and Rational Software Analyzer adapters" on page 380.

Table C-1 Rational Build Forge adapter templates

Adapter template name	Description	Type
ClearCaseBaseline.xml	► Scans a directory in a Rational ClearCase view. ► Writes branch and version information reported by Rational ClearCase to the Bill of Materials report.	Source
ClearCaseByBaselineActivities.xml	► Creates a new baseline from the contents of a Rational ClearCase view. ► Compares the new baseline and the baseline from the previous adapter execution to identify change activity. ► For each change activity, writes the activity, files changed, user, date, comments, and version to the Bill of Materials report. ► For each changed file, writes change details (from the `diff` command output) to the Bill of Materials report.	Source

Adapter template name	Description	Type
ClearCaseByBaselineVersions.xml	► Creates a new baseline from the contents of a Rational ClearCase view. ► Compares the new baseline and the baseline from the previous adapter execution to identify changed files. ► For each changed file, writes the file name, version, date, user, and comments to the Bill of Materials report. ► For each changed file, writes change details (from diff command output) to the Bill of Materials report.	Source
ClearCaseByDate.xml	► Queries a Rational ClearCase view for changes between two dates. The default dates are the current time stamp and the time stamp of the previous adapter execution. ► For each changed file, writes the file name, version, date, user, and comments to the Bill of Materials report. ► For each changed file, writes change details (from the `diff` command output) to the Bill of Materials report.	Source
ClearCaseByLabel.xml	► Creates and applies a new label to the contents of a Rational ClearCase view. ► Compares the new label and the label from the previous adapter execution to identify changed files. ► For each changed file, writes the file name, version, date, user, and comments to the Bill of Materials report. ► For each changed file, writes change details (from the `diff` command output) to the Bill of Materials report.	Source
ClearQuestCaseByActivity.xml	► Finds Rational ClearQuest defect records that are associated with a list of Rational ClearCase activities. For each defect record that is found, it adds job information to resolve the defect record within Rational ClearQuest if the Rational ClearQuest status allows it to be resolved. ► Writes the files that are associated with Rational ClearCase activity IDs and the Rational ClearQuest defect status to the Bill of Materials report.	Defect
ClearQuestCaseByDate.xml	► Queries a Rational ClearCase view for changes between two dates. The default dates are the current time stamp and the time stamp of the previous adapter execution. ► For each changed file, looks for a CrmRequest hyperlink attribute that identifies a Rational ClearQuest change ID. Attempts to resolve the change ID by adding job information to resolve the defect record in Rational ClearQuest if the Rational ClearQuest status allows it to be resolved. ► For each changed file, writes the file name, defect ID, defect status, and any Rational ClearQuest errors to the Bill of Materials report.	Defect
CVSv1Baseline.xml	► Scans a CVS directory on a Rational Build Forge agent looking for changed files. ► Writes the changed file name, status, working version, repository version, and sticky tag to the Bill of Materials report.	Source
CVSv1ByDate.xml	► Queries a CVS view for changes between two dates. The default dates are the current time stamp and the time stamp of the previous adapter execution. ► Writes the change type, date, user name, version, and file name to the Bill of Materials report. ► For each changed file, writes change details (from the `diff` command output) to the Bill of Materials report.	Source

Adapter template name	Description	Type
CVSv1ByTag.xml	► Applies a new tag to a CVS module. Compares the differences between the newly tagged module and a module that is tagged during the previous adapter execution. ► Writes the file name, revision, state, date, time, change author, and commit comments to the Bill of Materials report. ► For each changed file, writes change details (from the **diff** command output) to the Bill of Materials report.	Source
CVSv2ByDate.xml	► Queries a CVS view for changes between two dates. The default dates are the current time stamp and the time stamp of the previous adapter execution. ► Writes the change type, date, user name, version, and file name to the Bill of Materials report. ► For each changed file, writes change details (from the **diff** command output) to the Bill of Materials report.	Source
PerforceByDate.xml	► Queries a perforce client for changes that occurred since the adapter execution. ► Writes the change, date, time, user, perforce client, and comments to the Bill of Materials report. ► Writes change details (from the **diff** command output) to the Bill of Materials report.	Source
PerforceByRev.xml	► Queries a perforce client for changes that occurred since the last repository revision. ► Writes the change, date, time, user, perforce client, and comments to the Bill of Materials report. ► Writes change details (from the **diff** command output) to the Bill of Materials report.	Source
Quota.xml	► Queries a folder to determine if any of its subfolders exceed a specified threshold size. ► For each subfolder, writes the folder size, owner, and last modified date to the Bill of Materials report. ► Writes to the Bill of Materials report a list of subfolders that exceeded the threshold size.	Source
StarTeamBaseline.xml	► Queries the folder for a StarTeam view to gather information about files. ► Writes the file name, status, revision, and branch to the Bill of Materials report.	Source
StarTeamByDate.xml	► Uses the StarTeam API to query a StarTeam view to identify changes between the current date and the previous adapter execution. ► Writes the changed files and directories, user, version, date, and change comments to the Bill of Materials report. ► Writes change details (from the **diff** command output) to the Bill of Materials report.	Source
SubversionByDate.xml	► Queries subversion for repository changes that occurred between a past date and the current date. ► Writes the change type, revision, user, file or directory, and change date to the Bill of Materials report. ► Writes the file name, status, revision, and branch to the Bill of Materials report.	Source

Adapter template name	Description	Type
SubversionByRev.xml	► Queries subversion for changes to a repository that occurred between the current revision and an earlier revision. ► For each change, writes the revision, user, change type, file or directory path, and change date to the Bill of Materials report. ► Writes change details (from the **`diff`** command output) to the Bill of Materials report.	Source
VSSByDate.xml	► Queries a visual source safe directory for changes between an earlier date and the current date. ► Writes change information for projects and files, which includes the project or file, version, user, date, time, project activity, file project and action information, to the Bill of Materials report. ► Writes change details (from the **`diff`** command output) to the Bill of Materials report.	Source

D

Code review rules

The Code Review for Java domain comprises rule categories that contain rules that focus on several aspects of software quality. Such aspects include design principles, globalization, Java 2 Platform, Enterprise Edition (J2EE), and Java 2 Platform, Standard Edition (J2SE™), best practices, J2EE security, and software performance. Java source code analyses apply these rules to identify code that does not conform to recognized standards.

This appendix includes the following sections:

Rule categories and subcategories reference

Table D-1 describes the rule categories and subcategories that belong to the Code Review for Java domain and that you can use in a Java code review.

Table D-1 Rule categories for Code Review for Java domain

Category	Subcategory	Description
Design principles		Contains rules for the design principles of object-oriented programming
	Complexity	Contains rules to prevent unnecessarily complex code
Globalization		Contains rules that are based on globalization coding best practices, which help to ensure that code runs correctly in localized environments
	Cultural formatting	Contains rules that address data formatting options that are used in different parts of the world
	Encoding	Contains rules that validate encoding for globalization
	Locale handling	Contains rules that validate locales for globalization
	String handling	Contains rules that validate string operations for globalization
	Translation	Contains rules that validate code for translation
	UI specific	Contains rules that validate user-interface layout and content for globalization
J2EE best practices		Contains rules based on the best J2EE development practices and supports Web projects that are targeted for IBM WebSphere servers
	Correctness	Contains rules to detect incorrect method calls
	Data race	Contains rules to detect method invocations that can cause data race conditions in J2EE applications
	Garbage collection	Contains rules to detect method invocations that can delay garbage collection
	Maintainability	Contains rules to detect code that might be difficult to maintain in J2EE applications
	Performance and scalability	Contains rules to detect method invocations that hinder the performance or limit the scalability of a J2EE application
J2EE security		Contains rules that validate compliance with Java security standards
	J2EE security	Contains rules that validate compliance with Java security standards in a J2EE perspective

Category	Subcategory	Description
J2SE best practices		Contains rules that validate code for compliance with J2SE best practices for Java development
	Abstract Window Toolkit (AWT)	Contains rules that detect issues that are related to using the AWT library
	Casting	Contains rules that detect issues that are related to casting and coercion
	Clonable	Contains rules that detect issues that are related to object cloning
	Comparison	Contains rules that detect issues that are related to comparing objects and testing object equality
	Conditional	Contains rules that detect issues that are related to the usage of conditionals
	Constructors	Contains rules that detect issues that are related to defining and implementing constructors
	Declaration	Contains rules that detect issues that are related to declaring constants, variables, and fields
	Exceptions	Contains rules that detect issues that are related to exception handling
	Initialization	Contains rules that detect issues that are related to the initialization of primitives and objects
	Loop	Contains rules that detect issues that are related to using loops
	Null	Contains rules that detect issues that are related to using null
	Portability	Contains rules that detect issues that are related to portability
	Reflection	Contains rules that detect issues that are related to using reflection
	Serialization	Contains rules that detect issues that are related to serialization
	Statement	Contains rules that detect general issues in statements
	Switch	Contains rules that detect issues that are related to using switch statements
	Threads	Contains rules that detect issues that are related to using threads
J2SE security		Contains rules that validate compliance with Java security standards
	J2SE security	Contains rules that validate compliance with Java security standards in a J2SE perspective
Naming		Contains rules for naming conventions for elements in Java source code
	Conflicts	Contains rules to ensure that elements are named consistently in Java source code

Category	Subcategory	Description
Performance		Contains rules that enforce suggestions for improving performance and reducing the memory footprint in Java applications
	Memory	Contains rules that detect performance issues that are related to memory usage
	Profiling	Contains rules that detect potential performance issues that are related to profiling activities
	Speed	Contains rules that suggest ways to improve the speed of Java code execution
Private API		Contains rules that locate APIs that do not belong in Java code
	Sun™	Contains rules that locate APIs that do not belong in Java code
	WebSphere	Contains rules that locate APIs that do not belong in Java code

Architectural discovery patterns

The Architectural Discovery for Java domain comprises four rule categories that contain the rules for identifying patterns and antipatterns in Java source code:

- Design Patterns rule category
- Object-oriented Patterns rule category
- Structural Patterns rule category
- System Patterns rule category

This section explores each of these categories, which describes the patterns that an architectural discovery analysis can detect automatically.

Design Patterns rule category

The Design Patterns rule category for architectural analysis contains common solutions or pitfalls in designing and writing source code, including some of the classic Gang of Four patterns. Table D-2 lists each pattern and its purpose in the Design Patterns rule category.

Table D-2 Design Patterns rule category

Pattern	Purpose
Decorator	The Decorator pattern adds responsibilities to an object dynamically, without changing its interface. The Decorator pattern acts as a wrapper because it implements the original interface, adds capabilities, and delegates work to the original object, so that you can use it as an alternative to creating a subclass. The architectural discovery algorithm identifies this pattern as consisting of two classes: decorator and the wrapped component.

Pattern	Purpose
Factory method	The Factory method pattern defines an interface for creating objects without knowing the class of the object it creates. Each Factory method pattern can define the class to be instantiated based on the input parameters and specifics of the situation. The architectural discovery algorithm identifies this pattern as consisting of a Creator class, Concrete Creator subclass, Product interface, and Concrete Product object. The Creator class specifies the interface for creating a product. The Concrete Creator subclass implements this interface by instantiating a Concrete Product object.
Marker	The Marker pattern declares a semantic attribute of a class. The architectural discovery algorithm identifies the Marker pattern as a single empty interface without methods or constants.
Observer/Observable	The Observer/Observable pattern communicates the changes in the state of an object to other system objects. The architectural discovery algorithm identifies this pattern as consisting of Observer and Observable. The Observable class maintains a list of Observer classes that it notifies when a state change occurs.
Singleton	The Singleton pattern ensures that a class allows only one object instance. The architectural discovery algorithm identifies the Singleton pattern as a class with a private constructor and a public static field or method that provides global access to the instance of a Singleton class.
Utility	The Utility pattern models a stateless utility function. The architectural discovery algorithm identifies Utility as a class with a private constructor that contains only static methods.
Visitor	The Visitor pattern performs specific operations on the elements of an object structure. The Visitor pattern allows additional operations without changing the classes of the elements on which they operate. The architectural discovery algorithm identifies the Visitor pattern as consisting of Visitor class, Concrete Visitor subclass, Element (optional) class, and Concrete Element subclass. The Visitor pattern is an interface that declares the Visit operation for every element. The Concrete Visitor subclass implements the Visitor interface and acts on each Concrete Element subclass.

Object-oriented Patterns rule category

The Object-oriented Patterns rule category contains patterns that show abstraction and inheritance trees. Table D-3 lists each pattern and its purpose in the Object-oriented Patterns category.

Table D-3 Object-oriented Patterns rule category

Pattern	Purpose
Abstraction	The Abstraction pattern represents a system concept. The architectural discovery algorithm identifies abstraction as an abstract class or an interface.
Inheritance tree	The Inheritance tree pattern is based on standard object-oriented inheritance relationships between objects in a system. The architectural discovery algorithm identifies inheritance as the implementation of an interface or an extension of a class.

Structural Patterns rule category

The Structural Patterns rule category contains patterns that show various types of structural elements, including structural antipatterns. Table D-4 lists each pattern and its purpose in the Structural Patterns category.

Table D-4 Structural Patterns rule category

Pattern	Purpose
Component cyclic dependency	A Component cyclic dependency pattern is a structural antipattern that consists of interdependent components. A cyclic dependency between components is considered a major architectural flaw. Such a dependency makes the code difficult to understand and maintain. More important, cyclic dependencies compromise software testing, parallel development, and reuse. Large-scale software with many cyclic dependencies is fragile and unstable.
Component global breakable	A Component global breakable pattern is a structural antipattern for a system component that is often affected when any other component is changed. Except for high-level concrete implementations, global breakables are undesirable because they indicate fragility and a lack of modularity in the system.
Component global butterfly	A Component global butterfly pattern is a structural pattern for an object that has many global dependents. Changes to a global butterfly often have a significant impact on the rest of the system. For this reason, a global butterfly should only be either a basic system interface or a utility class.
Component global hub	A Component global hub pattern is a structural antipattern for a component that has many global dependencies and many global dependents. A global hub is often affected when anything is changed, and it affects a significant percentage of the system when it changes. Global hubs are undesirable because they indicate fragility and lack of modularity in the system.
Component local breakable	A Component local breakable pattern is a structural antipattern for a component that has many immediate dependencies. Such a component carries excessive responsibility and is usually identified by many long methods. Breakables make the code difficult to understand, maintain, and reuse.
Component local butterfly	A Component local butterfly pattern is a structural pattern for a component that has many immediate dependents. Changes to a local butterfly often have a significant immediate impact on the rest of the system. For this reason, a local butterfly should only be either a basic system interface or a utility class.
Component local hub	A Component local hub pattern is a structural antipattern for a component that has many immediate dependencies and many immediate dependents. Such a component carries excessive responsibility and serves as a utility or commonly used component. Hubs make the code difficult to understand, maintain, and reuse. Hubs also make the code fragile and unstable.
Package cyclic dependency	A Package cyclic dependency pattern is a structural antipattern that consists of interdependent packages. A cyclic dependency between packages is considered a major architectural flaw. Such a dependency makes the code difficult to understand and to maintain. More importantly, cyclic dependencies undermine testability, parallel development, and reuse. Large-scale software with many cyclic dependencies is fragile and unstable.

Pattern	Purpose
Package global breakable	A Package global breakable pattern is a structural antipattern for a package that has many global dependencies. Such a package carries excessive responsibility and usually contains several components with many global dependencies.
Package Global Butterfly	A Package global butterfly pattern is a structural pattern for a package that has many global dependents. Changes to a global butterfly often have a significant impact on the rest of the system. For this reason, a global butterfly package should only consist of either basic system interfaces or utility classes.
Package global hub	A Package global hub pattern is a structural antipattern for a package that has many immediate dependencies. Such a package carries globally excessive responsibility and serves as a utility or commonly used package. The package usually contains many components that have several global dependencies and dependents. A Package global hub pattern breaks the reusability of modules and makes the code difficult to understand and maintain.
Package local breakable	A Package local breakable pattern is a structural antipattern for a package that has many immediate dependencies. Such a package carries excessive responsibility. The package usually contains a large number of components or several components with many immediate dependencies. A Package local breakable pattern makes the code difficult to understand, maintain, and reuse.
Package local butterfly	A Package local butterfly pattern is a structural pattern for a package that has many immediate dependents. Such a package serves as a utility or commonly used package.
Package local hub	A Package local hub pattern is a structural antipattern for a package that has many immediate dependencies and dependents. Such a package carries excessive responsibility and also serves as a utility or commonly used package. The package usually contains many components that can have many immediate dependencies. Hubs make the code difficult to understand, maintain, and reuse. Hubs also make the code fragile and unstable.

System Patterns rule category

The System Patterns category contains one pattern, the Package pattern, that detects all packages in a project or resource working set of source code. Table D-5 describes this pattern and its purpose in the System Patterns rule category.

Table D-5 System Patterns rule category

Pattern	Purpose
Package	The Package pattern represents logical and physical groupings of classes and interfaces. The architectural discovery algorithm identifies all classes and interfaces that belong to a certain package and presents them as a group.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 636. Note that some of the documents referenced here may be available in softcopy only.

- *Application Lifecycle Management with SCLM on System z, SG24-7592*
- *Global Development and Delivery in Practice: Experiences of the IBM Rational India Lab*, SG24-7424
- *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM*, SG24-6399

Online resources

These Web sites are also relevant as further information sources:

- Jazz

  `http://jazz.net`
- The Eclipse Way (presentation)

  `http://www.eclipsecon.org/2005/presentations/econ2005-eclipse-way.pdf`
- Rational Unified Process

  `http://www-306.ibm.com/software/awdtools/rup/`
- JUnit

  `http://www.junit.org/`
- The Enterprise Unified Process

  `http://www.enterpriseunifiedprocess.com/`
- “Generalizing Specialists: Improving Your IT Career Skills”

  `http://www.agilemodeling.com/essays/generalizingSpecialists.htm`
- Examining the ‘Big Requirements Up Front (BRUF) Approach’

  `http://www.agilemodeling.com/essays/examiningBRUF.htm`
- Open Unified Process (OpenUP)

  `http://www.eclipse.org/epf/`
- OpenUP

  `http://epf.eclipse.org/wikis/openup/`
- “Application lifecycle management with ClearQuest 7.1.0.0: Part I”

  `http://ltsbwass001.sby.ibm.com/cms/developerworks/rational/library/edge/08/mar08/pampino-pierce/index.html`

- Business Intelligence Reporting Tool (BIRT) Eclipse project

  http://www.eclipse.org/birt/phoenix

- "IBM Rational ClearQuest general schema design performance"

  http://www.ibm.com/developerworks/rational/library/07/0717_patel/index.html

- Rational technical resources and best practices for the Rational software platform from IBM developerWorks

  http://www.ibm.com/developerworks/rational

- *Jazz Platform Technical Overview* (sign-on required)

  https://jazz.net/learn/LearnItem.jsp?href=content/docs/platform-overview/index.html

- Manifesto for Agile Software Development

  http://agilemanifesto.org

- "How to use the Scrum project management method with IBM Rational Team Concert and the Jazz platform"

  http://www.ibm.com/developerworks/rational/library/08/0701_ellingsworth/index.html?ca=drs-

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Collaborative Application Lifecycle Management with IBM Rational Products

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

Collaborative Application Lifecycle Management with IBM Rational Products

An IBM blueprint for Collaborative Application Lifecycle Management

Green-thread reference scenario showing the Agility-at-Scale approach

IBM Jazz products incorporated into an enterprise solution

In this IBM Redbooks publication, we provide a blueprint for Collaborative Application Lifecycle Management (CALM) and show how the new IBM Rational products support this evolving market. Driven by the business demands of global software delivery, many large organizations are seeking guidance in how to incorporate agile methods.

In this book, we provide a reference scenario and tool architectures for deploying the new IBM Rational products into an existing enterprise environment. We also provide a set of blueprints that define each of the key disciplines in the development life cycle to help you understand the domain and how the Rational products support the need of that discipline. Our primary focus is to highlight the value of CALM by providing a user view of the solution that is used to support a distributed enterprise development team that incorporates aspects of the "Agility-at-Scale" approach.

While most Redbooks publications provide details about a single product, this book provides a "green-thread" reference scenario that details one end-to-end path through an iteration of a software development project. The scenario demonstrates a reference architecture for an enterprise that uses the new Rational Jazz technology-based products along with the existing Rational team products. The scenario includes Rational Build Forge Enterprise Edition, Rational ClearCase, Rational ClearQuest, and Rational RequisitePro, and introduces Rational Quality Manager, Rational Requirements Composer, Rational Software Analyzer, and Rational Team Concert.

SG24-7622-00 ISBN 0738431974