TP328, 08/02

Rational. software

IBM

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

Part II of II: Have Changes in Architecture Affected Testing and Automated Testing Tools?

Jack Wilber
Writer/Interviewer
The Rational Edge

TP319, 08/02

INTRODUCTION

In *Part I* of this series, the author, Jack Wilber, talked with industry analysts *Theresa Lanowitz* of Gartner, *Hung Nguyen* of LogiGear, and *Sam Guckenheimer* of IBM Rational about how some of the same trends affecting developers are impacting testers. In Part II, these experts take a closer look at how changes in architecture have affected testing and automated testing tools, including the challenges inherent in embedded and real-time testing.

.NET IMPACT

What impact do you think the release of the .NET server platform will have on the ongoing .NET/J2EE battle, and how will it affect testing?

Theresa Lanowitz, Gartner: [jd1]Microsoft has the .NET suite, and you're going to see more and more Java providers coming out in support of the J2EE platform and offering more tools for J2EE. When it comes to testing, the way you test applications residing on each will be a little bit different. That is just inherent in the architecture of .NET and J2EE. Large organizations are certainly going to have both .NET and J2EE in house. If you're a small to mid-size organization you are probably going to choose one, based on the skill set that you have internally.
What impact will the .NET release have on testing? Vendors that have been around for a while will continue to do well. Rational being on stage with Bill Gates when Microsoft announced .NET was a big win. Rational has been able to do what seems impossible: have a good relationship with Microsoft as well as IBM and Sun. That is a tough thing to pull off.

Sam Guckenheimer, IBM Rational: The first observation about .NET and J2EE is that the choice is not either-or; you can choose both. They are going to coexist. They are going to interoperate, through things like Web Services. I agree with Theresa, that in any significant enterprise you will have both to contend with. Both .NET and J2EE provide a profiling interface in the runtime environment, which I believe is a fantastic improvement for testability.
In both cases, we come back to an understanding of the technology, deployment topology, and failure patterns as part of the knowledge a tester needs to have. All of this can be different in .NET and J2EE, and in J2EE it can be different from one app server to the next. There is nothing unique to J2EE and .NET that makes that so; after all, it is inherent in the nature of large distributed applications. You can think of the GUI as a very small window into what is happening behind. And those inner workings have many points of failure and enormously complex interactions. Testers need to be very sensitive to that.

Also, I believe the languages for test automation will change to become much closer to the development languages. The richness of those frameworks means that tests will need to be built on top of the same frameworks in order to test effectively and get the full benefit of what they offer. Rational RobotJ is a good example of that kind of tool.

Hung Nguyen, LogiGear: I don't see much impact on testing with respect to .NET and J2EE competition. We had NSAPI (Netscape Server Application Programming Interface), then ISAPI (Internet Server API from Microsoft), then ASP (Active Server Pages), JSP (Java Server Pages), and Servlets – and the list goes on. It's just a fact of life: Competition will be there, and therefore there will always be at least two major players. That means we constantly have to be educating ourselves about new technologies and their use and get the testers involved earlier. But I don't think that is something new, brought on by .NET or J2EE.

TESTING MULTI-TIER ARCHITECTURES

What about API and protocol-level testing? Traditionally, most testing has been driven from the GUI, but that has changed with multi-tier architectures. Web Services, for example, have no GUI to test at all. What is the impact of this shift on testing practice, test tools, and test automation?

Theresa Lanowitz: I think test tools will continue to evolve over time. In the past three to four years, these tools have improved immensely, and people better understand the benefits of automated testing. One is repeatability. Once you have created the scripts, it is just so much easier to do testing, because you don't have to spend time creating new tests. Still, not everyone has adopted automated testing.

But when you move to more complex, more pervasive types of technologies like Web Services, failure is simply not an option. Testing has to be done, and you need more experienced and more technical individuals to do it. If you look at tools for developers, the message is not about ease-of-use; it is about power and functionality, and that same message should go out to the testers.

Contrary to what was said earlier, I think it's fine for organizations to outsource testing in some instances. Organizations are being asked to do incredibly difficult things in creating these customer-facing, revenue-driving applications. Yet they are not necessarily given the people to do it with. So for them, outsourcing – whether through something like a hosted service or a lab situation – is a positive thing to do. The people who run the test may not share your domain expertise, but they really understand how to test and will bring less bias to what they are testing. And you'll have a skilled project manager coaching them. Organizations cannot do everything in house these days. They don't have experts on staff who understand every single OS you might use and every single piece of technology, let alone the time to research every new technology to determine what the organization should be moving to.

Sam Guckenheimer: This is another one of the principles that Marick and Pettichord are promoting under agile testing – the idea that test automation should be not at the GUI level, but rather underneath the presentation layer. It is absolutely the right way to build robust automation. GUIs are highly changeable for all sorts of reasons. People change their Web sites every week, for example, simply because they want them to look new. But the underlying transactions – to purchase a book, for example – change less frequently. So if you're testing at the GUI level, your test has to change, whereas if you're testing underneath that presentation layer at the business logic, then your test is stable. So clearly it's a good practice. And as we move into Web Services and related technologies, testing underneath the presentation layer goes from being a good practice to being a necessity. It's already a requirement in the embedded software world, which is why Rational Test RealTime is not focused on testing at the GUI, but rather at a layer beneath it.

Hung Nguyen: I have already talked about API and GUI testing but not about automation yet. Automation is the part that we have to do better. That is, we need to understand exactly what we are trying to accomplish before we buy a tool. Model-driven testing, for example, helps automate test-case design, but you still need to execute and manage the tests. Some engineers still think of test automation as translating a manual script into an automated one. We need to focus more on better practices and methodology for automated testing, which are just as important as the tools.

DESIGN-FOR-TESTABILITY

Many people recognize design-for-testability as one of those best practices. How can it help ensure the quality of Web Services and software in general? And are you seeing a lot of teams adopting design-for-testability practices?

Hung Nguyen: I think design-for-testability is essential for testing the Web Services model. Testing is much more complex now. Before, with components, at least you still had the binary, but now you don't see anything. All you know is the API, and that can be very scary! Developers and architects need to think beyond what they normally do, that is to think about design-for-testability not only in the code they write, but also in the larger context of the system architecture. If I have an e-business *system*, I want to know how to integrate the *system* so that it is testable. So if we're thinking only about code when we think of design-for-testability, that is too low-level in my opinion.

This goes back to getting testers involved with the extended development team. We want to extend the notion of testability to the system level. For example, if I am going to use the Google search engine as a Web Service on my site, I want to be sure I have hooks that enable my testers to test that component. In other words, I identify what I have control over and what I don't have control over, so I can determine the best method to test each component or service. For developers and analysts, it is important to think about the larger context, not just at the code level.

Before, we used to design software and not even think about testability. Now we are moving in the right direction and thinking about it, but still in terms of the software we *used* to develop in many cases. Now we're moving toward Web Services and distributed systems; the whole paradigm has shifted, but we're a step behind, still learning how to do design-for-testability for the old paradigm. It's a very good first step, and I suggest that we also need to look forward and think about testability for the new development paradigm. By the way, I am very encouraged about the direction that Rational is going with UML, looking into how we can make UML more useful for testers.

Sam Guckenheimer: One of the principles of design-for-testability is that you have access to that layer we talked about below the GUI. You need access to the business logic or the behavior of the software under test, just beneath the presentation layer. Brett Pettichord, I think, correctly characterizes design-for-testability as being about visibility and control. The visibility you need is through exposure at lower layers, with lots of open interfaces that allow you to see into the state of the software under test. Similarly, you need interfaces that allow you to control the application so you can drive it from an automation framework without engaging the GUI. Frankly, that is good component-based development. It is also what you get if you follow standards like Web Services.

The layer beneath the presentation layer is not necessarily HTTP. HTTP is a transport protocol; it does not tell you anything about the content. That's the big leap between the first generation of using the Web and the move to Web Services. With Web Services, the important thing is that you have highly structured content in XML (eXtensible Markup Language). The content is self-describing, so it does not rely on custom parsing of whatever strings happen to be in the HTTP without any guidance. Web Services take componentization to that layer of self-describing messages. That's the big difference. Obviously you can test under the browser at HTTP and then render the HTML and so forth, but the HTML is really just the GUI to be rendered. If you look at what happens under Microsoft .NET with ASP .NET page definitions, the HTML itself varies, depending on the capabilities of the browser. So just relying on the HTML is not a great testing strategy.

Where you capture the data is beneath the GUI, but more important is *what* you capture and with what intent. Capturing it in the API or on the wire is common in both cases, and the data you capture would be the same. The difference is in the way you build the application. In design-for-testability you ensure that what you get is meaningful and testable, and not just some arbitrarily assembled character string.

I think that people are embracing design-for-testability more and more. It's happening in two ways. First, it is becoming part of standards and frameworks – for Web Services, for example. A simpler example is the opening of profiling interfaces in J2EE and .NET that allow tools to inspect what is happening in the runtime *environment.* Second, people are building applications from design patterns now with tools like Rational XDE – and part of the design pattern is testability. Components constructed from patterns include exposure to test interfaces – with appropriate getters and setters on the components. And I see that as a very real and beneficial trend.

Theresa Lanowitz: I think the adoption of design-for-testability practices will evolve along with Web Services. As more and more organizations build and use Web Services, certainly design-for-testability is something that people are going to do. They have to, because they cannot afford to have failures. Testing is more crucial now than it ever was before; the more people you have using your software, the more it has to be tested. As I said in the previous article, enterprises really need to behave more like commercial software development companies; although they are not going to sell their products, the products are driving revenue. The more people you have using them, the more visibility you have, and the easier they need to be to use. At some point the software needs to function like a utility: Without understanding anything at all about how it works, people should just be able to use the software – just as they would turn on a light switch or a water faucet.

Given all these changes, do today's testers have the tools, processes, and resources they need to do their job effectively? What are they missing?

Hung Nguyen: I'd say testers have the tools, but they need more resources, better strategies, and education to do what they need to with those tools. What testers need to know – and they might need to collaborate with developers on this – is what potential errors they might discover if they use SOAP (Simple Object Access Protocol) or XML, for example. If they understand those errors, then they'll have a much better understanding of how to test for them. Once they have that, they can start thinking about the most effective way to automate that testing. In my opinion, developers understand how to use tools more effectively than testers do because traditionally, their training has always involved some type of tool, like a compiler or a debugger, for example. Basically, I think testers can improve on their understanding of how they can effectively use the tools that are available.

I will say that one category of tools I would love to see is in the area of change management for testers. What we typically control with version control and configuration management is really at the source level. Now things have changed drastically, and when we do testing, we test against an environment. In that environment, if any one of many

might invalidate all the testing we have completed. I think if we had tools that enabled us to have control over the environment at the file level, and at the service or environment level, that would be very useful.

Another tool I'd like to see is one that gives testers the ability to trace a transaction. When a transaction is submitted, it traverses a number of components and servers. If there is a failure, it would be nice to have a record of where it happens. In other words, the analysis side of testing is lacking. That's because it is very difficult to do. But if we can figure out how to do SOAP, then we have to figure out things like that for testing and traceability – especially in the .NET or Web Services environment.

Sam Guckenheimer: One thing I see as a very good trend is the combination of runtime analysis and testing. It addresses precisely the point Hung just discussed: the need to diagnose problems within the system under test. Being able to do memory profiling and performance profiling, tracing, and a code coverage check as part of a running test is a tremendously valuable thing, particularly in a distributed system. We are doing a lot of work to bring testing and modeling together with runtime analysis with tools like Rational® XDE,™ Rational® PurifyPlus, and Rational® Test RealTime. Rational PurifyPlus enables runtime analysis, and Rational Test RealTime, with its ability to trace transactions, is the best harbinger of where we're going.

Being able to visualize that behavior in a model is also really important. Tests can produce a huge amount of data, which, unless you can browse it in an accessible visual form, can be overwhelming.

That leads to the need for artifacts that describe the software under test to be easy to share among the extended development team. They are valuable resources you can use at all of the steps in an iterative software development lifecycle. I think that we'll see a demand for increased integration in a few directions: integration of test and runtime analysis; integration of test and models; suspicion analysis from the change management system; and integration of test results into the full change management system so that you can understand them relative to differences in development streams.

Theresa Lanowitz: In many cases I think what organizations are missing is a willingness to allow the process to work. They need to understand that process is necessary, and that they really need things like product lifecycle documents to help them understand what their processes are going to be. A product lifecycle document is a living document that identifies what steps have to be taken to produce and release an effective application. An organization can set alpha criteria and beta criteria, and specify definitions so that everyone is working from the same game plan. The document can identify what has to happen at a requirements gathering meeting, and identify the roles and responsibilities of each group. As things change, the document is updated. Without such a document, you are always working in a chaotic mode. Applications are becoming more and more crucial to businesses that develop them, and they only have one chance to get it right, so an effective product lifecycle document is crucial to success.

Organizations also often lack skilled professionals to do their testing. Tools always keep up with technology, but the skill sets within an organization may not. As technology changes, and more and more organizations start to build Web Services, the tools will naturally evolve to keep up with both user- and enterprise-level demands. The people on the leading edge should be the ISVs themselves. Rational, for example, uses its own tools during its development process. So if they are attempting to develop Web Services and discover that something is lacking in their tool set, they can develop that and then offer it out to customers.

In short, right now the tools are sufficient, processes need to be given a chance, and the need for skilled testers is really the crucial factor. To keep those skilled testers in place, you need to give them parity with developers, in terms of respect, salary, job titles, everything. I've found that companies that are wedded to the Capability Maturity Model (CMM) are better able to maintain parity for corresponding titles. They will have a test engineer level one, for example, on a par with a development engineer level one; the two will make the same salary for the same level of experience. And these organizations are able to continue that kind of parity all the way to the top, so they don't have a lot of people wanting to leave the test organization.

Some commercial software companies have QA engineers who stay for five or ten years because they are rewarded professionally and financially. That's not saying you don't want any entry-level people in those positions, because you do. Evolution and change is always good for an organization. But if you don't hang on to a group of people with core knowledge, then you'll end up solving the same problems over and over again because there will be no continuity.

how to use the latest and greatest automated test tool at what is basically a vocational school. But courses on software quality engineering are needed at the curriculum level of universities that offer computer science degrees. That would be a positive step, but the courses have to be not just about how to use tools, but also about understanding what a process means. There is definitely a lack of such instruction today, and I think it is because testing is perceived as a vocation instead of a profession.

EMBEDDED AND REAL-TIME SYSTEMS

Let's talk about embedded and real-time systems. Are the challenges inherent in these systems being addressed by current testing tools and methods?

Hung Nguyen: I think the key problem in many embedded and real-time systems is that you don't get to see much through the user interface. But you still have to test a lot of code that passes through a lot of logic, performing a lot of calculations; so more extensive testing must be done at the source level. The challenge is that in embedded products there are many custom solutions, and it is difficult to have generic tools for all custom solutions.

Unit level and source level testing are very helpful here. The question is, should developers continue to do these kinds of tests, or should we train more testers to be effective in testing at the source level? In my opinion, we need to get testing people to be more familiar with testing at the development level. In the long run, I believe it is more effective to train testers to be technically adept at development-level testing and testing at all levels. But today, I feel that it is probably more practical to get developers to do more testing as part of their job. I expect the industry will move in both directions.

Theresa Lanowitz: People who are developing embedded and real-time systems often use homegrown tools because they have so many target platforms they must deploy to. These in-house tools are the biggest competitors for tools on the market, so these organizations need to be shown that commercial tools are every bit as good, if not better, than the tools they can create themselves. And they need to understand that going with a commercial offering is a whole lot easier than committing people to maintaining and writing their own internal tools. Rational has a whole suite of embedded and real-time tools that can help with every phase of the software development lifecycle; it would cost an organization a lot of time and resources to develop and maintain a comparable toolset on its own.

I believe that all enterprises need to replicate the quality expectations of organizations that build embedded and real-time systems. Telecommunications, aerospace, transportation, medical, and automotive all have embedded and real-time systems, and all have been producing high-quality products. Now, other enterprises have to be able to emulate that model because complex technologies like Web Services demand it.

The people creating real-time and embedded systems are typically very disciplined and know they have to produce quality products. They infuse quality from the top down because they're operating with an underlying philosophy and methodology. In contrast to other organizations that get caught up in process, real-time and embedded organizations never get hung up on any of that. If anything, many of them are only looking for ways to fine-tune their process because they are very disciplined and very mature because the systems they are building are life-critical, not just mission-critical.

Sam Guckenheimer: Rational Test RealTime is the market leader in the embedded and real-time systems area, according to Venture Development Corporation. So we have some experience in this kind of testing. Basically, we identify three challenges in this area:

The first is the criticality of the software. Everything that is true for cost-of-failure in e-business is amplified when you're dealing with embedded and real-time. The cost of failure is enormous. The liability is enormous. The implicit reliability requirements are enormous. You cannot reboot a telephone switch that is serving hundreds or thousands of subscribers. You cannot replace a piece of hardware the way you can offer a new download from the Web.

The second challenge is that the footprint issues with embedded are always a factor. You don't have the luxury of always getting another 512 MB of RAM if you need it. You're stuck with what you've already got, and it is usually very constrained. It means you need to design for testability much more carefully and not take things for granted.

The third challenge is that the opportunity for the tester to see what is going on is usually very constrained. Typically testers are not able to work on the device itself and have only a narrow communication pipe to it.

Also, in the embedded and real-time market you are typically running on a RTOS (real-time operating system) with a cross-compiler and chip, and the market is very fragmented. Any particular RTOS-compiler-processor combination will account for only a small fragment of the market. In the past, what prevented the growth of a robust tools market was the fact that it was too expensive to support that huge variety of targets.

Today, Rational Test RealTime addresses those issues. It is built to allow testing up to DO-178B, the FAA's standard for air-worthiness in software. The footprint has been optimized for minimum overhead on the target device. And we solved the problem of limited observation opportunity by tying testing closely to runtime analysis – for memory, performance, code-coverage, and trace. We do this with an extremely portable target deployment package that allows us to support virtually any combination of RTOS, compiler, and chip. And we can support new combinations through field enhancements, using an XML document to describe the target and specify how the test instrumentation and observation is done. So Rational Test RealTime is addressing these challenges in a way that no other product does.

SKILLS FOR THE FUTURE

Is there one thing that testers or testing teams are not doing but *should* be doing to be successful in the next two to three years?

Sam Guckenheimer: If there is one thing that testers need to learn about, it is modeling. Absolutely. The use of models for visualizing the software under test, communicating across the team, and documenting the design and the results of tests is *the* thing that is going to make an enormous difference in testing over the next few years.

Theresa Lanowitz: In the next two to three years, testers certainly need to be more technical. And that means that vendors cannot be afraid to take messages about the power and capability of their tools straight to the users.
Testers really need to represent their constituents, to be the advocate for the customer. This means working across artificial boundaries within the organization because "customers" can assume many different shapes.
Testers also need to be unbiased. They cannot be caught up in an organization's internal, cultural politics. You can argue that testers should be outsourced, which will give you good requirements, good use cases, and a bias-free perspective. But you do need to have software quality professionals within your internal testing organization, to keep a core group of institutional knowledge. That's what makes an organization strong – having that same core group of people who understand how things work and how things should be done from release to release.

Also, if you require more accountability from the test organization, you empower them. When you make testers stakeholders, rather than thinking of testing as an ancillary function, that is a positive step. In a commercial software organization, once a general release candidate is ready to go, everyone has to sign off on it – from the product manager down to the development, test, and sales managers. This gives the testers more power, because they can say "No. Based upon the quality of this application right now, it is not ready to go." And that will give them respect within the organization as well.

Finally, there's a simple practice that I recommend to clients, and that is to have cross-functional team meetings. If you're working in a commercial software organization, you probably have these meetings once a week with the engineering manager, the QA manager, the product manager, the product-marketing manager, and a manager from sales. These meetings give each team insight into what the others have to do. So when the engineering manager walks in and says, "I think we have to slip the schedule by three weeks," it is really not a surprise. If everyone has been honest in these meetings, everyone gets insight into the roles and responsibilities in other departments, and everyone feels their pain; I think the test organization benefits greatly from that. It puts everybody on equal footing. So holding these meetings is a really simple but really powerful thing to do.

Hung Nguyen: I see underlying issues from three perspectives.

First, from a business perspective, testers need more visibility within the business. Testing must be recognized as an

groups should have their own budgets so that they will have higher visibility, and certainly, more clear-cut accountability.

Second, from a development perspective, testers need to collaborate with developers and figure out how to fill in the gap between what developers test and what testers test. They need to figure out how to work with developers to make testing an integrated process. For example, agile development and extreme programming are great development approaches, and I am very excited about these cultural changes. However, I see that there still is a huge gap because these approaches focus on developers and not testers. The real challenge is using an integrated process that gets testers and developers working together to figure out what testability is all about. It requires real leadership and a combination of development expertise, testing expertise, and business management expertise. That is the right direction to move toward.

And finally, from the tester's perspective, we need to focus more on better education. In particular, testers will benefit from better education in advanced test engineering, current technology, and trends. With all the new technology, developers often have an advantage in the learning process: They have the opportunity to use the technology first; and there are always tons of reference materials for developers. There are very few technical references available that speak to the testing folks. Technology training *has* to be a part of the testing and business budget to help testers move in that direction. Testers should ask what we can do to be a major contributor in this process. With any luck the economy will turn around soon, and with good leadership in the testing area, we can make good progress and good products over the next few years.

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**