Continuously ensuring quality: A case study

Level: Introductory

Laura Rose, Quality Engineering Manager, IBM

15 Dec 2004

from The Rational Edge: To produce a high-quality product, you must have a process that involves all project participants in ensuring quality throughout the software development lifecycle. This case study describes some of the tools and techniques IBM Rational uses to develop its own commercial software products.

No matter what their background and experience, software development professionals recognize that you need knowledge about a few basics to have a successful project: the product, the staff, the customer, and the schedule. Of course, knowing these things is just the start. Where we stumble is usually in the execution. In this case study, I share with you some of the approaches and tools we use in IBM Rational development projects to improve our execution, through a description of our development process for a new product.

Know the product

This may seem like an obvious point. But at IBM Rational, we have come to understand that, to really know a product, you must continually ensure quality throughout its lifecycle. This is true for both packaged products you buy from software vendors and business applications you develop internally. Some of the better-known practices for continually ensuring quality include:

- Developing high-quality requirements and conducting requirement reviews
- Testing code early and often
- Tracing features back to requirements and business models
- Analyzing root causes for defects and preventing further defects

These practices are embodied in IBM® Rational Unified Process,® or RUP,® which guides our development activities. As Figure 1 shows, RUP divides project work into four phases: Inception, Elaboration, Construction, and Transition. I will describe the work we did in each of these phases to ensure quality as we developed our Rational product.

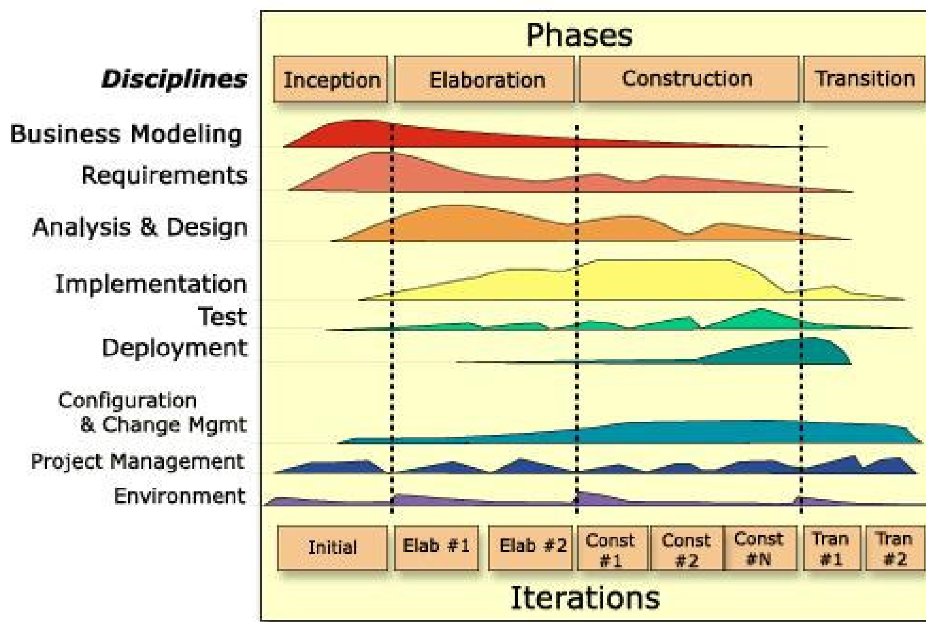


Figure 1: RUP phases, iterations, and activities

Developing and reviewing product requirements

At the beginning of the project, the business analyst and product managers supplied a long wish list of features and a timeline for product delivery that fell within the competitive market window. Missing this window would mean not only losing sales for the year but also allowing competitors to establish a stronghold in our client space.

Then, it was up to the team's product manager, developers, testers, and technical writers to whittle down the number of requests (which consisted mainly of one-line descriptions), based on their understanding of the business problem and what functionality might be feasible within the defined timeline. Once the list was a manageable size, we refined and amplified the descriptions into clear requirement statements and assigned a high, medium, or low rating to each one, based on estimates of how much effort each would require and its business priority. At this stage we included only a few managers, because we were working with few details.

After prioritizing the request list, we invited more team members to join us, including a technical team to assess feasibility and risk for each request. A design team defined prototypes and a shared architecture for the common components, using IBM Rational Rose® and UML (Unified Modeling Language) to visualize the components' interaction and their impact on use case flows. As we came to understand more about the potential feature set, we either removed or refined the riskier items.

In the Elaboration phase, we identified system imperatives and requirements. The product we were developing is part of a larger offering package whose products must comply with system imperatives. Although these requirements did not always add value to our product or our specific client base, we had to invest the time and resources required to implement, test, and document compliance features. This meant cutting our feature list, which required another round of evaluations. At the end of Elaboration, we entered the remaining requirements into IBM® Rational RequisitePro®, our requirements management tool.

We planned three iterations for our product during the Construction phase; the team defined a feature subset for each iteration. Using Rational RequisitePro, we continually reviewed and updated this initial feature delivery schedule, modifying and eliminating requirements throughout each Construction cycle.

At the end of Elaboration, we also completed the requirement documents, describing the features as clearly as possible. Based on past experience, we knew that most feature and system defects we discovered in testing resulted from ambiguous requirement definitions. If we could find defects at this early stage of product design, we knew we would save huge amounts of time and resources.

To reduce the level of ambiguity in requirement specifications and documents, we used a simple checklist of "good specification attributes" and "problem words to watch out for" (see Appendix A). Using these guidelines to conduct requirements reviews resulted in better design documents.

Involving developers in feature- and system-level testing

A strategy we used to pursue the "test code early and often" best practice was to involve developers in both feature verification and product verification (system-level) testing. We had thirty developers and only four testers assigned to this project, so this made sense from a workflow standpoint: the number of features that thirty developers could create would easily flood the available test support.

Our developer/tester partnership yielded good results. We had one automated test framework that ran both unit- and feature-level test cases (it ran automatically whenever a build was generated.) By using this framework, developers learned more about how to "break" software, and testers gained detailed knowledge of how the product worked. With an understanding of how the testing is conducted and the need to verify certain data points, developers could design more testable code. Developers and testers also worked together on using IBM® Rational PureCoverage® to produce a code coverage metric. This enabled us to increase test coverage as we progressed through the different Construction iterations.

Developers and testers shared other tools as well: Rational TestManager® to manage test assets and log test results and Rational Application Analyzer® to analyze the application and record test results for review against iteration exit criteria. Developers and testers shared responsibility for reporting test results and meeting these criteria.

Know the staff

We all know that poorly defined requirements and ineffective communication concerning changes cause headaches for developers, who must struggle to meet delivery deadlines and quality standards. However, a software development project involves many people outside the development realm: business analysts, project managers, configuration managers, testers, and technical writers each have their own different -- but interconnected -- quality concerns and pain points (see Appendix B). For example:

- Business analysts struggle with writing good requirements specifications, effectively communicating requirements and their changes, and correctly interpreting customer needs.
- When testers do not participate in requirement validation, they struggle with assessing application quality and meeting delivery deadlines.

Poor software quality is a business issue that affects every part of an organization. An application lifecycle process that acknowledges and provides guidance for dealing with these interdependences can greatly improve communication among the different stakeholders and improve the organization's quality capability.

In addition to the methods and tools we described above, we included test plan reviews in our product development process to reduce pain along the chain of key players. Our goal was to get everyone to agree on what we planned to cover in our testing, but it was difficult to get the key players involved. Everyone was already overloaded, the reviews were not explicitly scheduled in the program schedule, and it was hard to define what made a "good review" versus "a quick read to say it's done." In addition, most of the key players did not have to depend upon the test plan to complete their work.

To offset the participation problem, the project team reduced the number of required reviewers. We noticed that if we distributed the plan to a large group for review, we received very few comments. Probably everyone thought that "someone else" would do a review, so they did not have to. But when we asked only one representative from each major department, these people felt accountable for representing their department's interest and goals and took the responsibility more seriously.

Once we had whittled down the required reviewer list, we could afford to give each respondent one-on-one attention. Acknowledging that they had busy schedules, we asked them to commit to regular meetings far in advance. Often we were able to schedule weekly feature meetings to go over specific test plan areas that were of interest to multiple parties. During these meetings, we made sure to show reviewers how much we valued their opinions and how the specific test area related to their respective jobs. For instance, we gave technical support and field experts an opportunity to learn enough about the product to support after deployment. For the product manager and business analyst, we emphasized the customer scenario flows and customer application environments, information that they later used in beta programs. For developers, we outlined shared test environments and platforms they could leverage for unit and feature testing and for Construction cycles. This strategy produced invaluable, insightful comments from our reviewers.

Surprisingly, the biggest complainers were the testers who conducted the reviews. The process required a fair amount of time;

it would have been easier, they said, to just release the test plan to a large group of people and sit back to wait for comments. If no comments came back, we could easily say, "It's not our fault. We gave everyone the opportunity to review our plans, and they didn't do it." However, such a strategy runs counter to the underlying reasons for conducting a quality test plan review in the first place!

Know the customer

Experts have defined quality in many different ways, but the basic question they address is, "Does the product satisfy the target customer?" The best way to ensure customer satisfaction and control costs is to continually get customer input. What channels can you use to do this? Here are some of the methods we used:

- Invite customers to participate in your design and scenario definition efforts; ask them to engage directly with the development teams.
- Hold customer focus groups.
- Arrange for customer visits to your site.
- Gather feedback and conduct surveys on beta configurations.
- Ask customers to do heuristic UI reviews.
- Ask customers to provide competitive snapshots and formal reviews.
- Conduct milestone assessments based on your scenarios and requirements.
- Invite customers to do targeted usability evaluations and validations.

For the product we were developing, we targeted ease of use as a primary differentiator, because automated tools in this product's marketplace are sometimes complicated to use, and they generate complex reports. Since "ease of use" means different things to different users, getting comments from customers early on was paramount. For every Construction iteration, we had various customers try out the build. We did not wait until the product was frozen or in the deployment phase, as we had in the past. We provided test cases and installation documentation to the technical support teams who would be responsible for supporting the product after release; we packaged the builds for the training team that would design customer classes for the product; and we sent similar packages to the field support people who would be responsible for deploying the product at customer sites. We collected comments from all of these groups and tracked them, using the defect tracking mechanism in IBM® Rational ClearQuest.®

We also conducted usability studies for each iteration cycle, starting with the Elaboration 4 (E4) phases. Usability experts continually reviewed the product and supplied formal evaluation reports at each iteration exit point. Testers were encouraged to include enhancement requests aimed at ease of use early in the Construction cycles.

In addition, we networked with IBM's internal product departments to deploy the product for teams interested in running performance tests on an application under development. We assigned highest priority to ease-of-use issues early in the Elaboration and Construction phases, and developers fixed them within those same phases. We did not exit an iteration until we had closed the cases for all issues. Finally, we notified upper management about these usability and ease-of-use changes on a weekly basis.

Know the schedule

The greatest challenge to maintaining a development schedule is that "the only constant is change." The best defense against a runaway schedule is to develop and test your software iteratively. By dividing your effort into milestones and tasks that allow

you to illustrate proof of concept to managers and customers early and demonstrate the evolving product throughout the lifecycle, you can eliminate risks and errors, accommodate changes when it is least costly to do so, and ensure that you are building the product the customer really wants.

Rather than describe each iteration in detail, I will focus on what we accomplished during our Construction iterations.

The Construction phase for our development project actually consisted of three iterations, each with the following activities: planning, coding, feature verification testing (FVT), defect fixing, and assessment (see Figure 2).

Figure 2: Activities in project Construction iterations

Our Construction iterations overlapped; planning activities in the second iteration -- Construction 2 (C2) -- took place in parallel with later activities in the first iteration, or Construction 1 (C1). This was necessary because of component dependencies and the number of people involved in the project. It was also an effective way to maintain our schedule. For instance, developers who did not have to deliver features or fix defects in C1 worked on their C2 items rather than remain idle until C1 was finished.

Earlier I touched upon all the tasks we completed during Construction; Figure 3 summarizes the phases in each Construction iteration, along with their corresponding tasks. Before exiting each iteration, we reviewed all artifacts and results to verify that they met our exit criteria.

Iteration activity	Task summary
Planning	Specification updates
	Requirement updates
	Requirement reviews
	Test planning
Coding	Test case Identification
	Test case implementation
	Programming
FVT	Test execution and logging
Defect fixing	Some developers conducted defect fixing, while other developers and testers continued with FVT testing and defect validation
Assessment	Review of exit criteria
	Usability and test assessment reports
	IBM Rational PureCoverage metrics review
	Application analysis metrics review
	Test pass/fail results review

Figure 3: Construction iteration tasks for Rational software product development project

Summary

This article only touches upon the measures we take at IBM Rational to continually ensure quality as we develop a software product. I hope I have described enough to illustrate that all the measures are interrelated and affect everyone involved in the project. Each one relates to one of the basic areas of knowledge required for project success: product, staff, customer, and schedule.

Establishing a process that recognizes the interdependencies among these areas and provides guidance for the entire software development lifecycle is key to creating high-quality software. This case study illustrates how an iterative process can ensure that the product you develop will ship with minimal defects, operate efficiently, and contain the features your customer needs and wants. To learn more about how to continually ensure quality for your products, be sure to read the article on "*The Business Value of Software Quality*" and Part III of "*Building the right software development infrastructure for your business needs*" in this issue of *The Rational Edge*.

NOTE: Appendix C shows all the automated tools we used in developing the product discussed in this article.

References

Keith Eades and Jeff Fisher, *The Formula that Helps Win Sales.* No Boundary Press, 2000, ISBN: 0964501392. (Discusses pain chains.)

Mike Perks, "IBM Best Practices for Software Development Projects." Available at: http://www.computerworld.com /managementtopics/management/project/story/0,10801,85198,00.html

Andy Owen, "Best Practices for Software Development." Available at: http://owen.typepad.com/tt/

Ron Patton, *Software Testing*.

Cem Kaner, James Back and Bret Pettichord, *Lessons Learned in Software Testing*.

Tom Gilb, *Principles of Software Engineering Management*.

Murray Cantor, *Software Leadership: A Guide to Successful Software Development*.

Tom Mochal and Jeff Mochal, *Lessons in Project Management*.

Philippe Krutchen, *The Rational Unified Process: An Introduction*.

Appendices

Appendix A: Requirement review tools (from *Software Testing* by Ron Patton)

Requirement review checklist

List of attributes to test against:

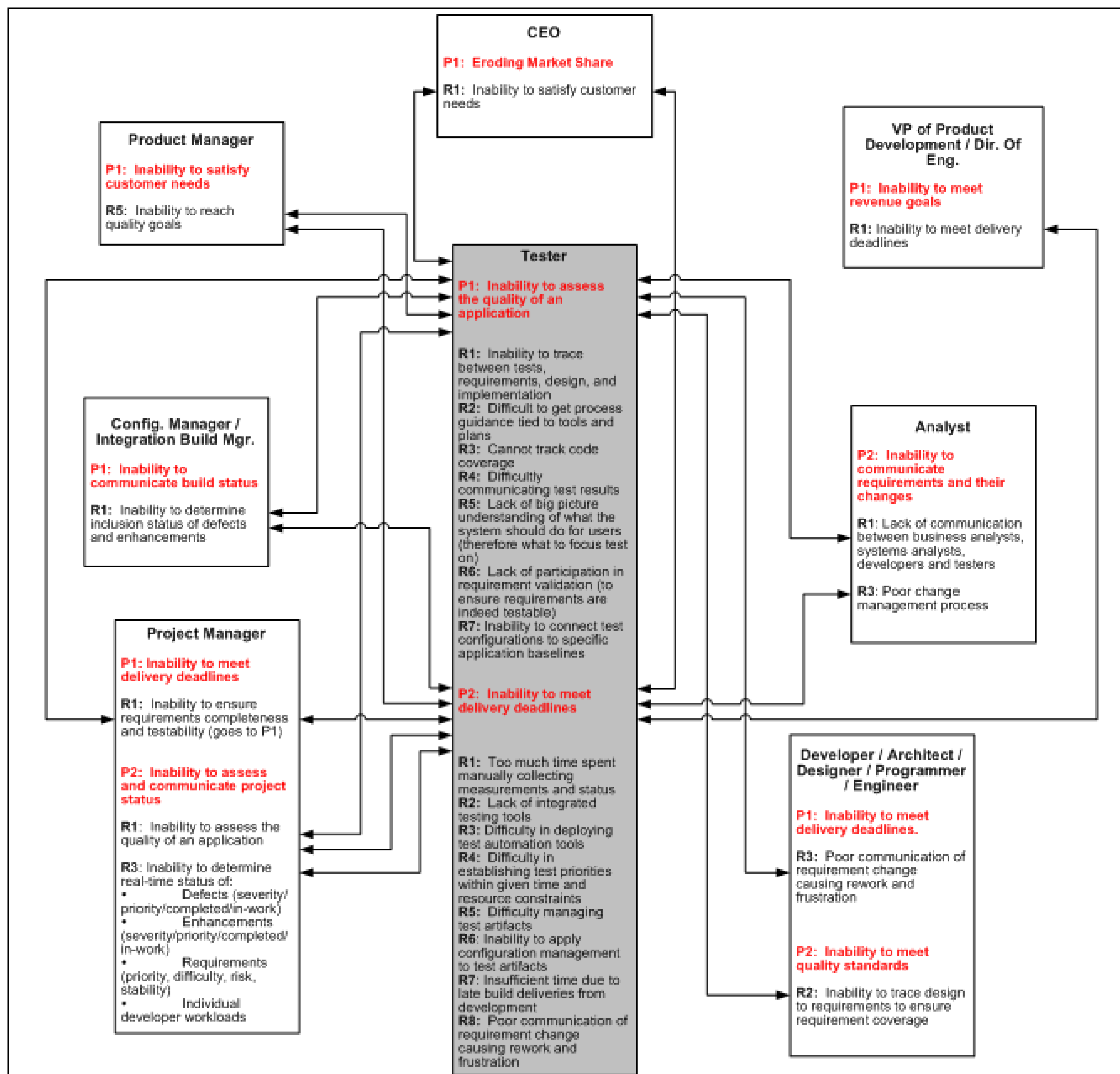
- **Complete.** Is anything missing or forgotten? Is it thorough? Does it include everything necessary to make it stand alone?
- **Accurate.** Is the proposed solution correct? Does it properly define the goal? Are there any errors?
- **Precise, Unambiguous, and Clear.** Is the description exact and not vague? Is there a single-interpretation? Is it easy to read and understand?
- **Consistent.** Is the description of the feature written so that it doesn't conflict with other items in the specification?

- **Relevant.** Is the statement necessary to the feature? Is it extra information that should be left out? Is the feature traceable to an original customer need?
- **Feasible.** Can the feature be implemented with the available personnel, tools, and resources within the specified budget and schedule?
- **Code-free.** Does the specification stick with defining the product and not the underlying software design, architecture, and code?
- **Testable.** Can the feature be tested? Is enough information provided that a tester could create tests to verify its operation?

Problem words in a specification

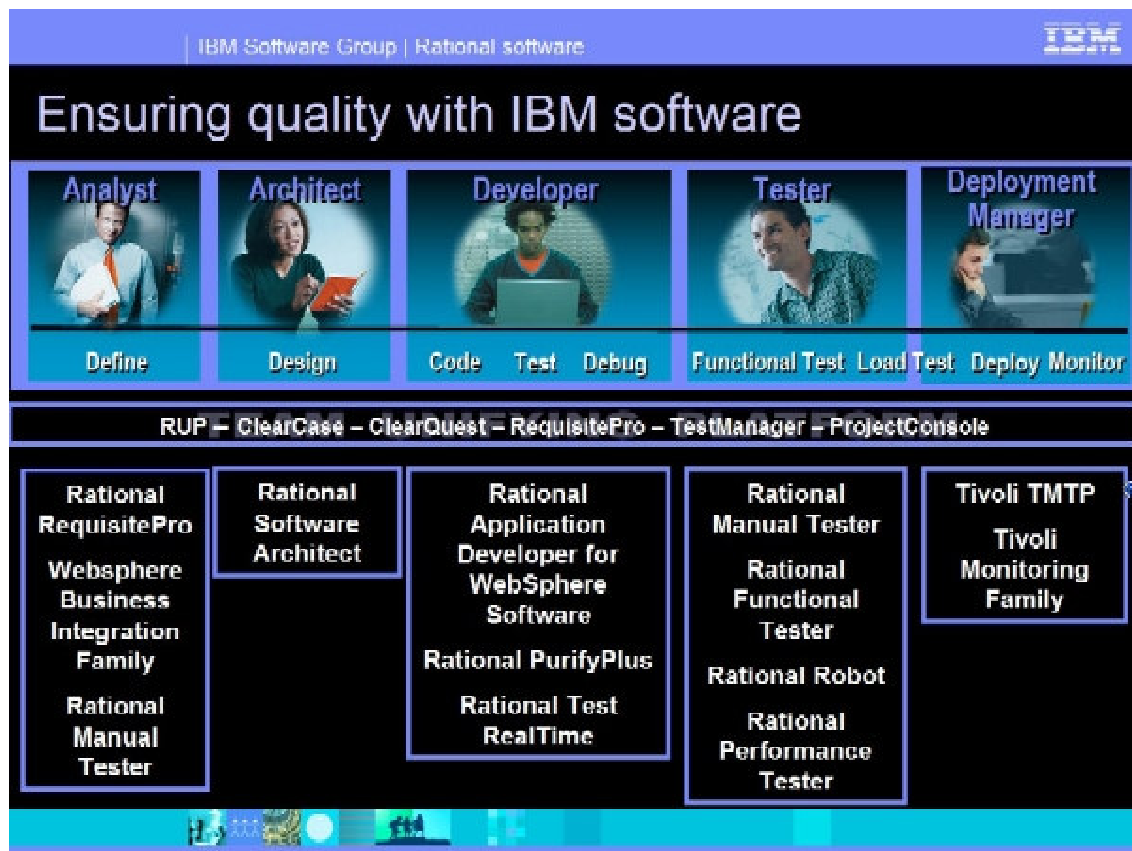
- **Always, Every, All, None, Never.** If you see words such as these that denote something as certain and absolute, make sure that they are indeed, certain. Think of cases that violate them when reviewing the spec.
- **Certainly, Therefore, Clearly, Obviously, Ordinarily, Customarily, Most, Mostly.** These words tend to persuade you into accepting something as a given. Don't fall into the trap.
- **Some, Sometimes, Often, Usually, Ordinarily, Customarily, Most, Mostly.** There words are too vague. It's impossible to test a feature that operates "sometimes."
- **Etc, And So Forth, And So On, Such As.** Lists that finish with these words aren't testable. There needs to be no confusion as to how the series is generated and what appears next in the list.
- **Good, Fast, Cheap, Efficient, Small, Stable.** These are unquantifiable terms. They aren't testable. If they appear in a specification, they must be further defined to explain exactly what they mean.
- **Handled, Processed, Rejected, Skipped, Eliminated.** These terms can hide large amounts of functionality and need to be specified.
- **If... Then (but missing Else).** Look for statements that have "If...Then" clauses but don't have a matching "Else." Ask yourself what will happen if the "If" doesn't happen.

Appendix B: Tester Pain Chain (created by Leonard R. Callejo, IBM Rational)



(click here to enlarge)

Appendix C: Glossary of tools for the Rational product development project



IBM Rational® Unified Process,® or RUP® -- Set the stage for the iterative software development lifecycle.

IBM Rational RequisitePro® -- Document, communicate, and control changing requirements to maintain project focus.

IBM Websphere® Business Integration Family -- Define, model, analyze, simulate, and report business processes, and extend IBM Websphere MQ Workflow with business tools to visualize process impact. Also monitor business processes -- in real-time and using visual dashboards -- for improved business decision-making.

IBM Rational Manual Tester -- Perform more frequent, efficient manual tests.

IBM Rational Software Architect -- Unify architecture, design, and development with one tool.

IBM Rational Application Developer for Websphere Software -- Quickly design, develop, analyze, test, profile, and deploy Web, Web Services, Java, J2EE, and portal applications with a comprehensive, Eclipse-based IDE that is optimized for IBM Websphere software and provides capabilities for development on other technology platforms.

IBM Rational PurifyPlus™ -- Increase unit testing and debugging effectiveness with runtime error detection, bottleneck discovery, and code coverage analysis.

IBM Rational Test RealTime -- Perform component testing and runtime analysis for applications targeting embedded and other real-time environments.

IBM Rational Functional Tester -- Optimize defect discovery through test automation for complex Java, VS.NET Winform, and Web-based applications.

IBM Rational Robot -- Automate functional testing of multi-tier client/server applications.

IBM Rational Performance Tester -- Verify Web application performance, scalability, and reliability under high multi-user loads.

IBM Tivoli TMTP (Tivoli Monitoring for Transaction Performance) and IBM Tivoli Monitoring Family -- Monitor essential system resources to detect bottlenecks and potential problems, and to recover automatically from critical situations.

About the author

Laura Rose is a quality assurance expert and the product manager responsible for automated performance test tools at IBM Rational. In addition to leading projects in both software programming and testing environments, she has thirteen years of programming experience and ten in test management. She has been a member of the American Society for Quality, the Triangle Quality Council, and the Triangle Information Systems Quality Association. She is published and has presented at various test and quality conferences including IBM Test Symposium West, Practical Software Quality Conference (East and West), the American Quality Society Conference, Better Software Conference & EXPO and StarWest. You can reach her at llrose@us.ibm.com.

Share this....