Innovate2011
The Rational Software Conference
11th and 12th of October
Let's build a smarter planet.

Agile Test Management Practices with IBM Rational Quality Manager

Simon Norrington
IBM Rational Quality Management

Agenda

- **What is agile software development?**
  - How it differs from a traditional approach
  - Common terms and definitions
- **Testing in agile**
  - Test Driven Development
  - Concurrent testing
- **Rational**
  - Rational Quality Manager
  - Rational Team Concert
- **Questions**

What is agile software development?

Agile software development is -

A group of software development methodologies based on iterative and incremental development.

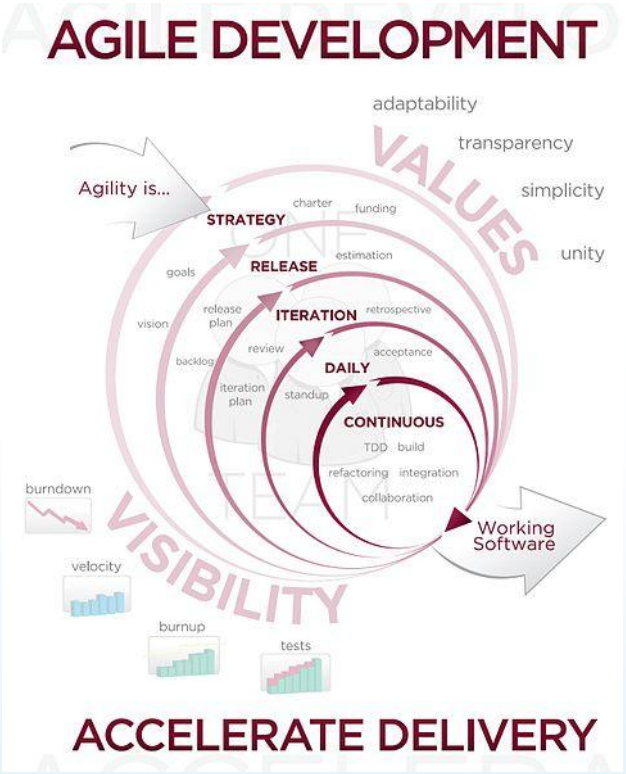
Where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

It promotes adaptive planning, evolutionary development and delivery; time boxed iterative approach and encourages rapid and flexible response to change.

It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

The Agile Manifesto introduced the term in 2001.

agilemanifesto.org



How is Agile development different from traditional methods?

Agile	Traditional
Iterative cycles	One long waterfall
Fewer people	Lots of people
Whole team	Many fixed teams
User stories	Use cases
Varying scope and requirements	Rigid scope and requirements
Time boxing, fixed iteration end dates	Delayed milestones, missed deadlines
Lots of customer/user involvement	Minimal user involvement
Test continuously	Test at the end

Terms and definitions used in this session

Scrum:

- One specific way to do agile development, it is a concept and practice functioning as project management utilizing whole teams, sprints and other agile concepts.

Whole team:

- A cross functional group which fulfills the roles for development, but with flexibility where a given person may fill different roles at different times.

Iteration:

- The fundamental time period used in agile, it is a fixed duration period in which most development activities are based on, and an iteration always produces working code.

Sprint:

- A particular type of iteration that is used in scrum. Typically a few weeks, a set of requirements are chosen to be implemented during the sprint.

Done:

- Better known as “DONE done”, this refers to the completeness of code produced by every single iteration/sprint. Done done means it has been fully implemented, tested, and accepted by the customer.

Terms and definitions used in this session

Backlog**:**

- The requirements for what is being produced (**product backlog**). A set of requirements are selected from the backlog for each iteration or sprint (**sprint backlog**).

Burndown**:**

- The work completed and remaining for either an iteration or a release.

Release**:**

- Either a final, or in some cases incremental, build which is the “finish line” for an agile project. A release also refers to the time period spanning multiple iterations.

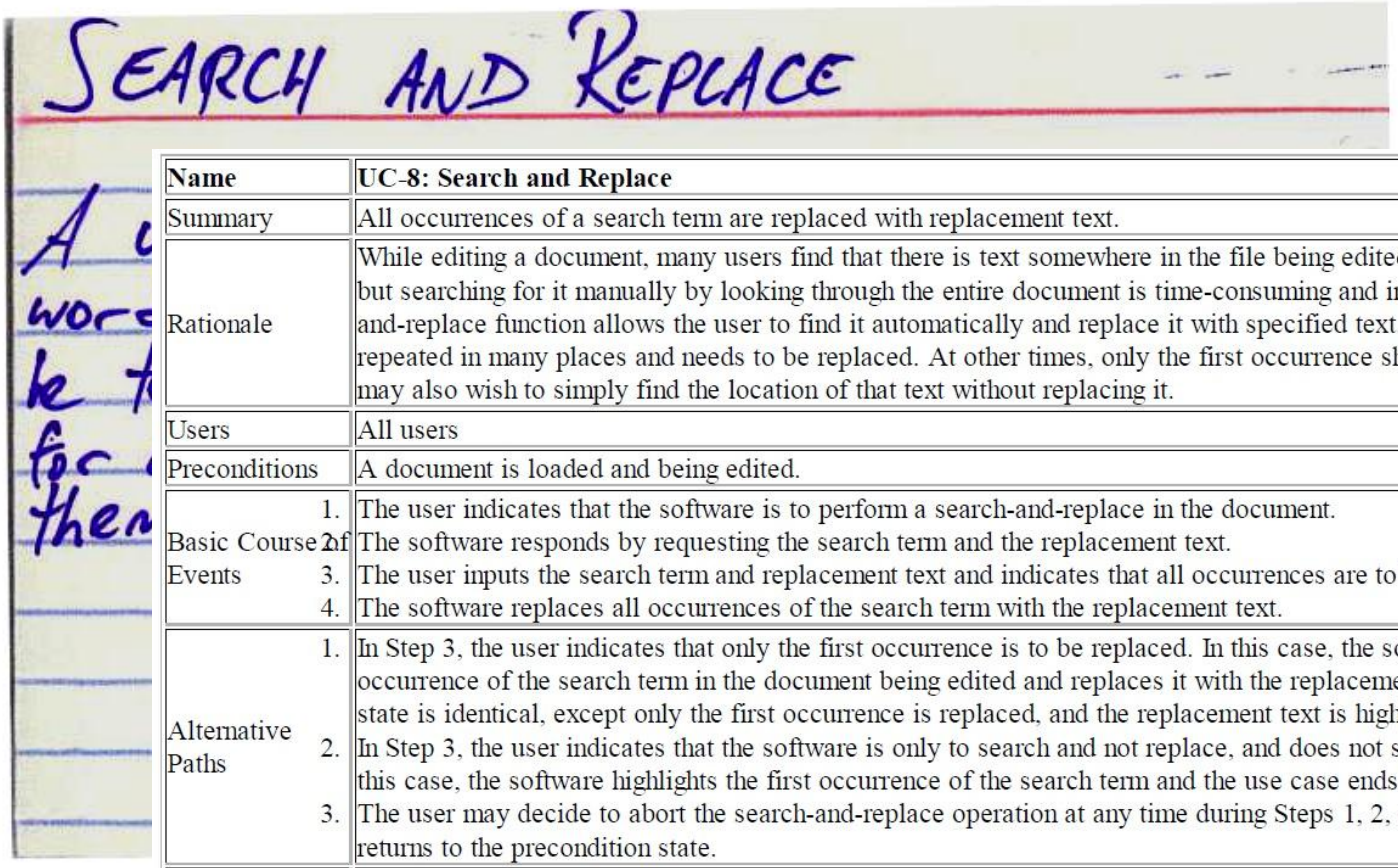
Build**:**

- An executable piece of software. An iteration/sprint will always result in a build.

Story**:**

- Also called a user story, a description from a user’s point of view of what they want to do, which is used to come up with requirements.

An example



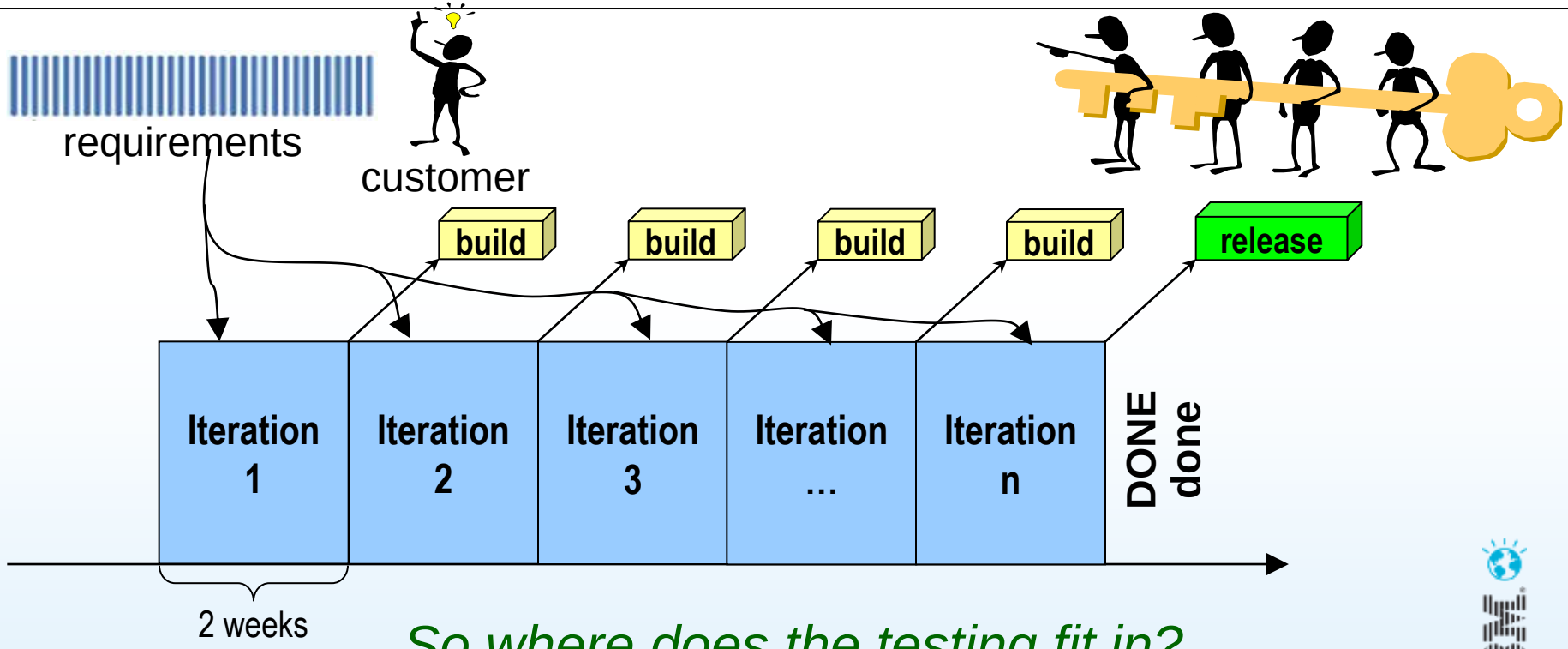
Name	UC-8: Search and Replace
Summary	All occurrences of a search term are replaced with replacement text.
Rationale	While editing a document, many users find that there is text somewhere in the file being edited that needs to be replaced, but searching for it manually by looking through the entire document is time-consuming and ineffective. The search-and-replace function allows the user to find it automatically and replace it with specified text. Sometimes this term is repeated in many places and needs to be replaced. At other times, only the first occurrence should be replaced. The user may also wish to simply find the location of that text without replacing it.
Users	All users
Preconditions	A document is loaded and being edited.
Basic Course of Events	1. The user indicates that the software is to perform a search-and-replace in the document. 2. The software responds by requesting the search term and the replacement text. 3. The user inputs the search term and replacement text and indicates that all occurrences are to be replaced. 4. The software replaces all occurrences of the search term with the replacement text.
Alternative Paths	1. In Step 3, the user indicates that only the first occurrence is to be replaced. In this case, the software finds the first occurrence of the search term in the document being edited and replaces it with the replacement text. The postcondition state is identical, except only the first occurrence is replaced, and the replacement text is highlighted. 2. In Step 3, the user indicates that the software is only to search and not replace, and does not specify replacement text. In this case, the software highlights the first occurrence of the search term and the use case ends. 3. The user may decide to abort the search-and-replace operation at any time during Steps 1, 2, or 3. In this case, the software returns to the precondition state.
Postconditions	All occurrences of the search term have been replaced with the replacement text.

Challenges with agile and agile testing

- Extensive and efficient collaboration is required
  - Email communication is error prone and not enough
- Planning is agile and continuous
  - Document-based plans alone are ineffective
- Many tests have to be created and executed quickly
  - Some, if not a lot, of test automation is required
- Tracking and communicating test results is critical
  - Spreadsheets are possible but not efficient

- Agile does <u>not</u> mean reckless or “wild” efforts

An over-simplified example of agile development



So where does the testing fit in?

Traditional vs. Agile Testing

Traditional Testing

Only the Test team is responsibility for delivering a quality application.

Traditional test team is comprised of

- Quality Manager
- Test Lead
- Tester

Most of the testing activities start towards the end of development

Agile Testing

The entire team is responsible for delivering a quality application

- Not a separate activity that only “testers” do
- Test activities are part of work plan for team

Testing is an integrated activity on an agile team

Testing activities are shared by entire team including

- creating, executing and building automated acceptance test
- Clarify stories, flush out hidden assumptions
- Help the team automate tests
- Make sure the acceptance tests verify the quality specified by the customer

Form an integral part of the continuous feedback loop that keeps the team on track

Testing in agile

Testing is not a phase

- It is done closely in conjunction with development

Testers use lightweight documentation

- Not traditional comprehensive test documentation
- Test plans are just enough to get the job done

Testers reuse existing artifacts to create test cases quickly

- User stories, use cases, story boards

Testers use tools to automate testing as much as possible

Testing is always focused on essential requirements

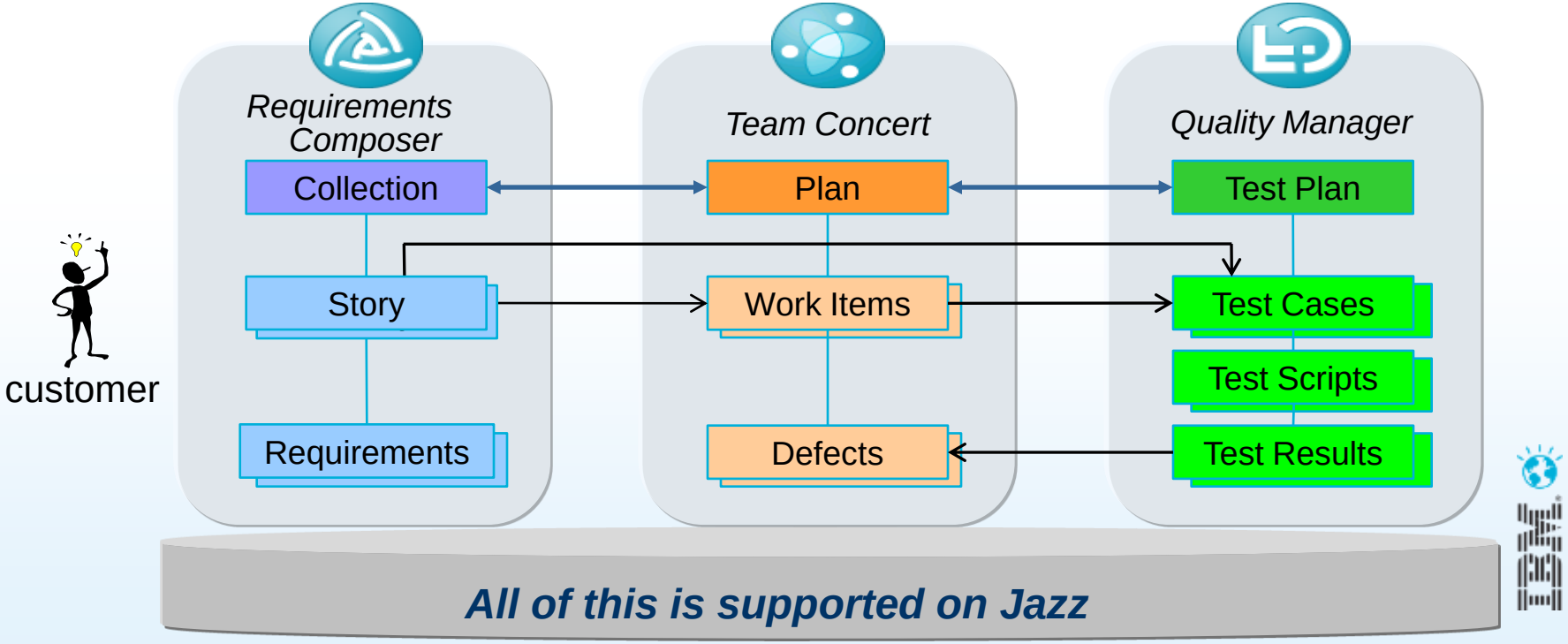
- Acceptance based testing

Everybody is involved in testing

- The Whole Team approach means anyone can test

Agile application lifecycle

- Testing is not a phase - it is integrated throughout the lifecycle



Using tools in agile testing

Tools increase the efficiency of identifying and tracking tasks

Type of work items used for an agile project include:

- **Requirement** – can be created in Quality Manager, Requirements Composer, or linked from another requirements tool
- **Quality** - work associated with a test artifact such as a test plan, test case, or test script
- **Defect** – used to track software defect found during testing
- **Review** - assigned to a user to review or approve a test artifact, such as a test plan or test case

Team Concert and Quality Manager access the same work items

- This can be used to link other items such as test cases and test scripts to help automate testing

The focus in agile testing

What to test, where to focus

- Like traditional testing, agile teams must test at multiple levels to ensure quality
- Agile teams often use acceptance based testing approaches
- Focus on customers requests

Tests often based on existing User Stories or Use Cases

- Provides end-to-end approach

Unit and Integration tests are also done

- Tools such as JUNIT and other developer test tools to automate unit / smoke / integration testing

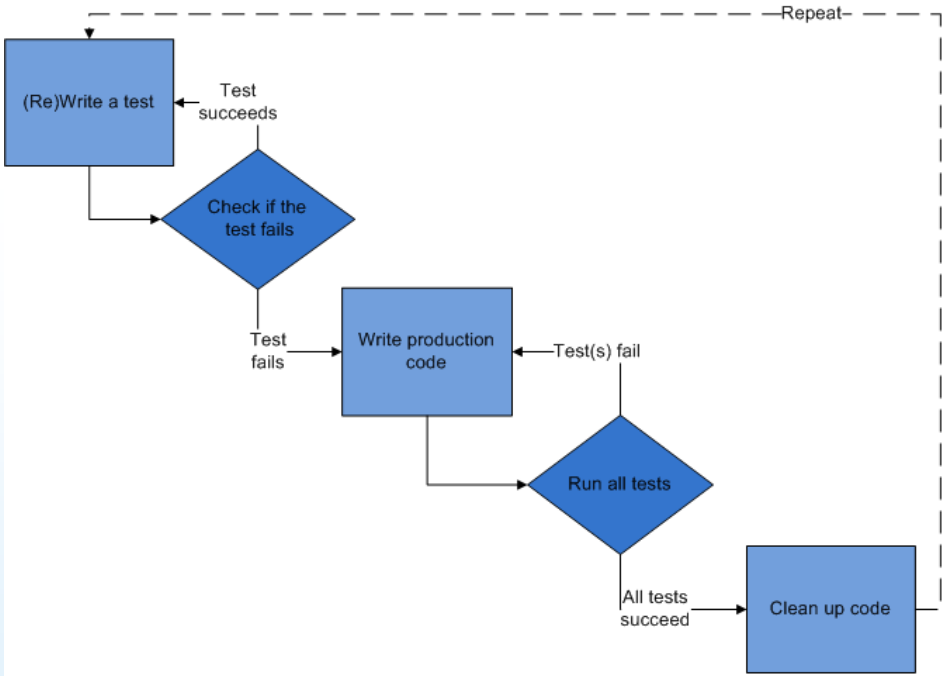
Test driven development addresses unit level quality

- Write the test first, then code to “fix” it

customer

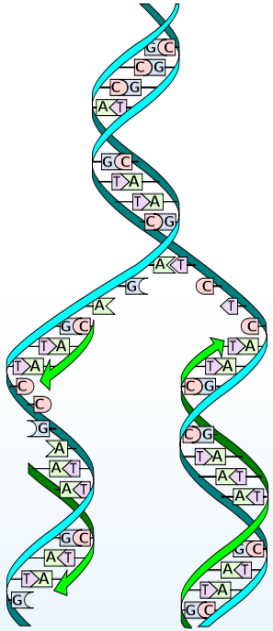
What is test driven development?

- Test driven development is a practice for developing code
  - It is a developer practice that involves testing
- How it works:
  - Tests are developed from the work items list
  - Code is written to cause the test to pass, then it is refactored
  - Code is only written in response to a failing test



What is concurrent testing?

- Concurrent testing is:
  - Testing at the same time as development
  - Testing as the code is delivered
  - Involving developers, testers, and other roles into testing
  - Testers trying to make the product better by working with the developers



- Concurrent testing is **not**:
  - Testing in “mini waterfall” cycle
  - Building up a bunch of tests for a series of big test runs
  - Defining a group of testers who are separated from development
  - Testers trying to break the product and find mistakes made by the developers

How is concurrent testing unique?

Tests are created and run at the end of every iteration

- For example, this could be every two weeks
- "Traditional" testing is done at the end of a development effort, just before the final release

Quality is measured very early

- Quality measurements are made visible to stakeholders throughout the project

Issues and risks are raised much earlier

- This saves time in development, as well as testing
- This leads to higher quality software

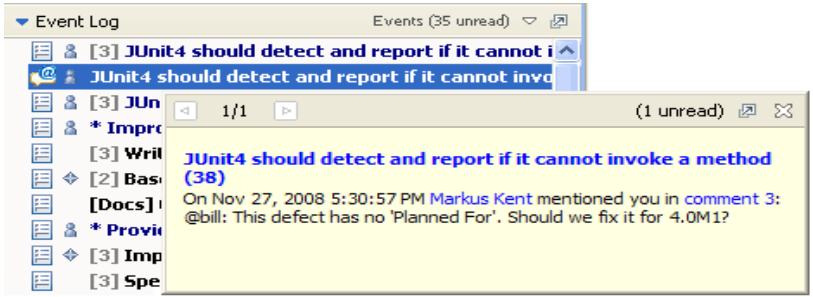
The tester's attitude is different, not as oppositional

- Testers and developers must collaborate
- Roles are not as absolute – developers may do some testing and a tester might be somewhat involved in the development

Tool support for agile concurrent testing

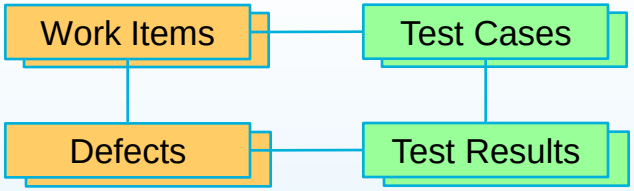
Rational Quality Manager provides:

- Test asset organization
  - Test plan, test cases, test scripts and test suites
- Control and reporting for other test automation tools



Rational Team Concert provides:

- Tracking and reporting for different types of work items
- Traceability through work item assignments
  - Ownership of work items
  - Identifying parent-child and related work items



Test management in agile

Agile teams need test management

- Making sure that there is not chaos when it comes to testing in agile

Agile teams use test management practices that support incremental and iterative development activities

- Multiple, smaller test plans are used
- Agile test management uses lightweight documentation

Agile allows teams to use evolutionary approach to organizing, planning, authoring, executing and reporting

- Agile teams embrace the need for concurrent testing

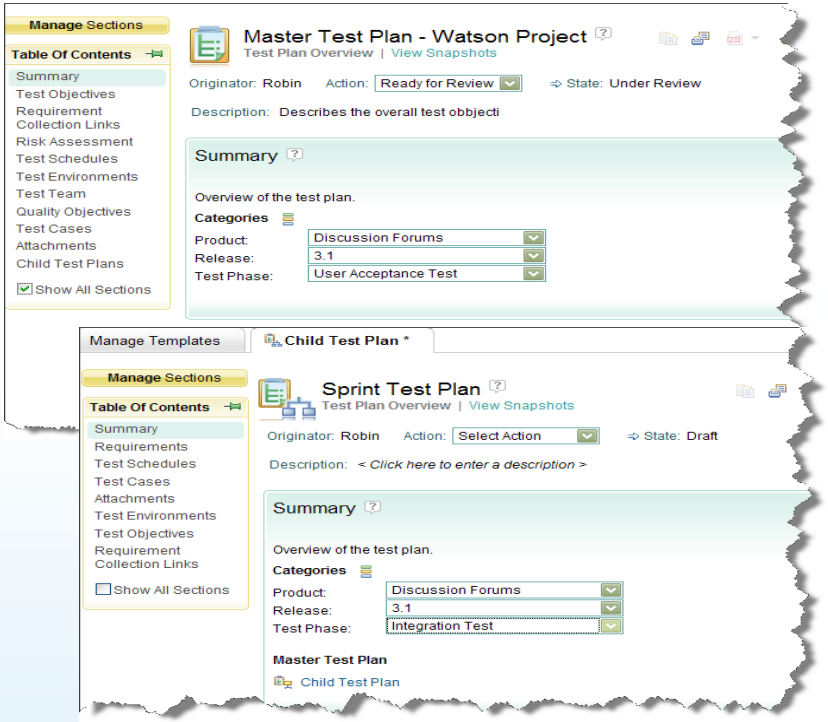
Agile test management involves the whole team – not just testers

Documentation in agile testing

Release Test Plan (the master test plan)

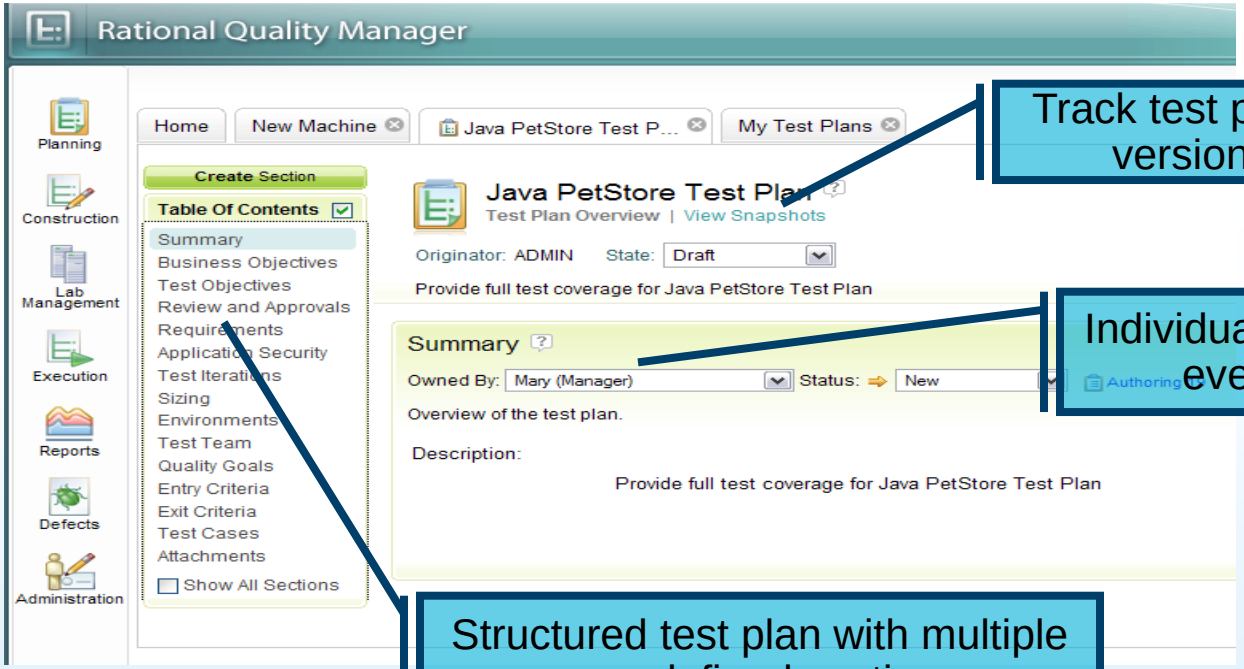
- Communicate test strategy
- Quality objectives
- Entry and exit criteria
- Resource estimates

- Iteration/Sprint Test Plan
  - Requirements validated during each Sprint
  - Test Schedule and Burndown status
  - Test Environments and Test Case
  - Reporting Overall Status
  - Defects, issues and tasks



Collaborative and adaptive test plan management

Test plans that are easy to create and evolve with the project



Track test plan history with version snapshots

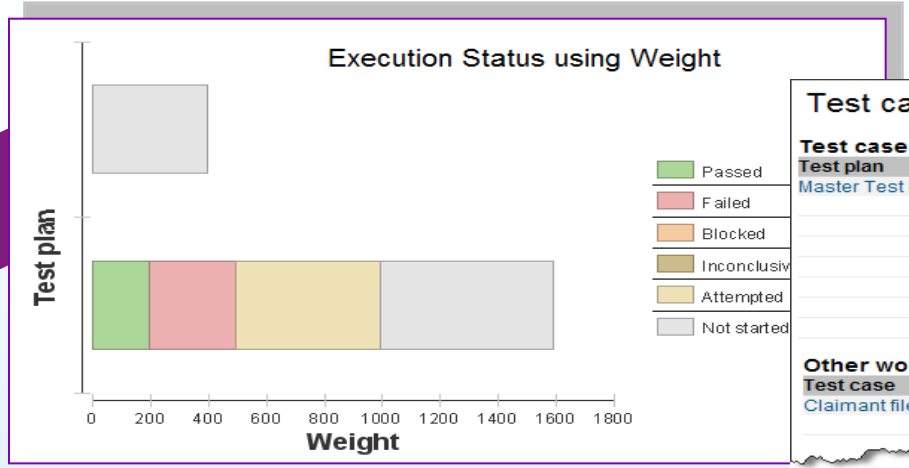
Individual ownership for every section

Structured test plan with multiple user defined sections

Rational Quality Manager support for agile testing

Testing in agile shows where you are at and where you are heading

- Rational Quality Manager reports and dashboards provide real-time information during each iteration or sprint
- Evaluating test results help the team identity additional areas to focus testing



Test case Review

Test case

Test plan	Test case
Master Test Plan - Watson Project	
	Claimant file associated by file number
	VCE LTS search contains apostrophe and hyphen
	VCE searches VADIR claimant using Wild Card
	TC: Assess Web site
	Claimant file associated by SSN
	VCE searches LTS claimant using Wild Card
	TC: Login Web

Other work items by test case

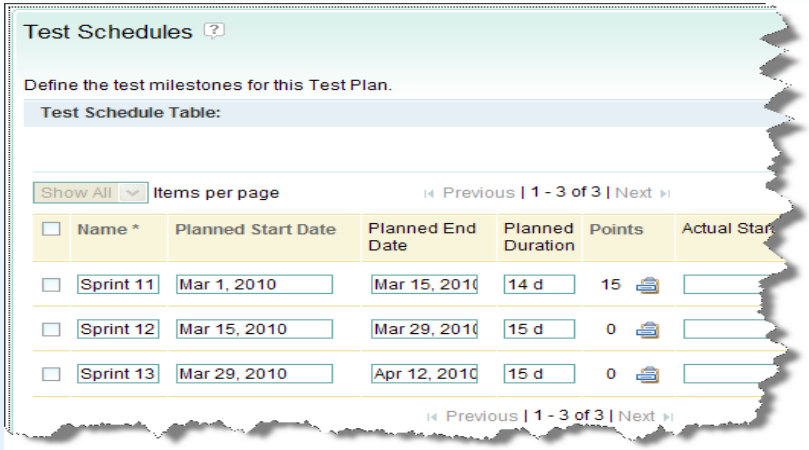
Test case	Work item ID	Work item name	Work item type
Claimant file associated by SSN			
	6	Must write Scenario for New Test Case SSN	authoring

More Rational Quality Manager support for agile testing

Acceptance testing often requires a set of test scripts to be run in order to validate the user scenarios or business requirements

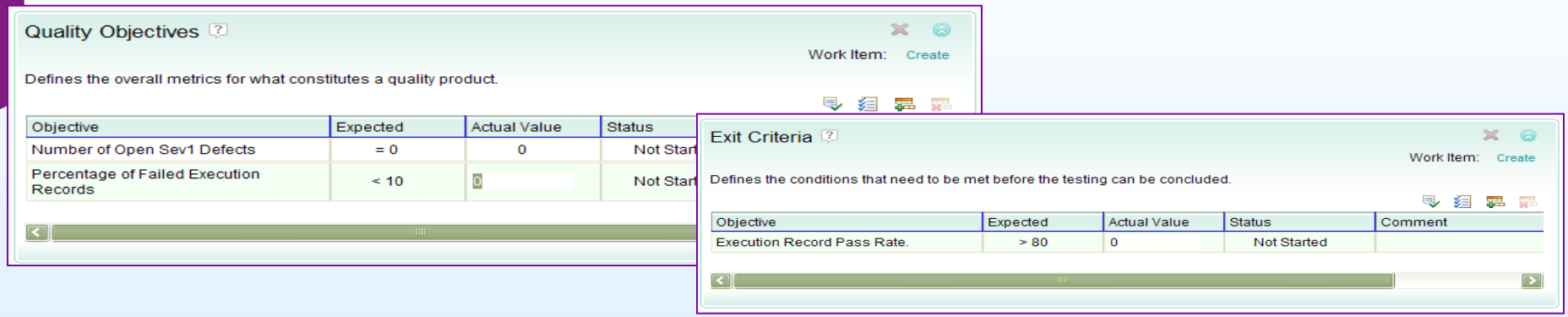
- Test Suites provide teams with an easy way to execute several test scripts (either in parallel or one after another)
- Test environments are used to capture the hardware and software configuration for a specific test case
- Test Execution Records are used to capture every test run results

- Testing follows agile's "Done done" principle
  - Rational Quality Manager allows you to specify a set of Quality Objectives for each test plan
  - Entry and Exit Criteria are defined to for each test plan



Test planning in agile using Rational Quality Manager

- The agile test planning approach:
  - Allows an agile team to define "Done" for each sprint/iteration
  - Links testing activities to rest of team
  - Allows the whole teams to reuse test assets
    - Reduces amount of rework
  - Jazz tools can provide dashboards for better collaboration



Getting up to date work progress information

Task management for individuals and the team

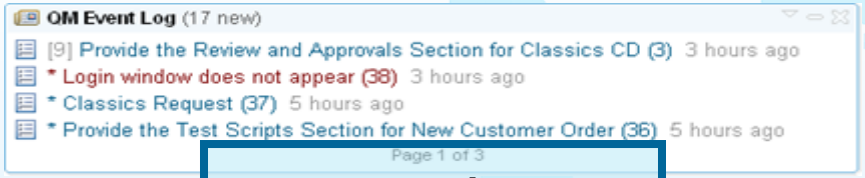
Challenge: **Assigning and coordinating test plan ownership and events across distributed teams**

Solution: **Visualize commitments, reduce rework, track tasks and monitor events**

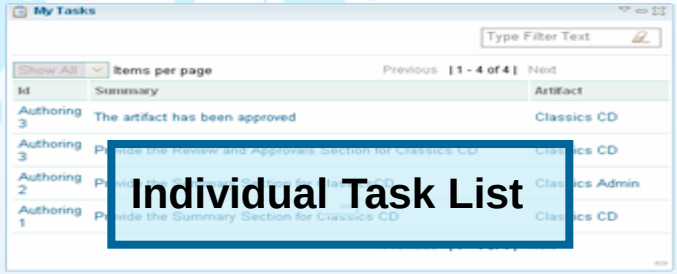
Requirements, application security

Test objectives, test case assignment and sign-off

Test iterations



Team event log



Individual Task List

Know what others are doing, know what others expect from you

What to watch out for with agile testing

Agile team may continue to use waterfall testing practices where Quality Assurance (QA) activities lags behind development activities

Common waterfall testing practices still sometimes used in agile:

- Testing not considered part of the agile team's responsibilities
- Iteration planning does not include the time required to execute testing activities in tests within the short time period
- Iteration backlog does not include testing planning, execution and reporting activities

Situations where waterfall testing practices are common:

- Teams are distributed, not co-located
- Test automation strategies are too hard to maintain
  - Teams do not have enough time to create and maintain automated test assets each iteration

Another thing to consider with agile testing

Whole team, <u>everybody</u> tests - *That's a good thing, right?*

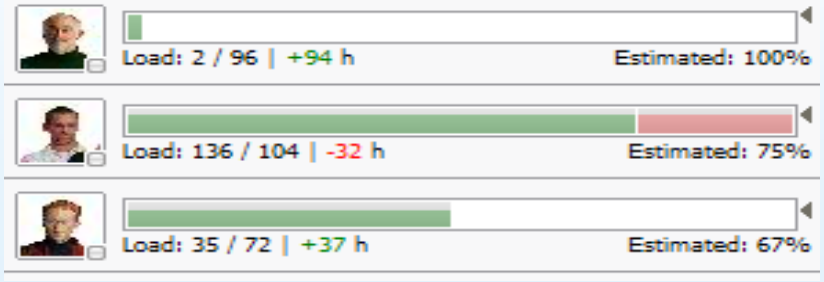
- Yes, because it maintains responsibility for quality in the product
- Yes, because it increases the resource constraints of traditional testing
- The risks is: Not everyone is a natural tester!

Example of a good tester versus a not-so-good tester

- A good tester will take the pessimistic view of whether the quality is good enough or not
- A not-so-good tester will always ***assume that everything works***, until proven otherwise
  - Unless they experience a bug themselves, and it affects them, they may resist the recognition that something must be done to fix it

Managing resources with Rational Team Concert

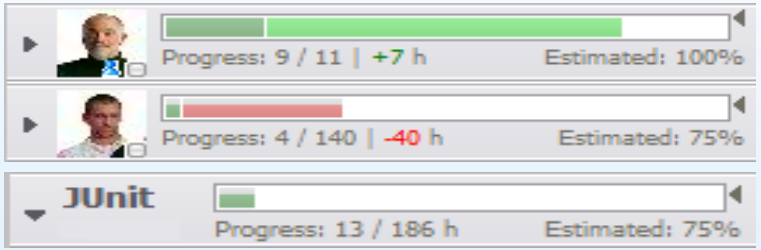
- ***Problem***: How to avoid overbooking team members
- ***Solution***: Team Concert uses work load bars to show how much work is assigned to a contributor, and whether he is overbooked or not.
- A work load bar shows:
  - Horizontally: The ratio of remaining work time (for an iteration) and upcoming work
  - Vertically: The percentage of estimated open work items
  - The less work items are estimated the less accurate the information is
  - This encourages estimation



Upcoming work: 2 hours / Remaining work time: 96 hours ← not overbooked
100% of open work is estimated
Upcoming work: 136 hours / Remaining work time: 104 hours ← overbooked
75% of open work is estimated
Upcoming work: 35 hours / Remaining work time: 72 hours ← not overbooked
67% of open work is estimated, but since 33% of the work items aren't estimated there will be very likely not much work time left

Managing schedules with Rational Team Concert

- ***Problem***: Team lead needs to know how the team is doing in terms of progress
- ***Solution***: Team Concert uses progress bars to reflect the progress of a team, a contributor, or other grouping elements (tags, category, etc.).
- A progress bar shows:
  - Horizontally: The ratio of resolved and total work. If an iteration information and work assignment is available a projection of this ratio onto the ratio of spent work time versus total work time is shown as well.
  - Vertically: The percentage of estimated open work items
  - The less work items are estimated the less accurate the information is



Resolved work: 9 hours / Total work: 11 hours

Spent work time: 17 / Total work time: 96

Resolved work: 4 hours / Total work: 140 hours

Spent work time: 53 / Total work time: 168 hours

Resolved work: 13 Hours / Total work: 186 Hours

No projection available

9 - 11/96 * 17 == 2 hours ahead

4 - 140/168 * 53 == 40 hours behind

A few proven techniques to manage scope

Some lessons learned from past agile projects:

- Have the customer focus on requirements and scenarios
  - This prevents rework and ensures development is always on the right track
- Create detailed storyboards, completed before each iteration
  - This increases the success of code produced by the iteration
- Estimate short iterations based on time to complete work
  - This will result in working consumable code
- Knowing what is new in each iteration build is crucial
  - It takes both developers and testers to efficiently know this
  - There is no time to waste on repeat or irrelevant tests

Agile collaboration essentials for quality management

Establish good relationships

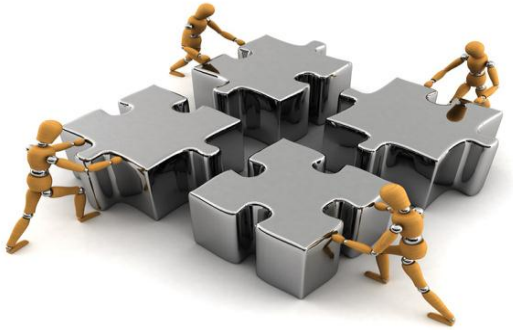
- Teams are built from people using many products and projects:
  - Learn who your product's team members are
  - Establish accessible repositories for all team members to use
  - Learn what other products your product will integrate with and who to contact on each team

Participation is key to staying informed

- Project level meetings (managers and team leads)
- Iteration meetings
- Scrum meetings
- Work product reviews
- Customer meetings and Beta programs
- Cross-functional team meetings
- Lessons learned session

Review of agile testing

- Agile Testing
  - Focuses on meeting quality from the customer's perspective
  - Centers on application capabilities that are really being used
  - Focuses on quickness, lightness, and helping the team deliver demonstrable quality code
  - Starts as early as possible, tests often as code becomes stable
  - Requires combination of manual and automated testing approaches to be effective
  - Is performed at multiple levels where unit, integration and acceptance tests are executed



Whole Team is responsible for quality