Cover Story
Software and the New Business Economy
While software continues to change the world, we face new challenges in improving software development. -- by Rational co-founders Paul Levy and Mike Devlin
Editor's Notes

Every year for the next four years, roughly one trillion dollars will be spent around the globe on the creation and maintenance of our software systems. Why such an unprecedented commitment of global resources? Here's one way to answer: It reflects widespread belief in new opportunities. At the highest level, this investment represents optimism about our ability to fulfill age-old desires -- to communicate, learn, heal, entertain, and do all sorts of things more efficiently. And let's face it... for those of us working in software development, there's a practical reward for seizing those opportunities: business success.

The Internet, of course, looms large on the new business landscape, as more companies focus on profitable models for their e-business initiatives. The infrastructure needed to run and improve our software systems presents endless opportunities; so does the embedded software needed to drive a host of new electronic devices on which our economy will increasingly depend. Yet, as important as software is to success in the new economy, the software development industry itself is not doing nearly enough to realize its full potential.

In this month's feature article, Rational Software co-founders Paul Levy and Mike Devlin offer their view of the overwhelming importance of software's role in the new economy, and the challenges we face in delivering on the promise that IT holds for us. In our Management section, Sid Fuchs defines...
the organizational building blocks that lay the foundation for successful technological change in a rapidly evolving business environment, and Karen Dasey examines the advantages that Web Content Management (WCM) brings to these environments. We have a much more extensive range of Reader Mail to share with you this month, and our Technical section offers a raft of practical information to help you improve your software development practices.

In other words, we're here not only to define challenges, but also to help you face them. Welcome to the third issue of The Rational Edge!

Mike Perrow
Editor-in-Chief
Software and the New Business Economy

by Paul Levy
Founder and Chairman
and Mike Devlin
Founder and Chief Executive Officer
Rational Software

Software is the world's most critical industry and will be for years to come. Yet most companies are spectacularly unprepared to create the software that will redefine how they interact with customers or that will help deliver their goods and services in new ways.

This is the first in a series of articles Rational will present called "Improving the Economics of Software Development." This first installment takes a look at the changing landscape of business and business growth in the new economy, and explores the challenges we as software professionals face in realizing the full potential of information technology over the coming years.

As we look at the drivers of what has been called "the new economy" and see why software is so critical to the success of companies around the world, it is clear that organizations need to closely examine -- and, in most cases, dramatically change -- how they create the software that will serve at the heart of their business. Many companies have already learned the hard way that the Internet is not a quick fix, nor is it a trivial matter to build a Web site that works well for both the customer and the business.

Over the past four to five months, some in the media have pointed to the demise of certain dot-com companies as if their disappearance indicates a trend regarding the importance of high-tech in business. In fact, many dot-coms have disappeared because bad business models were unable to sustain the promises made to their boards and shareholders. Does that mean technology is no longer important? Not at all. Technology still drives the new economy. The difference is, the stakeholders in the new economy have remembered how important profitability is. Underneath all the former hype lies the enabling power of the Internet -- and the software and communications technologies that drive it -- all of which constitute a more dramatic and fundamental transformation of business than anything we've
ever seen. The rate of change is just as rapid as it was last year, and the
direction is unequivocal.

Despite what happens on Wall Street -- bull or bear -- and no matter
which "dot-coms" are up or down, these fundamental changes in our
society and in our business plans will keep on occurring. What we've
learned this year is that all roads lead to Rome: Whether a company is a
brick-and-mortar establishment or a virtual enterprise, almost every
organization is becoming an Internet company in some way.

Now, let's look at these changes more closely.

**Drivers of Today's Business Transformation**

The major characteristics of today's business opportunities create a much
more open landscape of competition. These new characteristics include:

- **Time to market**: Both new and existing companies are being pushed
to develop new products more quickly than ever before or risk
losing to a competitor that has taken maximum advantage of the
new environment.

- **Increased productivity**: Both corporate and private, has been one of
the major factors contributing to growth of the new economy, and
this growth has occurred without inflation. In turn, the Federal
Reserve estimates that the use of information technology and the
production of computers accounted for about two-thirds of the
productivity growth between the first and second halves of the
1990s.

- **The "weightless economy"**: We're witnessing the increased
valuation of intellectual property -- displacing oil, gas, and other
types of durable or physical goods from the old economy as the
primary means of making money.

- **Globalization**: The opportunity to become biggest and best, not only
by expanding beyond the country of origin, but also by merging or
partnering with companies overseas.

- **Consolidation**: Global mergers and acquisitions continue to redefine
virtually every industry, especially telecommunications.

The Internet -- with the software and communications technologies that
drive it -- is changing everything. The cover story of *Fortune* magazine's
Oct 9, 2000 issue notes that we're living through one of the most profound
transformations in human society since the dawn of the Iron Age, and calls
this the "Age of the Network."

So, software has become the way we engage the world. And software is
ubiquitous. It's everywhere, from the cars we drive (with more software in
the average car than in the Apollo spacecraft); to military equipment
(today, 80% of a fighter jet's capability is performed by software,
compared to 6% in 1960); to the cell phones that are allowing us more
freedom and mobility. Software even exists in household appliances like
refrigerators, and in tiny devices like the cochlear ear implant for the
The Internet: Impacting Every Business

As software is becoming more present in more places, so too is the Internet impacting every business. For businesses the world over, the Internet continues to drive more change and more opportunity. That includes all the software that runs the infrastructure of the modern economy -- the computing and telecommunications platforms, all the operating systems and middleware, and all the underlying switching and routing software. It also includes our electronic devices and embedded software systems, everything from phones to refrigerators and transportation vehicles of every kind. Then, of course, there's e-business, whether it's done by traditional brick-and-mortar firms that are embracing new ways of developing relationships with customers or by dot-coms that exist only online.

Half of all adults in the U.S. go online each month, according to a December 2000 survey by Mediamark Research in New York. That's a 27 percent increase over 1999 in the number of people who surf the Web. Although most Web access happens from home, some 10 million more Net users are accessing the Web from work since last year, Mediamark says. This, in turn, is causing businesses to re-evaluate how they reach their customers and to redefine their products and services offerings. It's also causing a sea change in customer service, as enterprises are learning how to provide 24x7 support.

The Click and Mortar Revolution
Traditional companies are moving various portions of their businesses to the Web. At Swedish powerhouse Ericsson, for example, its future heavily depends on the growth of the Internet. Along with other networking companies, Ericsson is helping build the infrastructure of the Web in 140 countries -- especially in the wireless arena -- investing 15% of its sales revenue into research and development.

Financial services are also racing toward the Internet. At a recent panel discussion moderated by a Federal Reserve assistant vice president, both speakers agreed that the successful providers of financial services will "model themselves as Internet portals or hubs," where consumers will return frequently.

Automotive giant Ford Motor Co. is also staking a large part of its future on the Web. Ford recently announced the Wingcast initiative to bring wireless mobility and information services into cars and trucks by 2003. These services will include voice, entertainment, Internet access, and safety features.

Additionally, Ford, GM, and Chrysler in February announced they are building a business-to-business parts exchange on the Internet to speed communications with their thousands of suppliers. They see the parts exchange as only the first step toward putting all their business on the Web. If so, they will be building the world's biggest B2B site, the foundation of an Internet-based industry that designs and engineers cars faster and better, and eventually builds them to order, delivering them directly to customers within a few days.

Internet pioneer Dell Computer has been selling its PCs over the Internet since 1994. For its most recent quarter, ended Oct 27, 2000, it reported that sales from its site have grown to an annual run rate of $16 billion, up from $12 billion a year ago. By itself, the current run rate for Dell's Web-enabled revenue would rank No. 110 on the Fortune 500 roster of companies.

Cisco Systems tells a similar tale -- not only is it the world's leader in building the infrastructure of the Internet, Cisco also uses the Web to serve customers in ways not possible before. By using networked applications over the Internet and its own internal network, Cisco is seeing financial benefits of nearly $1.4 billion a year, while improving customer/partner satisfaction and gaining a competitive advantage in areas such as customer support, product ordering, and delivery times. Some 90% of Cisco's orders are transacted over the Web.

The Web and New Business Opportunity

As for newer businesses, the Internet translates into brand new opportunities that simply didn't exist for entrepreneurs in the past. Software and services provider Commerce One enables online trading networks, allowing buyers and sellers around the world to trade in a barrier-free environment and creating new business opportunities for all trading partners. Another pioneer, eBay, created a format for auctions impossible to imagine even 10 years ago.
And while it's true that many e-businesses have taken a beating over the last few months, this simply demonstrates that poorly planned and executed business models do not generate long-term, or even short-term, success. Clearly there are successful, profitable e-businesses like eBay who are breaking new ground, and demonstrating that innovation and solid execution are just as important in the virtual domain as they are in the brick-and-mortar domain.

**Software: The Key Differentiator for Businesses in the New Economy**

As companies are changing the way they think about technology and how they use it, so too are organizations changing the way they buy the software they need. As recently as the 1980s, software was the domain of a centralized information technology group that typically made all the purchasing decisions and controlled the back-office systems used to automate business processes.

By the 1990s, departments and systems were becoming more integrated through distributed architectures and corporate intranets. More corporate data became visible to more people. This led to a more robust front office - a business could have a point-of-sale system integrated with an inventory system, so you knew as you sold that you decreased your inventory and could place a reorder with your suppliers.

Today, in the new economy, we're witnessing a much more collaborative environment for doing business. To stay competitive, every business must embrace the technologies of its customers, its supply chain, and its partners. Software is increasingly used as the basis of this connection. And because software investment is becoming a much more strategic investment for the company, this means the decisions about software and its use are increasingly being made by senior management today.
Because software has assumed such a critical role in business success, companies all over the world are taking their software investments VERY seriously. Every year over the next four years, a trillion dollars will be spent by companies and governments to build the software that the world runs on, according to analyst firm IDC. This figure takes into account software used for the design, development, maintenance, and analysis of the business applications being built in corporations and government agencies. It also includes all the salaries and benefits provided to the employees who do the work -- this, in fact, represents the bulk of the costs.

In June of last year, the U.S. Department of Commerce published a study called "Digital Economy 2000" in which they noted that business investment in IT equipment and software more than doubled between 1995 and 1999. They also reported that over the last five years IT industries have contributed nearly one-third of the real U.S. economic growth. This is because industries in many sectors of the economy are incorporating more and more software, embedded devices, and computing infrastructure into their products and their business operations.

Based on all the evidence -- including the degree of spending on software and the Internet as well as the widely embraced move to e-business -- we can draw one obvious conclusion: Software will be the key differentiator for every business in the new economy.

If we're going to thrive over the next decade, we will have to be using software in a creative way to deliver our products and services or to delight our customers in new ways. Mere software competence is not enough; software excellence is becoming essential.

**Software and the Internet: Predictions**

This past summer, Rational's chief scientist, Grady Booch, surveyed 100 people who are busy inventing the future -- the best minds in the software industry -- to get an idea of where we are all headed as stakeholders in this new, Internet-driven economy. These are the four most important predictions based on this survey:

- Software will continue to become increasingly complex.
- The unwired Web will have far-reaching implications.
- The new economy requires software that can be continuously updated.
- There are unlimited possibilities for software's growth -- but there are some problems, too.
Let's take each of these in order, starting with the complexity issue.

**Increased Complexity**

Grady Booch has often said, "We cannot reduce complexity in how we build software. The best we can do is manage it." Because so many of the products and services we value highly in the new economy are more complex than their predecessors, there are manifold processes required to support those new types of offerings. Today's software engineers are feeling pressured to deliver more complex applications that require more mastery of more computing concepts and techniques than ever before. In building for the Web, the trend is toward even more concurrent, more distributed, and more connected applications.

We see this happening in telecommunications, which has evolved from the hard-wired rotary phone to the global wireless infrastructure. Photography offers another example of increasing complexity, where the process has gone from silver halide film to all the intricacy of digital imagery. And we see increased complexity in the securities industry, where brokers hired to handle stock transactions are now being augmented by online trading services. But the software designed to handle the volume, the data security, and the transactional requirements of online stock trading is highly complex, and growing in sophistication every day.

**Emergence of Wireless Connectivity**

Our second prediction has to do with the huge consumer demand for wireless access to nearly everything electronic. Today, half a billion people currently use mobile phones -- and within three years, that number is expected to reach one billion. According to Ericsson, clearly one of the leaders in the wireless industry, wireless devices have penetrated less than 1% of the market, and there's enormous growth ahead of us.

An important aspect of the growth is wireless access to the Internet. Consider this:

- Ultimately, one-sixth of the world’s population will be equipped for the unwired Web. Almost every facet of our daily routine will be affected as the Internet further snakes its way into our cars, homes, and businesses.

- There is an emerging OnStar architecture that in effect makes every car its own mobile IP address. Companies like Ideo are giving us electronic paper of various sizes, so we can more easily read our e-books. Imagine carrying this around as opposed to toting a traditional bound volume!
As the wireless infrastructure continues to evolve, we see the growing sophistication of devices and embedded systems designed to take advantage of it. One of the most significant aspects of the unwired Web is its infancy. Within three years, there will be more than 400 million users of mobile Internet services. And that number will quickly exceed the number of fixed line Internet users. As this occurs, text services will give way to support for photos and blueprints. Eventually, increased bandwidth will mean that handheld mobile devices will access video over the Web as easily as do our machines wired to T1 and T2 lines.

**Continuously Evolving Systems**

In the past, most new software systems were developed by an internal IT organization, and the predominant value of software was in reducing the cost of doing business. Time pressures and feature content pressures always drove these software solutions, but there was generally little impact to the top line revenues of the business, and, consequently, these pressures were usually second order. Most software projects were one-of-a-kind systems that took years to develop, then they were upgraded and released back into the user community about once each year.

Today, however, software's role has shifted from cost reducer to revenue...
producer, and competitive pressures are demanding ever-faster cycles of development and upgrade. While disciplined software management and common sense will remain the primary discriminators in software engineering success or failure, the Internet has truly revolutionized our ability to both include the user during development and deploy software products transparently to a broad user base. In contrast to the huge annual upgrades of the past that typically required a total shutdown, most markets today demand frequent releases that allow us to keep our systems available to customers and partners 24x7.

This need to support continuously evolving systems has driven today's software development processes away from the traditional sequence of waterfall development activities toward a highly iterative model. In the past, many software domains drew a distinct line between development and maintenance, but future software projects (legacy system upgrades, new applications, or some combination of the two) will probably not differentiate much between development and maintenance.

Iterative development and the Internet are also driving software engineering toward a more homogeneous software management approach. This includes process frameworks, advanced requirements and design notations, and Web-based architectural patterns. Ultimately, this should produce dramatic improvements in software release cycles along with several other major trends regarding continuously evolving systems:

- Greater use of commercial components that will reduce implementation activities, resulting in earlier transitions to demonstration-based assessment.
- More mature iterative development methods and Web-based architectures that will create a larger role for deployment activities within the life cycle.
- More mature iterative development environments (processes and tools) that will enable further reduction of life-cycle scrap and rework.

And Unlimited New Possibilities

We have come a long way in a short time to overcome many major constraints in the computing industry. Compared to the early 1990s, today we have ample:

- Power -- this translates into lightning-fast processing, ample memory, and virtually unlimited storage capacities.
- Accessibility and usage -- Although software's features and functions are ever-more complex, we've managed to vastly simplify the user interface, so it's easier than ever to get email, jump on the Internet, figure out how to do word processing, and use computers for personal and business productivity gains.
- Bandwidth -- In all honesty, we're not quite at the "all bandwidth, all the time" stage. But that's only a matter of time, as companies continue to build or buy the necessary infrastructure. Some
analysts predict that 40% of all homes and businesses will have DSL speeds by 2003. And we are connecting the world through both fixed lines and mobile devices to the rapidly growing Internet.

**So, Where Are the Problems?**

Despite the tremendous strides this industry has taken in computing power, we will not be able to take full advantage of those benefits until our software achieves similar levels of excellence. Our next major hurdle as an industry is software development itself. That's our current brick wall.

![Diagram showing power, accessibility, bandwidth, and software development]

**Figure 4:** Compared to the early 1990s, today we have ample power, accessibility, ease-of-use, and bandwidth. But our ability to achieve excellence in software development represents our next serious brick wall.

Until we eliminate the difficulties in this area, we will not come close to realizing excellence in the software accessed by our customers, partners, and employees -- and we will be less likely to become leaders in the new software-driven economy.

**The Failure Rate for Software Projects**

Let's take a closer look at the difficulties facing us in this area, some of which we can quantify. Figure 5 below shows the results from the CHAOS Study, by the Standish Group in 1999.
Figure 5: While companies have seen an improvement in the success rate of their software products over the past six years, the Standish Group reports an astounding 74% failure rate for all software projects undertaken today.

This study covered both commercial and government IT projects and identified several disturbing statistics. In 1994, only 16% of all IT projects were completed on time and on budget. By 1999, this statistic had improved, but still only 26% of all IT projects were considered successful by this standard.

We at Rational Software are fortunate to have worked with a number of customers who report far better results. In fact, we see extraordinary achievements here and there among companies pursuing both commercial software development and development of their own internal systems. Unfortunately, these represent the exceptions to the rule. What's more, we need to be getting closer to the performance rate in other industries. In the automotive safety and healthcare industries, for example, a failure rate of even 1% is considered unacceptable. We need to have the same kind of goals for software.

**Labor Shortages**

The high failure rate in software projects is not a problem we can solve by adding more people. For one thing, there just aren't enough qualified IT workers to go around. We will not examine in detail the labor market in every area of the world. But we certainly have a well-documented shortage in the United States. A study by the Information Technology Association of America predicted that U.S. businesses would try to add 1.6 million IT workers in 2000, but would succeed at hiring only half that many because of the shortage of skilled employees.
The evidence suggests that the labor shortage problem for software development is worldwide, and growing year by year. Analyst firm IDC issued a report in June that projected even greater labor shortages in Europe, up from today's deficits by as much as 30%.

To overcome this particular problem, IT investments need to become more focused on existing personnel, so that software development teams can produce more bottom line results with better tools and processes. Measuring your return on IT investments can be difficult, but as software becomes a competitive differentiator for more businesses, we are beginning to see studies and methodologies emerging that can help business leaders better address their ROI questions regarding software investments. IDC, for example, has recently published such a study.¹

The New Economy and the "Software Development Paradox"

To sum this up, while the possibilities for software are seemingly limitless, so is the growing demand. We must become more competent in our ability to deliver high quality software. And this is a huge challenge. As we've seen, most companies are struggling in the effort. Of course, success is possible, and Rational sees it on a limited scale. We have been privileged to work with some of the market leaders in different industries, the ones who understand how closely the success of their business is tied to their success in developing high-quality software.

The answer for most companies is not going to come through additions to their software development staffs. There just aren't enough skilled workers anywhere in the world to meet the demand. And as quality experts in any field know, throwing more people at a problem is not necessarily the answer, either.

The problem is that speed and quality have typically been opposing forces in software development, and they still are. In the past, businesses could sacrifice software quality to make their ship date, or compromise on software features to meet time-to-market deadlines. In the new Internet economy, we have no choice: we must produce higher quality software in Internet time.

And this conundrum forms the heart of what we at Rational call the "software development paradox." Fast time-to-market is absolutely critical for business success. The competitive features that once took years for the market to embrace now, overnight, have become absolute checkbox requirements for consumers. So now we must respond with our own product plans, or risk losing market share. At the same time, a poorly engineered product that gains wide visibility simply because it captures "first mover advantage" will ultimately fail, due to low customer satisfaction. Products rushed into market will typically not live up to the expectations in the vast marketplace of the Internet.
On the other hand, if we can get our business to a point where we are not obligated to trade off speed for quality, we can enjoy a sustained, competitive advantage in our respective markets. Our business success will depend on how well we execute in the process of software development. We need speed AND quality in our software development process, in order to escape the "software development paradox."

**Conclusions**

Let's summarize the key points covered in this first installment of our series, "Improving the Economics of Software Development."

1. **Software will be the key differentiator for every business** in the new economy. It connects businesses to customers, suppliers, and partners. It empowers everyday devices. And a high degree of competence in our development and maintenance of software is becoming essential to every business.

2. But **we can't simply rely on the past decade's improvements in computing power**, bandwidth, and accessibility to realize the full business benefits of the digital revolution. While future possibilities -- and demand -- for software are endless, we need to vastly improve our methods for designing and building software to sustain healthy business growth and success in the new economy.

3. **This is a challenging situation. Most companies struggle** with their software development projects. While success is possible, the answer will not be found through the addition of more people to our software engineering teams. There's simply not enough qualified
help available.

4. The answer lies in improving the economics of software development -- that is, **reducing complexity and improving processes at all levels of software engineering**. It can be done. With proper investments in productivity, companies can escape both burdens of time-to-market pressure and quality-of-product pressure.

We will share with you how Rational Software believes these goals are best accomplished in an upcoming edition of *The Rational Edge*. In the meantime, we want you to know that Rational's mission is to ensure the success of customers who depend on developing or deploying software. You’ll find answers to the software development paradox in the product pages of www.rational.com, in our services, our best practices, and in the combined solutions that our partners help us deliver to the IT industry.

---

1 This whitepaper offers a case study of Rational Software customers using Rational ClearCase for software configuration management. To view this paper online, click here.

---

*For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!*

---

Copyright Rational Software 2001 | Privacy/Legal Information
Configuration Management -- It's Not Just For Source Code

by Ralph Capasso
Product Manager
Rational Software

"It's not just for breakfast anymore." Several years ago, this was the slogan for a very successful advertising campaign for orange juice. As part of an effort to dispel a common myth that orange juice was strictly a morning drink, these advertising spots insisted that orange juice was a healthy and refreshing drink to be enjoyed any time -- regardless of where the hands happened to be positioned on the clock.

What does orange juice have to do with software development? Not much; but myths similar to the "morning only" belief have evolved over time in software development circles, too. One such myth concerns the role of configuration management (CM) in the development of software projects. Many development shops believe that configuration management is a solution suitable only for managing source code, when in fact, many of today's configuration management systems are capable of managing a wide variety of electronic artifacts. The time has come to debunk this myth about configuration management -- it's not just for source code anymore!

Don't get me wrong. Many technology shops regard their proprietary source code as the company's crown jewels, and rightly so: An adequate source code management system is certainly critical to the success of any software development endeavor. The point is that artifact management doesn't have to end there. Successful teams can leverage even more benefits from their configuration management tools and practices by applying them across a broader scope of development artifacts.

Breaking the Source Code Barrier

As we look back on how this narrow-minded view of configuration management developed and why it still persists, there's no great mystery. First-generation configuration management systems were capable of
versioning only ordinary text files and had self-limiting names like Source Code Control System (SCCS). In fact, Microsoft's CM system still has such a name: Visual SourceSafe.

Actually, configuration management vendors were among the first to discover alternative uses for their configuration management systems, and they've been busy evolving their early systems' capabilities ever since. Today, for example, systems like Rational ClearCase offer management and versioning capabilities for all file system artifacts -- including directories and binary file formats. By expanding the breadth of file system artifacts they are capable of storing and managing, these modern-day systems have greatly expanded configuration management's potential in the development environment.

For some development shops, recognizing and capitalizing on this new potential has been a natural progression. When configuration management systems became capable of managing binary files, for example, some saw it as an opportunity to capture more and more of their build environments within the system. In addition to source code, build and release engineers started to manage and version their compilers, linkers, system libraries, and other binary artifacts that had been difficult, if not impossible, to manage previously. By managing more of the build environment in the CM system, release engineers immediately began to realize more meaningful build audits. These audits included not only the versions of the sources that went into a build, but also all the system files, libraries, and tools that played a role in the build.

In addition to build environments, modern software development tools and practices involve many other types of electronic artifacts: requirements, visual models, source code, binaries, documentation, test scripts, images, and Web content, to name just a few.

**Benefits of Using CM for Artifact Management**

Very much like source code, test artifacts -- including test plans, test harnesses, and test scripts -- evolve over time. As new functionality is added to any software application, new test cases are also developed to test that new functionality. Existing test cases may also be modified in order to reflect changes to an application under test. Managing test baselines, therefore, becomes extremely important as both an application and its associated test artifacts evolve. Testing a new product with old test assets, for example, will leave a gap in test coverage for new functionality. Likewise, testing an older version of an application with newer versions of test artifacts may result in false results (failures) if the new tests cover functionality not present in that older version.

Proper management of all development artifacts, from requirements documents to test assets, can lead to better communication among analysts, developers, and testers. Moreover, a federated repository of development artifacts promotes a common model for asset management and reuse.

**Rational Suites: A Powerful Federated Repository**
Like other configuration management vendors, Rational has evolved Rational ClearCase to manage a wide array of artifacts, and the company's vision is to provide a total artifact management solution through its Suites offerings, all of which currently include ClearCase. Because all of the products in these Rational Suites are tightly integrated, they already take configuration management a step beyond most other systems, making it possible for software project teams to manage a wide range of development artifacts within a federated repository.

Today, for example:

- Rational Suite Development Studio users already have the ability to store and version Rational Rose model artifacts within Rational ClearCase, right alongside the code their team is generating and/or reverse-engineering using Rose.

- Analysts using Rational Suite Analyst Studio can leverage capabilities that enable them to create baselines containing appropriate versions of both the requirements documents and requirements database artifacts for their projects; they can also create new requirements projects from baselines of existing projects.

- Rational TestStudio users can create and manage baselines of versioned test artifacts. Managing baselines of test assets enables testing professionals to ensure that the versions of test artifacts they employ match up with the applications they are testing.

**Looking Ahead**

The Rational Suites vision doesn't end with creating and managing baselines of development artifacts. In fact, that is only the beginning. In future releases Rational Suites will offer the benefits of configuration management capabilities and practices for other types of development artifacts. Features will include advanced operations required for parallel development, such as automated differencing and merging utilities for modeling, requirements, and test artifacts. These features will extend the existing capabilities, automation, and integrations currently offered by the Rational Suites products.

It's not hard to grasp the impressive benefits a federated repository of development artifacts could provide in terms of integration, usability, process enforcement, and metrics collection. Extending configuration management practices beyond the traditional limits of source code management can have a profound impact on all members of a project team, not just those who write the code. Configuration management really isn't just for source code anymore. Now, it's 4:30 p.m. Please pass the orange juice...

*For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided.*
Thank you!
The one constant in the Internet Age is change. Continual advances in software development and Internet and communications technology, coupled with increasing demands for information, have created a new, constantly evolving business environment driven by requirements to deliver quality products and services faster, the challenge to exploit new technologies, the prospect of new markets to discover and conquer, and a growing need for better customer relationship management.

For the software industry, these new demands mean that we will never again be "finished" with a plan or product. In the past we delivered new software releases on an annual basis; in the new economy, it is the norm to deliver them several times a year -- and even weekly or daily for some Internet-based companies.

Clearly, in a world that is characterized by such a heightened sense of urgency, companies and organizations must understand, embrace, and drive change if they hope to survive. What factors and dynamics must organizations understand to manage and drive change effectively? Then, based on this understanding, what are the best management practices for implementing change? This article addresses these questions.

Laying the Foundation for Organizational Change

When it comes to change, too often high-tech organizations focus almost exclusively on technology (new tools, processes, equipment, etc.) and all but ignore the need for modifications in culture and behavior. Change, by nature, intrudes on people's "comfort zones," so many equate it with pain, whether or not they think it will result in improvements. Inevitably, workers meet change with resistance, so it is critical for organizations to address this key issue up front.
In fact, it's important to recognize that cultural and behavioral factors are the essence of organizational change -- basic building blocks that lay the foundation for it. Unless these blocks are in place, technology introductions will fail to satisfy expectations and may even produce adverse results.

Briefly, here are some descriptions, along with questions that can help organizations assess their readiness for a change initiative:

- **Leadership** -- Is there adequate leadership at all levels? From the top executive down to the practitioner, leadership is necessary for driving and sustaining changes. Leadership comes in many forms, but commitment, conviction, and the ability to guide the organization through unknown territory are critical qualities for success.

- **Vision and Planning** -- Has the organization formulated a flexible plan, and does everyone understand its goals and objectives? Change, of course, does not always result in progress, so if the value proposition of a change is unclear, that is the first problem to address. The plan should be abstract enough to allow the organization to adapt to changing environments but precise enough to produce desired results in the early phases. The plan and vision document should also identify key dependencies (skills, equipment, etc.) and risk factors.

- **Reasonable Alternatives** -- Has the organization considered and studied options and alternative paths? The best-laid plans may have to be put aside if the unpredictable occurs. Organizations should identify and then examine alternatives within a "trade space" bounded by fixed constraints (budgets, timeline, needs vs. wants, etc.).

- **Culture and Behavior** -- Is the organization changing or willing to change the way it is structured and the ways in which it thinks, distributes incentives and rewards, and approaches new challenges? Leaders need to ensure that all stakeholders have a sense of urgency about implementing the prescribed plan and responding to new demands. Also, they may need to raise the bar for performance and quality.

- **Skills, Resources, and Personnel** -- What new talents, equipment, and experiences will be required to drive and sustain this change? In addition to the latest technology skills, project management, leadership, vision, interpersonal skills, and commitment are key. In addition, the organization must value training programs and make sure they play a key role in the overall solution.

- **Technology** -- What new technologies (tools, process, equipment, etc.) does the organization need and who will use them? To answer these questions, you must size up the leap between where the organization is now and where you want to go. Then evaluate whether you have the time, budget, and personnel to support that leap. Also, be sure that the new technology will allow the organization to scale and adapt to changing environments. And
Finally, assess whether the new technology will yield enough return on investment to make the effort cost effective.

Obviously, the extent to which each of these requirements needs to be addressed is dictated by the scope of change and the organization's complexity. Overhauling the business systems in an organization delivering a DoD Weapons or Avionics System (which typically involves many subcontractors, partners, and vendors) requires more planning, coordination, and management than developing a Web-based interface for an e-commerce customer within a small, nimble systems consulting firm. Both the plan and process for change should be scalable and tailored to meet the particular needs of the organization.

**Top Ten Best Practices for Implementing Organizational Change**

What are the best ways to ensure that these building blocks are in place and that your organizational change initiative will be successful? Although the primary mission of Rational Software's Strategic Services Organization is to assist customers in adopting new technologies and processes to accelerate their software development capabilities, we frequently get an inside view of our customers' organizational dynamics. Because of this, customers frequently turn to us for help in driving change throughout their organizations.

Through many years of experience, we have identified a number of best practices that are consistently successful in reducing risk and driving change. The following list, arranged in descending order of priority, is by no means exclusive, nor is it intended to be a recipe. Rather, it provides a high-level overview of key requirements and suggestions for bringing about effective organizational change, including many we touched upon in the "Foundations" discussion above.

These best practices are not unique to the high-tech industry; any organization in any market or industry can benefit by employing them. Also, keep in mind that in the Internet economy, changes affect not only your organization, but also your customers, suppliers, and partners. This is especially true if the change is a pervasive one that affects all components of your organization's business.

1. **Identify and agree on key change drivers.**

   Organizational change should be driven by a key business concern (time to market, quality, efficiency, etc.) that is significant, well understood, and articulated throughout the organization. If people perceive that you're making a change simply for change's sake, then you'll cause more problems than you will solve. If they perceive that there's a real problem or opportunity and understand the negative effects of not taking action, however, then stakeholders will support your change initiative.

2. **Create demand for change; don't mandate or force it.**

   This relates closely to number 1. If you want everyone in your
organization to adopt a change, then you must create demand for it at every level. Make sure each person understands the problems you are addressing and has a feeling of ownership for the solutions you're proposing. This "buy in" is required if you expect stakeholders to drive change and withstand setbacks and hurdles. When managers give orders demanding change, they meet with resistance, halfhearted attempts to adopt the change, false starts, enormous amounts of wasted time and money, and sagging morale. If you recognize that the change will not benefit all stakeholders involved -- some may suffer reductions in territory, budgets, or titles, for example -- it is best to address these issues immediately and formulate agreements with these stakeholders concerning their new roles and responsibilities. If you can't get their buy-in and support and you discover that they're undermining your efforts, then you may need to take more severe measures.

3. **Exercise consistent leadership and communication practices.**

This applies to everyone from top managers on down to group leaders. Lack of leadership and vision will cause the change effort to stall and lose credibility. Changes to your plan are inevitable, and it is critical to keep the troops informed about when and why you make them. In environments with poor top-down communication, people consistently look for signals or signs that let them know how things are going. Often, they identify the wrong signals or misinterpret correct ones. For example, if a project start is delayed, or if management is holding more "closed door" meetings than usual, then employees may incorrectly assume that things are not going well for the project.

It is important for the management team to communicate in all directions and to behave as a cohesive unit with a consistent message. If managers appear to be fractured and out of sync, then team members will quickly become concerned. A consistent regime of talking openly, sharing information, and helping others understand the common vision will ensure cohesiveness across the organization. It is also critical to keep the message simple so that it can remain consistent over time and not promote confusion or misinterpretation.

In particular, if managers decide to depart from the path they were on and pursue another, then they need to re-seek support and buy-in from other parts of the organization. If they fail to do this, then much time and money will be wasted trying to get everyone on the same page.

4. **Continually update and fine tune the vision and project plan.**

In the initial planning phase, creating the vision and project plan helps managers identify key issues and risks, establish good communication and strong relationships among stakeholders, and fix a starting point that everyone can agree on. It's important to remember, however, that these plans are not set in stone. The vision and project plan are living artifacts, not static documents (as they are frequently communicated). They provide a roadmap for change, but it will be necessary to correct the course and validate the direction throughout the implementation. Also, as new employees and teammates come on board, it is important that they understand and adopt the vision and plan.
5. Achieve incremental, demonstrable success.

Nothing can stall an effort like starting with an overly ambitious plan that tries to address every issue or project at once. Instead, the best strategy is to strive for incremental, demonstrable success as you implement the plan in stages. This will help stabilize the environment and build the confidence, credibility, and experience needed at each phase to manage chaos and sustain performance. The same approach works well for adopting a new process, including the Rational Unified Process. Start with a few key areas or workflows and add more as the organization's confidence and capability evolve.

For your first success, it's strategically important to choose a fairly complex project that management regards as highly important. In the course of such a project, you can identify dependencies, risks, and unforeseen problems up front. Plus, involving top players will not only increase the project's chances for success but also ensure more management attention and support.

6. Find champions for your solutions at all levels.

Early on, it helps to identify champions at all levels who support your solutions and can get the change process moving forward on the right track. These people can also provide valuable insights on the working environment and team morale. Remember that potential champions may take different forms and hold various titles, so be sure to do a little digging for the right people.

7. Acquire and develop new employee skill sets.

Change creates demand for modern techniques, skills, and mindsets. As you formulate a vision and plan, it's important to identify what new skills will be required to facilitate change and operate in the new environment. Very likely, you'll need a training program to keep the skills of current and future employees up to date. In addition to strong technical skills, you'll need top-level project, business, and personnel management skills. Software organizations often lack depth in these non-technical areas, but their importance should not be underestimated.

8. Establish a collaborative environment.

At the outset of any change initiative, it is essential to establish a collaborative working environment among all stakeholders and partners. Successful organizational change requires risk sharing, a collective sense of ownership, and the ability to leverage the knowledge and skill sets of people across the organization. Technical change, for example, requires bilateral knowledge transfer in disciplines such as domain expertise and modern software development techniques. Throughout the project, teams should strive to maintain a high-trust environment that fosters collaboration. If they employ on-site contractors to do the bulk of the process work, they can invite them to collaborate, but staff employees must retain ownership of the change process. After all, the organization's success is on the line, and employees have a far greater stake in the
9. Collect and use metrics to monitor progress.

How do you monitor project status and know when things are going well or going wrong? Too often, organizations spend an enormous amount of time planning how they are going to implement change, but too little on understanding which metrics are critical to monitor a project's performance. It is important to capture the right metrics (schedule performance, open/closed discrepancy reports, action items, cash burn rate, manpower profiles, EVA analysis, etc.) on a regular basis and use this data to perform course corrections as needed.

An initial metrics assessment can help team members identify which metrics are of value, learn how they are typically collected and analyzed, and understand what they reveal about an organization's current state. Then, this assessment can be used as a benchmark for moving forward and to get people thinking about how metrics can help their effort.

10. Overhaul incentives before you begin.

If you're asking employees to adopt a change that will require modifications in behavior, skill sets, and responsibilities, then you must also ask yourself, "What incentives do they have to make these adjustments and ensure the initiative's success?" Making changes to incentive programs before the change process begins will generate the energy and behaviors critical to success. This includes providing incentives for those who may lose something in the transition to keep their motivation and self-esteem intact. You can structure incentives around achieving milestones on or before schedule, a certain level of quality, or other specific goals, for example.

Find the Right Pace

How long will change take? Organizations move at different speeds when adopting new technologies or processes. The pace of change depends on many things, including how well management internalizes and implements best practices, and creates an environment conducive to continual change. And even within an organization, the speed of individual projects will be different. Although it's important to move as quickly as possible to achieve time-to-market goals and competitive advantages, it's also important to determine the correct pace for each team. Move too fast and quality will suffer; teams will get frustrated and lose confidence. Move too slowly and the window of opportunity will close; plus, teams may begin to question the value of change and lose patience.

As with any learning experience, you can expect to encounter setbacks and hurdles. View these as opportunities to gain knowledge that the organization can build on, and don't allow your teams to get discouraged and lose focus. Make an effort to create an environment that advocates and rewards some level of risk taking. Without this, people will not be bold enough to propose new ideas and push for innovation. When change ceases, so does improvement and growth.
For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
Enterprise Java and Rational Rose - Part II

by Khawar Ahmed
Technical Marketing Engineer
Rational Software
Loïc Julien
Software Engineer
Rational Software

This is the second installment of a two-part series exploring the synergistic relationship between Java 2 Enterprise Edition (J2EE), a popular development platform for distributed enterprise applications, and Rational Rose, the industry’s leading CASE modeling tool. Part I provided an introduction to J2EE from an architectural perspective. In Part II, we’ll first explain how Servlets and Java Server Pages (JSPs) work within the J2EE architecture and then go on to show how Rational Rose can help developers build better J2EE applications.

What Is a Servlet?

As we saw in Part I, an enterprise application may (but does not necessarily) consist of several tiers. Tiers are primarily abstractions to help us understand the architecture. The J2EE architecture usually involves four distinct tiers, as shown in Figure 1.
Servlets reside on the Web tier and typically act as the intermediary between the client tier (end users) and the other tiers (e.g., the business or the EIS tier).

A servlet is really a Java class that is invoked in response to a user's command. The invocation of the servlet follows a procedural request/response process. The Servlet 2.2 specification introduces the notion of a Web Application: the component that is deployed within a servlet container. A Web Application is packaged as a file with the extension ".war" in a format similar to that of the Java Archive (.jar) file.

You've probably used or seen Web sites that use CGI scripts to create an HTML page on the fly. That is exactly the kind of functionality servlets provide, but they are much lighter weight. Unlike CGI scripts, servlets don't require a new process for each HTTP request. All they need is a thread. Plus, they are much easier to maintain.

When a servlet container starts up, it typically creates a pool of servlets. Once the servlets are initialized, they are ready to process requests. For an idea of how servlets work, let's say you visit a Web site that has a servlet for processing registration requests. When you fill out a registration form, and press "Submit," the servlet receives the request and extracts the information you provided in the form via a "request object." (The request object is created by the container and provides an object-based access to the form data.)

Then the servlet does whatever it needs to do (save the information in a database, for example, or ask other objects such as EJBs to do so) and then composes an HTML page via the "response object." The page is then sent back to you, perhaps displaying a "registration successful" message and confirmation number.

By definition, a servlet is stateless. If you want a "stateful" servlet, then
you can use either the HttpSession object or the Servlet Context object. The HttpSession object is dedicated to a client session; the Servlet Context object lasts for the life of the application.

If you are familiar with the pre-J2EE servlet specification, you might be wondering where the servlet engine fits into the Servlet 2.2 architecture. As it turns out, the servlet engine is now called the servlet container, which we mentioned earlier. The new specification also introduces a few new concepts, such as an application that can be distributed within several containers.

Although servlets are wonderfully fast, lightweight, and powerful, servlet-based development suffers from one very fundamental flaw. The specification states that

\[\ldots\text{the servlet must contain the presentation aspects as well.}\]

This requirement can be difficult to implement and even more difficult to maintain, since the servlet developer and the presentation creators are, more often than not, different people. The presentation is embedded inside Java statements, which makes it difficult to comprehend. Moreover, the presentation is likely to change more frequently than the logic behind it.

So what's a developer to do? Use JavaServer Pages, or JSPs.

**JSPs: Stepping into the Breach**

Built on top of the Servlet architecture, the JSP technology allows for a clear separation between presentation logic and business logic. It provides all the power of servlet technology while allowing the servlet developer and the content developer to work independently.

JSPs are great for creating dynamic content. They excel at rendering dynamic HTML/XML documents and provide an easy way to associate template data (XML or HTML) with directives and actions that supply dynamic behaviors on Web pages. These behaviors can be tied to specific requests or entirely independent of any request.

Basically, the JSP technology is a mix of HTML code and Java bits of logic (called scriptlets). A JSP gets compiled dynamically into a servlet at runtime, essentially yielding servlet capabilities without the drawbacks.

Figure 2 shows a sample JSP.
The J2EE and the Model-View-Controller Paradigm

The J2EE programming model applies the Model-View-Controller design pattern to J2EE development. This allows enterprise application developers to separate tiers efficiently and associate many views with one model. In fact, for any client-server architecture, it's important to isolate the real logic of your application from your end-users so that you can easily change the look and feel of your application without impacting the logic.

Figure 3 shows one representation of this design pattern.
Figure 3: Model-View-Controller Design Pattern Applied at the Enterprise Level

Many design patterns (best practices) recur in J2EE applications: shopping patterns, security patterns, transaction patterns, patterns that involve dependent objects such as value or data access objects, patterns that require a session-to-entity exchange, and so on. If you're designing your own e-commerce application with a Web-centric architecture, remember that an EJB-centric solution will be easier to adopt later if your design clearly separates presentation and business logic, as shown in Figure 4.

Figure 4: Class Diagram of a Typical Shopping Pattern

Figure 5 shows how to map your Web-centric application to an EJB-centric application.
Developing J2EE Applications with Rational Rose

Rational Rose introduced support for J2EE via the Rose Tech Preview release in July 2000. Rational Rose supports the following capabilities related to J2EE:

- Servlet development, including generation of a J2EE compliant deployment descriptor.
- Entity Bean development, including generation of a J2EE compliant deployment descriptor.
- Session Bean development, including generation of a J2EE compliant deployment descriptor.
- Ability to invoke a Java compiler to compile files.
- Creation of Web and EJB archive files from within Rose.
- Synchronization of home and remote interfaces with the EJB class.

We introduce you to these capabilities in the following examples but do not cover the dialogs in great depth.

Creating and Modifying Servlets

Rational Rose provides a quick and easy way to create and modify servlets. To create a new servlet, right click on a class diagram. Then, from the shortcut menu, select J2EE>New Servlet. This brings up the dialog shown in Figure 6, which allows you to quickly configure the servlet you want to create. Note that the dialog is multi-level (Servlet Class Properties, Advanced Properties, Deployment Descriptor). You can move back and forth among the different levels by clicking the desired item listed on the left.
Figure 6: Servlet Configuration -- Class Properties Dialog

In the Class Properties dialog, you can specify the name of the servlet, whether it is based on the generic servlet or the http servlet class, and so on. You can also choose the exact operations you want to generate for the servlet: e.g., doGet, doPost, destroy, init, etc. Let's specify the name as "myServlet," a subclass of the http servlet class, and select the following operations from the list: init, service, doGet, destroy.

If we click on "Advanced Properties" on the left, we bring up the Advanced Properties dialog shown in Figure 7. This lets us specify whether we want to use a context object, and the exact operation of the servlet in which we want specific activities to take place. For example, if we check getSession and choose doGet from the list, a call to getSession will be placed in the doGet operation at code generation time. Similarly, if we set a BufferSize limit of "7," then a setBufferSize() call will be placed in the doGet operation at code generation time.

The resulting doGet operation is shown below:

```java
/*
Method: doGet
*/
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

/**
   Servlet Code Generation Process
   -- Content Type --
/**
   response.setContentType("text/html");

/**
   Servlet Code Generation Process
```
Next, let's click on Deployment Descriptor on the left, which brings up the dialog shown in Figure 8. Servlets use XML-based deployment descriptors as a way to communicate various configuration details to a container or Web server. At code generation time, Rose J will generate the XML deployment descriptor based on the information we specify. This may include a more descriptive name for the servlet, initialization parameters required for the servlet, a session timeout value, and so on. We can specify the Init and Context parameters by using the radio buttons. We can toggle between Name and Value, and enter the name and value in the text field on the right.
We can also modify an existing servlet (created via the process described above) by selecting the servlet class on a class diagram, right clicking, and then choosing J2EE>Servlet-configuration from the shortcut menu. This brings up the same dialogs, pre-populated with the appropriate information.

Creating and Modifying Enterprise JavaBeans

Working with EJBs is equally simple, and we can create them in a couple of ways. One way is to click on the class diagram and then choose J2EE>New EJB from the shortcut menu. The preferred method, however, is to make a design/implementation decision by turning an analysis class into an EJB. We can select the class, right click to bring up its shortcut menu, and select J2EE>EJB-Configuration. This simplifies the process because we do not have to specify names for the EJB or the remote and home interfaces. Rose uses the name assigned to the class we are converting as the basis for establishing the other names!

Selecting J2EE>EJB-Configuration brings up a dialog similar to the one for creating a servlet. The dialog has three levels: EJB Class Properties, Persistence Properties, and Deployment Descriptor.

The Class Properties dialog, shown in Figure 9, allows us to choose the type of EJB we want to create, as well as the bean, home, and remote interfaces. For instance, we can choose to create an entity bean with container-managed persistence. We can call it MyEntityEJB, and name the home and remote interfaces MyEntityHome, and MyEntity, respectively.
Clicking on Persistence Properties on the left side brings up the Persistence Properties dialog shown in Figure 10. When we create entity beans, we can specify that the primary key class should be based on some pattern, or we can assign a specific name for the class. We can also define finder methods. (Note that the first method is defined this way; the rest can be defined by selecting the EJB, right clicking, and then selecting J2EE->EJB-New ejbFind Method from the shortcut menu, which brings up a dialog similar to the one in Figure 10.)
Clicking on Deployment Descriptor on the left side brings up the Deployment Descriptor Configuration dialog shown in Figure 11.

![Deployment Descriptor Dialog](image)

**Figure 11: EJB Configuration -- Deployment Descriptor Dialog**

### Compiling Java Files from Within Rose

Rose J also introduces a modeless, auto sync editor for use with Java. Once it’s installed, the new editor becomes the default browser for Rose, accessible via the Java>Browse Java Source command.

Via the new editor, we can link in an external Java compiler and then compile files from within Rose. This is useful for quick compilation and is not intended as a replacement for a full-featured IDE. To link in our Java compiler, we’d enter the path to it via Build>Java Compiler Options. The dialog is shown in Figure 12.
To invoke the compiler, we choose the Build>Compile menu option from the modeless editor main menu.

**Building EJB and Web Archive Files from Within Rose**

Once we've compiled our Java classes to .class files, Rose J offers a convenient way of building EJB and Web archives. Access to this capability is via the J2EE>New EJB-Jar file, the J2EE>New War file menu options off the class shortcut menu, or the Tools>Java>J2EE menu. Figure 13 shows the dialog.
In the first field, we can enter the path to the jar utility. Then, we click on the [...] next to Archive Name field, choose the directory path, and enter the file name for the jar file via the file dialog. We can then select and add the appropriate .class files and the XML deployment descriptor we generated from Rose for the EJB (or servlet, if we're dealing with a Web archive file).

**Automated Interface Synchronization**

One of the challenges developers face when using EJBs is the need to keep the EJB class and the home and remote interfaces in sync. For example, each time you add a new business method and want to expose it via the remote interface, you must also add the method to the remote interface class. Rose automates this process by providing a synchronization menu option that's invoked from the J2EE>EJB-Update>Update Interfaces (from EJB) or from J2EE>EJB-Update>Update EJB (from Interfaces menu options).

Rose relies on stereotyping to distinguish between different types of methods, so if you create the methods yourself, be sure to assign them appropriate stereotypes. For example, business functions must be stereotyped <<EJBRemoteMethod>>. Find methods must be stereotyped <<EJBFinderMethod>> and so on.

**The Future of J2EE and Rational Rose**

The J2EE platform is evolving rapidly to meet the demands of current users and vendors. A new J2EE 1.3 specification is now in development via the Java Community Process, and drafts are already available. This new version includes revisions and refinements for many of the technologies and features we have discussed, introducing greater consistency and tighter integration throughout the platform.

The new specification also introduces connector architecture. A weakness of the J2EE 1.2 specification is that it doesn't provide an easy way for enterprise

---

### A Quick Preview of the New EJB 2.0 Architecture

- This new version emphasizes portability. Although a goal for the EJB framework is the ability to move ejb-jars from one container to another without any rewriting or recompiling, an area such as persistence, for example, might be managed differently in different environments. The container-managed contract was rewritten to resolve this problem.

- In the EJB 1.1 architecture, an entity bean was likely to be represented as a row in a database. In the 2.0 container-managed architecture, the persistent state schema of the bean is much richer.

- The EJB 2.0 architecture brings an asynchronous and messaging dimension to the EJB framework with Java Messaging Service technology
and the definition of a new, message-driven type of bean. A message-driven bean is defined as a single class with no home and no remote interfaces (to avoid any internal or external direct access).

- The new specification introduces EJBQL, an SQL-like query language for finder and select methods. This powerful language will operate at the logical level, not the data level, and is a portable definition for finder methods.

- The EJB 2.0 specification introduces support for an additional type of method: the home methods, which are equivalent to static methods. They are not attached to one particular instance. Home methods are declared in the home interface and defined in the implementation class.

- Interoperability between J2EE application servers has been enforced with a requirement to support the CORBA/IIOP protocol. In this configuration, IORs (interoperable object references) are associated to the EJBHome and EJBRemote, and IIOP messages are sent from one J2EE server to another. This powerful requirement allows EJBs to be accessed by non-Java clients.

- Other technologies have also been revised, including the Java Servlet specification, which introduces new capabilities (e.g., support for event notification and filtering), and the JSP specification, which has added mapping from JSP to XML.

As a future direction, J2EE v1.3 points to the XML data binding specification, which is a way to realize XML schemas by Java classes. Although it's still under development, this specification previews the important role that XML will play in the future specification of the Java 2 platform, Enterprise Edition.

Given J2EE's responsiveness to user demands, its popularity as a development platform -- which is already growing rapidly -- is likely to accelerate in the future. Although the learning curve for J2EE is steep,
developers will find that Rational Rose can ease the transition to this platform by automating and simplifying many aspects of the development process. Visit the Rose Upgrades page at http://www.rational.com for free Rational Rose Evaluation software.

References


For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
Designing Component-Based Architectures with Rational Rose RealTime

by Reedy Feggins
Senior System Engineer
Rational Software

Rose RealTime is a comprehensive visual development environment that delivers a powerful combination of modeling notation, design capabilities, and debugging tools to meet the challenges faced by real-time designers. Rose RealTime provides software architects and embedded system developers with a unique tool for rapidly prototyping, designing, and constructing executable models. Many new Rose RealTime designers, however, find it difficult to create robust models when they first start using the tool. This article first examines the goals and challenges of developing and deploying real-time systems and then describes an easy-to-follow approach to architecting a scalable, component-based system with Rose RealTime.

The Challenges of Developing Real-Time Systems

Real-time systems range from small embedded applications (e.g., for cell phones, handheld devices, or set-top boxes) to highly evolved distributed applications (e.g., for airplanes, switches, and automotive systems). Developers have created and deployed real-time systems for years, but the process offers challenges not normally encountered in other development domains:

- Most real-time systems must run on small, customized operating systems with limited memory and strict performance requirements (e.g., response times and throughput).
- Because many real-time systems are deployed in highly critical applications, developers must provide fault-tolerant capabilities that isolate the result of a subsystem fault/failure, so that the application can continue to operate at a degraded level.
Real-time systems require tight integration among application software, the real-time operating system (RTOS), and the target hardware; this is contrary to the goals of most good system architectures.

**Architecting Real-Time Systems**

So how do developers meet these special challenges? We can begin to address that question by looking at the basics of architecting a real-time system.

The main goal of a real-time system architect -- like that of all other system architects -- is to create a "good" system. In this instance, "good" means the system will meet corporate and customer needs for:

- **Reliability** -- Does the system have just a few bugs (note that no system is completely free of bugs)?
- **Flexibility** -- Can the system evolve to meet changing user needs?
- **Affordability** -- Can the organization afford to build and maintain the system? To sell the product in which it is embedded?
- **Availability** -- Can the system run on available operating systems and hardware?

System architectures are used by diverse sets of people at different times in the development cycle, so often these architectures are separated into different views. This approach enables all the different groups of stakeholders to communicate needs (i.e., requirements), resolve conflicts, and document decisions. The Rational Unified Process (RUP) identifies a standard set of views called the *4+1 Architecture Views*, depicted in Figure 1.
The +1 refers to the **Use Case View**, which contains the key use cases that drive the architecture. The other four views are:

- **Logical View** that describes the software structure and is used to identify major design packages, capsules, and classes.

- **Process View** that addresses the concurrent aspect of the system at runtime: tasks, threads, or processes, and their interactions.

- **Deployment View** that shows how the various executables and other runtime components are mapped onto the underlying platforms or computing nodes.

- **Implementation View** that provides the software's static organization in terms of packaging, layering, and configuration management.

Together, these views provide the architect with the necessary perspective to adequately model and build a complex real-time system. Although the 4 + 1 Views don't map exactly to the UML models (or, therefore, to Rose RealTime), let's take a look at how Rational Rose RealTime supports them.

**The Use Case View**

As for most software systems, architecting for real-time systems involves understanding the short- and long-term requirements and deciding on necessary tradeoffs to achieve the highest quality system. The Use Case
Architecture View, which describes functional requirements from the user's perspective, is documented in a set of diagrams. This view (the +1) maps precisely to the Use Case View in Rose RealTime. For most systems, only a subset of these diagrams are architecturally significant. These architecturally significant diagrams describe usage scenarios that:

- Require multiple software components.
- Identify how components communicate or are organized (see Figure 1 for an example).
- Describe configuration, administration, or maintenance scenarios for multiple components.
- Specify qualities that make the system competitive, cost-effective, or worth building.

![Figure 2: An Architecturally Significant Use Case Diagram](image)

The RUP also suggests using a "Robustness Analysis" to map a system's use cases into three classes:

- **Control Class** -- used to manage application services or resources typically mapped to active classes.
- **Boundary Class** -- used to identify interfaces between actors and the system under development.
- **Entity Class** -- used to contain persistent data typically mapped to passive classes.

In most cases, these "Robustness" classes are mapped as shown in Figure 3.
The Logical View

The Logical View provides the architect with a mechanism for describing the system software structure in terms of major design packages, capsules, and subsystems. The logical view is represented in Rose RealTime by class and structure diagrams\(^2\) used to show the architecturally significant artifacts in the view:

- Major classes that have high visibility throughout the rest of the application
- Business domain and persistent classes
- Active classes (or capsules)
- Protocol classes
- Classes critical to system integrity

Grouping and organizing the application elements into packages is the best approach for depicting architecturally significant components and subsystems. Typically, system architects use packages to illustrate dependency relationships between different parts of an application. Packages are useful for showing architecture layers, as depicted in Figure 4. They are also units of configuration management that can be used to organize the building and evolution of actual software elements.
Capsules are the basic building blocks that Rational Rose RealTime modelers use to provide encapsulated data and operations as well as support for concurrent events. They also represent capsule-related roles and responsibilities. Possessing many of the characteristics of a class (e.g., attributes and operations), capsules are class stereotypes used to depict active (or concurrent) responsibilities that the application must exhibit. Each capsule represents an independent flow of control in the system under development. Using Rational Rose RealTime, application developers use state diagrams to model class behavior and structure diagrams to depict object interaction. Like passive classes, capsules communicate with each other through messages, via well-defined interfaces called "ports."
Many Logical View diagrams can be used to provide architecturally significant information for constructing real-time systems, but the structure diagrams associated with each capsule provide the most complete representation of the software's dynamic structure. The example in Figure 5 depicts the structure diagram for an Asymmetric Digital Subscriber Line (ADSL) project.

**The Process View**

In Rose RealTime, vital aspects of the Process View -- namely determining the threads of control within the application -- are focused in modeling elements: capsules and components. By definition, each capsule (i.e., active class) has its own independent thread of control that is implemented using its state diagram. Each capsule can be mapped to a specific component (or set of components), which can then be built as an executable library or external library. Each component implements a non-trivial set of functionality and may not necessarily be a single unit of code or a single binary. System architects can use both capsules and components to depict units of distribution, assembly, and deployment. Further, as shown in Figure 6, designers can assign capsules to specific runtime threads, thereby enhancing the modeling capabilities of the Process View.
The Deployment View

The Deployment View gives the architect the means for describing how components and subsystems are distributed across system nodes. Deployment diagrams depict architecturally significant elements such as processor nodes, device nodes, connections, and runtime components (e.g., executables). Figure 7 is a typical deployment diagram.
**Implementation View**

The Implementation View focuses on the software development environment and how it is organized. Typically, applications constructed using Rose RealTime are organized into a set of hierarchy layers, realized using subset of the packages found on class and component diagrams. Each layer should provide a well-defined interface (using capsules and, in some special cases, interface classes) to the layers above it. The Implementation View assists architects in describing partitioning, reuse, and visibility guidelines to the team. In Rose RealTime, the Implementation View is realized using class diagrams to depict model organization -- with packages, interfaces, and subsystems. Architects must continuously review and assess the impact on these guidelines as changes are made by team members.

**Resolving the Architect's Dilemma**

Any well-designed architecture separates design concerns from abstract implementation details, supports the testing of independent components, enables greater reuse of larger subsystems or frameworks as well as low-level classes, and ultimately increases system longevity. It also provides several key benefits during the development process:

- Good communication as well as understanding of tradeoffs among
stakeholders regarding major design decisions;

- Efficient resource management (development teams, hardware and software environments, and capital resources);
- Effective coordination of project schedules and risk assessment efficiency.

In designing a "good" architecture, real-time architects face a few special challenges. They must balance the need to focus on the big picture with the need to design component-based architecture that will work seamlessly with the RTOS and target hardware. Rational Rose RealTime provides the support they need to accomplish this. Using the 4+1 Views, system architects and application developers can better model system responsibilities and create reusable elements.

References


1 An architecture view is a simplified description (abstraction) of a system from a particular perspective or vantage point, covering particular concerns and omitting entities that are not relevant to this perspective.

2 Structure diagrams are stereotyped collaboration diagrams used to graphically depict the active objects and their communication paths.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
Some Guidelines on How to Use the Universal Design Pattern

by Koni Buhrer
Software Engineering Specialist
Rational Software

In a two-part series published in the December and January issues of The Rational Edge I introduced a Universal Design Pattern based on a first principle of software development. (See "Searching for First Principles of Software Development" and "Rules of Software Design.")

As several readers have observed, applying the Universal Design Pattern to the design of an entire software system can be a challenge. It is not always obvious how the design elements -- data entities, I/O servers, transformation servers, and data flow managers -- should be used and combined. This article is a response to these readers’ specific questions; I welcome additional queries on this topic and will respond to them in future issues of The Rational Edge.

A Pattern for an Entire Software System

Let me begin this article by emphasizing that the diagram in "The Rules of Software Design" represents a whole system rather than a building block for constructing systems. The Universal Design Pattern characterizes the way an entire system should look, not its building blocks. The building blocks of the Universal Design Pattern are individual data entities, I/O servers, transformation servers, and data flow managers.

What I’m proposing is that the fundamental structure of any system, no matter how complicated, would still be similar to that of the simple system represented in the diagram. See, for example, the more complicated system design in Figure 1 below, which has several data flow managers surrounded by many I/O servers and many transformation servers, exchanging many different data entities.
Note that the Universal Design Pattern withdraws somewhat from the position of pure object technology embodied in the Rational Unified Process. Object orientation is not a silver bullet for success in software development, and other approaches have merit, too.

While the Universal Design Pattern is firmly based on object orientation, it incorporates concepts that are often considered to be functional design concepts. For example, the Universal Design Pattern forces a software developer to:

1. Divide the objects of a software system into four realms with different functional responsibilities.
2. Use discipline when designing objects and object interactions, such that functional complexity is exposed at a high level.

Although we would accord an artist complete freedom in using objects, an engineer needs to restrict object use to ways that result in sound designs.

**Using the Design Elements to Create Large Systems**

The design elements of the Universal Design Pattern have different purposes within a software system, and different guidelines apply to each of them. We'll start with the data flow managers.

As a rule, a data flow manager must always perform a task that is substantial and meaningful for the system as a whole. A data flow
manager should never be used merely to perform a simple computation step like "have internal data element A transformed into B." Although that may be one of many actions the data flow manager performs, it should not be the only one. Avoid designs with chains of data flow managers, in which each data flow manager performs a simple action and then hands off the output data to the next one.

Ideally (and this is where the name comes from) a data flow manager should guide the flow of a data stream across the entire system, from the I/O server where the input data is created to the I/O server where the output data is consumed. Of course, it does not always work out this way. Nevertheless, because a data flow manager must always perform a substantial, meaningful task, it cannot be broken down into any smaller objects, other than the actions it performs.

Furthermore, data flow managers are never hidden within other system elements, only to emerge at a later design stage. All data flow managers must be present in the highest-level architectural design.

Unlike data flow managers, the other system elements can be -- and in large systems usually are -- composed of more fundamental objects of the same kind (!):

- Each data entity is an object of a data class. As such, data entities are subject to whatever modeling techniques the object-orientation paradigm provides. Data entities are often derived from other data entities (inheritance) or composed of other data entities (aggregation). Data classes can be instantiations of parameterized classes.

  Neither the detailed structure nor the full set of primitive operations of a data entity needs to be exposed in the highest-level design. New data entities always emerge during lower-level design of a data entity, as its data structure and its primitive operations are refined.

  Note that it is a mistake to start data entity design by identifying objects in the problem space. The top-level data entities should be the objects a software system receives or transmits through its I/O servers. As these data entities are refined, relevant problem space objects will eventually reveal themselves, but they should not be the starting point.

- Each transformation server implements an algorithm. These algorithms can often be broken into multiple steps, each represented by a subordinate transformation server. Furthermore, a transformation server may perform a transformation step that is already implemented (as an algorithm) by an unrelated transformation server. The former can thus invoke the operation of the latter.

  None of these details needs to be exposed in the highest-level design. New transformation servers and new data entities normally emerge during detailed design of a transformation server, as its algorithm is refined.
Each I/O server provides operations to obtain input data from -- or send output data to -- an external interface of the software system. Higher-level I/O servers often rely on subordinate, lower-level I/O servers to perform the actual input or output task. The higher-level I/O servers thus invoke operations of the lower-level I/O servers.

The details of how an I/O server performs its input and output operations, and whether it relies on any lower-level I/O servers, do not need to be exposed in the highest-level design. New I/O servers and new data entities normally emerge during detailed design of an I/O server.

**Interactions between Data Flow Managers**

Data flow managers are allowed to *explicitly* synchronize with each other. Explicit synchronization is an action of each data flow manager involved. During explicit synchronization, data flow managers can also pass data entities to each other. Hence there is no need to arbitrarily introduce I/O servers representing internal data buffers just so that two data flow managers can exchange data.

Of course, if system requirements call for an internal database, then by all means use an I/O server to represent it. Multiple data flow managers that access the same I/O server (database or external interface) are said to *implicitly* synchronize with one another. Implicit synchronization is also called *data-related* synchronization. Implicit, data-related synchronization between data flow managers is fairly common.

**Note, however (and this is a heavy one), that you should keep data flow managers as freewheeling as possible. Avoid explicit synchronization among them if at all possible!** Never create software designs in which data flow managers routinely synchronize with each other just to make sure they are in synch. Implicit, data-related synchronization is less problematic, but it too should not be introduced arbitrarily.

Sometimes there is no other solution than to let data flow managers synchronize. But because of the mechanistic way in which we tend to think, we often introduce synchronization points unnecessarily.

Let's assume, for example, that two data flow managers are in control of processing the same data entity for two different purposes, and must do so concurrently. The obvious solution would be to have the two data flow managers synchronize at the point where one retrieves the data entity (from an I/O server). Each data flow manager would then take a copy of the data entity and go off to perform its actions. We might even consider introducing a third data flow manager, which would only retrieve the data entity from the I/O server and then hand it off to the two original data flow managers for processing.

There are two alternative solutions, however, that are far better. They do not require the data flow managers to synchronize explicitly, but instead make clever use of the I/O server that provides the data entities to be processed:
Before the I/O server makes a new data entity available, it checks that both data flow managers have retrieved the previous data entity. The I/O server thus forces a data flow manager to wait if it gets too far ahead, which gives the other data flow manager a chance to catch up. This solution avoids explicit synchronization between the data flow managers but introduces implicit, data-related synchronization.

The I/O server lets a data flow manager skip a data entity if it falls too far behind. This lets the data flow manager catch up at the cost of a minor, temporary performance degradation. This solution avoids synchronization between the data flow managers altogether. Of course, skipping a data entity is acceptable only if it doesn’t happen too often.

As a rule, then, avoid explicit synchronization between data flow managers and avoid arbitrary I/O servers to represent temporary data buffers.

**Using Concurrency Wisely**

It is quite obvious that the Universal Design Pattern yields systems that have some amount of concurrency. And designing for concurrency is a good strategy up to a point, but I am not suggesting that you use massive concurrency. Let me elaborate.

Say your software system presents a single user with a sequence of screen pages. On each page the user can enter data or perform some other action that the software records. The software then processes the user input, stores it in a database, and goes on to the next page. This system is very simple really, as shown in Figure 2 below:

![Figure 2: A Simple Input Processing System](image)

A single data flow manager performs a long sequence of actions,
including:

- Invoking the user I/O server operations, both to display output pages and to read input data -- possibly many times.
- Invoking transformation servers to process input data, format output data, analyze data already stored in the database, and compute predicates that determine further actions - possibly many times.
- Invoking Database I/O server operations to store and retrieve data - possibly many times.

The data flow manager does not need to perform all its actions sequentially. It can also perform actions repeatedly or conditionally, depending on predicates computed by transformation servers.

So how much concurrency does this system require? You'd need one thread for the data flow manager, a second thread for the database I/O server, and a third thread for the user I/O server. So the design would imply that the system has three concurrent threads. If the same software system had to serve multiple, independent users, you'd need a separate data flow manager for each user. So for an N-user system you'd need N data flow managers and N + 2 concurrent threads in your design. I think that's very acceptable.

One final thought: There is no requirement that you have to implement each design thread by an operating system thread or an Ada task. Sometimes you can get away with less. Say, for example, that you have a typical Win32 application on a PC. The application has a window on the screen and waits for user input or some other user action. When the user provides input or performs an action, the application processes the event, updates the window, and waits for more input. This can be modeled easily, as shown in Figure 3:
The data flow managers CB1, CB2, and CB3 each wait for an event at different Win32 interfaces. When an event occurs, a data flow manager springs into action to have the event and its associated data processed.

"Wait!" I hear you say. "That doesn't work, because Win32 doesn't provide operations a client can call to wait for events!" Quite right. The Win32 interfaces are based on callback routines. A client has to provide Win32 with a callback routine, which Win32 then calls when an event occurs.

So let's just implement each of our data flow managers CB1, CB2, and CB3 by a callback routine!

Please consider: When a callback routine executes its final *return*; statement, the callback routine effectively waits for the next event. (Yes, I agree: *return*; is a rather funny way to implement the "wait for next event" action, but that's what the callback routine effectively does.) Similarly, when Win32 calls the callback routine, it effectively stops waiting for an event and goes on to perform its sequence of actions. The callback routine may, for example, invoke transformation servers to process input data or invoke operations of the Win32 interface to display output data. Eventually the callback routine will reach its *return*; statement again and start waiting for the next event.

So in this case you can actually implement three data flow managers (in fact, any number of data flow managers) and an I/O server with a single operating system thread.

**How to Treat External Files and Databases**

There are two fundamentally different ways in which we can look at an external file or a database:

- As a data object with somewhat peculiar and convoluted component access rules.
- As an external interface from which the software can read and to which the software can write data records.

Both views can be valid and useful, depending on the circumstances.

If we consider a file (or a database) to be a data object, it makes a fine data entity. A transformation server can read from or write to the file just as it can access the public components of any other data entity. Reading or writing a file thus becomes a data transformation: The data is transformed either from its file format to some internal representation, or from an internal representation to the file format. And a transformation server can read from or write to a file data entity anywhere within its guts without intervention of data flow managers and I/O servers.
Treating files and databases as data entities is very common. I myself have written numerous programs doing just that. And I believe that modeling files and databases as data entities is fine, as long as the program is essentially sequential and free of real-time constraints. If a software system contains substantial concurrency or is subject to substantial real-time constraints, however, then data entities with hidden external interfaces -- like files and databases -- are very problematic.

Here's why: External interfaces typically cause an execution thread to be delayed (due to hardware latency), to synchronize with other threads (due to mutual exclusion), and to block (due to data unavailability). Sure, these issues are well known, and a software design can deal with them -- usually. What makes data entities with hidden external interfaces so dreadful is that they can introduce delays, synchronization points, and blocking into a design secretly and in subtle ways.

Consider this: To determine whether a software architecture is viable (for example, whether it can meet timing requirements), a software developer usually looks at the high-level, architectural design. At that level the details of data transformation operations and data entities are rarely exposed. It is therefore very easy for the software architect to overlook the delay, synchronization, or blocking issues that are introduced by a data entity with hidden external interfaces.

Moreover, even if the software architect is aware of the data entity, it may be difficult for him to gauge the impact that the hidden external interface will have on overall system performance. After all, a data entity can be used in any number of places in any number of ways. And even if the software architect takes all those possible interactions into account, there's still a chance that during detailed design, someone else may use the data entity (and its hidden external interface) in a new and unexpected way, thus wrecking havoc on the system.

This is poison for a real-time software project! Data entities with hidden external interfaces have a high capacity to induce late design breakage and late scrap/rework -- the most common causes for project failure. To make it worse, a human factor may aggravate those risks: Software developers have a tendency to defer problems they don't know how to deal with. An architect may thus be tempted to hide a hard-to-deal-with external interface inside a data entity, thinking the problem can be solved later, during detailed design. That's dangerous, because that data entity might introduce delay, synchronization, and blocking issues that will affect overall system viability.

It is a goal of the Universal Design Pattern to expose the architecturally complex issues at the highest design level and to encapsulate the tedious but architecturally simple bulk of the software in data entities, transformation servers, and I/O servers. Hiding external interfaces inside data entities can defeat that goal. Hence, as a rule, data entities with hidden external interfaces must be used only in software systems that are essentially sequential and free of real-time constraints.
For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
From Use Cases to Role-Based Security Components

by Enricos Manasssis
Technical Architect
Rubus

The sophisticated, built-in security functionality of application servers such as COM+ and J2EE has greatly simplified the development process. The crucial and often complex decision-making process surrounding security role assignments within an organization, however, still puts the development process, along with project timetables, at risk. This article describes a systematic method for assigning role-based security when installing a component-based application in an application server. The method is based on the application’s use cases and a UML System Model, which can be constructed with Rational Rose.

Security: Benefits of Built-In Server Functionality

Modern software systems are typically implemented as applications running within the controlled environment of an application server. This control takes the form of a series of services, typically transaction monitoring, connection thread and object pooling, queued components, loosely coupled events, and security.

For application developers, the primary benefit in using a controlled environment is that you do not have to worry about implementing these services; the application server can implement them better and more effectively. You can focus on the business logic of the application and leave the rest to the server. In the past, it was common for software engineers to develop, in system after system, a means of access control to the various functions of the system. Obviously, very few developers today enjoy developing from scratch things like security mechanisms for their applications. Most experienced developers know how difficult and tricky this can be, so they welcome ready-made solutions. And that is exactly what an application server like COM+ and J2EE offers for role-based security.

Every object within an application has a series of publicly accessible methods. The principles of role-based security are as follows:

- Application developers identify roles that can access the objects. A role defines a category of actual users.

- Each role is associated with a list of one or more actual system users. Conversely, an actual user can be associated with more than one role, reflecting different roles the user plays within the organization.

- Each object and each method of the application is associated with a specific role.
All this is done using the configuration tools that come with the application server. The application developer can work on the application without concern for the security services. When the application is installed in the application server, the role-based security configuration can be completed. The application server will monitor all calls to the objects and their methods and enforce security access according to the configuration. The application server has the responsibility to identify the actual user who is making a particular call. The outcome of this identification depends solely on the configuration of the application in the application server.

The Fly in the Ointment: Organizational Decision-making

Having this built-in functionality can certainly make the development and deployment process nice and easy, but there are usually obstacles along the way. The biggest one is typically the decision-making that developers must undertake in cooperation with business managers concerning security role assignments. This involves:

- Deciding which roles are actually needed for the application.
- Deciding which objects and methods can be accessed by each role.

Unfortunately, these decisions can engender endless debate and slow down development and deployment. The method I'll discuss below can help an organization avoid these obstacles by removing guesswork and systematizing the decision-making process. By cutting back on "freeform" decision areas, it can help developers not only improve the effectiveness of the process, but also achieve consistent quality throughout the application.

The method is based on the analysis of use cases and a UML System Model based on those use cases. To be fully effective, it requires that each external service (third-party product to be integrated into the application) be component-based in the same technology (COM+ or J2EE) as the application itself.

Overview: A Method for Designing Role-Based Security

The first step in this method for designing role-based security is to identify roles. We do this by reviewing the UML System Model specified by use cases. Every UML model has a Use Case View that shows the Use Case Model and defines the actors. Each actor, in turn, defines a role in the Role-Based Security Model. Inheritance relationships between actors define a hierarchy of role definitions, which may or may not be implemented in the application server. If the hierarchy is not implemented, then each terminal role will translate to one role in the application server.

The next step is to decide which objects and methods each actor can access. This requires a careful review of the UML System Model's sequence diagrams. A well-designed UML System Model clearly associates each sequence diagram with the use case from which it originated. Each sequence diagram, in turn, clearly identifies the actor as the originator of the interaction. To assign the correct role to an actor, you must ensure that all the objects with which the actor needs to interact directly are accessible by that role.

And finally, you should look at the UML Design Model to determine how the application is separated into layers, typically presentation, business, and data layers. An actor should be able to access the presentation layer but not the business or data layer. Our method introduces a number of extra roles that do not map to any actor identified in the use case analysis. These roles represent the various layers of the application and enforce an extra level of security.

We can represent findings from these analyses using a matrix. The first column lists the system's objects, and the second column lists all the methods for each object. The first
row lists the roles we identify. If a role has access to a method of a particular object, then we place an X in the box where the role column and the method row intersect.

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
<th>RoleA</th>
<th>RoleB</th>
<th>RoleC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object1</td>
<td>Method1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object1</td>
<td>Method2</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Object1</td>
<td>Method3</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Object2</td>
<td>Method1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object2</td>
<td>Method2</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Identify Roles**

As we saw in the overview, this is the easy part of the process. First we review the Use Case View of the UML System Model and then check the Use Case Model for definitions of the actors (see Figures 1 and 2).

![Figure 1: Use Case View of a UML Model](image-url)
By looking at Figure 2, we can see that our system should include four roles:

- Online Investor
- Online Customer
- Anonymous User
- Credit Card Authorization System

The three first roles are internal to the system, but the fourth represents an external system. For role-based security to work well within an application server, the external system should have a boundary component that integrates with the application server. This is the only way the application server can control the complexity of the security set-up within its boundaries.

The "Anonymous actor" associated only with the "Create Account" use case represents all users who have not yet registered with the system but want to create a new account. Reviewing the use case model in Figure 3, we can also observe that the "Create Account" use case is associated only with that actor. This means that other actors cannot create new accounts. In other words, an actor must have an "anonymous" status in order to create a new account.
Figure 3 shows us that the "Online Customer" actor inherits from the "Online Investor." This means that any function accessible by the "Online Investor" will also be accessible by the "Online Customer." Note, however, that in association with the "Logon to the System" use case (the "Online Customer" must logon before using any shopping feature), the converse is not true: the "Online Investor" cannot access customer features. This observation is important for the security model. It means that the role representing the "Online Customer" should systematically be granted access to all objects and methods that "Online Investor" can access.  

Decide Which Objects and Methods Each Actor Can Access

As we saw in the overview, the next step is to decide, via a careful review of the UML System Model's sequence diagrams, which objects and methods to make accessible to each actor. This involves reviewing each sequence diagram that represents the interaction between an actor and the system's classes in the context of a use case. The underlying assumption is that all use cases are modeled with interaction diagrams.

We should consider two levels of sequence diagrams:

- The analysis level that shows interactions between the actors and boundary types of classes.
- The design level that shows interactions between the actors and classes of:
  - The presentation layer (for an interactive user)
  - The business layer (for an external system)

For the purposes of this discussion, we will assume an interactive user. For an external system, a business layer would access another business layer, possibly with a presentation layer at the boundaries of the external system.

It is also good practice to build two levels of role-based security models:
• An analysis level that shows all the classes the actor can access. There is no need to list the methods in detail. The assumption is that the security model at this level is coarse, and that the design model will refine it. If a role accesses an object in the analysis model, it could well be that later, in the design model, the role will access only part of that object or will not access it at all.

• A design level that accurately reflects the object that will be visible during the configuration of the application in the application server. For each role, this model specifies all the details regarding access to objects and methods.

**Analysis Level Security Model**

In our sample system, let's consider the "Create New Account" use case. Figure 4 is the sequence diagram for the UML Analysis Model. The "Default Page" and the "New Account Page" are the two boundary classes with which the "Anonymous User" directly interacts. But for the Analysis Security Model we also consider the other classes involved in the interaction. The security matrix below reflects the level of detail covered by the UML Analysis Model.

<table>
<thead>
<tr>
<th>Object</th>
<th>Anonymous User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Page</td>
<td>X</td>
</tr>
<tr>
<td>New Account Page</td>
<td>X</td>
</tr>
<tr>
<td>Create New Account</td>
<td>X</td>
</tr>
<tr>
<td>User Profile</td>
<td>X</td>
</tr>
<tr>
<td>Account</td>
<td>X</td>
</tr>
</tbody>
</table>
The next step in the process is to review the UML Design Model.

Design Level Security Model

Following the same process as for the Analysis Model, we next examine all the sequence diagrams that were developed to cover use cases. At this level we also list all the methods for the objects, considering only public methods. Figure 5 presents the design model sequence diagram for the "Create New Account" use case. The actor is not represented at this level of detail; it is understood to be the "Anonymous User."
As a first cut, we could decide that all the objects in the sequence diagram can be accessed by the role representing the actor who originates the interaction. At this level of detail we also specify which methods of each object can be accessed. The matrix below shows the methods of all the objects that participate in the collaboration:

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
<th>Anonymous User</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_NewAccount</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>cp_NewAccount</td>
<td>OnCreateAccountClicked</td>
<td>X</td>
</tr>
<tr>
<td>sp_NewAccount</td>
<td>ProcessPage</td>
<td>X</td>
</tr>
<tr>
<td>spDefault</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CreateNewAccount</td>
<td>CreateAccount</td>
<td>X</td>
</tr>
<tr>
<td>UserProfile</td>
<td>Add</td>
<td>X</td>
</tr>
<tr>
<td>UserProfile</td>
<td>VerifyUser</td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td>Add</td>
<td>X</td>
</tr>
<tr>
<td>Account</td>
<td>Update</td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td>GetAccountInfo</td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td>GetSummary</td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td>ListPositionsForSale</td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td>ListPositions</td>
<td></td>
</tr>
<tr>
<td>pAccount</td>
<td>Add</td>
<td>X</td>
</tr>
<tr>
<td>pAccount</td>
<td>Summary</td>
<td></td>
</tr>
<tr>
<td>pAccount</td>
<td>VerifyUser</td>
<td></td>
</tr>
<tr>
<td>pAccount</td>
<td>GetAccountInfo</td>
<td></td>
</tr>
<tr>
<td>pAccount</td>
<td>Update</td>
<td>X</td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>GetConnectionString</td>
<td></td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>RunSPReturnRS</td>
<td></td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>RunSQLReturnRS</td>
<td></td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>RunSPReturnRS_RW</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>RunSQLReturnRS_RW</td>
<td></td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>RunSP</td>
<td></td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>RunSQL</td>
<td></td>
</tr>
<tr>
<td>IpDBHelper</td>
<td>RunSPReturnInteger X</td>
<td></td>
</tr>
</tbody>
</table>

Note that we can already select exactly which methods of the objects to be installed in the application server need to be accessed by the particular role.

**Additional Security for Layered Architectures**

We must also consider another dimension of modern system design. An application is segmented into a number of layers -- typically the presentation, business, and persistence layers. Furthermore, the presentation layer can be split into a controller and a view layer when implementing the Model View Controller (MVC) approach.

In our model, we can analyze the system's structure by examining the UML Design Model (see Figure 6). We can immediately see that the actor must be able to access the presentation layer but not the other layers.

![Figure 6: A UML System Design Model](View full size graphic in new window)
We can also see that there are two major subsystems: Online Broker and Online Shopper, which correspond to two major roles we defined: "Online Investor" and "Online Shopper." In reviewing the Online Broker subsystem, we can clearly identify the following system segments:

- Application
- Business entities and business services
- Persistence

Note that the organization for the application segment echoes the divisions for use cases. We can now add corresponding classes from our previous security matrix to the list of segments:

- Application: f_NewAccount, cp_NewAccount, sp_NewAccount, spDefault, CreateNewAccount
- Business entities and business services: UserProfile, Account
- Persistence: pAccount, pDBHelper

To achieve a higher level of security configuration, we can separate the roles that are allowed to access each of these segments. Clearly, the application segment will be accessible by roles associated with use case actors. In our example, the role/actor is "Anonymous User," and the part of the security matrix that covers the application segment will not change for that role.

For the other segments, we must introduce two new roles that do not correspond to any use case actor. We will assign them distinctive names to avoid confusing them with roles that do correspond with use case actors. The "sysApplication" role will be granted access to the business entities and services segment. Each class in the Application segment will be configured to play that role when it collaborates with the business entities and services segment. We can conceptualize the classes of the application layer as one actor (hence the role sysApplication) that interacts with the system defined by the business entities and services segment. Then, in the same way, we will define "sysBusiness" as a role that can access the persistence segment.

We can now review the security matrix, adding the new roles and shifting the access rights to the correct role, based on the segment to which the class belongs:

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
<th>Anonymous User</th>
<th>sysApplication</th>
<th>sysBusiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_NewAccount</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cp_NewAccount</td>
<td>OnCreate</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>AccountClicked</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp_NewAccount</td>
<td>ProcessPage</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spDefault</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateNewAccount</td>
<td>CreateAccount</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>UserProfile</td>
<td>Add</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>UserProfile</td>
<td>VerifyUser</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In the application server, each component will be installed to run with a specific identity (actual user account). For the "User Profile" class to be accessible by "Create New Account," as documented in the sequence diagram, we need to configure "Create New Account" to run under an identity that will be part of the list of actual users assigned to the "sysBusiness" role. In addition, we should avoid associating users who will be associated with the "Anonymous User" role with the "sysBusiness" role; otherwise, we defeat the purpose of having two roles.

At this point the security matrix has to be implemented as a configuration for an application server. How this will actually be done depends on the application server used for the implementation.

**Automating Specifications with Rational Rose**

This systematic approach for deciding how to configure role-based security when installing a component-based application in an application server facilitates the work of technical architects and introduces a rational dimension to this activity. Because it works from the system design specified in detail within a UML System Model, it would be highly desirable to have a tool that could automatically analyze such a model and configure the application server with the correct roles and access rights for the implemented interfaces.

Actually, Rational Rose comes close to achieving this. Rational Rose exposes all its functions as COM objects and can also be extended with Visual Basic for Applications (VBA). If COM+ is the target application server, then we can access its COM interface
from VBA and thus create a Rational Rose add-in that will directly install the components with the correct security settings in COM+. This brings us one step closer to the objective of a completely automated system specification based on design.

1 The method described in this article is based on work done for an e-commerce sample system for Rubus, named Rubus Stocks. This system, in turn, is based on a sample system named FM Stocks, a Web application commissioned by Microsoft and developed by Vertigo Software, Inc. as part of a Windows Distributed InterNet Architecture (DNA) performance and scalability study. Complete documentation for this original sample can be found at Fitch & Mather Stocks 2000: Introduction and Article List.

2 This sample was designed to exemplify various modeling techniques. The rationale behind the usage may not necessarily reflect a typical e-business scenario, but it is internally consistent.

3 This detailed matrix could be used as the input for an automated configuration program. This would be particularly useful if the application server supports configuration through an API (e.g., COM+ defines a set of COM+ objects to support the configuration features that are also exposed through system tools).

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
Getting Help with Rational ClearCase Performance Problems on Windows NT and Windows 2000

by Carem Bennett
Rational Technical Support

Many factors affect Rational ClearCase performance on Windows. Because ClearCase has a client server architecture, we at Rational Technical Support rely heavily on network resources and remote procedure calls (RPCs) to accomplish much of our work. ClearCase is also dependent upon network architecture, and the location of DNS, WINS, and other resources requires careful planning. In a mixed Windows Unix environment there are additional considerations: authentication and network access (NFS or SMB) implementations are crucial.

This article introduces you to some of the procedures we use when customers call about performance problems with ClearCase on Windows. First, however, we’ll discuss a few things you can try on your own to improve performance.

Before You Make a Call

Before you pick up the phone to call a Rational Technical Support engineer for help in diagnosing a performance problem, you can try these simple suggestions for improving overall performance:

1. Be sure that ATRIA\bin is within the first three variables in your PATH statement. An example looks like this:
   C:\WINNT\system32;
   C:\WINNT;C:\PROGRA~1\Rational\common;C:\Program Files\Rational\ClearCase\bin;C:\Program Files\Rational\Attache\bin;

2. Be sure that your clients have ready access to network resources such as DNS and WINS. If DNS and WINS servers are not on your local subnet, and name resolution calls have to go over a router or switch, performance will be slow.

3. Add the hostnames of your ClearCase servers to the local HOSTS file on your Windows client. This will speed lookups.

4. Adjust your Network Provider order so that ClearCase is first in the
If, after following these suggestions, you are still experiencing performance problems, then it's time to contact Rational Technical Support for help: support@rational.com or (800) 433-5444.

## Reviewing Clearbug Data

When a customer calls Rational Technical Support with a ClearCase performance problem on Windows, usually the first thing we request is the output from a utility called clearbug. This utility gathers a number of logs that we ask customers to examine for specific information. We then guide them in taking corrective actions if necessary. The table below shows what we look for and recommend for each log.

<table>
<thead>
<tr>
<th>Log</th>
<th>What to Look for and Corrective Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearCase log files</td>
<td>Look for RPC timeouts and socket disconnects in the view and VOB logs.</td>
</tr>
<tr>
<td>Microsoft Diagnostics Report</td>
<td>● Check list of services for online defraggers and anti-virus software (if either is present, then we ask you to disable and run a trial to see if performance improves).   &lt;br&gt;   ● Check Environment Variables for the existence of an MVFS drive in the PATH statement (well-known cause of system hangs that should be removed). Also check Environment Variables for TEMP and TMP configured as System Variables.</td>
</tr>
<tr>
<td>ClearCase version information</td>
<td>Check patch levels: all recent or relevant patches should be installed (download from our customer-only Web site at <a href="http://clearcase.rational.com">http://clearcase.rational.com</a>. Ask your Rational Technical Support Engineer for a temporary login if you don't have a permanent one.</td>
</tr>
<tr>
<td>ipconfig output</td>
<td>Check for use of DHCP, as well as the location of DNS and/or WINS servers. Any machine that is configured as a ClearCase server (hosting local views and/or VOBs) should have a static IP address to reduce ARP calls for name caching. If you are using DHCP, try using a static IP address.</td>
</tr>
</tbody>
</table>
Running a Network Trace

If reviewing the clearbug data and taking the suggested actions does not yield any improvement, then we may request a network trace. Network Monitor is our preferred tool for generating network traces on Windows NT and Windows 2000. There are two versions of Network Monitor for Windows NT 4.0: the standard version that ships with Windows NT, and the more robust version that ships with SMS. If it is available, the SMS version is best for generating your trace. If you are working in a mixed environment, then support may also request that you run a network trace from the Unix server while the Network Monitor trace is being run from the Windows side.

Your Rational Technical Support engineer will supply detailed, step-by-step instructions for running a Network Monitor trace, but in general you should: 1) start the trace in promiscuous mode (this means that the trace will capture all traffic on the network, with no filters applied); 2) reproduce the problem (if a checkout takes ten minutes, then perform a checkout); 3) stop the trace and send it to Rational Technical Support via e-mail or our ftp site. To access the Rational ftp site, ftp to ftp://ftp.rational.com. Then use the login anonymous, and enter your full email address as a password. Once you are logged in to the site, do a cd command to switch to the /apg/incoming directory, and create a directory with your case number. Then put the data into the directory and alert your Rational Technical Support engineer that you have put the data up on our ftp site.

When we examine the network trace, we look for several things. If your site uses an SMB interoperation solution, we check for the success of local or proxy authentication. In an all-Windows environment we check for calls to the local domain controller, as well as name resolution calls via WINS or DNS. We also check for excess network load in terms of broadcast traffic (browser service election broadcasts, for example).

To determine whether the problem is related to Rational ClearCase or to the network, your Rational Technical Support engineer may request that you run some small tests. Questions that he or she may ask are:

- How long does it take to copy a file from the VOB server to your local machine, outside of ClearCase?
- If you are running in a mixed environment, how long does it take to create a file on the Unix export from the Windows client, outside of ClearCase? What are the permissions on that file?
- What are the ping times to your closest domain controller?

- What is your network architecture? Are all your ClearCase servers local to your site, or are you using a WAN to access one of your servers? If you are working over a WAN, then is Rational ClearCase MultiSite a valid option for your company? How about the ClearCase Web client?

- Does performance improve with the use of snapshot views?

It's important to bear in mind that the procedures and suggestions we've outlined above will not apply to every site. If you've tried the tips in the "Before You Make a Call" section above and are still experiencing performance problems, it's best to contact the experts at Rational Technical Support and pursue further diagnostic procedures under our guidance. We're here to work with you until we find the right solution for your system.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
TECH TIP - Installing Unix ClearCase Family Patches

by Kristy Gleason
ClearCase Unix Technical Support

Please note: v4.1 and sun5 are used only as examples in these instructions; please replace v4.1 and sun5 with whatever version of Rational ClearCase and/or its family of products and whatever platform you use in your individual release area.

Use this procedure any time you apply a patch to your ClearCase release area. In the readme files associated with downloadable patches, the explanation of the patches directory location can be confusing; this tip provides a clear explanation of what to do.

After downloading the necessary .tar.gz files to your Unix host, you will want to step through the following instructions. Be sure to apply multiple patches to your release area from lowest to highest. Patches listed on the patch matrix page for each Unix platform and each version of Rational ClearCase do not supersede one another; you must apply all patches listed as mandatory on the patch matrix page.

1. Make a patches directory for your compressed patches (if one doesn't already exist) in your ClearCase release area (in most cases, /usr/ccase_rls/v4.1/sun5).
2. Copy the *.tar.gz file(s) to the ClearCase release area (/usr/ccase_rls/v4.1/sun5/patches).
3. Use gunzip/gzip to uncompress the individual patch(es) (gunzip -d clearcase_p4.1-6, for example); Use tar to extract the patch(es) into its own directory structure beneath the current directory (tar -xvf clearcase_p4.1-6, for example).
4. Cd into the individual patch directory; cd into the install subdirectory.
5. Execute the apply_patch script (./apply_patch).
1. CREATE A PATCHES SUBDIRECTORY

Create a 'patches' subdirectory in the release area to be patched. This will probably require super-user privileges.

```bash
% su
Password:
# mkdir /usr/ccase_rls/v4.1/sun5/patches
```

2. COPY THE GZ FILE INTO THIS DIRECTORY

```bash
# cp -p ... /usr/ccase_rls/v4.1/sun5/patches/clearcase_p4.1-6.sun5.tar.gz
```

3. EXTRACT THE PATCH

Change directory to the 'patches' directory.

```bash
# cd /usr/ccase_rls/v4.0/sun5/patches
```

If the patch is in the form of a compressed tar archive (*.tar.gz), extract the patch using the gzip and tar commands.

```bash
# /usr/atria/etc/Gzip -d clearcase_p4.1-6.sun5.tar.gz
# tar -xvf clearcase_p4.1-6.sun5.tar
```

If the patch is in the form of a shell archive (*.shar), extract the patch by executing the shell archive with /bin/sh.

```bash
# sh clearcase_p4.1-6.sun5.shar
```

Either extraction method creates a subdirectory for the patch in the 'patches' directory.

4. PATCH THE RELEASE AREA

**WARNING:** If any Rational ClearCase hosts at your site have been installed using either the Standard or Link installation models, patch the release area only when these hosts are not in use, and patch the individual hosts as soon as the release area has been patched. Otherwise, inconsistencies between the software in the release area and the software installed on the individual hosts will make ClearCase fail or operate erratically on these systems.

To patch the release area, you must be logged in as the super-user.

Change directory to the 'install' subdirectory of the directory that was created when you extracted the patch.
5. APPLY THE PATCH USING THE COMMAND

```
# cd /usr/ccase_rls/v4.1/sun5/patches/clearcase_p4.1-6/install
```

The apply_patch utility validates the extracted patch contents, then overwrites files in the release area with newer ones included in the patch.

6. RUN SITE_PREP

Run site_prep to synchronize permissions in the release area, using a series of commands such as these. The value you use for the -setuid option may be different from the one shown here. Check with your ClearCase administrator if you're uncertain.

```
# cd ../../../install
- or -
# cd /usr/ccase_rls/v4.1/sun5/install
# ./site_prep -chmod ; ./site_prep -setuid
```

7. PATCH CLEARCASE HOSTS

Once the release area has been patched, you may patch individual ClearCase hosts using install_release as described in the ClearCase Product Family Installation Notes. Only the changed files will be installed.

```
# cd /usr/ccase_rls/v4.1/sun5/install
# ./install_release
```

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
Understanding the Ugly Duckling: Reliability Testing and Rational TestFactory

by Brian Bryson
Technology Evangelist
Rational Software

It's widely acknowledged by industry professionals that reliability testing is an effective strategy for infusing quality into the software development process. So why do so few development teams actually do it? In last month's issue, Brian Bryson examined some of the obstacles impeding comprehensive unit testing's widespread acceptance and introduced Rational Quality Architect, a new technology that overcomes those obstacles. In this article, he looks at how testing and development teams view reliability testing and explains how Rational TestFactory can transform their perspective.

In 1844 Hans Christian Andersen penned "The Ugly Duckling." As you may recall, it's the poignant story of a swan born into a duck's family. Being unlike any of his siblings, the lone swan is chastised and runs away. Just as he's about to give up all hope, the swan happens upon a group of other swans and spies his reflection in the water. It is only then that the swan understands and appreciates his own beauty.

How could Andersen, a 19th century Dane, have the foresight to write such a perfect parable about software testing in the year 2001? It's true: he succinctly described the current plight of reliability testing in the software development environment.

Functional, Performance, and Reliability Testing
Typically, testing activities are classified as functional-, performance-, or reliability-related. These are fairly broad categories, so let's start with some definitions.

You're probably already familiar with functional testing, which IEEE defines as "...testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions." The Rational Unified Process defines it a bit differently, as "...a test focused on verifying the target-of-test functions as intended, providing the required service(s), method(s), or use case(s). This test is implemented and executed against different targets-of-tests, including units, integrated units, application(s), and systems." What both definitions are basically saying is that functional testing verifies that an application works as intended from a user's perspective. Simple enough.

Most likely you're also familiar with performance testing, which IEEE says is "...testing conducted to evaluate the compliance of a system or component with specified performance requirements." The Rational Unified Process calls it "...a class of tests implemented and executed to characterize and evaluate the performance-related characteristics of the target-of-test, such as the timing profiles, execution flow, response times, and operational reliability and limits." Again, to simplify, both definitions describe the task of verifying that the system meets response time requirements defined by the user.

And then we come to reliability testing, which you may know about only in theory. The IEEE defines it as "...the ability of a system or component to perform its required functions under stated conditions for a specified period of time." Here again, the Rational Unified Process's definition is similar, although it breaks down reliability into Integrity, Structure, and Stress tests components, defining each along the way. Collectively, these components parallel the IEEE's definition.

Given these relatively clear definitions of reliability testing, why hasn't the practice itself gained the same degree of acceptance and universal understanding as functional testing and performance testing? Why is this poor creature the ugly duckling of the testing world? Maybe it's time to re-examine this misunderstood fowl.

**Understanding Reliability Testing**

Before we talk about any technology, let's look at what we're trying to achieve with reliability testing. As we can see from the definitions above, reliability testing helps ensure that an application is stable -- that it won't crash now because of a general handling exception or over time due to a memory leak. Put simply, we hope it will validate structure.

Because of this goal, many view reliability testing as a by-product of functional testing. This makes sense from one perspective. If a functional test validates that a given function point within the application works, then
didn't you implicitly validate that the application won't crash along the way? Absolutely.

Thorough functional testing of an entire system, however, is a rare achievement.

Unless a testing team is blessed with an infinite amount of time, it confines functional testing to the most critical areas of an application. Instead of thoroughly testing every area of the system, testers pick a small subset of the system's functionality and test many variations and combinations of those functions. Functional testing usually means "going deep" into key areas of your application. The problem is, it ensures good coverage of these areas to the detriment of less significant areas.

That's where reliability testing comes in. Reliability testing is a way to ensure a minimum quality standard across a very broad spectrum of an application's functionality. It represents an attempt to "go wide" and test everything. The key to success is to keep it simple. These tests should not be complicated validation systems for complex business functionality; instead, they should simply validate that the application doesn't -- and won't -- crash.

**Finding the Right Technology for Reliability Testing**

To reiterate, the goals of reliability testing are:

- Cover the whole application.
- Verify that the application doesn't -- and won't -- crash.

Again, note that validation of business functionality is not a goal of reliability testing; it's a goal of functional testing.

This simple approach works out very well from a technology implementation perspective. The two goals (and one non-goal) described above map very well to available technologies. Rational Software produces Rational Robot, a tool capable of playing back actions against an application. Rational also produces PureCoverage, a tool that can measure source code coverage for a given sequence of actions taken against an application. The missing piece, then, is a spidering technology that "crawls" across an application, mapping it out, to ensure maximum coverage. Rational has that, too. In fact, when it comes to applications, we're the only spider in town.

Spidering technologies for Web sites have long existed, and hundreds of them are currently available. They all work on the same basic concept: The spider traverses the site, parses all the Web pages, and discovers all the links. It then tries all the links to detect broken ones, as well as slow-loading pages. You may be familiar with Rational's offering, Rational SiteCheck. Curiously, however, there's only one technology on the market that performs the same types of operations against an application. That technology is Rational TestFactory.
TestFactory: A Spidering Technology for Applications

This article is not meant to be a marketing piece for Rational TestFactory. You can head to rational.com for that. I simply want to talk about the ways this tool can make reliability testing a viable option for testing teams.

Basically, Rational TestFactory provides an intelligent spidering technology for applications. Without user input, it will traverse Visual Basic and Microsoft C++ Applications or Java applets to discover all windows and controls. Along the way, it will insert data into fields, click on buttons, and select menu options until it has touched every window, control, and line of source code in the application.

TestFactory's goal is not to test business logic. (Remember that simplifying assumption?) It cannot determine whether or not a system is correctly responding to a given set of inputs and pre-conditions. Instead, the goal is a simple one: to ensure that a system or component can perform its required functions under stated conditions for a specified period of time.

Taking the Sting Out of Two Tough Tasks

Rational TestFactory is an automated reliability testing tool designed to uncover the ugliest of defects: a crash. Now, when a tester encounters such a defect, he or she should always seek to answer two key questions:

- How can I reproduce this problem?
- What change(s) in the underlying application could be causing the problem?

Rational TestFactory addresses both of these issues.

First, any tester will acknowledge that reproducing a crash is one of his or her most time consuming and frustrating tasks. TestFactory can make the task quite painless. Should an application crash during testing, TestFactory not only identifies the line of source code that caused the crash, but also creates a test script, which can be run in Rational Robot to reproduce the crash. Any tester who has reported a defect only to have it countered with, "Doesn't happen on my machine" will immediately appreciate the value of this capability.

Second, testers usually find the other question so difficult to answer that they simply throw it back at developers. The technical debugging tools they'd need to pinpoint what changed are not typically available to testers. Unless they have Rational TestFactory's source code coverage technology, that is.

TestFactory has visibility to an application's source code. For every action it takes on an application -- be it clicking on a button or entering data into a field -- TestFactory is aware of which lines of source code are executed to process its actions. Through an add-in to Microsoft Visual Studio,
TestFactory is also aware of what source code files are affected when a developer makes code modifications. TestFactory can then map these source code changes to scripts it has run to discover which scripts are affected by a given change in source code. Then, it can run the scripts to discover if a change in source code has caused an application failure. This functionality can be extended to functional testing as well. It can help determine if a given change not only caused a crash but also caused the software to malfunction from a business rules perspective.

The Spider and the Swan

To fully appreciate reliability testing, you must understand its role within a testing project. Its goal is not the same as the goals for functional or performance testing. The goal for reliability testing is to perform basic validation for a broad portion of your application. Sometimes there's a lot of time for reliability testing, sometimes there's little -- but it should always be done.

Using automation tools such as Rational TestFactory provides a very simple way to do more in less time. Effectively, TestFactory provides free testing. Because it runs without human input, you can run it overnight with little set-up. If it finds nothing, then you've come out ahead by confirming the solid structure of your application. If it uncovers crashes, then you're way ahead. You've found defects early in the game, when they're still relatively easy and inexpensive to correct.

Rational TestFactory, our friendly spider, can help you appreciate the true beauty of that poor, lonely swan: reliability testing. Understanding them both for what they are, and making them an integral part of your comprehensive testing strategy, can greatly improve the effectiveness of your quality assurance efforts.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
Straight from the Source: The Need for Distributed Authoring in Web Content Management

by Karen Dasey
Senior Writer
Rational Software

This article is the first in a series about Web content management (WCM) business needs. It explores the advantages that businesses can realize with Rational Suite ContentStudio, a comprehensive WCM tool that empowers diverse, non-technical content owners to interact directly with dynamic Web applications.

If your company’s external Web site or intranet site is like most, the number of pages is multiplying at a rapid rate. And chances are, the number of content owners contributing to these pages is increasing just as quickly, making it more and more difficult for your overworked Web team to keep up. A December 14, 2000 feature on PlanetIT cites a Network Computing poll which indicated that almost 24% of e-businesses do not have any strategy for Web content management (WCM). The same study found that nearly two-thirds of the e-businesses that do have a WCM strategy use a home grown solution.

How well you manage your Web page content can dramatically affect many areas of your business, including company-wide productivity and your competitive position in the market, along with customer and employee satisfaction. Although a pieced-together, home-grown system may offer a cost-effective way to manage and post content for a small, static Web site, it becomes less and less effective as the site expands in size and complexity. Typically, home-grown systems can neither scale to manage the greater number and wider variety of artifacts comprising the site nor continue to meet demands for fast turnaround as an expanding pool of diverse content contributors submits changes.

Once a Web site becomes more than just a vehicle for creating an online
presence, only an end-to-end solution that supports the flow of content across the e-development lifecycle -- from creation, to editing and approval, to staging, deployment and expiration -- can be truly effective for managing content.

This past fall, Rational announced a major new addition to its lifecycle management product line designed to address these problems. It's called "Rational Suite ContentStudio," and it offers an end-to-end solution for Web code and content management. It integrates easily with back-end and end-user desktop applications, and it includes key functions such as automated deployment and workflow management, a feature Forrester Research\(^1\) cites as the top reason e-businesses invest in WCM systems.

A comprehensive WCM tool also features **distributed authoring**, a function we'll discuss in this article. Distributed authoring includes workflow management capabilities and enables authorized content contributors to make controlled changes to a Web site directly -- without requiring Web team interaction or impacting site design. Future articles in *The Rational Edge* will address workflow management in more detail, plus automated deployment and other unique advantages offered by Rational Suite ContentStudio.

**The Chaos Is in the Content**

At the core of any corporate Web site or intranet sits the Web team, which can consist of groups of people in roles such as Web engineering, design and editorial, and production. Typical Web team activities include:

- Site design and navigation
- Configuring back-end hardware
- Developing and changing e-business code
- Synchronizing Web code and content deployment
- Establishing and enforcing release procedures
- Load balancing
- Testing
- Setting and maintaining graphic and editorial standards

Without an efficient WCM system, the Web team also inherits responsibility for managing the influx of content from contributors spread across the company and beyond. In most cases, they deliver the content in an array of disparate formats, such as Microsoft Word, e-mail, PDF and graphic files, video, and databases. And rarely is this content right the first time. So Web team members find themselves in the middle of roundabout approval cycles between multiple content owners and reviewers, spending unproductive time chasing down content owners for approvals. Often, these highly skilled, highly paid Web professionals wind up in the role of glorified typists. Add to these challenges the pressure of converting content to Web-compatible formats, such as HTML and XML, and keeping up with aggressive release cycles that require the daily or even hourly
deployment of new Web applications.

Such scenarios often result in Web team bottlenecks that prevent the timely deployment of business-critical Web applications and foster errors that can destabilize the Web site. To prevent these bottlenecks, you need to remove the Web team from the content input, editing, and review cycle. The most effective way to do this is to leverage the distributed authoring capabilities in Rational Suite ContentStudio.

**Control the Chaos with Distributed Authoring**

Distributed authoring relieves Web teams of tedious content input, editing, and conversion tasks by enabling authorized content owners to make changes directly to a Web site, using pre-formatted templates. A template is a Web application consisting of two separate components:

1. A presentation component that provides a pre-defined blueprint of the Web page layout and its navigational and transactional functions; and
2. A non-transactional content component that consists of any combination of text, graphics, and audio elements.

By separating form from content, templates enable everyone responsible for your Web site to focus on his or her respective areas of expertise. Web team members can focus on improving page design and navigation, maintaining editorial and graphic standards, and administration. And content owners can create and make changes to information, and see how that information will be organized on a Web page, without getting the Web team in the middle.

The result: A major increase in both productivity and accuracy. When you combine a distributed authoring system with a workflow management process that automates the flow of content through the stages of creation, review, approval, staging, deployment, and expiration, there are fewer opportunities for errors to creep in. (Watch for an in-depth exploration of workflow management in the March issue of *The Rational Edge.*)

**Two Types of Templates**

The most sophisticated Web content management systems provide two types of templates:

1. Content management templates
2. Content delivery templates

**Content Management Templates**

A content management template provides an internal input form for creating, editing, and managing information requiring deployment to any number of Web pages on your site. Let's say, for example, that you manage a Web site for a retail business that provides online sales of
classical music CDs. When your company wants to launch a new CD series, you need to coordinate the simultaneous publishing of product information across the site, on pages that meet established creative and legal standards. To support online sales of the CDs, you will also need to manage changes to the code that enables sales transactions in tandem with the associated content updates. (In this example, we'll deal only with how templating helps you manage content changes.)

Without a templating function, a product manager would need to find each and every Web page on which the new series is to be promoted. The product manager would then have to either communicate these page-by-page changes to the Web team via email, or print out and mark up all pages requiring edits before hand-delivering them to the Web team. Once the requested changes are implemented, the Web team would then email for the marketing manager's review a staged URL address for each updated page. The product manager would then have to email back additional edits for approval.

It's easy to see how a manual, circuitous process such as this would be prone to error and riddled with inefficiencies.

A content management template greatly simplifies the editing process. The Web team can create separate input forms for displaying different types of information such as price lists, order form introduction copy, and online catalog product descriptions. To make the changes, content owners simply access the appropriate form with a Web browser and enter the new information. These changes may then be deployed from the form to all required Web pages. If the Web team also uses content delivery templates, then the information can be displayed using the appropriate design format for each of these pages.

**Content Delivery Templates**

Content delivery templates provide presentation blueprints that define format and functionality for the front-end Web pages your customers or internal end-users see. A typical e-business might design different delivery templates for the home page, corporate overview page, who's who pages, and product pages, for example. When marketing managers needs to edit content on a product page to promote a new version release, they can access the appropriate delivery template via a Web browser to input all text edits and graphic updates directly. Because the format of the page is kept separate from the content, the product manager cannot alter the order or appearance of any standard page elements, such as corporate logos and navigation links. Better yet, because the product manager is working in a template that mirrors the actual look-and-feel of the Web page, it becomes easier to organize the presentation of content within the context of the page design, without wasting time waiting for a staged layout.

**The Last Word**

With its state-of-the-art distributed authoring capabilities, Rational Suite ContentStudio offers many advantages. For your Web team, it simplifies
the process of standardizing a Web site's look and feel. Rather than using a page-by-page approach, designers can categorize the site into a manageable set of page formats, and then design templates to enforce the execution of these formats. And by freeing your Web team from time-consuming content input, editing, and review tasks, distributed authoring enables them to specialize in their areas of expertise in order to improve the quality, consistency, and navigability of your Web site. Templates also make it easier for your Web team to reuse page elements, saving them valuable time when the need arises to update the look and feel of your Web site.

For content managers, distributed authoring helps them ensure the accuracy of information posted on the site, by minimizing errors resulting from faulty, manual review processes. And because Rational Suite ContentStudio is integrated with content managers' preferred desktop tools, it enables them to work productively with the text and graphics applications they're most comfortable using.

To everyone who contributes to your Web site, distributed authoring provides a huge productivity boost, enabling your organization to respond more effectively to the fast-paced demands of today's e-economy. It's an essential link in the chain of the end-to-end WCM capabilities Rational Suite ContentStudio gives you to build and maintain a high-performance Web site that can satisfy the demands of your customers quickly and accurately -- both today, and as your business evolves.

---

1 Content Management, Forrester Research, Inc., December 2000

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!
Managing Teams, Part II

by Joe Marasco
Senior Vice President
Rational Software

In the previous installment of this article, I defined four principles distilled through years of managing and observing teams engaged in product- and service-related efforts.

1. Focus on building a strong team that can solve hard problems and add genuine value for the customer.

2. Leaders inspire; managers enable. To be both a good leader and a good manager, you need to communicate the vision and understand the details.

3. Anticipate obstacles, and eliminate them while they're small.

4. Take the time to listen to others carefully, but don't worry TOO much about what other people think.

In this final installment, I'll add six additional principles that can help you succeed as a team manager.

5. Focus on facts.

For many reasons, we frequently violate this guideline, almost always with disastrous consequences. Deal with reality. Always relate to what is, not to what you would have liked or what could have been or what might be in the future. Stay in the present, and deal with the facts.

Separate facts from opinions. Also separate facts from their consequences.
or implications; often people report these concurrently or confuse them.

When engaged in discussions, evaluations, critiques, and other issues that involve performance, stay focused on the facts as opposed to the personalities of the people involved. Evaluate data based on its factual content, not on the source. Gather facts first and reserve judgment until later.

I have found that writing things down helps me focus on facts. Sometimes this involves making lists, writing things in a standard format, or just creating notes for future reference. When I do this, it becomes very clear when I can legitimately use verbs like "is" and "are" rather than verbs like "appears to be" and "seems to."

6. **Provide stability by being an attenuator, not an amplifier.**

This is an important attribute.

Most of our information channels are "noisy." Every organization has a rumor mill that is constantly churning out misinformation. As managers and leaders, we need to avoid amplifying the noise so that we don't mask the signal. Rarely is a new situation as good or as bad as it looks at first. Take in the data, absorb it, and then decide on appropriate action. A measured response is almost always best. As crises spike within your organization, remember that "this too shall pass," and be the one who sets an example for the rest of the team. To lead effectively, you must keep your head while those around you are losing theirs. Your job is to dampen the spikes and surges and restore regularity to the daily flow of energy.

Occasionally, you may be forced into acting quickly, or an "unmeasured" response may even be desirable. For example, if someone lures away a key employee with a "Godfather offer," then you need to react quickly if you hope to turn the situation around. Or if your team makes a major breakthrough, you'll want to react with unbridled enthusiasm, no holds barred. You'll recognize these exceptional situations when they crop up.

7. **Never attribute to malice what can be explained by incompetence.**

If someone says or does something that may affect you in an adverse way, be very careful of jumping to the wrong conclusion; paranoia can get you into a lot of trouble. If the act seems wrongheaded, first assume that it was a mistake. Try to imagine the erroneous set of assumptions that might have led the person to this action. Put yourself in his or her shoes.

Only after eliminating all possible "error scenarios" should you even entertain the notion that impure motives were at work. Why? Consider the consequences. If you assume malice and you are wrong, then you will almost certainly make an enemy -- and enemies have a nasty habit of accumulating. It's silly to make them unnecessarily.
If you incorrectly assume incompetence, on the other hand, then, yes, you may get burned. But you will be burned only once. When you give the perpetrator a chance to reconsider his “mistake,” then he’s sure to expose his true colors. In the long run, you’ll have earned the trust of other team members by treating your enemy with respect.

I also believe that incompetence is far more widespread than evil. This may be a naïve view, but I believe statistics are on my side when I take this approach.

8. Cultivate a sense of humor as a counterweight to intensity: take the job seriously and yourself lightly.

I have sometimes been called an intense person. This quality is both a blessing and a curse.

Intensity is the flame to brilliancy’s spark. It allows you to focus, and it can help transmit a sense of purpose to the rest of the team. The refusal to give up, even in the face of adversity, is important.

But there is a dark side to intensity. It violates the Greek ideal of "everything in moderation": There's no such thing as "moderately intense." If you're a person who doesn't let go easily, then you need to be careful; don't let large, ongoing doses of your intensity poison your team.

Having a sense of humor helps. Even in times of crisis, you may need to step back enough to recognize the absurdity of it all. Laugh. Make fun of yourself. Recognize your mistakes and be proud of them, even though it hurts. My theory is that you have already paid for the mistake, and getting a laugh out of it at least brings you a little return on your investment.

I'm not talking about gallows humor here, which is scarcely better than no humor at all. I mean a real, robust appreciation for the follies that inevitably go along with working in an organization, trying to create something out of nothing, and being human.

Teams will forgive a lot of transgressions on the part of their managers, but incompetence, sloth, and humorlessness are not among them.

9. Have a life outside of work, and read twenty-five books a year.

Go back and read #8 again.

You cannot lead effectively -- or even survive -- if you don't have something else to think about besides your job. For me, it has been family, physics, golf, and a few other miscellaneous interests. Go out to the movies, see plays, play poker, dance, howl at the moon. Whatever works for you. But remember, alcohol is a depressant.
Hot baths and long walks have also helped me. I've found that you have to do some kind of exercise to stay in shape, even if the only muscle you use during the day is your brain. That's why world champion chess players do rigorous physical training.

If the stress gets really bad, then talk to people outside your project and your company. They can help you gain perspective. If one of them is your boss, be sure to make him your ally.

Regular reading is another activity that's crucial for effectiveness and survival. As we get older, we tend to recycle what has worked in the past instead of learning and trying new approaches. We gain more and more knowledge through experience, and less and less through formal channels. In addition to periodicals, you should set a goal of twenty-five books a year, which comes out to a book every two weeks. They don't all have to be new, and they don't all have to be technical. If travel is a part of your job, reading is an excellent way to use the time you might otherwise waste waiting around in airports or on the plane.

10. Trust your instincts: if it doesn't feel right, then it probably isn't.

It's easy to get overly analytical. Sometimes we run the numbers until we're blue in the face without bothering to examine the underlying assumptions that went into collecting those numbers. Then, we come to a conclusion that doesn't feel right.

What's maddening about these situations is that we're unable to articulate the reasons for our discomfort. Nevertheless, fearing that this perverse situation will paralyze us, we forge ahead with the analysis and then take action, even though it doesn't feel right.

In the vast majority of these cases, I have regretted the decision. Here's my advice: If it really doesn't feel right, then trust your instincts. You didn't accumulate your "gut wisdom" overnight; you are feeling the sum total of all your past experiences when this happens. At the very least, force yourself to try to understand what is causing the discomfort and then address it.

In my case, most of these bad decisions have revolved around hiring. Never hire someone with whom you don't feel comfortable, which is not to say that you should never take a risk. If the risk level is high enough to make you uncomfortable, however, go with your gut, and don't make the offer.

I've always believed that it's better to make a bad decision than to make no decision at all. When your important decisions and your tummy disagree, however, be careful!

Parting Thoughts
If the ten ideas I've presented in this two-part article seem like a bit of a grab bag, that's because they are. Management and leadership are still arts, not sciences; disciplines that attach the word "science" to themselves -- such as computer science, management science, and social science -- are suspect.¹ We learn what works empirically: by trying things, observing success and failure over a large number of attempts, and then trying to discern patterns. These ten ideas have worked for me over an extended period of time, and all I can do is pass them on for your consideration and use. Your results may vary, because it is all in the application. Best of luck!

¹This observation was originally made by Wayne Meretsky, a former developer at Rational.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!