Community, modularity and empowerment in software delivery
White paper
February 2007

Rational. software

IBM

The future of software delivery.

Danny Sabbah, PhD
General manager, Rational Software

Contents

Introduction

"*Every generation laughs at the old fashions, but follows religiously the new.*"

—Henry David Thoreau

Today's software design and deployment professionals cannot ignore important current trends: service-oriented architecture (SOA), community-based software (a.k.a. open source software [OSS]) and globally distributed development (GDD).[1] One analyst expected 62 percent of Global 2000 firms to have implemented an SOA by the end of 2006[2]; another puts that number even higher: "9 of every 10 companies are adopting or have adopted service-oriented architectures and will exit 2006 with SOA planning, design and programming experience."[3] OSS is expected to be included in mission-critical software portfolios within more than three quarters of large international enterprises by the end of this decade, and many surveys indicate that a majority of us are using OSS in production *today*.[4]

Although SOA, OSS and GDD appear to be unprecedented phenomena, it must be remembered that they did not arise from nothing. Before embarking on a new architecture or a new software model or a new method of making and maintaining software, it is wise to consider carefully what SOA, OSS and GDD are, what they represent and where they come from. In this way we can better prepare ourselves for their eventual mass adoption.

Highlights

SOA is not entirely new. The promise of homogeneous services was behind the development of Enterprise JavaBeans Java™ technology, and before that, CORBA. Good SOA is open, standards-based, community-driven, governable, modular and easily available—some of these concepts have been around for a long time. For OSS, it's the same. In fact, in the context of software development, the underlying invention behind OSS is nonexistent.

The process is the key driver.

Open source software has been in use for 25 years—emerging largely from academia—but it has evolved into a major force only over the past three to five years, primarily because of a much more important trend: community-based software. The most visible features of OSS—pervasiveness, innovation, low cost—are redux. True value is added to software delivery by way of a three-step process:

1. Good-enough software modules encourage community participation to create OSS.
2. Innovation and component commoditization occur in an open development environment.
3. Standardization leads to industry adoption.

So while the product may be OSS, the process is the key driver—and that's community-based software.

So it's not SOA, OSS, GDD or other fashionable TLAs that really matter; it's their underlying precepts and processes: communities of interest, modular systems and empowerment. By focusing on the "ilities" (qualities, abilities) of the trends rather than the buzzwords themselves, it's possible to begin incorporating those benefits into our software right now (if we're not already doing so).

Highlights

IBM is working to incorporate broader communities of practitioners.

Incorporating today's rapidly broadening requirements and methodologies into our own software delivery—distilling time-tested best practices together with today's innovations to meet our precise needs—is the focus of this essay. The IBM Rational® group has already folded these lessons into the creation and maintenance of existing products and solutions, and IBM is hard at work expanding the scope of these principles in upcoming Rational product releases to incorporate broader communities of practitioners.

The remainder of this essay is divided into four sections: a brief description of past software delivery issues that have contributed to the current environment; identification of key trends that drive our industry today; an outline of the Rational group's future software delivery strategy; and a compendious summary.

Legacy software development

The reality of software and systems development today is that it can be faster to construct a new manufacturing plant than to deploy a new enterprise resource planning (ERP) system. It's faster, it seems, to integrate a parts supplier into a physical supply chain than it is to integrate a supplier into an IT supply chain.

For example, in May 1986, Toyota broke ground on its new North American assembly plant in Georgetown, Kentucky. Less than two years later (23 months, to be exact), the first automobile rolled off the assembly line. The average installation of an SAP ERP system takes almost three years (33.6 months).[5] Creating a structure for routing digital artifacts, ideas and processes around an IT system takes more time than building an assembly plant from a hole in the ground, routing automobile factory parts from around the world, and driving the first car out the door.

Highlights

The software industry is still in its adolescence.

The software industry is over 50 years old now, yet the gap between physical implementation and software implementation remains wide. The problem is that the software industry and software processes are still not fully formed—they are not yet mature. The software industry is still in its adolescence—experiencing a growth spurt—and one prime cause of that volatility is the very foundation upon which we run our software: the base of our IT infrastructure is evolving ever more quickly. And faster CPUs have allowed us to become inefficient in how we develop and deploy our software.

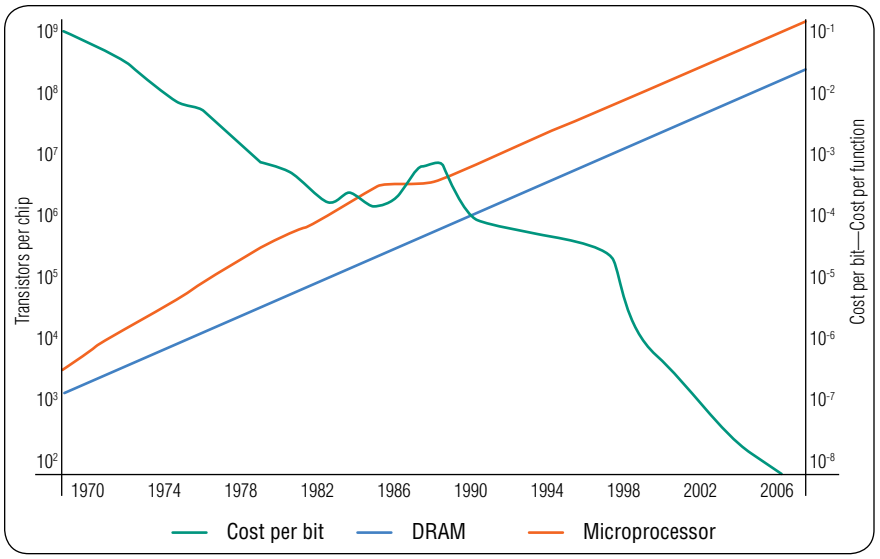

Figure 1. As memory and processing power increase, cost decreases.

Figure 1 shows the advance of processor speed in accordance with Moore's Law,[6] and it shows the price of computing power plummeting at the same time. Rapidly evolving hardware and software enable new applications of technology, and because of that we can now build and deploy software that wasn't even *possible*

Highlights

a few years ago. Having cheap processing and memory available to all developers has led to myriad new software frameworks that leverage these advances. Legacy applications that are not able to integrate with these newer frameworks have a disadvantage beyond limited scope—they themselves act as inhibitors of change.

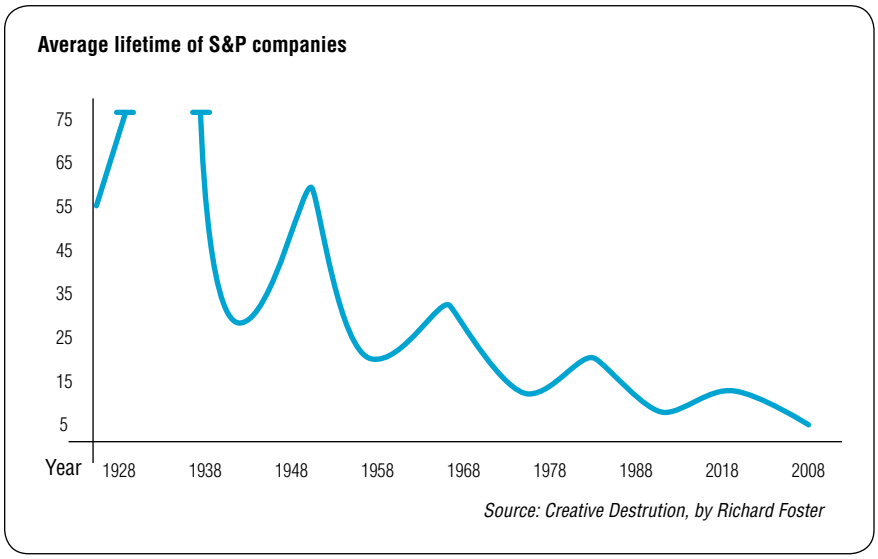

Figure 2. Average lifetime of S&P companies has decreased dramatically since 1930.

Today's marketplace looks very different from even 5 or 10 years ago.

As computing and communications technologies accelerate, so too does the average life span of an S&P 500 company. In 1930 the average life span was 75 years; today the average is 15 years. The rapidly evolving global marketplace and the principles of marketplace capitalization contain some valuable lessons for our consideration. Today's marketplace looks very different from even 5 or 10 years ago, and it's easy to think that all the rules have changed. But this is not so.

Highlights

Let's remember the "new economy" of the late 1990s and P/E ratios in the 2000s, which heralded a new reality in enterprise valuation. Then let's reflect on 2001, the year this was overwhelmingly disproved. Companies are forming and disbanding with ever-increasing speed, but, at least for now, it seems that the old rules still govern marketplace valuation.

The same lesson can be applied to understand how to approach the onrush of technological advancements occurring now. Because we are currently, quite plausibly, in the vortex of the computer technology tornado, it's easy to lose sight of our grounded experience in the fields of commerce, construction, software development and systems integration. If we try to step outside that tornado for a few moments, it's easy to see what's left in its wake.

Software architecture has not yet reached the same level of maturity of traditional building architecture; nor has software supply chain technology reached the same level of capability and robustness of the physical supply chain of manufacturing. Software engineering methodologies, project management paradigms, programming and scripting languages, and enterprise frameworks are all in flux. Yet, as we have learned from the stock market over the past 10 years, the principles governing the processes that lead to true value tend to remain the same.

Software engineering methodologies, project management paradigms, programming and scripting languages, and enterprise frameworks are all in flux.

Discerning the true value of existing processes and methodologies, understanding where to focus our energies, and knowing when we are on the right track are invaluable insights that inform us of what we should keep and what we should discard from the many options available today. The IBM Rational group has identified a number of trends that have informed strategic vision, and they warrant examination.

Highlights

Affordable broadband data access and social networking have had a profound effect.

Trends that matter today

Years from now, when we look back at the early 21st century, it's certain that one of the most important phenomena impacting not just the software community but the entire world will be the advent of affordable broadband data access and the social networking made possible by the Internet and cellular and wireless technologies. The availability of cell phones in emerging nations, for example, has already had a profound effect on many emerging countries' economies. And that advance has come exclusively through the spoken word. Imagine what will happen when those societies expand their commerce onto the Internet and begin developing intellectual property, freed from the traditional physical boundaries of time, space and capital resources.

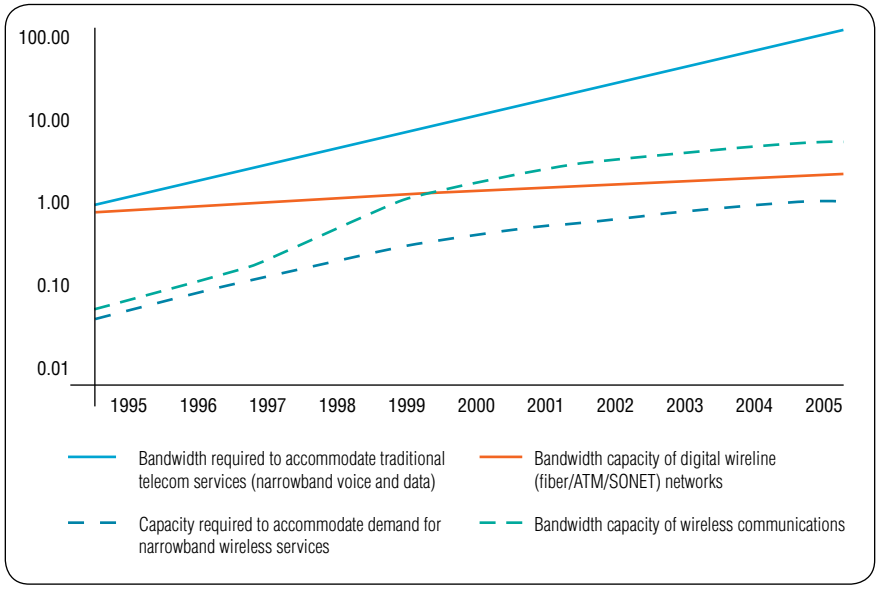

Figure 3. Bandwidth continues to increase over time.

Highlights

Broadband capacity continues to grow year over year, so we are only witnessing the beginning of this phenomenon. One result of increased bandwidth across different media is the realization that the data delivered, or the content of the data, is more important than the delivery mechanism. The telecom industry has already seen a convergence of data and information streams; it is no longer about television versus the Internet versus telephony. It's about the speed, reliability and availability of the data transfer, letting the end user decide the specific location, format and device hosting end-delivery content.

Widespread data access is changing our lives. Metcalfe's Law[7] states that the value of a network is proportional to the square of the number of users of the system: $\frac{n(n-1)}{2}$. In other words, the more people connected, the more value generated from the connections. We already have evidence of this from social networking.

New communities or subgroups form around interesting and innovative ideas.

As social networking communities have specific agendas, so software communities have equally transparent common goals: Linux® contributors are interested in their libraries; Eclipse contributors are interested in their projects. Software developers today have lots of opportunities to participate in existing communities or to create their own. In the absence of a formal governance structure, these communities are often spawned and governed by ruthless meritocracies. New communities or subgroups form around interesting and innovative ideas. Anyone who's monitored these discussion groups has witnessed the vitriol directed at the originator of malformed ideas or poor code.

Highlights

Technology communities are potent: the Rational software community, Microsoft® users, system administrators and others. In fact, such noncontributive communities, by their sheer size, are actually the largest force in determining de facto standards. Think about the community of C developers, Java developers, personal home page (PHP) users, Web site developers, et al., who have shaped not only their domains but the way in which we create software. The difference today is the speed at which these communities can form. Whereas it took C technology 10 years to gain popularity, it took Java technology only 5 years. And the relatively new Subversion technology is replacing 20-year-old Concurrent Version System (CVS) technology within its first 2 years of existence.

The difference today is the speed at which these communities can form.

Speed is but one dimension of today's communities. The tipping point occurs—as it always has—when a critical mass of leaders and adopters decide that this particular technology has what we need.[8] What is truly exciting in today's rapidly evolving software landscape is that a very small group of people can prove to be the tipping point—the community of change can be led by the needs of the individual.

Let's now look at a number of trends that have been fueled by easily available bandwidth and connectivity: service-oriented architecture, community-based software and the environment that governs their success. Within these trends we will find a number of time-tested principles at work, including the important distillation process that reduces a complex technology to its essence, which is sort of like applying Occam's razor to software issues. It's important to remember, for example, that Linus Torvalds did not invent an operating system; he simplified existing UNIX® modules. SOA, covering many complex technologies, is

Highlights

at its core a very simple concept: atomic homogeneous services. This community-driven evolutionary trend, from complexity to clarity, drives all of the trends under discussion here. The Rational group has long recognized that successful software trends embody the same three fundamental properties: community, modularity and empowerment, each of which we will discuss at length.

Community-based software development

- Evolution of programming-in-the-large[9]

Programming-in-the-large has evolved.

If you've ever wanted to witness a philosophical debate akin to arguing the number of angels that can fit on the head of a pin, the blogosphere is rife with examples. Google a thread that contains the following: "What constitutes *real* open source software?" What you'll soon discover (once you get past the invective) is that there are extremely few frameworks that everyone can agree are "pure" OSS. Even Linux technology—arguably the most successful open source software on earth—is split into (at last count[10]) more than 350 distributions, with varying degrees of "open-sourciness." Ruby is controlled (mostly) by one guy in Japan, and *everyone* is looking for funding. Present-day commercially minded open source contributors have come a long way from the beneficent Free Software Foundation (FSF). The trick here is not to get bogged down into religious arguments, but instead to judge each software framework and product by its attributes. Let's start with the most basic definition of OSS and work our way out.

Open

The best definition of open source software is implicit in the name itself: OSS lets you peek at the source code. The benefits derived from having access to the original code are implicit, but not guaranteed, as we will explore.

Highlights

Easy to modify, easy to fix

This modification is not guaranteed by every OSS framework, as there are over three dozen OSS licensing schemes, each with a slightly different take on what you're obligated to do once you start fiddling with the bits. Modifications can also negatively affect support contracts. And the freedom to "hack" cuts both ways: While the ability to repurpose software is a boon for developers, it's a headache for the IT managers who have to maintain the new software once the developer is gone.

Easily available

For decades, commercial software vendors have held their source code under lock and key because it was believed (and still is, to a large extent) that their competitive advantage would be lost once everyone had access to its inner workings. This secrecy also had a negative effect on software adoption, requiring a salesforce and lots of marketing literature to draw back the veil. The source, once public, can be rapidly available to all, driving the point-of-sale price down to zero, and allowing anyone to evaluate what the software can really do. Open source software is never shelfware.

The true cost of OSS begins* after *installation.

Free

Practical experience tells us that open source software is free like the family dog's puppies are free. The true cost of OSS begins *after* installation, with development, deployment and maintenance costs. This is nothing new—the same is true for all software. Modified OSS will also have greater costs associated with these activities, due to the uniqueness of a one-off solution. We have already seen that the dominant business model of successful OSS companies is to make money offering support, services and education, and this makes sense. In other words, the cost of the puppy lies in the care and feeding. As with a puppy, making a poor choice with OSS results in a high cost of continual poop scooping.

Highlights

All these things are the result of reasonable governance applied at every level.

Modular

Modularity is not implicit in OSS, but it is a hallmark of good software design in general. This includes clean lines of demarcation, reasonable and thoughtful application programming interfaces (APIs), and an extensible framework that may include pluggable components. The best OSS frameworks are modular, as are the best commercial software packages.

Community-based

This is the heart and soul of successful OSS, and it's also at the heart and soul of all successful software design and execution. Communities and open standards exist side by side, as the dominant community will also dictate the de facto standard. The problem here is ensuring the best standard, and the key to that is governance at multiple levels. Maintaining a healthy bug-tracking system, ensuring lively debate, developing utile standards—all these things are the result of reasonable governance applied at every level.

To summarize, OSS may have, but is not required to have, many known attributes of quality software: modularity, availability, reasonable cost (price plus maintenance), open standards, robustness, performance and a thriving, well-governed community of users. The point here is that these qualities are not *exclusive* to open source software, but they're the ideal for all software. In this regard, we can agree with the spirit of the OSS developers' debate above. We're all looking for software that has these positive qualities, but the industry has yet to agree on exactly how these qualities will be realized.

Highlights

Service-oriented architecture

With SOA, we see many of the same issues from the OSS discussion above; however, the scope of SOA includes the entire enterprise, not just an individual framework. SOA is, at its core, a style of information systems architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services. These services interoperate based on a formal definition (or contract) that is independent of the underlying platform and programming language.

SOA services should be well defined (encapsulated), discoverable and loosely coupled so that they are sufficiently autonomous, abstract (reusable) and capable of being composed into more complex services, which themselves have the above attributes.

It's easy to fall into the trap of thinking that an SOA equals a particular technology.

Because SOA standards are still being defined, it's easy to fall into the trap of thinking that an SOA equals a particular technology or a particular solution. A few years ago many thought *SOA = Web services*, because Web services happen to satisfy the primary requirements of SOA services. That *SOA ≠ Web services* is self-evident today, but the same mistake is being replayed by those who think *SOA = ESB* (enterprise service bus). Different technology, equally wrong equation. In fact, Web services and ESBs are useful tools within a well-formed service-oriented architecture. Malformed Web services and ill-conceived patterns instantiated through an ESB in the absence of an elegant and thoughtful SOA is simply a faster path to chaos.

Highlights

SOA is 1 percent services and 99 percent governance.

The mistake of course is to assume that a single technology or a combination of standards will infuse your enterprise with SOA. The truth couldn't be further away. The reality is closer to the famous Thomas Edison maxim "Genius is 1 percent inspiration and 99 percent perspiration." Thus, SOA is 1 percent services and 99 percent governance.

A successful SOA is *hard work*, and it requires many disparate departments and organizations agreeing to subordinate their own concerns for the good of the whole. This unity of purpose is difficult enough to achieve within a small team of individuals, but it's even more daunting when one considers the geometric accumulation of inflection points between all the teams and all the organizations within an enterprise that must work together to achieve an SOA.

Difficult, but not impossible. An SOA has a much greater chance of success when leveraging open standards, modularity and a well-governed community. As with OSS, the one attribute required for the success of all the others is governance. Governance controls standards compliance, modular component contracts and, ultimately, the ability of your SOA community to work together in a meaningful way.

Highlights

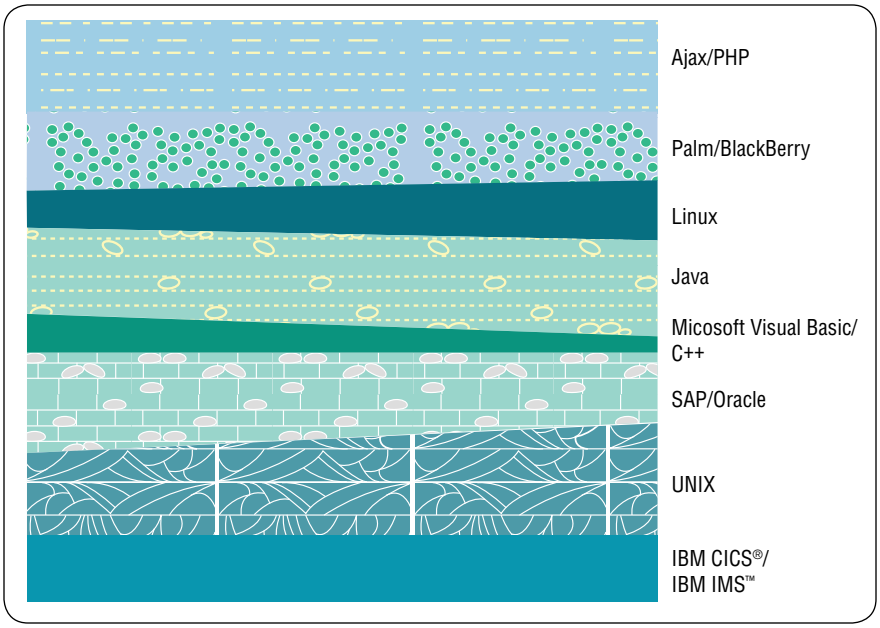

Figure 4. Layers of data, services and processes

Think of what the banking and financial industries are already doing with SOA-surfaced data.

Unlocking the services, data and processes buried under decades of software layers requires a concerted effort of excavating and repurposing those artifacts that are of value to your enterprise. Think of what the banking and financial industries are already doing with SOA-surfaced data. Accounts, investing, credit cards, mortgages and loans can now be managed through a common user interface, and these services are now interconnected with variable payment plans, automatic withdrawals and deposits, dividend rollover, financial analyses, tax calculators—the list goes on and on. Those services represent a lot of unburied artifacts transmitted securely around the world, with close to zero percent failure. And that's just what's happening today in banking; there is much, much more to come.

Highlights

As we'll see, the Rational group is making strides toward easing your SOA development process. IBM understands the importance of community development, of information sharing and of appropriate governance during the entire application lifecycle.

Governance and empowerment

To many, the mention of governance immediately leads to thoughts of compliance: compliance with Sarbanes-Oxley and the Health Insurance Portability and Accountability Act (HIPAA). It also leads to thoughts of *noncompliance* and stiff penalties for regulatory infractions. The umbrella of governance as it applies to software delivery covers much more than the implementation of extant legal dicta. Governance is more of an oversight framework, a set of standards that, when enforced, can empower a software team rather than hinder it. The framework of project governance evident in the workings of the Eclipse project, for example, has empowered one of the most creative, rapidly growing open source communities.[11]

Governance is more of an oversight framework, a set of standards that, when enforced, can empower a software team rather than hinder it.

Successful compliance, then, becomes a *result* of successful governance, rather than the focus of software delivery efforts. C, Java, the Internet, XML, Linux, open source software, SOA—none can exist without governance. Every one of these technologies, architectures and frameworks has flourished *precisely* because there are standards in place, norms that set boundaries of behavior, of discussion, of implementation. These allow ever greater numbers of users to rely on the stability of the technologies, and to utilize them effectively. C could not have become the lingua franca of the PC world without its adherence to American National Standards Institute (ANSI) and International Organization for Standardization (ISO) governance; nor the Internet without the World Wide Web Consortium (W3C) and Internet Corporation for Assigned Names and

Highlights

Numbers (ICANN); nor Linux solutions without C and the Internet. OSS and SOA stand on these broad shoulders, and they are no less bound by the requirements of proper governance than are their ancestors. Without governance, you're leaving a lot to chance.

Driving with cruise control allows an automobile to maintain the speed limit—this is different from a "governor," a device that only sets an upper limit. Cruise control is a lot like effective governance in that both ensure operation at the highest possible legal speed, maximizing efficiency while minimizing travel time and trivial activities. Let's further the analogy by imagining this same car taking advantage of timed traffic lights, which require computerized governance. This illustrates how we can use governance to our advantage, achieving our goals with the utmost speed and efficiency. Contrast this scenario with one in which we fear compliance—when we see a traffic cop waiting behind every tree. The difference could not be greater.

Using governance to one's advantage means agreeing to a standard, a framework or a product that minimizes unnecessary effort, leveraging the implicit control to turn a project in the right direction. Governance is a crucial part of every successful software delivery, OSS integration and SOA implementation—in fact, it's a crucial part of every facet of software creation, implementation and maintenance. It's up to each individual or organization to use those sets of controls wisely.

Governance is the hidden glue that binds every community together.

Governance is the hidden glue that binds every community together—there must be a defining reason for the community to exist at all: common goals, common beliefs, common property—there always exists an implicit contract that binds. Many communities are created, but very few succeed for more than a short time.

Highlights

The key ingredients of a healthy community include both governance and empowerment.

For a community to exist long enough to exert influence over the broader marketplace, it must be healthy so that it can harness the intellectual vitality of its members. The key ingredients of a healthy community, we have found, include both governance and empowerment, and empowerment of the individual through modularity of choice is a key to the future success of software delivery.

Future software delivery

> *"People can have the Model T in any color—so long as it's black."*
>
> —Henry Ford

Introduced in the early 20th century, the Ford Model T had the right price point, was available in mass quantities and performed pretty much as advertised. It was the right product at the right time, but completely unconfigurable—available in one size, in one color, with no options whatsoever.

Fast forward to 2007 and witness how the automobile industry has changed, allowing you to communicate your specifications directly to the factory, customizing your new car online. Options include GPS satellite positioning, satellite radio, terrestrial CD or MP3, climate control, rain sensors, fuel consumption regulation, hybrid and alternative-fuel controls, guidance systems, robotic parking, space visualization and many other options that justify having computing power under the hood. Oh, and you can choose the color, too. The automobile industry has embraced software integration both within the chassis and throughout the manufacturing supply chain, delivering products on demand.

Highlights

We expect to be able to configure our technology to our needs.

Witness also the success of just-in-time retailers such as Amazon.com, Netflix and others, which now stock thousands of products—many more than a bricks-and-mortar site could handle—catering to very specific tastes. We live in an age of customization; we expect to be able to configure our technology to our needs, and there is no reason to expect any differently when we do systems integration or software delivery. We know that customization comes with a price, and we're willing to pay it.

Supply chains of intellectual property

Welcome to the age of the Long Tail. Popularized by Chris Anderson's book of the same name (and sold on Web sites that daily prove Anderson's theory), the Long Tail theory holds that the future of business is to sell fewer items geared for majority acceptance and more items customized for very small groups, or even individuals.[12] For software vendors, this will mean developing fewer big-ticket items and more customizable components. In short, generic frameworks with lots of customized plug-ins.

The Long Tail argument focuses more on the supply side. Technology enables a shift in the supply-side cost model—from retail storefronts to the distribution center, which in turn allows for greater inventory of disparate items. This then enables the supplier to capture the long tail of the demand curve by providing a wider range of choices.

Highlights

In applying this phenomenon to software delivery, demand-side ramifications translate backward onto the supply-side vendors of software. In one manner of speaking, SOA is both an expression of the desire for choice and a mechanism for accommodating the lack of flexibility buried in the sedimentary layers of software architectures. We're talking about nothing less than surfacing information, data and artifacts as we construct supply chains of intellectual property.

Compounding the issue for software development is that most of us do not efficiently and effectively address the long tail of usage patterns for various development roles. Too often the tail wags the dog, forcing roles on development and testing organizations through conformity to the various features and functions of the tools we use. Software tools providers bemoan the fact that testers are overly focused on testing for performance rather than application business intent. However, software providers are also culpable because they provide tool suites that define the features that encourage functional pigeonholing.

The best way to craft these products comes from listening to the community.

The best way forward requires providers to work backward from the demand side of the equation, and then understand how to shape the supply side to provide the choice and flexibility required to address the issue. The best way to craft these products comes from listening to the community.

Highlights

Eclipse technology is embedded in more than 200 IBM products.

Opening the community to participation

IBM's goal in releasing Eclipse to the open source community in 2001 was to bring developers closer to the more open Java technology-based middleware, envisioning a world in which a customer's development environment comprised a heterogeneous combination of tools. This was accomplished by creating a robust thick client framework and by making all services accessible to plug-ins—plug-ins that could have been developed as existing Eclipse projects, third-party vendors' products or independent developers' products, but all of which had been built against a common platform, thus comprising a software tools ecosystem.

Eclipse has rapidly become the most popular enterprise Java development platform in the world, being used today by 65 to 75 percent of Java developers.[13] This testifies to the popularity of Eclipse with vendors as well as users. Eclipse technology is embedded in more than 200 IBM products, and standardization on a single development and deployment platform has contributed to greater product interoperability.

The Eclipse project's stated goals of transparency, predictability and constant feedback have led to an amazing level of project health—one of the most important ingredients in creating quality software. A healthy community, strong team spirit, robust builds, solid milestones, promising betas and continuous testing all contribute to the Eclipse project's health.

Highlights

The Rational group's involvement in the Eclipse project over the past six years has proved that project health is ultimately more important and complementary to software quality. Because the state of a project is constantly changing, staying healthy becomes a team responsibility—it takes a village. And a healthy developer community—one that values openness, responsibility and inclusiveness—can better react to the normal stress of working on continuously evolving software: the unanticipated changes, the new requirements and the constantly moving target.

Rational thinking

Governance will be the key to creating the synergy necessary to pull these threads together.

The IBM Rational group sees the future of software development involving open, dynamically defined communities working with maximum freedom of configuration and customization to build modular solutions that support agile, responsive organizations. Governance—the ability to harness the energy of disparate groups and heterogeneous technologies—will be the key to creating the synergy necessary to pull these threads together. No single standard or technology will dictate the speed at which development organizations will move forward on business initiatives. It's not SOA or OSS that will be the driver—it's the effective governance of multiple drivers that will lead to healthy communities and drive responsiveness.

This effective governance will take the form of a common data structure that enables real-time sharing of information from one application to another, regardless of platform. It's a way to integrate communication so project team members can be messaging one another in the context of the project that they're working on and the applications they're using—from the business analysts all the way all the way out to the production specialists. It's the ability to construct software from globally distributed teams, and have those products interlock seamlessly.

Highlights

The future of software delivery looks a lot like the IBM Rational Jazz project.

It's the ability to pass a governmental audit with zero additional effort because compliance has been embedded into the framework. It's a fundamental shift from role-first to team-first thinking. The key is reassigning the team to be more inclusive in all of the different phases of the project.

The future of software delivery looks a lot like the IBM Rational Jazz project, a technology that integrates team tasks across the entire software and systems delivery lifecycle. Organizations using the Jazz framework will be able to assess the impact of changing a requirement to see how it affects builds, thus enabling organizations to more accurately determine what needs to change and who needs to change it. This lifecycle integration will incorporate tool-supported process guidance, where the tools *understand* the development process the team has decided to use. And, through automation, the lifecycle integration helps ensure that team members can follow the process without it getting in the way.

To realize full value from a team-first lifecycle governance platform, organizations must also supply flexible modular components based on open standards that ride on top of the platform. One cannot exist without the other. The decomposition and componentization of existing features and functions are parallel and equally important work efforts.

It's a community effort; no one can get there alone—much like Eclipse. That's why the IBM Rational group has taken community-driven software to the next level, both as committers to Linux, Eclipse, Geronimo and other OSS projects, as well as

Highlights

committers to community building within the company's own software organization. The IBM Rational group's process guidance capability is being designed to scale in complexity from agile practices to structured approaches. And it will scale to support development environments from very small teams to large, distributed organizations. Organizations will have the ability to mix and match software delivery tools and technologies for the optimal solution at every stage in the transformation of their development environments.

Conclusion

We live in exciting times—rapidly changing times—and our software is continuously evolving to meet ever-expanding needs. The harsh reality, as true as it ever was, is that we must adapt to our times or be swept aside, with the threat of our obsolescence coming in months, not years. This does not mean that everything has changed; it's quite the contrary. Software creators still face the same realities: Software modeling and business requirements gathering are still essential, and high-quality utile software is always better than the alternative. What has changed, of course, is the speed, the scope and the reach.

The answer lies in agility.

The solution is not completely new, but it incorporates new ideas. While it will become essential to customize your software delivery to meet ever more segmented customer demands, you'll go broke if you focus on providing the n^{th} feature or function. The answer instead lies in your agility: the ability to surface meaningful data and relevant assets and artifacts through your current layers of intellectual property, as doubtless your solution will incorporate a mix of many different types, brands and layers of software.

Highlights

There are three key principles of future software delivery: community, modularity and empowerment.

Getting your data working for you will mean atomizing existing artifacts and repurposing them across your enterprise. That means you have to remain nimble and adaptable so that your customers can individualize their interaction with your business. Success here depends on how faithfully you keep your eye on the three key principles of future software delivery: community, modularity and empowerment.

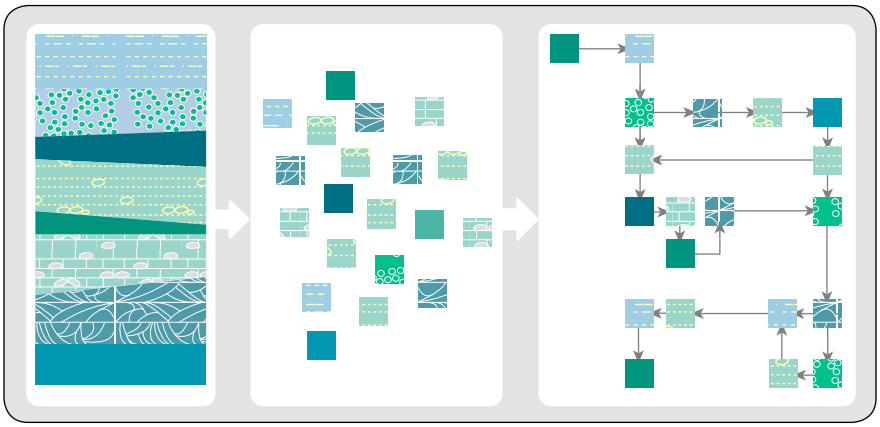

Figure 5. Surfacing layers of data, services and processes, and putting them to use

Future software delivery will more closely resemble a supply chain: horizontal organization that is loosely coupled yet effectively governed to accommodate your SOA and GDD requirements, surfacing the data, processes and services

Highlights

The IBM Rational group is taking bold steps.

you need in a streamlined way to provide a positive return on investment (ROI). The future is less concerned about individual applications, operating systems or software choices; rather, it's more concerned with the delivery of the service, the modularity of the system and the involvement of your defined and flexible community in making choices that matter.

The IBM Rational group is taking bold steps to make this future vision a reality. Community, modularity and empowerment guide and infuse current Rational products, the Jazz framework and all future Rational product offerings.

The future of software delivery looks exciting. And you can rest assured that the IBM Rational group will continue to play a leading role.

Endnotes

1 *The World is Flat: A Brief History of the Twenty-first Century;* Thomas L. Friedman; 2005; Farrar, Straus and Giroux.

2 *Service Oriented Architecture: The Foundation for Digital Business;* Diego Lo Giudice; Forrester Research; 2006.

3 *Enterprise Service Bus and SOA Middleware;* Aberdeen Group; July 2006.

4 *Open Source in Global Software: Market Impact, Disruption, and Business Models;* IDC; July 2006.

5 Nucleus Research various ROI reports; 2003 (http://nucleusResearch.com).

6 http://en.wikipedia.org/wiki/Moore's_Law.

7 http://en.wikipedia.org/wiki/Metcalfe's_law.

8 *The Tipping Point;* Malcolm Gladwell; 2002; Little, Brown & Co.

9 http://en.wikipedia.org/wiki/Programming_in_the_large.

10 http://distrowatch.com.

11 *The Eclipse Way;* Erich Gamma and John Wiegand; 2006.

12 For more examples of the Long Tail postulate, visit http://www.longtail.com/the_long_tail.

13 This figure includes use of other products that embed Eclipse components into their commercial IDEs, such as IBM Rational Application Developer software.

© Copyright IBM Corporation 2007

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
02-07
All Rights Reserved

CICS, IBM, the IBM logo, IMS and Rational are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product and service names may be trademarks or service marks of others. The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided ìas isî without warranty of any kind, express or implied. In addition, this information is based on IBMís current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

This publication contains other-company Internet addresses. IBM is not responsible for information found on these Web sites.

RAW10983-USEN-00