Rational. software

IBM

Q&A with Industry Experts: How Are e-Business Trends Impacting Testers and Testing Teams?

Part I of II:
Challenges for Testers

Jack Wilber
Writer/Interviewer
The Rational Edge

TP319, 08/02

Introduction

In this two-part interview, industry analysts shared their views on how e-business trends are affecting the testing community. Of all the professionals that comprise an extended development team — project managers, analysts, testers — — perhaps none feel the impact of current trends more than testers. They are the people tasked with ensuring the quality of complex applications with very limited resources and in the face of rapidly approaching project deadlines.

For insights and opinions, the author turned to a panel of three respected testing experts and analysts: *Theresa Lanowitz*, Research Director at Gartner; *Hung Nguyen*, President and CEO of LogiGear® Corporation; and IBM Rational's own *Sam Guckenheimer*, Senior Director of Technology for Automated Software Quality. In the first interview, they share their thoughts on the challenges testers are facing and the technologies, skills, and strategies needed to meet them. Part II focuses on how changes in architesture have affected testing and automated testing tools.

Skills

Let's begin by talking about skills. In the last two or three years, how has the explosive growth in distributed applications affected the skills and domain knowledge that testers need to be effective?

Hung Nguyen, LogiGear: I think the effect of this growth has been tremendous. In the past, everything in the testing environment was very self-contained: You got a deliverable, you ran the installation program, and you started testing. But when you go to a more distributed, or e-business, model, there are two main problems for testers. First, on the technology side, everything has changed. You don't have control over your environment because your system might have components distributed all over the place, some that your team developed, and some third-party components. So just trying to understand the environment and figuring out how to test effectively within it is a big technical challenge.

Second, on the business side the rules have changed as well. In the old days, users bought a package, installed it, and used it. Now you have users who might buy a package, or they might just use your e-business infrastructure to conduct business transactions. So on the business side, testers need a lot of education to be effective. For example, consider performance, just one dimension of testing. In this new environment, the tester needs to understand non-functional issues such as, What is performance? How do I come up with a "reasonable" response time and test for it? That is something they don't always see in the functional spec. Another area of concern is security testing. Testers need to ask, How do I know that my users are protected or that the business is protected? Right now, that's a gray area for testing, because few testers know how to do it effectively.

We have begun to realize that in order to test effectively, you need technical skills. Because the field is not mature enough, we still have non-technical people doing testing. Now, there is nothing wrong with that at the business logic and user level. But you also need to fill the gap on the technology side. Until everyone understands that we need skilled people to do the job, I think that testers will, unfortunately, continue to be underdeveloped, and earn less on average. Ideally, the more skilled testers would know as much about the technology as a developer and would therefore deserve to be paid comparably or commensurate to their ability — and management needs to understand this. If the salary structure shifts and there is a budget for bringing more talented people into the mix, then more developers will be interested in becoming test engineers.

Theresa Lanowitz, Gartner: Even though we have seen explosive growth in distributed applications, we have not seen explosive growth in skills, either for developers or for test engineers. With many distributed applications, the application is the business. Suddenly, the enterprise has all these customer-facing applications, and the IT organization in the traditional enterprise is now responsible for creating revenue-producing products, not just applications.

But skills have not grown; in fact I'd argue that they have diminished, because as more and more enterprises rushed to create these customer-facing, revenue-driving products, they could not find enough skilled people, so they hired inexperienced people. We saw a lot of that back in 1999 and 2000, which accounted for a lot of high-profile Web site failures. Around the same time, you would hear a lot of hype about testing tools that were so easy, you didn't

have to be technical to use them. But I think that is the wrong message to send; you really do need technical skills to do what we're expecting testers to do with these applications.

And distributed applications are only getting more complex. From an evolutionary perspective, with mainframe applications you knew who the users were, you knew what the architecture was. Then there was client-server, and then the Internet world, and now you have wireless applications. And in order to handle the new complexity on the testing side, you want skilled quality engineers — people who understand process and what quality engineering is all about.

Companies know this, but few act on it. Through surveys, we know that getting people with solid technical skills is a top concern for enterprises. However, one of the things they're least likely to spend money on is training. So it's a constant conundrum.

Another problem is that testing is often the first thing a development organization cuts when the budget needs to be pared back. Also, testers may be perceived as entry-level people, or testing regarded as a position you accept first, before moving on to become a developer. As Hung pointed out, testers are really not given the professional equity and respect that they deserve. So organizations constantly have the same problems over and over again because they don't do enough to keep a core group of testers with institutional knowledge.

Sam Guckenheimer, IBM Rational: So what we're saying is that once upon a time people believed that you could test without having deep technical knowledge of the software under test, but when you are looking at a distributed application — on the Web in particular — that assumption breaks down. Hung's book[1] on the subject of testing Web-based applications is excellent on this point. Testers need to understand how the technology affects the kinds of errors and risks that they can see. They need an understanding of technology issues — such as the deployment topology — as well as understanding of and the kinds of errors inherent in the technologies they're examining. Even understanding details like the difference between bean-managed and container-managed persistence on an application server — all these issues affect what kinds of faults you are going to find. Today, testers need to understand the technology and the domain as well as generic testing techniques.

For example, suppose you see an error message that says "404 - Page not found" in the browser. That error might be caused by a broken link, or it might be because some service has become unavailable. A good tester will not only suspect the unavailable service, but will also be able to confirm his suspicion — for example, by looking at other pages that depend on that service. This is a critical technique for isolating a bug.

Another skill that has gotten a fair amount of attention recently is the ability to be a good explorer. Historically, a lot of what was described as testing was very scripted and planned, but in reality good testers are good explorers. They see things that may be hints, and they know how to follow up on them. It may be something as simple as a page that takes surprisingly long to load. A good tester will ask, Why would that be? and knows what paths to go down. James Bach has written the best material about exploratory testing and has the best exercises on the subject. I think it certainly is a critical skill, and one that a testing team needs to have.

Pressures

For years, as organizations have tried to develop software "faster, better, cheaper," testers have been there to ensure the "better" dimension. Is there now more pressure to help with the "faster" and "cheaper" dimensions?

Theresa Lanowitz: What we're really talking about is the age-old triangle of choices: budget, schedule, or quality. Your question asserts that testers have been there to ensure the "better"; but have they really been able to do that? Consider the role that test engineers have been forced into. In traditional waterfall development, testing occurs only during a brief period before the application goes live. The test engineer really never has much input into developing either the use cases or the test cases. And if the schedule slips during engineering, it's the test engineer who feels it on the back end. I would argue that testers have not always been able to ensure the "better."

To do so, they really need to be the customer's advocate. And I don't think they have been given the respect, time, tools, or even the right cultural settings for this. Organizations are always more concerned with faster and cheaper than better, and it usually takes a catastrophic or near-catastrophic event for most to realize that their development

abilities were not as good as they had assumed. We've seen this over and over again, with all the high-profile outages and site failures we've had over the past few years.
Building a high-quality application, within budget and on time, takes a very disciplined organization — in terms of both management and process. And that kind of culture is not yet pervasive in the industry.

What are the cornerstones for that culture? Skilled professionals; processes and procedures that you can document and repeat; strong tools and services. Often people think that a tool is going to be a panacea, but that's not the case. If you are focused on delivering faster, then you are probably sacrificing quality and maybe exceeding your budget as well. This is what we saw during the dotcom boom. The sad truth is that you are not really getting to market much faster either, because over time, new development costs will get out of control, surpass maintenance costs, and prevent you from getting to market with the right product at the right time.

Hung Nguyen: I think this issue can be traced to a lack of budget for testing groups. Management always wants to build better quality products — I have not met one who says otherwise — and that requires better process, better development methodology, and better testing strategy. Yet if you look at most business budgets, there is no line in there for
testing; it all goes to R&D or development. So there's no visibility for testing within the organization and no budget at the business strategy and management level, but testers still have all the responsibility of making sure the system works.

Another problem is that there are really very few reliable metrics to show how much you've done before, so there's no traceability you can use to determine whether you are doing better or worse. If you have a huge failure in the product, where do all the fingers point? At first they point at testing, but eventually the blame spreads all over the place, and no one is accountable for one single thing. I think that is the number one problem from a management perspective. If you want "better," then you have to increase visibility for testing and quality engineering, and you can effectively do that through a budget. Team up testing folks with development staff to figure out how to get the job done. The testing budget can be a percentage of the development budget or, preferably, of the business budget; the actual amount is up for debate but it has to be something. That is how we allocate funds for marketing, sales, and R&D. So why not testing?

Developing "cheaper" is not easy, either. Tools can certainly help, and so can process. So can education, particularly on how to use the tools effectively. Actually, this goes back to the skills issue we just talked about. Finding good testing education is a problem; serious, skill-based software-testing curriculum is limited. Off the top of my head, the only example of a good program available today in the U.S. is the one offered by the Florida Institute of Technology, where Cem Kaner and James Whittaker teach. Programs delivered by the University of California at Berkeley and Santa Cruz Extension, LogiGear and SQE are also examples of limited useful course offerings on software testing. Other than that, I think there is a huge skills gap, and adding more education at the college level would be good step. Companies like Rational are constantly developing tools that support new technologies, but testers need to understand them in a larger context. Tools are just a means to solve a problem. To use them effectively, I need to know I have a problem, how the problem is defined, and that there are a number of ways to solve it. That is the kind of education I am talking about.

Sam Guckenheimer: The key to faster and cheaper is an iterative development process that brings testing forward in the development cycle, making it possible to find defects when they are cheaper and easier to repair. However, I don't think testers are well trained to work in iterative processes. Nor are project managers well trained to consider the testing role; that's why we've added a lot and are continuing to extend the Rational Unified Process and Rational University training to show how testers can work iteratively.

But even if you're not doing iterative development — if you're doing waterfall — the same concept applies: To save time and money, test basics first. You want to validate the spec and do function-level testing from simple tests first, in early iterations, and build up to complex scenarios and configuration testing and multi-variant combinations in later ones. Ideally, you build up a growing repertory of automated tests, though you also need to refactor them as you go. For example, automating tests for interface contracts is absolutely critical, and those should be run in regression all the time. But in user scenarios that may change based on usability, test feedback, or design changes, you also need to be sure that you're clear about what you're automating and how you're going to refactor the tests when the application under test changes.

What's really important is to understand the power of testing at many levels and not to think of testing just as something you do from the GUI on a finished system. As testers, we need to think carefully about unit testing and interaction testing, as well as what kinds of tests are appropriate and where.

Process Changes

Let's talk more about process. What changes have there been in the way testers work with the rest of the extended development team? Agile development processes have promoted awareness of test-first design and unit testing. Are test teams now getting more involved in code-level and model-driven testing?

Sam Guckenheimer: Let's take these one at a time, starting with the way testers work with the extended development team. I am a firm believer that testers need to be closer to developers; they should be working in a tight loop, iteration by iteration. I think that about half the market works that way now. The other half thinks that testers should be independent, and a lot of them outsource their testing. In my opinion, you lose half the benefit of testing when you do that. You get people to find bugs, but you do not create a process based on continuous and exploratory learning. If your testers are working close to your developers, then all can learn as they go, and they can both contribute to making a much better product. If you throw testing over the wall to outsourced testers or a test team outside your
project, then people can find mechanical bugs and report them back to you, but you have limited ability to really evolve the product or process in an iterative way.

This leads into the "agile development processes" part of your question. The notion of evolutionary development is fundamental to Extreme Programming (XP), which has grown into the agile movement. Testing in agile development is not well defined, and there are many views on what it might be. I tend to line up with the definition that Brian Marick and Bret Pettichord have been working on, which is based on six principles — actually they call them slogans — that capture practices. One is that you develop tests as the embodiment of design specifications;

essentially, the tests are the design specifications. So what the Rational Unified Process calls use-case realizations, they accomplish through tests. At the same time you do exploratory testing on the software that is built, and you continually iterate and refactor, focusing hard on design for testability. I think these are all great practices, and a lot of testers are starting to pay attention to them.

Do I see testers getting more involved in code level testing? Here, the nomenclature is a bit confusing. People who are called testers in one organization are called developers in another, and vice versa. In most organizations, testers do not get involved in testing directly from source code unless they and the developers are working in pairs. I think that is appropriate, because developers should take responsibility for the quality of the source code. We've known for a long time that the best person to test the source is the person who wrote it, and Rational offers strong tools to support developer testing activities.

Model-driven testing is another issue. Models offer a great way to document a system, visualize system behavior, and communicate shared work across the team in an accessible way that also reduces complexity. Interest in using models for testing is growing exponentially, and that is a fantastic trend. Model-driven testing has a few meanings. One is that models can be developed specifically for testing, separate from the code development, as a way of generating high-volume tests. (That is the meaning Harry Robinson of Microsoft uses on the Web site he maintains: www.model-based-testing.org.)

From Rational's point of view, on the other hand, model-driven testing means that the model depicts the software under test — its structure and behavior. The same model captures the definition of what to test and can also capture test results. We are actively contributing to the development of model-driven testing. Rational® Test RealTime, for example, shows the behavior of the software under test in a UML sequence diagram. Our concept of model-driven development is consistent with the work that's being done in the OMG[2] working group on a test profile for UML. Once the UML test profile is adopted by OMG, I predict that we will see an explosion in the use of models for visualizing results and defining tests.

We haven't talked yet about the way testers work with analysts. There has always been a relationship between these two roles, and even in the most "waterfallian" of processes (e.g., IEEE 829), people understand about testing requirements. The evolution of modeling into an analysis and development practice tied analysts and developers together, because it enabled developers to translate requirements into designs with progressively greater levels of

specification. On the flip side, it also allowed them to visualize these designs at progressively higher levels of abstraction. Testers weren't originally considered in that loop, but all of the same benefits apply. And indeed, everyone wins when they realize models can not only describe intent, but also capture actual system behavior. Frequently people skimp on use-case realizations in a model, but if teams could apply the same kind of roundtrip engineering to behavior that they apply to structure, that would change. And that is exactly where we are going. If you look at Rational Test RealTime, you'll see that is exactly the kind of value you get from capturing system behavior in a sequence diagram.

Theresa Lanowitz: Process, including a test-early approach, is critical to the success of any organization, but many still haven't realized that. Two or three years ago, Gartner heard a lot of organizations saying, "We developed this application for [fit-in-your-favorite-vertical-industry-here], and we want to take it commercial. Our plan is to sell it to others in the industry and spin ourselves off from the parent organization." But after we had a conversation with them on what it takes to be a commercial software company, they would retreat. We never saw any spin-off. But in fact, enterprises do need to be able and willing to behave much more like commercial software companies. They need to understand the build cycle, requirements, and schedules; they need product managers who can serve as liaisons with the engineering group, and so on. So far, we have not seen enterprise organizations en masse adopting this more structured behavior.

To make such a change, you need a culture that supports it. Practitioners often tell me that management does not want process because they think it will take up too much time. To have a good process, you have to understand what it should be, and keep the management and the philosophy intact long enough to get through the initial stages of adoption. You also have to keep in mind that the end goal is to deliver a high-quality application, on time and within budget, that everyone thinks about as a product. Unless the emphasis on quality is infused and travels from the top down, the organization will tend to run in a chaotic or reactive mode.

It's also important to remember that coding is really only a small part of any development project. Identifying the correct architecture, getting the process in place, making sure you are following the standards that have been established for the organization — those are the key things.

As for code-level testing, developers are now writing more unit tests, and that's a positive thing. Some really good tools have come on the market to help developers create unit tests. However, I still believe that, over time, testers need to become more technical, and the organization needs to invest more in training and keeping testers. Then, as their skill sets keep growing, so will parity and respect for the testing function within the organization. And testers will most definitely be more involved in code-level testing.

And for model-driven testing, the UML is a great thing for that. Once you have the use cases written, you have the test cases written. And it's a very positive thing if you can integrate a good solid process all the way through your software development lifecycle.

Hung Nguyen: Certainly the degree to which testers are involved with the rest of the development team varies greatly by company. One organization might have test engineers that are not very technical, but they have a great process and are able to get the testing, development, and business teams together to talk about requirements and features and document it all. But industry-wide, there's definitely a shift toward getting testers involved earlier in the process and working more with business analysts and the development team.

It's good to have the development team thinking about testability of their code at the source level, and thinking about unit testing. But if you look at where testing takes place — at the requirements level, source level, interface level, component level, and system level for integration tests — where testers are not doing well is at the source, interface, and component levels. I see some good collaboration at the interface (API) level, but at the source level, it is still a developer thing; testers have yet to understand how to be useful in that environment.

For example, Rational® Purify® is a dynamic tool that developers often use, and that is good. But to have better test coverage you need to execute more of the code, and developers don't have time to do that. So it would be wise to integrate the testing team into that process and have testers use Rational Purify during their tests as well. Likewise, it makes sense to have the developers do unit testing in one pass, and then let testers do it in another pass. We need to close the gap between development and testing people, although I still see code-level testing as mainly a developer

activity, probably because of the lack of education in testing. But testers can just run the tests, log all the errors, and send the results to the developer; they don't even need to be able to interpret the results.

I believe model-driven testing has a very important role in test design and analysis. For example, Rational® QualityArchitect can generate tests based on models and dependencies. And once you have an error, you can actually use the model to shorten the path to deduce the failure. So model-driven testing is key to test design and generation, and as a knowledge base for automating failure analysis and pinpointing problems.

Cost of Failure

People often talk about process as a means to reduce software failures. Much has been written about the increasing cost of failure associated with public-facing e-business sites. Has this business change affected testing practice in significant ways?

Theresa Lanowitz: Absolutely. When it comes to e-business, failure is not just a matter of people not being able to use the software; it is a matter of public image. Because you have non-technical people using these applications, and because the applications are moving toward ubiquity, the software has to be foolproof. With a less sophisticated audience you only get one chance. If they try to use something — like a Web service — that doesn't perform or doesn't work at all, then they'll just abandon it and move on to another site. And that speaks directly to the need to build higher quality into things like Web services.

Sam Guckenheimer: I think the increasing cost of failure has raised management awareness of the importance of testing and quality. Fortunately, we've moved beyond the practices of some former dotcoms that ignored quality and focused entirely on speed.

Management is more savvy and more careful since we had those highly visible dotcom failures.

Hung Nguyen: I don't think the high cost of failure is really new; we've faced it before. It does affect testers; it puts pressure on us to be more effective in finding errors. But the problem is more closely related to the quality assurance process.

How do we implement a quality process that capitalizes on people and technology? How do we get QA and development to work together to develop better practices?

In the context of e-business failures, the way we do testing now is different from the way we used to do it. Now, we don't stop when the product is released; we test on an ongoing basis. That is why a new monitoring market segment has opened up, and we're putting mechanics in place to alert us if there is a failure.

Also, the public is better educated now. They understand that if they pay for it, then you have to give them good stuff. They have options; there is so much competition that they are just going to walk if you don't give them a good, quality product.

A very positive result of these failures is that management has begun viewing quality issues in terms of dollars and cents. They are telling their development organizations, "I don't care if you call it a high-quality or low-quality product. If it shuts my site down for two minutes, it costs me a million dollars, and I don't want that to happen. So, you go back and figure out how to prevent that from happening." And management also knows that if they give the testing group a decent budget, then the testing group can be held accountable. You want to put testing at the top of the list when you create the yearly budget because that is one of the primary avenues to get quality. In the end, that will give testers authority, responsibility, and accountability.

IBM software integrated solutions
IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.*
- *Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*
- *Tivoli® software helps you manage the technology that runs your e-business infrastructure.*
- *WebSphere® software helps you extend your existing business-critical processes to the Web.*
- *Rational® software helps you improve your software development capability with tools, services, and best practices.*

Rational software from IBM
Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.

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

Printed in the United States of America
01-03 All Rights Reserved.
Made in the U.S.A.

The IBM home page on the Internet can be found at **ibm.com**