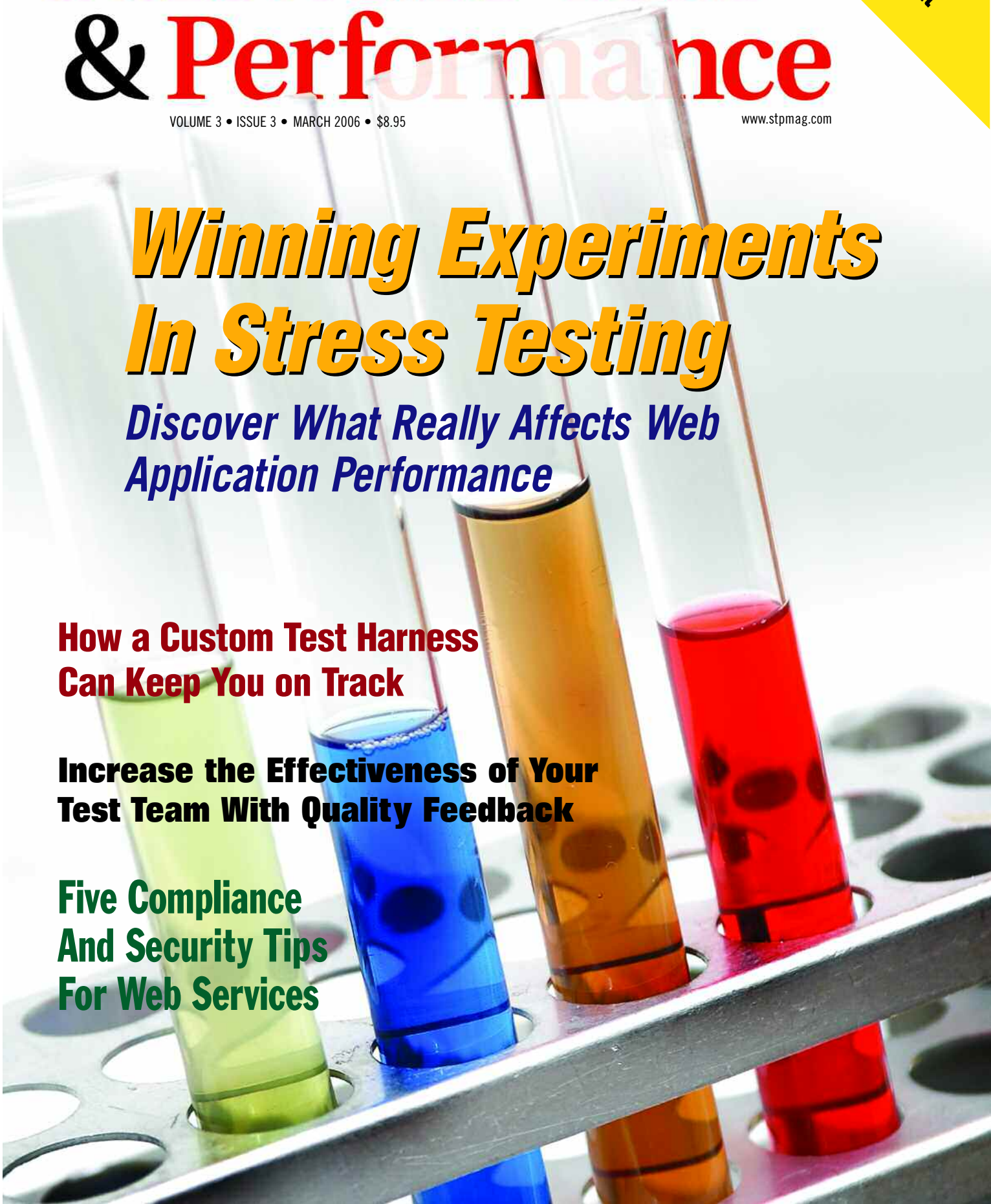
A BZ Media Publication

Software Test & Performance

BEST PRACTICES: Change Management

VOLUME 3 • ISSUE 3 • MARCH 2006 • $8.95

www.stpmag.com



Winning Experiments In Stress Testing

Discover What Really Affects Web Application Performance

How a Custom Test Harness Can Keep You on Track

Increase the Effectiveness of Your Test Team With Quality Feedback

Five Compliance And Security Tips For Web Services

Ensure Interoperability.

Validate functionality.

Eliminate security vulnerabilities.

Test performance & scalability.

Confirm compliance.

Collaborate and reuse.

Ensure Secure, Reliable Compliant Web Services

PARASOFT.

SOAtest™

As enterprises adopt Service Oriented Architectures (SOA) to deliver business critical data, ensuring the functionality and performance of Web services becomes crucial. Complex web services implementations require the means to thoroughly validate and test them to assure they are truly production ready.

Parasoft SOAtest is a comprehensive, automated tool suite for testing web services and complex Service Oriented Architecture (SOA) solutions to ensure they meet the reliability, security and performance demands of your business. SOAtest provides a total and holistic testing strategy for your SOA implementations including automated unit testing, graphical scenario testing, scriptless performance/load testing, security penetration testing, standards validation, message authentication, and more.

If you are building serious web services, you need SOAtest. For more information regarding Parasoft SOAtest, call 888-305-0041 (x-3501).

Download a copy of SOAtest for a free evaluation today at www.parasoft.com/STPmag

Parasoft SOAtest clients include: Yahoo!, Sabre Holdings, Lexis Nexis, IBM, Cisco & more.

PARASOFT.

We make software work.

Automated Error Prevention™

Eliminate the finger pointing with WebLOAD Analyzer™.

Want to accelerate resolution of Web performance problems?

You can with WebLOAD Analyzer—the only load testing tool with root cause analysis for .NET and Microsoft Web applications. WebLOAD Analyzer employs unique "black box" software to capture actual problems at multiple synchronized levels, enabling users to drill down to the individual component. By pinpointing the root cause of all kinds of application problems, WebLOAD Analyzer reduces problem resolution time by 60%—significantly improving application performance and eliminating finger pointing once and for all.

Get a FREE "No more finger pointing" T-shirt. > >

> > Learn more at www.radview.com/analyze

RADVIEW™

The Smart Choice in Web Application Testing

Software Test & Performance
VOLUME 3 • ISSUE 3 • MARCH 2006

BZ Media

President
Ted Bahr
Executive Vice President
Alan Zeichick

BZ Media LLC
7 High Street, Suite 407
Huntington, NY 11743
+1-631-421-4158
fax +1-631-421-4045
www.bzmedia.com
info@bzmedia.com

Software Test & Performance (ISSN #1548-3460, USPS #78) is published 12 times a year by BZ Media LLC, 7 High Street, Suite 407, Huntington, NY 11743. Periodicals privileges pending at Huntington, NY and additional offices.

POSTMASTER: Send address changes to BZ Media, 7 High Street, Suite 407, Huntington, NY 11743. Ride along is included.

The Human Factor in The Quality Equation

Lindsey Vereen
Editor

Sometimes free beer is just not enough. Once upon a time there was a high-tech company that held a party every Friday afternoon and provided free beer and chips for all its employees. I have to hand it to the company for at least making an effort, but long hours coupled with no profits to share in and low job security in this harried, perennial start-up were too much to overcome, and morale never rose very high despite the beer. And I'm not sure sending workers out to the parking lot with a buzz on at the end of the day is such a good idea anyway.

The gesture was a vain attempt to make things seem less dreary than they actually were, and it just didn't work. It did represent an understanding that the morale of employees had an impact on the productivity of the company. Quality depends to a great extent on the productivity of the people who are working to design and manufacture a product and assure its quality. I suspect that quality does not thrive well in an environment like the one depicted in David Mamet's play and film "Glengarry Glen Ross," in which salesmen are harassed and intimidated toward achieving their goals.

One of the reasons for implementing automated processes is to take human variability out of system manufacturing and test. I have read that the quality of an automobile once depended upon the day of its manufacture. Automobiles built on Fridays, as I recall, had a greater likelihood of being lemons than those coming off the assembly line on other days. As long as people are involved, quality is likely to vary. You can see that variability in athletic and musical performances as well as virtually every other human enterprise.

Companies propagate slogans such as "Quality is everybody's job" that sound like feel-good statements and may or may not mean anything at a particular company, but in a successful enterprise, quality is indeed everyone's business. And quality depends to some extent on such things as working relationships with managers and coworkers, fatigue, stress and ergonomics. Personal relationships with coworkers can also be a factor: hence the admonition against "fishing off the company pier."

You cannot underestimate the importance of the human factor in the quality equation. You probably wouldn't want to lie under the scalpel of a disgruntled brain surgeon or fly across the Atlantic with a flight crew mad at their boss.

In human relations oftentimes it is not what is done but how it is done. Companies have budgets to meet and stockholders to answer to, and workers may have issues independent of their jobs. But for the most part, everyone is rowing in the same direction; the question is, how best to facilitate the process.

Mutual respect goes a long way toward that end. In this issue Prakash Sodhani describes how the way personnel reviews are conducted can help boost the effectiveness of the QA team. It's just one way to improve product quality, and it's a better approach than handing out free beer.

Get More

Get 110% satisfaction with TestTrack Pro.

It's not often you feel 110% satisfied about a product. But, TestTrack Pro gives you more advanced issue management features, more performance, and more ease of use than any other solution on the market today.

- **More issue management** with defect tracking and change request capabilities, configurable workflows, extensive field customizations, and comprehensive reporting
- **More performance** with fast and secure remote access, scalable cross-platform client/server support, full Unicode support, and seamless integration with popular IDEs and SCM tools
- **More ease of use** with easy installation and administration, and uncomplicated licensing options

But wait... there's even more. With every Seapine solution, you get world-class support before and after the sale. In addition, every product is backed by our unconditional 30-day, money-back guarantee. Now it's your turn to get more. Download your **FREE** fully functional evaluation software now at **www.seapine.com/st&p** or call **1.888.683.6456**.

Seapine Software™
managing process, change, and quality throughout the enterprise

Software Test & Performance
VOLUME 3 • ISSUE 3 • MARCH 2006

Contents

A BZ Media Publication

Departments

CodePro AnalytiX™ is developed by the experts who brought you the popular book
Eclipse: Building Commercial Quality Plugins — Eric Clayberg & Dan Rubel

Unfair Advantage

CodePro AnalytiX™

for Eclipse, Rational® and WebSphere®

Download a risk-free trial copy:
www.instantiations.com/codepro

Drive out quality problems earlier in the development process.

Powerful code audits and metrics make every developer a Java expert.

Create tight, maintainable systems.

Wizards generate unit and regression tests, and analyze testing code coverage.

Enforce quality measures across teams.

Define, distribute, and enforce corporate coding standards and quality measures, across individuals and teams… no matter where they're located.

Spend less time and money to develop high-performance Java systems.

Dramatically improve your application development process and code quality… leading to more reliable, maintainable systems.

Key Features of CodePro AnalytiX™

- Detect & correct code quality issues... automatically
- Define, distribute & enforce quality standards across development teams
- 800+ audit rules and metrics, 350+ Quick Fixes
- Powerful management reporting
- Code metrics with drilldown & triggers
- Audit Java, JSP and XML files
- JUnit test case generation
- Code coverage analysis of test cases
- Dependency analysis & reporting
- Integrated team collaboration
- Javadoc analysis & repair
- Seamless integration with Eclipse, Rational® and WebSphere® Studio

instantiations
the development tools experts

www.instantiations.com 1-800-808-3737

No one outside of IBM has more experience creating Eclipse-based Java development tools

eclipse FOUNDATION MEMBER

IBM Business Partner

Compiled by Alex Handy

NetBeans Profiler Adds Swift Profiling

NetBeans Profiler is a newly created profiling tool for use within the NetBeans Java IDE. Sun Microsystems said that it created the profiler to address a number of issues that crop up during profiling, such as speed loss, information overload and complexity.

NetBeans Profiler is optimized to tackle a number of preset profiling tasks. Gregg Sporar, a technology evangelist at Sun, said, "Java profiling tools introduce too much overhead. As you introduce that overhead, you slow down the application. You might get into a situation where you can't even use specific profiling tools because the overhead they introduce makes them unusable. The second issue is the inability to do an appropriate level of filtering. You end up with an overflow of information. If I am trying to find a particular problem with memory allocations, I don't want this tidal wave of info on every thread."

NetBeans Profiler works within the recently released NetBeans 5.0 IDE and is available now from Sun's Developer Network Web site (developers.sun.com).

Shunra VE 4.0 Lets Testers Push Apps to Their Limits

Emulation is often associated with old hardware. The ability to run software under emulation has been around for years. Apple's Intel-to-PowerPC software Rosetta is one common example. But when it comes to emulating a network, the waters have been murky—that is, until Shunra Software (www.shunra.com) decided to offer its Virtual Enterprise network emulation software. The company released version 4.0 of this software in February, and the tools included with the suite should make testing SOA deployments, network services and high bandwidth software much easier.

Shunra VE 4.0 emulates a network environment for accelerated testing.

Shunra Virtual Enterprise version 4.0 includes a number of tools that help testers get a better handle on how their applications will behave under stress. The VE Profiler gives testers the power to automate their network application tests, and can be ratcheted up to push software to its absolute limits. To measure just how much all this stress is affecting the end user, Shunra also includes the VE Predictor, which gives testers a firsthand view of just how the tested application will behave when it reaches the end user.

And to make all the information gleaned that much more legible, version 4.0 of the Shunra Virtual Enterprise software includes a new reporting center. Here, users can correlate and compare data from the myriad tests available. You also can use the reporting center to get access to the information that will eventually make its way into the 8x10 color glossy images of charts and bar graphs that make management so happy.

Shunra Virtual Enterprise is available now for US$70,000.

Perforce 2005.2 Enables Offline Coding

The content and source code management and control suite from Perforce Software (www.perforce.com) is typically updated twice per year. The Alameda, Calif.-based company was slightly late in releasing its second update of 2005, however, and it arrived shortly after the new year.

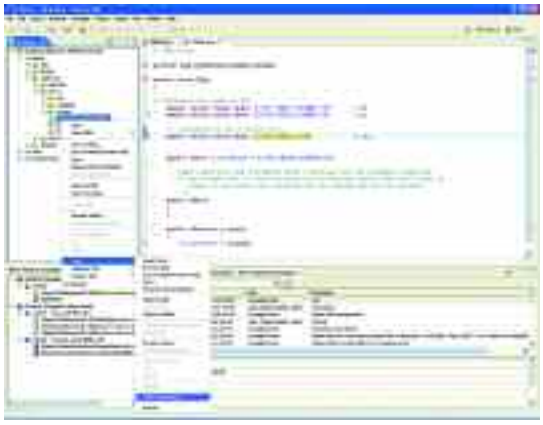
Perforce has a reputation for remarkably solid release builds, and 2005.2 is no different.

Featured in this release are external authentication triggers, server improvements and more. New to this edition is the ability to code offline, and then later upload additions to the central Perforce server.

Integration with Eclipse, the popular open-source IDE, has also been added this time around.

With the addition of offline coding, it was also necessary to add file and folder diff capabilities. These can be taken advantage of by just about anyone in the coding world—whether they use Mac, Windows, Linux or BSD.

This new version of Perforce is a free upgrade for subscribers.



Perforce allows users to leave the office to work on their code and upload additions to the server later.

Out of the Box

JUnit 4 Rolls Out Several High-Profile Improvements

After almost three years of silence, a new version of JUnit (www.junit.org) has finally surfaced for the unit-testing masses to enjoy. And what a triplet of years it's been for Java. With Java 5 now running around the world, JUnit has had to play a bit of catch-up to take advantage of all those new capabilities, such as annotations, generics and variable length argument lists.

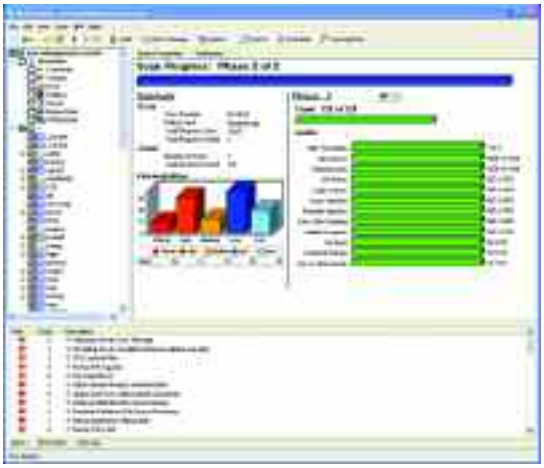
Since unit testing is a developer-centric testing method, you may find yourself having to play evangelist around the office to get your programmers to pick up their end of the bargain here. But with JUnit 4 in hand, that should be somewhat easier, thanks to the new additions to this indispensable tool.

First off, your old tests should function perfectly within this new version. The JUnit staff made certain to let everyone know that 99 percent of old tests should be compatible with this new version.

Also new to version 4 is the ability to set up and tear down tasks immediately before or after an operation. Thus, coders can reference a database connection before tests begin, and tear it down after all tests are finished. This should save time and resources for developers using expensive resources.

There are a number of other high-profile improvements to JUnit 4 as well. Examples of the improvements include exception testing, the ability to ignore tests with a new @ignore annotation, and improved timed testing.

SPI Dynamics Offers AJAX Tester



WebInspect offers a comprehensive tool set for testing AJAX applications.

Security testing tools are essential for Web applications, but it's gotten quite difficult to automate Web testing without the help of more robust QA tools. SPI Dynamics (www.spidynamics.com) has heard the cries of weary testers, and has released version 5.8 of WebInspect.

The tool offers support for automated security assessments of Web applications that use AJAX (Asynchronous JavaScript and XML) software development technology, according to the company. Erik Peterson, vice president of product management, said that "AJAX represents the future of Web application technology. SPI Dynamics says that by the end of 2006, 30 percent of all Web applications will be AJAX-based. Applications like Google Maps represent a new challenge for Web application professionals who have to ensure their security. It's a new technology that presents new security concepts."

WebInspect 5.8 wasn't the only thing SPI Dynamics offered in January. The company also announced its scalable Web application testing platform, Amp 2.0. The tool is said to allow security testers to expand, scale and make remotely administrable Web tests created with WebInspect. The tool also is available now, though the concept of a remotely accessible security tester is somewhat antithetical to proper security practices.

WebInspect 5.8 is available now starting at US$6,000, while Amp 2.0 starts at $60,000.

Compuware Updates Security Tool

DevPartner SecurityChecker 2.0 was released by Compuware (www.compuware.com) just before the end of January. This new iteration of the Web application vulnerability scanner includes a host of audits to find holes before they're exploited by malicious hackers.

The Web makes application security a much more complex issue for enterprises. Thanks to hacking Web sites hosted around the globe, one simple hole in your Web application can be taken advantage of by hundreds, even thousands, of unskilled mouse-based warriors. One particular site, known as Google Dorks (johnny.ihackstuff.com) lists hundreds of exploitable and vulnerable Web pages, with more added every day. The gist of this particular site is that security holes can be easily found by plugging magic phrases and snippets into Google.

But Compuware's DevPartner SecurityChecker 2.0 is designed to help prevent these types of problems from plaguing your Web applications. The tool integrates with Visual Studio 2005 to ensure that ASP.NET applications are quickly and effectively checked for security during development.

DevPartner SecurityChecker 2.0 is capable of finding pages within an ASP.NET application that could be vulnerable to Google hacking. That means it keeps an eye out for pages that contain too much information, illegal actions, faulty permissions, password listings and other data that shouldn't fall into the hands of hackers.

In addition, Compuware's updated tool is capable of finding cross-site scripting vulnerabilities and cookie flaws that could give users access to areas of a site in which they don't belong.

DevPartner SecurityChecker 2.0 is available now for US$12,000 per concurrent user. Volume discounts also are available.

Send product announcements to stpnews@bzmedia.com

The Shoulds And Shouldn'ts Of Software Testing


Scott Barber

Lately I've become rather fond of saying that *should* and *shouldn't* are the two most frightening words a software tester hears while testing. They probably aren't far down the list of words that frighten managers during status updates either.

Consider this example. While doing some performance testing for a bank, I noticed what looked like a fascinating pattern in the results with respect to response times. After applying some statistically flimsy grouping (by rounding the response times of the measured objects *down* to the nearest whole second), I confirmed that the pattern was, in fact, quite fascinating.

Reasonable Consistency

Response time results generally follow one of just a few patterns. An individual transaction will typically display response times that are reasonably consistent throughout the test, that increase linearly or geometrically throughout the test, or that increase during peak volumes and return to low-volume response times as the volume decreases. Pretty much any other response time pattern for a particular transaction is a good indication that something worth investigating further is going on. In this case, the response times seemed to oscillate unpredictably in four-second increments.

As I investigated further, I found several items worthy of note.

First, more than a third of the measured objects returned in less than four seconds. As it turned out, these objects were all graphics, style sheets and plain HTML (text) objects. Second, the majority of the remaining objects displayed response times of four seconds—with a few just over five seconds. Third, all of the objects returning in four or more seconds contained data that they retrieved from a remote, hosted database. Last of all, every one of these remaining objects returned in "four-and-a-little" seconds, "eight-and-a-little" seconds and so on.

Pinpointing the Problem

As I explained this behavior to the team (after defending my test, test tool, analysis and competence), the developers finally agreed to instrument their code and systems to time stamp each transaction as it entered and departed their areas of responsibility. Running the test again, all of them were happy to report that their code/system was responding in a quarter of a second or less—all except the middleware developer. He reported that the requests were spending under a tenth of a second in the middleware, but that the time difference between the outbound requests to the remote database and the responses was, in fact, showing the same four-second step pattern.

The overwhelming reaction to those findings by the managers and developers was, "Oh, it must be the database"—not surprising since the main reason for the project in the first place was to ultimately replace the outsourced database. I wasn't as immediately convinced, however, and asked, "Are we sure it's not something in the network? The database is two firewalls, probably upward toward a dozen switches, hubs and proxies, and over a thousand miles away." Once the team finished gawking at me and softly chuckling to one another, their response was, "That shouldn't be the problem; we've never had trouble with the network before."

It took us three weeks to verify conclusively that the database was not the problem, two more weeks to convince the network administrators to trace the packets, one hour to conduct the test, 15 minutes to analyze the results, and less than five minutes to fix the offending setting on the recently upgraded network appliance. So, basically, the word *shouldn't* cost the project five weeks.

Of course, the hardest part for me was biting back the urge to say, "I told you so!"

Sage Advice

In his book "More Secrets of Consulting: the Consultant's Tool Kit" (Dorset House Publishing, 2001), Jerry Weinberg offers memorable rules and principles based on his years of experience in the software industry.

For those of you who may not be aware, Weinberg has authored and coauthored more books than I care to count—virtually all related to his 40 years of experience in the software industry.

While it is true that many of his lessons come from the point of view of a consultant, every one of his books holds valuable information that is applicable to anyone in the software industry (and most any other industry for that matter). So don't think that his work is interesting only to consultants. Weinberg's books and his lessons that others have shared with me have had, and continue to have, a

Scott Barber is the CTO at PerfTestPlus. His specialty is context-driven performance testing and analysis for distributed multi-user systems. Contact him at sbarber@perftestplus.com.

significant influence on how I approach software testing, my writing and my values and principles as a consultant. If you've never read any of his work, I highly recommend it.

If only Weinberg had gotten this book out a year sooner, and we had read and paid attention to Chapter 3, we might have avoided our five-week delay.

The first three key points in Chapter 3 are: "If they're *absolutely sure* it's not there, it's probably there," "Don't bother looking where everyone is pointing," and "Whenever you believe that a subject has nothing for you, it probably has something for you."

A mere two pages later, Weinberg introduces a concept he calls "lullaby words" that he summarizes this way:

"Later, I reflected on the deeper lesson underlying our discovery of all these lullaby words. In effect, the words discourage feedback by putting both the speaker's and the listener's mind to sleep. When feedback is discouraged, the meaning of a statement cannot be clarified. If it's not clarified, the statement can mean almost anything—and that's always the beginning of trouble. If you want to avoid such trouble, start converting those lullaby words to alarm words—words that wake you up to potential misunderstanding, rather than lulling you to sleep. Just do it!"

I'm guessing that you won't be surprised that Weinberg includes *should* as one of his lullaby words; the other six are *just, soon, very, only, anything* and *all*. While Weinberg's insights are both brilliant and useful (as usual), I have to say that I think Weinberg missed a valuable opportunity to write about why people use these lullaby words—or maybe Weinberg and I just have had some very different clients. What I have seen are members of the client's team who use lullaby words very intentionally as a method of all but begging the tester (or consultant) *not* to look too hard in a particular place.

It's a sad but true fact that team members often try to keep the testers off their turf and thus keep them from finding problems that will put their team under the microscope.

Sometimes this effort is an act of self-preservation brought on by organizations or managers who are particularly harsh when critical defects are traced back to an individual; sometimes it's a side effect of nontechnical testers trying to tell developers how to do their jobs; and other times it's simply organizational politics.

'Whenever you believe that a subject has nothing for you, it probably has something for you.'

In this case, politics is exactly what led to a five-week delay that was kicked off by the use of the word *shouldn't*. I later found out, months after phase 1 (and my official involvement with the project) was completed and in production, that there had been a long history of contention between the development teams and the infrastructure teams in this organization, allowing the development teams to basically dictate tasks to the infrastructure team and blame them for any number of defects.

Making matters worse, a few months before I came on board, a new executive vice president who, among other things, made it exceptionally difficult for the development teams to get unscheduled assistance, was placed in charge of the infrastructure team. By all reports, many of the real and perceived infrastructure issues went away very shortly after the change in management, but the balance of power had taken a 180-degree turn.

The gawks, chuckles and "shouldn't" that I received when I asked about the network actually meant, "We really hope it's the database because we don't want to fight with the infrastructure team again...especially after they assured us that the network wouldn't be an issue for this project." Now, if only someone had told me that from the beginning, we could have created a fairly simple test to prove that the problem was between the middleware and the database, as opposed to spending weeks designing, creating, scheduling, executing and analyzing a test to prove the problem was the database, which would have quickly implicated the network by default.

In this particular case, at least, it was much easier to disprove the database theory of poor performance than to prove it.

All in all, after the network fix, the remainder of the performance testing effort went well, and last I heard, the application was still performing well.

Look Where You Least Expect

But that isn't the big lesson. The big lesson that I've had to relearn time and time again since this project, which is the first encounter I had with lullaby words that I can recall, was that no matter the reason behind the use of *should* or *shouldn't*, as a tester, there is only one single response that makes sense every time: "I understand that shouldn't be the problem, but let's test it really quickly just to make sure. If nothing else, it's worth being able to document the test and conclusively rule it out."

So now whenever someone says, "This shouldn't be the problem" or "This should work," what I hear is: "Make sure *this* is something you don't forget to test."

SORRY.

We know how much you enjoyed the late nights...

You've been there. Working the long hours to make a release deadline or to resolve critical errors that surfaced post-release.

Wouldn't it be great if you could eliminate the late-night grind by adopting a more effective approach to application quality? You can. With Software Quality Optimization™ (SQO™) from Segue.

Segue's quality platform — SilkCentral™ — provides visibility into application quality throughout the entire software application lifecycle, from requirements tracking to regression and performance testing to ongoing production monitoring.

Make late nights at the office a distant memory. Contact Segue today — call **800.287.1329** or visit **www.segue.com/lh?sorry**

segue®

You Don't Have to Be Mad to Experiment

A Stress Testing Lab Can Bring You Peace of Mind

Photograph by Maartje van Caspel

By Robert Sabourin

A major educational evaluation service provider had just experienced the worst possible problem– a complete system collapse during a live interactive exam. Luckily, a couple of astute technicians were able to install a fresh server and get the exam restarted within a "reasonable" delay of about two hours. But the damage was done. Why had the server failed when a couple of dozen applicants started using the system within a few milliseconds of one another?

Looking at the details, we found that 14 concurrent examination requests within a 20-millisecond window threw a wrench at the server, consuming more resources than were available and triggering a series of software failures because of inadequate failover or recovery software.

Somehow testing overlooked this eventuality. Even though load had been tested, some very important scenarios, such as varying the arrival patterns of transactions, were not considered.

What can we do to avoid such disasters? Who are you going to call? What

are you going to do? Stress testing of multitiered Web applications can be an expensive and time-consuming activity. Organizing stress-testing environments into three decoupled dimensions helps focus and organize projects effectively.

For the uninitiated, stress testing means observing the operational characteristics of the system in a harshly constrained environment. Whatever constrains the environment harshly is applying a stress to it. The operational characteristics being observed can vary. We could be looking at how the system performs, what the throughput of the system is, or what the response time of the system is to certain transactions. We could be looking at characteristics of the system, how it consumes resources, how much memory the system is using, how much CPU capacity the system is using or how much network bandwidth the system is using. We can also look at things like how many bugs there are in the system and whether there are failures in executing tasks or functions. We can observe how basic system operations or different usage scenarios are impacted by stress.

We can perform stress-testing experiments to predict behavior, isolate bugs and to perform what-if experiments to see the impact of different technologies on system performance and availability.

The elements include transaction generators, load generators and system monitors. By separating these elements, a company can dramatically reduce testing costs using appropriate tools and technologies for each element individually—without the need for a one-size-fits all solution.

This article shows how such an environment can be set up and used effectively to stress test applications. I'll offer examples to illustrate how different performance characteristics can be measured, how bugs can be isolated, and how development alternatives can be explored without breaking the bank.

Robert Sabourin has more than 20 years of experience leading teams of software development professionals. An adjunct professor of software engineering at McGill University, he often speaks at conferences on software engineering, SQA, testing and management issues. You can reach him at rsabourin@amibug.com.

Such crises as the one I have described have occurred too many times. Even if companies have mature performance- and load-testing capabilities, they sometimes forget some of the basic what-if analysis that could have predicted such failures.

I will begin by looking at a Web application stress-testing environment as being composed of three critical elements: a transaction simulator, a load generator and a system monitor/console (see Figure 1). Each of these elements is often found in commercial, and sometimes quite expensive, performance- and load-testing tools. The instrumented load test configuration illustrated shows the relationship between these three critical elements.

The *transaction simulator* is a system that accurately simulates typical transactions of the system under test. Transactions are simulated one at a time, as opposed to many instances running in parallel. Generally, I reserve workstations that match closely to a typical user's workstation for each transaction simulator.

The purpose of the transaction simulator is to provide a means to observe how a typical transaction behaves while the system under test is operated under many different conditions. If the system under test is an online bookstore, then a typical transaction would be buying a book.

Generic client-based test automation tools can be used to implement the transaction simulator. Generally, I recommend getting the lowest-cost tool that can repeatedly simulate a typical transaction many times in sequence.

Transaction simulators should provide a capability of measuring the amount of time it takes to complete all or individual components of a transaction.

The transaction simulator may be implemented using manual testing. In this case, the tester would follow a typical scenario and take note of the time to complete all or part of the transaction using simple timers or a stop watch.

Generally I recommend using a commercial test automation tool for transaction simulation that required detailed timing data for all or part of a simulated transaction. Tools such as Mercury WinRunner, IBM Rational Robot or Segue Silk will do the job well for transaction simulation. Keep in mind that you usually need one basic license of the test automation tool for each workstation dedicated to transaction simulation. I would recommend using open-source and home-brewed tools to support the load-generation part of the test environment.

The load generator is a series of

FIG. 1: WEB APP STRESS-TESTING ENVIRONMENT

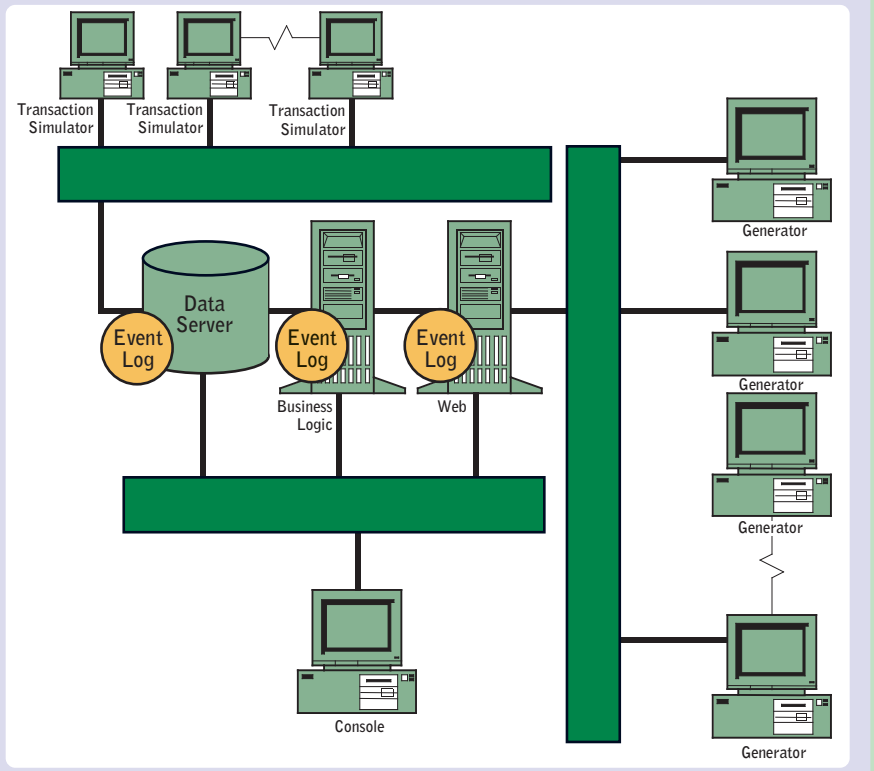
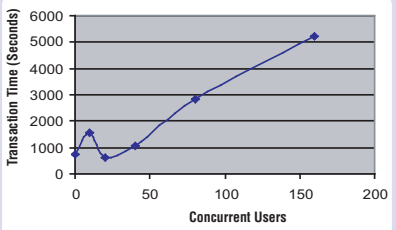


FIG. 2: TRANSACTION TIME VS. LOAD



computers dedicated to the job of generating a load on the system under test. The load generators are used to keep the system under test busy. Load generation should be varied based on the goals of any load test required.

Load-testing tools can be used to simulate typical user transactions in any mix required. But sometimes precise modeling of typical end-user transactions is not required to simulate a system load. At some of my customer sites, we run a series of custom-made applets or scripts on each load-generation system that invoke the system's business logic directly or call stored procedures to exercise the application data layer. The load generator's job is to keep the system under test busy.

Tools for load generation can vary dramatically in cost. Open-source tools such as OpenSTA or home-brewed scripts written in Perl or Ruby may be all you need to generate a very useful load to stress test the application.

Note that the network bandwidth should be sufficient to allow for the amount of data required during load generation. It is also important to be able to detect passed, failed or incomplete simulated transactions. This information may be extracted from log files or reported by testing tools. More sophisticated load statistics are available from commercial load generators.

The system monitor, or console, is a series of tools designed to study the system under test. While the system is experiencing a generated load, the system monitor allows for the study of resource usage and performance of the servers. Invariably, I use a combination of standard operating system tools, vendor-specific database management tools and some custom home-brewed scripts and filters to study system behavior during stress testing.

When servers are based on Microsoft Windows Server technology, an excellent tool to study system behavior is the standard Windows performance monitor tool *perfmon*.

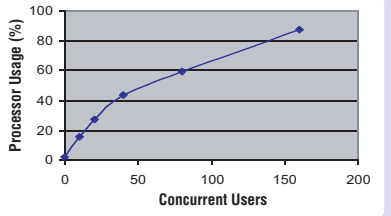
The *perfmon* tool can be used to record logs of almost any system resource under use, including memory used, processor capacity usage, file system access metrics, Web page access statistics and a myriad of database parameters including cache hit ratios and various timing measures.

When servers are based on Unix- or Linux-based technologies, several script-based tools such as *vmstat* can be used to gather system and resource usage information of the system under test during a stress-testing experiment.

Setting Up an Experiment

A Web system-under-test generally will include a series of services that manage the presentation layer, application logic layer and the data layer. Each tier of services may be implemented on one or more physical servers. I like to ensure that tools I use to monitor transactions and generate load are completely independent of the technology and software used to implement the Web system under test. On the other hand, it is common for the system monitor/console to be implemented in a technology-dependent manner. Test labs should avoid implementing load generators or transaction simulators that are tightly coupled with the applications service technologies. I want to make sure I can completely replace components of the system under test without requiring new load generation or transaction simulation tools.

FIG. 3: PROCESSOR USAGE VS. LOAD



It is also a good idea to implement the test environment so that the hardware and software used for load generation, transaction simulation and system monitoring can be reused across many different applications under test. This reuse is facilitated by managing and implementing the system console, load generator and transaction simulator completely independently.

Stress-testing experiments can be implemented using the environment I have described here. In a stress-testing experiment, we will be studying how the system-under-test behaves as we vary the environment and context in which the application is running. A common experiment is to study how the application behaves with different numbers of end users operating the system concurrently, but we also can vary how the application behaves in any harshly constrained environment. The key point is that we change the environment and then study the same basic characteristics using the transaction simulator and system monitor. The results of stress-testing experiments help us decide on design alternatives, provide information that may help us to determine if we can offer availability necessitated by service-level agreements, and help us to understand the behavior of the system under load from the end user's perspective.

Experiment 1: Traditional Performance Testing

A traditional performance testing experiment can take place by studying typical transaction time while varying the load on the system. In this case, the transactions simulated on load generators should be consistent with the expected behavior of users on a live system. The transaction simulator is set up with a separate workstation for each type of transaction being observed. Test data related to system resources and processor usage is collected at the system console. Test data is collected at the transaction simulator related to the time to complete a typical transaction as a function of load.

As you can see in Figure 2, transaction time increases linearly in proportion to load beyond 20 concurrent stressful users.

Note in Figure 3 that even with a load of 160 concurrent users (at a stressful workload), the processor is

FIG. 4: MEMORY USAGE VS. LOAD

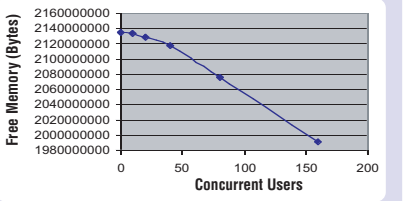
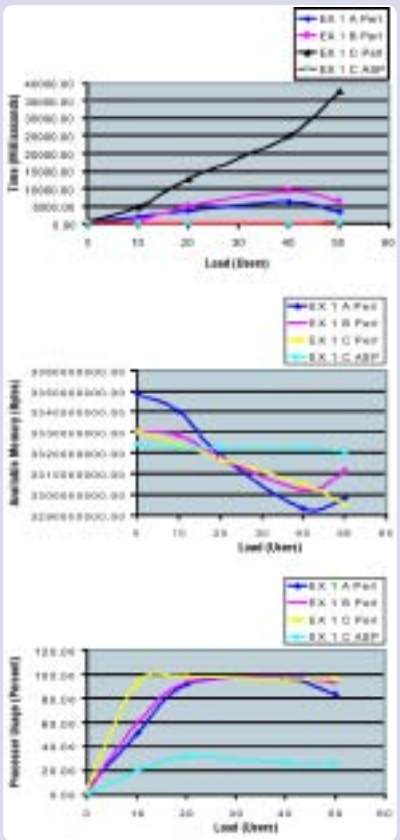


FIG. 5: EXPLORING DESIGN ALTERNATIVES



not saturated.

In Figure 4, we can see that the memory usage increases linearly in proportion with the number of concurrent system users from a load of 40 through to 160.

Experiment 2: What-If Analysis

This second experiment is an example of a series of stress-testing experiments designed to help a company decide among four different design alternatives for implementing data interfaces from applications running on a Microsoft Windows Server. Microsoft SQL and ActiveState Perl were used in the experiment.

Using the stress-testing environment described in this article, four major what-if experiments were done to study different design alternatives for implementing the business logic of a Web application under test.

Recently, a client used a stress-testing framework to explore design alternatives for implementing critical parts of its business logic. The technology was written in Perl running in a Windows NT server environment. The architecture team wanted to do a better job of understanding how different data communication design alternatives impacted performance and system resource usage. Design alternatives included accessing the database with the standard database interface module of Perl (DBI) using the ODBC (Open DataBase Connectivity Standard) protocol (Experiment 1 A Perl), alternatively using the Microsoft Windows Win32 API ODBC library (Experiment 1 B Perl) and using an OLE (Object Linking and Embedding) library or using Active X OLE/ADO library (Experiment 1 C Perl). The development team also wanted to explore the impact of switching from Perl to ASP technology to see if it may be worth considering a technology switch (Experiment 1 C ASP).

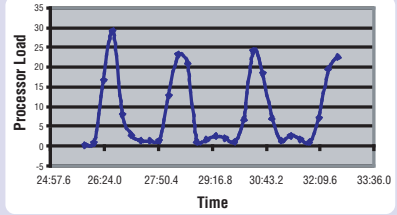
The three graphs in Figure 5 represent the result. It is interesting to note that the ASP alternatives had the fastest transaction time and lowest processor usage. Each point of data was an average of five different test samples.

Each stress-testing experiment was indicated using a different color line in the graphs in Figure 5. As a result of these experiments, performance and system resource impacts of different design alternatives can be considered. It is very important to note that each design alternative explored has many other risks and factors to consider before a decision is made to implement it. Keep in mind there may be different costs across each alternative as well as risks related to the code changes needed to implement them.

Experiment 3: Spike-Testing Analysis

The same stress-testing environment can be used to determine how a system will react to different load patterns. Using open-source tools and some Perl scripts to simulate the load, and running transaction simulation using a low-cost Web testing tool, the following stress test was performed. Exactly the same environment was used as indicated in the previous what-if analysis.

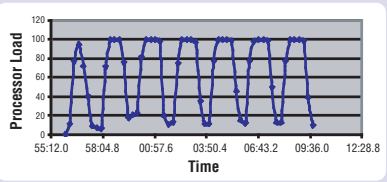
FIG. 6: 10 USERS SUBMITTING SIMULTANEOUSLY EVERY 2 MINUTES



The type of testing used here is often referred to as *spike* testing since we are forcing the system to have a lot of processing to do in a very short period of time thus causing a spike in processor usage.

The next two graphs (Figures 6 and 7) represent two data points of test results of between 1 and 80 concurrent submit operations in the application under test. Load is generated in different patterns for each experiment. The system console is used to track processor usage during the experiment. Developers are concerned about the percentage of CPU capacity used as different numbers of users submit requests at the same time.

FIG. 7: 80 USERS SUBMITTING SIMULTANEOUSLY EVERY 2 MINUTES

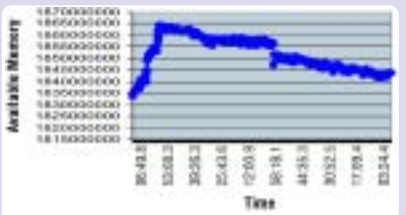


Processor saturation is observed at 80 concurrent users submitting exam sections at exactly the same time. The system successfully processes all transactions and then gracefully returns to a more typical usage of about 20 percent. No side effects or system failures occur.

The spike testing in this experiment demonstrated that with loads of 80 and more concurrent simultaneous submits, the system processor was saturated causing the width of the pulses on the CPU usage graph to grow wider. If we increase the load further, the pulses would flatten and get still wider. Once the width of the pulse exceeds the time between pulses, we observe there is not enough processing capacity to handle the load.

With spike testing, we can study how the system reacts to different numbers of concurrent user submits while we vary time between concurrent submits. This is analogous to stimulating a system with a pulse varying both the amplitude (number of concurrent users) and the frequency of these concurrent submits. Studying the response of the system to this stimulus gives insights into the behavior of processes and use of resources. (Some people may call these graphs response

FIG. 8: MEMORY USAGE OVER TIME



curves.) I often study response curves to try to identify the impact of amplitude and frequency to resource usage and response times.

An incredible advantage I gain using this approach is that I can study the behavior of a system without the need for an accurate usage model. What is the spike response of a system? I consider these curves a bit like signatures that characterize aspects of the system. You do not need perfect usage models to generate the load required to study the spike response of a system. In this experiment, I used a blend of open-source (OpenSTA) tools and low-cost Web-testing tools (Evalid from Software Research) to generate the load. Operating system tools (like the Windows performance monitor tool *perfmon*, script-based tools like *vmstat* and spreadsheet programs like Excel) were used to collect and analyze the results.

Experiment 4: Bug Isolation

Sometimes a stress-testing environment can be very useful in bug isolation. When developers and testers have trouble systematically reproducing a bug, tremendous costs are often incurred trying to identify the root cause. I have often used the same stress-testing environment to study the system under testing. The following experiment illustrates a chart of memory usage over time during a load-testing experiment. In this case, a constant load was generated over a long period of time. Memory usage data was collected in an attempt to identify a memory leak that shows up only after prolonged system use during normal operation.

We observe in Figure 8 that at the start of the test session, some memory is being freed up. This may be due to server garbage collection activities or the completion of system processes that may be unrelated to the stress-testing experiment.

We observe a decline in available free memory as the test continues through the night. In this case, the rate of decline was computed to be around 40MB per day. This is a similar curve to that observed during live operation.

Having reproduced the nasty memory leak, a new series of experiments was identified to repeat the situation monitoring specific memory usages for active processes during the testing period.

Memory Used by the Perl Runtime Environment

Figure 9 shows the memory usage of the Perl runtime environment, which was suspected as the root cause of the problem—but clearly the memory used was consumed and then restored during the course of the stress-testing experiment. This was considered to be a well-behaved environment.

Figure 10 shows that the SQL Server process did not release memory resources during the course of the stress-testing experiment.

As observed during stress testing, SQL Server continues to consume memory at a constant rate. In this case the rate is 40MB per day.

The technical staff at Microsoft confirmed that the memory behavior of SQL Server was as expected during the test period. These results confirmed the source of the memory leak and were used by development and network support staff to identify several ways to tune system performance and resource usage.

Other Sources of Stress

When the stress-testing environment is established, a few other types of experiments may be useful depending on your technical and business needs. Here is a short summary of some stress tests that I have frequently implemented in the same environment. Some of these stress-testing ideas might help you.

Forced Errors: Study system behavior when you force or simulate intermittent failures of processes or servers.

Data Sets: Investigate how performance is affected with different sets of

FIG. 9: MEMORY USAGE OF THE PERL RUNTIME ENVIRONMENT

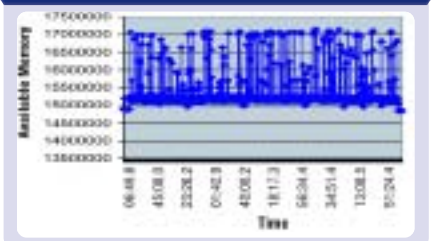
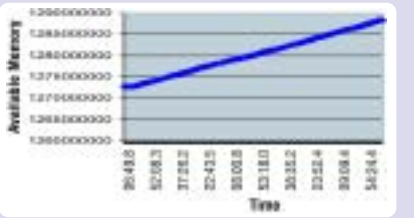


FIG. 10: SQL SERVER MEMORY CONSUMPTION



data. Explore what happens if the data set is nearly full, nearly empty or configured with different database cache settings.

Resource Hogs: Run programs on the servers that consume system resources in parallel to the normal running processes. Consume processing capacity, memory and other system resources, and then observe how the application behaves.

Shared Processes: Create applications that call processes used by the application under test. Stored procedures can be run to create and remove records from the database in parallel with normal application processing. Gain insights into how the application under test can perform if shared processes are used by other applications. Databases are shared by many applications other than the one being tested.

HTTP Attacks: Simulate denial-of-service attacks while the system is under normal load conditions. Study the impact on performance.

Network Bandwidth: Study system behavior as the amount of network bandwidth is reduced.

You Can Do It!

A cost-effective way to perform stress testing can be achieved with a versatile test environment. In order to keep the costs down and be able to perform many different types of stress tests, keep in mind that it is important to separate the load generation from the transaction simulation and the system console elements of the test environment. I recommend combining open-source and home-brewed technologies for load generation with commercial tools for transaction simulation. I also recommend custom-made scripts using system tools and log analyzers for the system monitor when possible.

Stress testing is not limited to studying system performance under load; you can also perform what-if analysis to study design alternatives, model different system usage patterns and help isolate tricky bugs.

Now on the East Coast!

June 5-7, 2006
Hyatt Regency | Baltimore, MD
www.S-3con.com

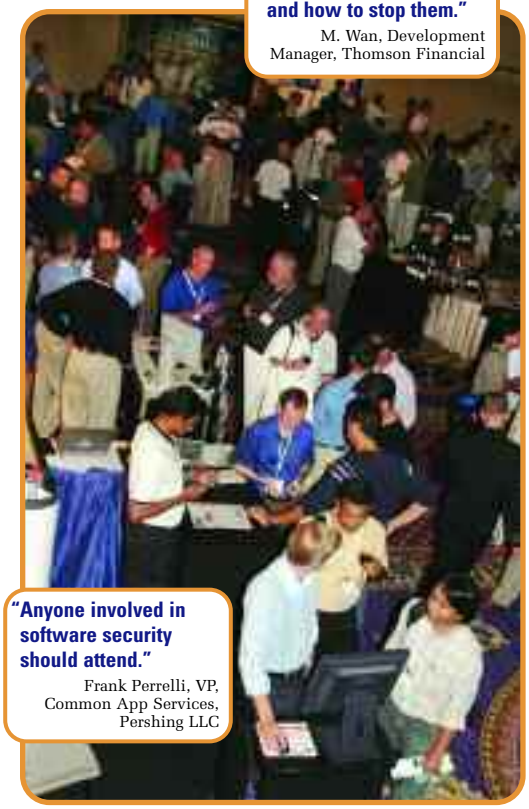
"Eye opening."
Amy Haselhorst, QA Engineer, Ceridian

SOFTWARE SECURITY SUMMIT EAST

The only technical conference focused on software security at the applications level

is coming to Baltimore!

"A good place to learn how hackers work and how to stop them."
M. Wan, Development Manager, Thomson Financial



"Anyone involved in software security should attend."
Frank Perrelli, VP, Common App Services, Pershing LLC

- Learn How to Build Secure Software
- Secure the Software on Your Network
- Test the Security of Your Software
- Implement a Layered Approach to Application Security
- Architect Security Into the Development Life Cycle
- Understand Software Security Vulnerabilities

"If you are interested in writing more secure software, go to this conference."
Richard Morris, Staff Firmware Engineer, Applied Biosystems

More Than 40 Classes & Tutorials!

"It's THE conference for developers and testers who are concerned with software security."
Wei Feng, Senior Architect, GrapeCity Inc. (Japan)

A BZ Media Event

www.S-3con.com

Hitch Up A Custom Test Harness To Your Next Project

By Ashwin Palaparthi

Many development shops write custom test drivers to automate some of their functional testing, which helps speed up the QA process and make it much more accurate.

Rein In Complex Test Flows And Plow Through Functional Testing

While there are numerous automation tools available, many factors, such as custom needs (for example, API-based tests), complex test flows and high cost, lead to the development of custom test harnesses. The purpose of this article is to describe the conditions in which custom test harnesses are created in the software testing space. While a variety of custom test harnesses are created depending upon specific needs, I will attempt to cover the common features and architectures of such creations.

If you are concerned that commercial test automation tools are not suitable for some of your test contexts, this article will answer questions such as: "What is a custom test harness and when should one create it?" "When there are so many testing tools available, why should one author a custom test harness?" and "What are the design considerations?"

Test harnesses are used in a wide variety of situations, but the major classification could be black-box-versus-white-box modes. In either case, there is a need for programmatic design and execution of the tests. This article deals with black-box custom test harnesses that perform specification/functionality-based testing. The white-box test harnesses are usually not custom created by each application development shop and are usually available as ready-made tools (for example, unit test frameworks such as xUnit; where x could be any programming language you may desire).

A Programmer's Solution

A custom test harness, as the name implies, is a custom tool that addresses and operates in a particular context. The "test harness" part of it could be defined as "a programmatic solution designed to exercise a variety of touchpoints/entry points in/of your system/application to

verify/validate the functional behavior in an automated fashion."

There are no boundaries to what a custom test harness can encompass, but it is very usual to exercise the programmatic external interfaces of a system and additionally validate the data stores—a dual approach to the system/application validation. Black-box custom test harnesses might appear analogous to white-box-based unit test driver programs, but they hugely differ in ways such as 1) not just checking the function return values but by validation at all the layers, and 2) sequencing multiple calls to form and test a user scenario rather than direct one-off function calls.

The Need

To make use of the functionality offered by a system or application, the interface exposed depends on the user of the functionality. For example, in the case of general human users, you would go with GUI applications that are essentially browser-based or rich clients, which can be automated using tools like IBM Rational's Robot and Mercury's WinRunner. In the case of the user being an external system by itself, you would go with application programming interfaces that link the functionality through programmatic functions/procedures. You would want to automate testing these APIs and also perform nontrivial validations by performing some logic in the back-end databases, doing such things as file comparisons. In a situation like this, we do not have a single centralized test controller application/product available to do what exactly we need, which leads to the need for a custom test harness.

The need described above increases when we have orchestrated scenarios with a mixed sequencing of GUI and API calls and intermediary file/database validations besides other programmatic validations. In scenarios like this, we would still need to build our custom test harness where it would additionally invoke the GUI automation scripts (written using GUI automation tools available in the market) partially and perform other custom steps.

The history of software testing strongly suggests that there is no single silver-bullet process for software testing. While test automation tools are getting increasingly commoditized with the availability of several options in the market, there are a lot of process-centric gaps that would induce someone to create a homegrown custom test harness. The simplest example would be to write an end-to-end smoke test driver that will perform basic but distributed validations on all layers of your application.

Philosophically speaking, if you want to program the logic of testing, as opposed to testing the logic/functionality of your application with predetermined test logic/methods, you would need a custom test harness.

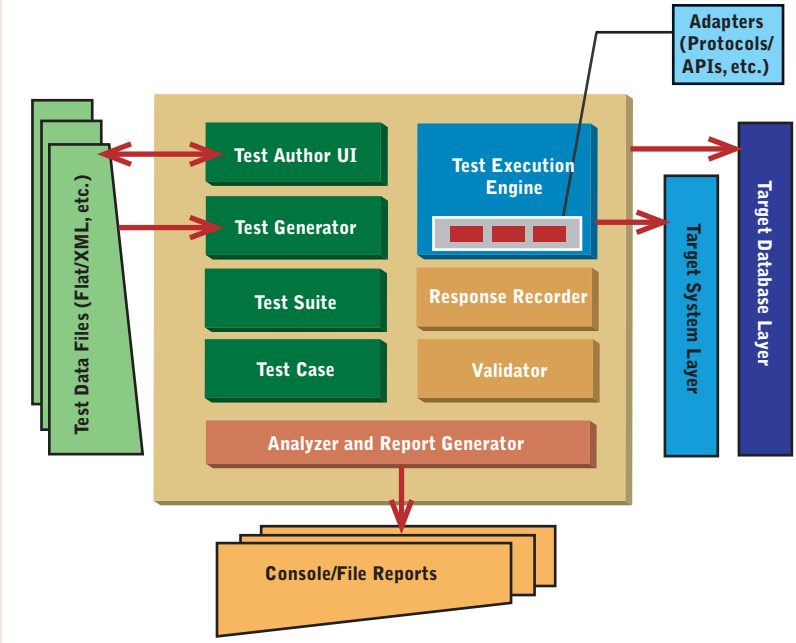
Common Features

While there could be several black-box custom test harnesses that are designed to serve different purposes, oftentimes there is a common set of features that are usually noticeable in many of them. Here are some examples:

- An intuitive GUI tool for authoring test cases and selecting tests that make a test suite. This becomes a mandatory feature in cases where the test engineers (that are the target audience of a custom test harness) cannot spend time on trivial and complex file-editing proce-

Ashwin Palaparthi is the principal architect of AppLabs, a global IT services company specializing in software testing and development. He has overseen the architectural aspects for many custom-test-harness-based QA projects. You can contact him at ashwin.palaparthi@applabs.com.

FIG. 1: COMMON CUSTOM TEST HARNESS ARCHITECTURE



dures to author test suites.

- A standardized way to specify setup and cleanup procedures. You would need a custom setup procedure to simulate the real-time environment and perform any other initialization needed by the tests.
- A magic engine that will parse the test files and create and run the test cases from a test suite (using the sequence and parameters as dictated by the input files).
- The ability to parameterize and invoke/spawn external programs for certain portions of the test.
- A set of validation features that include analyzing the output for expected (or unexpected) results, and performing database validations.
- Features to perform declarative analysis and diagnosis on the results in a custom fashion.
- A manager-friendly feature of result/failure reporting.
- A plug-in style architecture to bring in new adapters for new targets so that the rest of the tool is applied as-is in the new context.

Common Architecture

One of the essential things this article does not cover is the discussion about methodologies you could adopt in dealing with different segments of a custom test harness, which is a topic that would take up an entire separate article. For example, for test parameterization we could go with Category Partitioning, Markov Modeling and a few other methods. Regardless of the methodologies, there usually are a set of core elements that compose the architecture of a custom test harness (see Figure 1).

The components of the harness with their generic names/definitions could be as follows:

Test Author UI: Many custom test harnesses were created by the programmers who were themselves the original users. Many of these tools when passed on to the QA team were deemed not to be intuitive ones and hence discarded. The QA teams either resorted to manual testing or waited for the release of an off-the-shelf tool in the market. The fundamental principle here is that the purpose of a custom test harness is not just testing the target system but also providing intuitive usability features for the QA engineers. So, for editing any flat files such as CSV and XML files that need special editors, it is very important to abstract the editing methods into a nice and simple UI. This component should be able to construct test data files. For example, complex XML editing could be taken out by leveraging something like Xerlin (an open-source option).

Test Generator: Oftentimes it is not just the raw test data that is persisted in test data files—there will also be a lot of metadata that is stored. A programmatic parser would be needed to decode all those files and prepare programmatic objects. *Test Suite* is an ordered collection of *Test Case* components that represent the test targets, the parameters and the expected results. Again, these components should deal with generic placeholders more than hard-coded values. For example, Freemarker (again an open-source option) can be leveraged in replacing hard-coded values by templatized placeholders in XML files.

Test Execution Engine: This is the main controller component that performs all the commands as laid out by the *Test Suite* component and with the information generated by the *Test Generator* component. A key subcomponent of this component is the *Adapter*. An *Adapter* is needed for each type of command we perform. These could be wrappers that invoke the target APIs or *Protocol Adapters* that generate a specific type of message needed by the protocol in context. One or more *Adapters* could be used by the *Test Execution Engine*.

Response Recorder: This logs all the results for a later analysis, most of the time in a structured format, and the *Validator* at a minimum performs assertions on the response data. Validators usually perform additional functions such as database validation, file comparison and so on.

While architecting a custom test harness, enough care should be taken to ensure reusability. A situation in the future would be to move away from your custom tool and use an off-the-shelf tool that would have proven itself by then. In such cases, some of the code in your custom tool could be plugged into the new context if it follows the interface rules. The opposite situation could be one in which your custom tool remains to be the master controller and may invoke some off-the-shelf tools for certain portions of the tests. So, plug-in architecture and the facilitation of external program execution should be considered by the architect.

Advantages

There are many advantages to using a test harness. Some of them are as follows:

- Philosophically speaking, any method that is reusable has repro-

ducible behavior; this is why a custom test harness can be used to get a failure surfaced consistently; which would otherwise need evidence that something has (or has not) worked in a particular environment.

- Early adaptation of automation is possible on applications developed using new technologies. Testing tools are always after the fact, meaning the technologies that they target have to be used widely in the market. For example, two years ago automating applications developed using FIX protocol at the API level was not possible unless we authored a custom test harness. Today the market has started providing off-the-shelf tools in this area.
- Once in place, you can use a custom test harness to cut regression testing time and error rate while not having to pay huge dollar amounts to tool vendors. Added to this is the flexibility you have for modifications and enhancements.
- Companies that have built custom test harnesses over a period of time continue to use them even though the market has now released tools that relate to their context, as many of these tools do not make a highly appealing case.

A Set of Rules

Here are some rules of thumb for developing a custom test harness.

Never think of building a custom test harness if the market already has a tool for your context. You're not going to save money by building tools that are not core to your business. However, once the custom-test-harness path is chosen, due care must taken to strategize its usage.

When developing a custom test harness becomes necessary, plan an agile/rapid approach where you can quickly see it (or at least part of it) in action. Heavyweight waterfall approaches are too risky for lateral tools development that enables your core application/product development. Brief specifications derived from interviewing QA engineers on their needs and a brief design followed by fast implementation should yield better results for any custom test harness development.

The most important requisite that needs to be in place before attempting to write a test harness is that the actual designers/developers who author the harness need to understand the needs of the manual QA engineers.

It is very likely that the test harness will have a short life if it relies on the technical skills of people who use it. That suggests to us that we design it in such a way that it not only cuts the time and human intervention involved, but also offers its features in an intuitive way.

Use open-source tools to the greatest possible extent. Fifty percent or more of code that needs to be written in developing a custom test harness can be avoided by leveraging open-source tools such as Xerlin, FreeMarker, Regular Expression libraries, and STAF for distributed test control.

Architecturally speaking, your custom test harness should be assembled with object-oriented components that can be reused (or replaced). A purely monolithic scripting approach to custom test harnesses will not suffice, and in such cases you are better off writing some huge non-GUI-related complex test logic inside GUI automation tools.

As with any other automation tool, target only repeatable regression tests on stable features of your application and use other methods on the new features. Once test files have been created, we don't want our QA teams editing them. Our test suites should be open for extension and closed for modification.

If you are trying to do code-level white-box testing or functional-level "performance testing" through a custom test harness, a better alternative is open-source and commercial tools. A custom test harness makes sense only in a complex and unique functional testing context most of the time.

Not a Full Replacement

Factors such as automation of custom test methodologies and adaptation of automation on applications developed using newly born technologies create the scope for custom test harnesses. By no means are these custom tools competitors or full replacements for commercial test automation tools. You should build custom test harnesses only when you have a real need.

Building a custom test harness should never be a major project in itself, and the best approach is to author one rapidly when the need strikes and then let it evolve incrementally side by side with your other development efforts.

By Jack Quinnell

In a number of industries, businesses face pressure to comply with regulations that mandate data security and network integrity. Accounting scandals and well-

publicized security breaches have angered consumers and legislators. As a result, businesses are finding themselves compelled to put enforcement and monitoring processes in place for access control, financial reporting and permission marketing. Security regulations proliferate, and audits are becoming more frequent.

Meanwhile, software architects, IT managers and business managers are exploring new technologies, such as Web services and service-oriented architectures (SOAs). These technologies promise to increase business agility, accelerate time-to-market, and reduce IT and operational costs. An SOA solution replaces large, monolithic applications with configurations of smaller, reusable software components that are designed and quickly assembled to meet the specific needs of a business offering or service. While an SOA can be implemented without Web services, most SOA implementations today use Web services based on XML and HTTP. In large enterprises, Web services are most likely standards-compliant, SOAP-based Web services protected by technologies that comply with the OASIS WS-Security standard.

Are these two trends—increasing regulatory pressure and the migration to SOAs—related in any way? Do SOA security vulnerabilities make regulatory compliance more difficult? Or is the unfolding SOA revolution an opportunity for businesses to tighten controls over their business processes and online services?

If you're an IT manager, security professional, software developer or QA tester working with Web services and SOAs, here are five things you should know about compliance, security and Web services.

HOT TIP NUMBER 1: ***Because Web services are loosely coupled and granular, they provide a better infrastructure for protecting confidential data and securing business processes than traditional, application-centric security approaches.***

A West Coast financial services firm recently replaced its traditional LDAP-based identity management system with a new SOA-based identity management system because the company recognized that the security landscape is changing. New business models,

Jack Quinnell is CTO of Kenai Systems. He has more than 30 years' experience in developing and marketing voice and data network communications products. You can contact him at jquinnell@kenaisystems.com.

Photographs by Geoff Delderfield

Red Hot Tips

For Making Web Services Compliant and Secure

Internet applications and mobile computing have effectively eliminated the network perimeter. To provide comprehensive security that is robust enough to withstand increased regulatory scrutiny, every application and every information asset must be identity-enabled. The best way to implement identity management on this grander scale was to adopt an architecture based on end-to-end business processes. As the company put it, it needed "to get identity out of the apps, and into the business process."

The company replaced its LDAP directory and update scripts with a new set of internally developed Web services, including a metadirectory service that manages updates. This new SOA-based solution improved the coverage and the accuracy of the company's identity management functions. Since January 2004, the service has achieved 99.95 percent uptime.

For businesses such as financial services companies, telecommunications companies and health-care organizations that face increased regulatory scrutiny, the best approach for strengthening security and reducing regulatory risks is to use SOAs to build security and compliance into the business process. Designing solutions for end-to-end business processes, rather than for intermediary client/server transactions, ensures that data is always secure, wherever it happens to be in the course of a business transaction.

Financial services firms and similar organizations require a flexible software architecture, such as an SOA, in order to keep pace with growing lists of regulations.

According to IT analyst company RedMonk, "Leading with siloed applications may be adequate for initial, tactical compliance, but that approach introduces significant complexity and limitations over the longer term. The sheer variety and scope of compliance challenges require that IT organizations address compliance issues at an architectural level, using a fluid, adoptive approach. Organizations should deploy a

Growing Regulatory Pressure And The Migration To SOAs Present New Challenges

services-based architecture that can deliver compliance-specific services as necessary, based on specific acts and regulations."

Instead of addressing regulations in isolation, enterprises should deploy enterprisewide security capabilities that can be applied to meet the compliance requirements of each department and division. The same authentication capabilities can ensure compliance with both Sarbanes-Oxley and Gramm-Leach-Bliley. Enterprises do not need to invest in regulation-specific solutions; rather, they can invest once in security building blocks that can be deployed and redeployed as necessary to comply with whatever regulatory requirements are in force.

Creating these building blocks within an SOA makes sense, as the goal of an SOA is to deliver common, reusable services that can be applied wherever they are needed. The SOA model facilitates implementing broadly accessible components designed to ensure compliance in a cost-effective manner.

HOT TIP NUMBER 2:

The best way to ensure that Web services are secure and compliant is to build security and compliance into Web services components and verify security and compliance during the QA process, before the services are deployed.

Rather than relying on XML firewalls and other data center technologies to plug all the security holes in an SOA application, enterprises would do better to build security into their applications during the development phase. By systematically applying security policies and best practices, while testing for the presence of common vulnerabilities, enterprise IT departments can ensure that the Web services deployed in the data center are already reasonably secure and policy-compliant. Production-stage technologies, such as XML firewalls, simply become an additional layer of security, rather than the sole bulwark against all attacks.

This approach minimizes risks for four reasons. First, it minimizes the dependency upon firewalls and other defensive products; the software is protected, even if these products fail to block every intrusion or attack.

Second, it makes Web services running within the network perimeter less vulnerable to attack from insiders—users who might not be blocked by outward-facing defenses such as firewalls. Insiders are now responsible for 70 percent to 80 percent of attacks on enterprise networks.

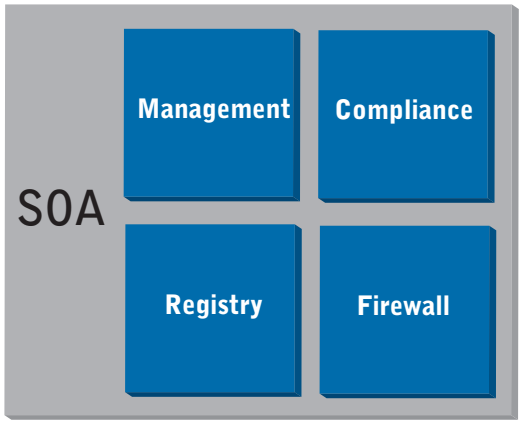
Third, if the Web service is going to be shared with partners, customers or other sites outside the control of the corporate IT department, built-in security minimizes exposures to risks caused by oversights and omissions in the data center security of those other organizations.

Finally, by incorporating security and compliance testing into the standard testing and release process, enterprises can make security a reliable feature of every Web services release.

Another advantage to this approach is cost. According to research firm Gartner, fixing a vulnerability in development costs only 2 percent of what it costs to fix that same vulnerability in production. That's a 50x saving for each vulnerability found.

To improve security while lowering costs, enterprises should test for security during development.

FIG. 1: KEY COMPONENTS OF AN SOA DEPLOYMENT



HOT TIP NUMBER 3:

To test for security and compliance, QA teams and development teams need an automated software solution with built-in policy knowledge.

Any enterprise interested in promoting a security and compliance testing practice within its IT organization is going to face several challenges that they can overcome through automation and knowledge sharing.

One challenge is that software developers and QA testers are not security experts, and enterprises cannot afford to train them to become security experts. Second, the security experts in the organization, such as the CSO, CISO and his or her security team, have no way of systematically transferring their knowledge of software vulnerabilities and security best practices to the development team. Security policies may be scattered across written documents and e-mail messages, but they are not encoded in a way in which they can be systematically applied and monitored in development and testing.

Enterprises need an automated framework that enables security experts to encode their knowledge in policies and rules that can be passed to development and QA teams in a reliable, usable format. This automated framework should take advantage of the domain knowledge of each group—security, QA and development—without requiring security experts to become developers or developers to become security experts.

Automation eliminates the errors and redundancies commonly found in manual processes. It scales to accommodate growing SOA implementations and supports growth that handwritten testing scripts would be unable to accommodate. By minimizing manual work, it ensures that lack of manual resources doesn't become an excuse for companies deferring the adoption of a

rigorous security and compliance testing methodology.

HOT TIP NUMBER 4:

A security and compliance testing solution should be able to assess the effectiveness of the security building blocks required to keep information confidential and secure.

While regulatory requirements vary from industry to industry and legislative act to legislative act, a few common security building blocks are common to just about all major industry regulations. For example, authentication, encryption and resilience to attack are capabilities common to almost any security solution to achieve compliance.

The Gramm-Leach-Bliley (GLB) Act compels financial institutions to implement security measures to protect the confidentiality of consumer data. At any financial institution, these security measures will include access control measures involving authentication, authorization and data encryption. Only users who present valid IDs should gain access to consumer data, and consumer data should be encrypted whenever it's exposed to public networks or other potentially hostile environments.

The Health Insurance Portability and Accountability Act (HIPAA) requires health-care organizations (HCOs), including payers such as insurance companies and providers such as hospitals, to protect patient data through a variety of security measures, including authentication and encryption.

GLB pertains only to financial services companies, and HIPAA pertains only to HCOs, but in each industry organizations are likely to turn to the same security technologies to achieve compliance. Top-down security policies, authentication controls for networks and applications, identity management and encryption can facilitate compliance in both these industries.

The success of compliance technology depends on its ability to implement and assure core building-block security measures. Compliance assessment for SOAs should include tools for testing these building blocks. Is consumer data encrypted when it traverses the network? If a specific business service requires authentication, are authentication measures being enforced? Does a portal withstand common attacks? Vulnerability assessment and policy compliance provide the visibility to help developers, QA testers and compliance officers answer questions like these.

HOT TIP NUMBER 5:

Once Web services are deployed in production, test for security and compliance again, using the same policies and automated test solution.

It's most cost-effective to test for security and compliance while Web services are still being developed. But to ensure that no new vulnerabilities are introduced by gaps in change management or on-the-fly configuration changes, enterprises also should regularly test Web services in the production stage.

The process used for development-stage testing should support production-stage testing as well. When a CSO or other security professional defines a security policy to follow during development, that same security policy should be applied in production too. Vulnerability profiles and other test metrics used in development should also be applied in production-stage testing.

Using the same testing solution for both development-stage testing and production-stage testing saves money by eliminating the need for an IT organization to invest in parallel but separate security and policy compliance testing solutions.

Administrators and compliance officers can discover vulnerabilities and exposures if they can compare test results across life-cycle stages. If security tests that passed during development suddenly fail in production, the IT department knows that a potentially serious change has occurred. They can then use the knowledge they gained from their development-stage testing to find and fix the problem quickly.

It's a mistake to assume that services deployed within the firewall are safe.

A Turning Point

We're at a turning point for Web services. Analyst firms agree that more than 75 percent of enterprises now have Web services projects under way. Web services have become essential business technology for companies like Amazon.com and eBay, and organizations such as banks are busy rewriting their online applications to run as Web services. We can expect to see Web services gain even more momentum in 2006. Microsoft chairman Bill Gates' recent endorsement of Web services business models is simply one more spark for the kindling.

We can also expect to see regulatory enforcement organizations, such as the SEC and the FTC, continue to scrutinize corporate operations and punish wrongdoers. Many regulations require organizations to monitor their own progress and to continually improve their performance. It's likely that in 2006 and 2007 auditors will look for tangible signs of improvement. Putting compliance and monitoring practices in place might suffice for year 1. But by year 2 or 3, it should be possible to measure progress. Auditors will be looking for reports and quantifiable results.

Enterprises can take advantage of SOAs while meeting regulatory challenges. At the same time, they need to recognize that the public interfaces and new technologies of SOAs pose threats to network security. Gartner estimates that Web services will reopen roughly 70 percent of the security holes that firewalls and other security products have closed over the past decade, so enterprises must build security and compliance into every Web services component they deploy.

Even Web services pilot projects should be built with security in mind. It's a mistake to assume that services deployed within the firewall are safe. The majority of security attacks now originate with insiders, including dis-

gruntled employees. Pilot projects need to be safe from these "trusted" users.

Pilot projects also provide an opportunity for enterprise IT teams to test and to develop the security and compliance mechanisms they will deploy along with their production Web services. Rather than assuming that mature, foolproof security and compliance mechanisms can be developed on the spot when Web services are ready to deploy, IT teams should design, implement and assess these mechanisms along with the Web services they are designed to protect.

What can developers do to lead in these efforts?

- If your development team isn't already exploring SOAs, start now. The technology is mature enough to offer real benefits. Survey results make it clear that your partners and customers will likely expect you to be delivering Web services soon.
- Work with your organization's CSO and security team to identify the common security requirements of the industry regulations that apply to your organization.
- Design and implement Web services that act as "security building blocks," providing these common security services, such as authentication, encryption and so on.
- Ensure that all new business services and SOA applications incorporate these "security building blocks."
- Invest in an automated testing framework that enables CSOs and security professionals to define policies (using assertions such as those in the WS-Policy standard), which can be automatically translated into test suites for use in development and QA.
- Make security assessment and compliance testing part of development. End users want to buy products and services that are secure out-of-the-box. The best way to create these products and services is to make security an intrinsic part of software design and development.

Through SOAs, developers have an opportunity to play an important role, not just in creating new applications for end users, but also in helping their organizations meet regulatory their requirements. It's an exciting opportunity and one that developers should not overlook.

In QA Staff Reviews, What Goes Around Comes Around

By Prakash Sodhani

Quality assurance is all about processes. It involves activities designed to maintain and improve processes in an organization. And just as quality assurance is about processes, so is maintaining and advancing the motivation and capabilities of members of the QA staff. The feedback involved in the staff evaluation process represents one such process.

Everyone has to go through this in one form or another. It gives QA staff members the opportunity not only to get feedback on their performance, but also to find out areas in which they can improve.

The process begins with a communication from a manager reminding staff members to complete a self-evaluation as part of a year-end appraisal process. Then, as the other half of the appraisal process, they are asked to schedule a meeting with their manager to discuss their performance. Almost everyone goes through a similar process. The obvious question is: "What do we get out of this process?" Is it just a formality so that a company looks better in terms of having a process in place, or is it just to determine how much of a raise and/or a bonus an employee gets based on the manager's rating? This article presents a few guidelines that, if followed in such feedback meetings, might help make these meetings more productive and successful than they would have been otherwise.

The Appraisal Loop

In the context of staff evaluations, a feedback cycle usually consists of a three-step process like the one depicted in Figure 1. First, staff members set a few goals to accomplish during some specific time period, generally a calendar year. These goals are normally aligned with each employee's future goals and aspirations. Next, they work toward achieving the goals during the set time period. Finally, they meet with their respective managers to discuss the progress at regular intervals. The manager gives them ratings during a final year-end appraisal meeting based on their performance during the year.

A feedback meeting should be well-structured and its goals properly stated. The feedback goals do not necessarily comprise only work-related goals; they can include a staff member's personal development goals as well. One of the important objectives of these meetings is to guide employees, through honest inputs, along the path each has envisioned for himself or herself.

From a manager's perspective, four objectives of this feedback process are to:

- Provide frank feedback on the staff member's performance.
- Instill confidence by praising his or her commitment to the team.
- Identify areas of improvement.
- Discuss both the positive and negative aspects of the staff member's performance.

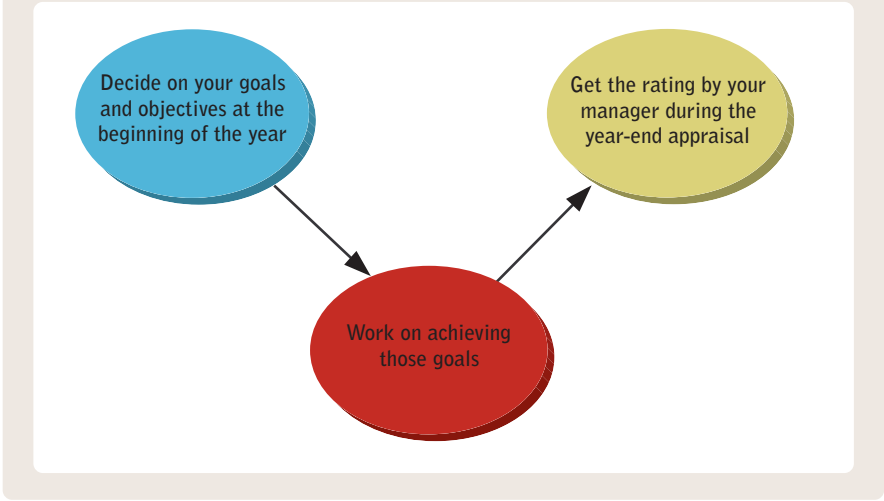
Feedback: Roles and Responsibilities

The two necessary participants in any feedback meeting are a manager and a staff member. The feedback process should not be a one-way street where a manager rates a staff member who merely listens to the input. Staff members are an important component of these meetings, and their participation, together with the participation of their managers, is one of the keys to achiev-

Prakash Sodhani is a quality control specialist who has worked in a number of IT organizations in different capacities as a quality professional. He can be reached at Prakash.Sodhani@gmail.com.

Photograph by Norman Chan

FIG. 1: A TYPICAL APPRAISAL LOOP



ing the goals set out for such meetings.

Let's discuss each participant's role in more detail. The manager's role is not limited to rating a staff member in feedback meetings. The feedback that managers have to offer is an important factor in accomplishing the objectives previously set for these meetings.

Here are some of desired goals that managers should try to achieve in this process. They should:

- Give feedback that helps in the overall development of staff members. Feedback such as "You are doing a great job" is not of much help.
- Be clear, concise and consistent with feedback. Managers should not just say what staff members want to hear or be too critical of their performance.
- Be ready with specific instances to illustrate their views. Just saying "Do it" will not be of much help. There should be a good justification behind everything they say.
- Always talk about the work and not about the individual. People might be willing to listen to criticism of their work but not to something that targets them personally. It's better to say, "Your last report needs a few modifications" than "You don't write reports well."
- Encourage staff members to ask questions. They need to be able to count on, rather than fear, their manager.

A staff member's role is often underestimated in feedback meetings. In these meetings, a manager rates someone who then willingly or unwillingly has to acknowledge that input. The role of the person being evaluated in these meetings is as important as that of the manager.

The following are some goals that staff members should try to achieve in such meetings. They should:

- Be willing to ask questions. It's better to ask than to remain in ignorance.
- Always ask what their manager thinks they need to work on. Some managers are too nice to mention such things. They prefer to just give feedback that sounds good. It's the staff member's responsibility to get this information by asking specific questions.
- Ask for specific instances where their manager thinks they should have done something better and where they made mistakes. This will help them to avoid repeating the mistakes in the future.
- Always mention that they want to learn and that the manager's feedback will go a long way in helping them achieve their goals. This will make it clear that their objective from this meeting is to get an honest opinion about them from management.

The Feedback Meetings

Typically, staff members should meet with their managers at frequent intervals to discuss their performance and get feedback.

Let's look at a typical conversation between a staff member, say Peter, and his manager, say Kate, during one of these feedback meetings:

"What do you think about my performance," Peter asks with intent to start the discussion.

"I think you are doing a great job, and I am really happy that you are a part of this team. Actually, I was talking to other members of the team and everyone was happy with your work," Kate replies.

"Thanks. Do you have any more comments," Peter asks with a big smile on his face.

"Not really. Keep up the good work. Let's just go over the appraisal forms and complete the formalities," Kate says.

Such a conversation raises a few questions:

- What was the outcome of such a meeting? Was the objective achieved?
- Did Kate's evaluation help Peter in any aspect?
- Did Peter do his part by not specifically asking for comments that might have been more helpful to improve his performance?

Feedback Dissected

Feedback can essentially be of two types: positive or negative. While positive feedback is crucial to let people know about their areas of strength, negative or critical feedback complements it by identifying areas in which improvement is needed.

Both types of feedback are essential for an employee's overall development. A feedback meeting is incomplete unless both positive and negative feedback have been offered.

Everyone likes to hear positive comments about themselves. Let's look at an example from Kate about Peter:

"Peter did a wonderful job as a test engineer during the past year. He completed his work on time and found good favor among the team members. He was also very punctual and handled his responsibilities well. His work finally led to a successful interim release of this project."

Let's look at the feedback above closely.

What does it achieve, and what does it miss? Feedback like this lets Peter know that his manager, Kate, is happy with his work. Peter, possibly, thinks that Kate wants him to continue doing what he has done till now. In all probability, he expects that he is going to get a raise and possibly a bonus based on this feedback.

However, such feedback misses the mark. It implies that Peter is a perfect

employee and encourages him to continue as before. It doesn't identify any areas for him to improve on, which is one of the key aspects of any feedback meeting. Moreover, it doesn't include any comments that might help Peter to achieve his future goals.

Learning is a continuous process. There is always something to learn and improve upon. While positive feedback achieves a part of what is expected in these feedback meetings, it fails to identify the areas in which an employee can be more effective.

No one likes to hear negative comments about themselves. Let's look at one example of feedback from Kate about Peter:

"I believe Peter's performance to be just satisfactory. I believe he needs to be more inquisitive and share his ideas and thoughts with the team. He should also be more exploratory, be more independent with regard to new technologies. Though he was somehow able to complete his work on time, I saw him struggle on most occasions. He needs to handle pressure better."

Let's look at the feedback above closely. What does it achieve and what does it miss? Feedback such as this lets Peter know that his manager, Kate, is not completely satisfied with him. It gives Peter a sense of insecurity about his job. As most people would acknowledge, there is no bigger threat in a job than an unhappy manager. In all probability, such an evaluation reduces the possibility of a significant raise and bonus.

Feedback like this misses some important things. It gives the impression that Peter didn't do anything of much significance during the year. It fails to instill confidence in him and discourages him, which could easily reflect on the quality of his future work. An unhappy employee cares more about his issues than the success of a project.

A good dialogue should offer feedback that achieves two major goals: It instills confidence in staff members by stating positives about their work, and it recommends areas where they can improve.

The Right Kind of Feedback

Let's look at another scenario in which Kate evaluates Peter. In this version, Kate says:

"Peter did a wonderful job as a test engineer during the past year. He completed his work on time and found good favor among other team members. He was an important part in the successful release of the product. Though I am very happy with his overall performance, I believe he still has a few areas where he can improve. I believe he needs to be a little more inquisitive and share his ideas and thoughts with the team. He should also be more explorative, be more independent with regard to new technologies. I believe Peter has a lot of talent still left to be unearthed. Having said that, I believe he is an indispensable asset to the team."

This kind of feedback lets Peter know that his manager, Kate, considers him to be an important part of the team. It lets him know that although he is doing a good job, he has some areas to work on. It doesn't give Peter the impression that he is the perfect employee with nothing to improve upon. And most important, it instills confidence in Peter about his work and encourages him to strive for more.

A key component of feedback is not only what is said but *how* it is said. The feedback above puts everything in a very simple and concise way that is designed to produce the most satisfactory results.

While positive feedback is essential to instill confidence and provide a metric for pay raises, it does little to present a complete picture. On the other hand, negative feedback, by itself, can be quite discouraging and may even facilitate a low level of work from staff members.

However, feedback, presented properly and including both positive and negative components, shows appreciation for staff members' accomplishments and helps guide them toward improving their performance in areas in which they are less effective. And it does so in a way that makes it palatable for the people being evaluated. On one hand, it lets employees know that they are important members of the team, and on the other it lets them know that they can be better if they can improve their performance in certain areas.

Active participation by both parties in such meetings is imperative in order to make such meetings successful and is likely to pay dividends in terms of the quality of the products the team is building. Product quality depends on several factors, not the least of which is the quality of the QA team.

SOFTWARE TESTING
ANALYSIS & REVIEW

The World's Largest Software Testing Conference

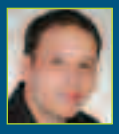
STAR EAST

2006

ORLANDO, FLORIDA MAY 15–19, 2006

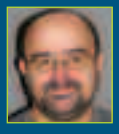
KEYNOTES BY INTERNATIONAL EXPERTS

Your Development and Testing Processes Are Defective
Mary Poppendieck, Poppendieck LLC



The Software Vulnerability Guide: Uncut and Uncensored
Herbert Thompson, Security Innovation LLC

Inside The Masters' Mind: Describing the Tester's Art
Jon Bach, Quardev Laboratories



Testing and the Flow of Value in Software Development
Sam Guckenheimer, Microsoft

Testing: The Big Picture
Brian Bryson, IBM Rational Software

Risk-Based Testing in Practice
Erik van Veenendaal, Improve Quality Services BV

www.sqe.com/stareast

REGISTER EARLY AND SAVE $200!

plus **THE TESTING EXPO**
MAY 17–18, 2006

IN-DEPTH TUTORIALS structured in a daylong workshop format to provide practical, hands-on information about specific testing issues and challenges

KEYNOTE SESSIONS given by international industry experts from a range of backgrounds on a variety of testing topics

CONCURRENT SESSIONS packed with information covering critical testing issues giving new real-world approaches to solving them for the beginner to advanced testing professional

EXPO EVENT provides the latest tools, products, and services from leading testing software and service vendors to give you the solutions you need

A Tiered Test Approach To Validating Software

Toss a Life Preserver to Your QA Team With This Automation Framework Model

By Aditya Dada

During the course of developing complex enterprise software, the testing effort usually inflates as new features are added to the product. Without effective management, chances are that instead of finding and fixing defects, too much time is spent in either trying to scale the manual testing effort, or in maintaining automation of existing tests. This becomes especially crucial to enterprise software where dozens of modules need to be integrated, and features are changed every day. A tiered approach to testing offers an easy way to manage complex testing, and works well with automated tests as well as manual tests.

As the lead architect for automation in the test development group for Sun Java System Application Server, I designed the workspace from scratch during the J2EE 1.3 timeframe. The test workspace has seen many releases since then, and is being used by many different groups—test teams as well as development teams—and has the potential to be automated with a single click. The workspace, which is used by more than 250 engineers, has withstood the test of time and accommodated changes with ease. This article offers some useful ideas to keep in mind if you are in a position to be designing a test workspace.

In the four years I have been working with the Sun Java System Application Server Quality Engineering team, our product has grown from a reference implementation with average quality standards, to a fully blown enterprise edition and a platform edition with rigorous quality processes. Along with the growth of the product, our test base has grown tenfold, from roughly 500 simple tests to about 5,000 complex, distributed tests that are complete transactions in themselves, making test execution very expensive.

Another factor that has made executing a complete test run time-consuming is the number of configurations supported. Four years ago there were only six configurations, with two databases and three operating systems. Those numbers have grown to more than 1,500 configurations, six databases, 13 operating systems and four machine architectures, with support for more browsers and high-availability features.

The tiered model was the only approach possible for us if we were to keep our turnaround time short and find defects quickly. Before I explain the evolution of the tiered model that worked for us, it is important that you first understand the concepts and processes we had to come up with, and their purpose.

Aditya Dada works in the Sun Java System Application Server Quality Engineering group. He is the lead architect for the automation framework being used to test the Application Server, and played a role in developing quality processes for the team. You can reach him at Aditya.Dada@Sun.com.

Photograph by Jeff Gynane

Release Models

With the product growing more complex, we needed to ensure that every stage of product development had the lowest possible number of defects. To achieve this goal, the product build was released to the test engineers and the development engineers in many different ways and in many different increments, so that each increment could be used to run a more comprehensive test suite, thereby catching defects at every stage. The following release models were established.

Tinderbox: What could be more explosive than product stability when 200 developers are checking in code every day? Tinderbox is the machine setup to continuously compile, and build the product. It tracks the changes made by engineers to the product using the source control and signals through its report page. If all is green, things are OK. If they turn red, then the build is broken and the product cannot be compiled into a bundle. Near the deadline, there are times when it feels as if tinderbox is celebrating the holiday season, with its status going green-red-green-red-green. And that is why tinderbox is so effective in tracking rogue check-ins. The gatekeeper for tinderbox monitors its status and takes the responsibility of involving the engineer who made the bad check-in, and rolling back the fault.

Nightly: Each night, the release engineer compiles and builds the product and publishes the bundle on a shared location. Anyone interested in using the nightly bundle, say, to verify a fix to a defect filed earlier, would access this shared location and use the latest product version available. A small set of tests is run against the nightly build by the release engineer, to catch any issues that might have escaped tinderbox.

Weekly: Every week, the release engineer compiles and builds the product and publishes the bundle on a shared location, much like the nightly build. The only difference is that the release engineer tests the weekly promoted builds more thoroughly than the nightly builds, assuring that they are of reasonable certifiable quality. The weekly promoted builds are used by the quality team to run tests on, and to report quality issues to the management team.

Milestone: Before a release of the product to the outside world, in the form of alpha, beta or update release, a milestone build is published by the release engineer. The process of publishing a milestone build is the same as the weekly build. A milestone build, however, usually takes place after a code freeze, and thus it is known that not much would have changed since the last promotion of a weekly or a nightly build.

Different Types of Tests

We discovered during our product development life cycle a few years ago that the development engineers have very little patience. They were given a set of tests that they needed to run before major feature integrations to assure that they were not checking in bad code. This set of "one-size-fits-all" tests took four hours to run, and as much as the development engineers would have liked to have run them, they didn't. Meanwhile, tinderbox was celebrating Christmas with its status lights going red and green all the time, so we decided to come up with sets of tests designed for specific tasks.

Quick-Look: This set of tests ensures that the product build is not completely broken. They take about 15 minutes to run, broadly touch all components of the product, and can be run on any platform with one command. The report is generated in two formats: text, so that the engineer can see the output of the run immediately, and detailed HTML format that can be accessed for failure analysis. These tests are primarily used by engineers to verify the integrity of their changes. The tests are also used by the release engineering group for nightly builds qualification.

When the product development is in the early stages, very simple tests are used for this suite. As the product matures, the simple tests are replaced with increasingly complex tests, thereby keeping the execution time constant and ensuring that more and more bugs are caught at the earliest moment. These tests are used by scores of developers, and it is therefore a necessity that the reporting be simple and understood by all.

Smoke: This set of tests is designed to help the release engineering group find problems in weekly builds. They take about 45 minutes to run and can run on any platform. Setup time is under five minutes. These tests touch all components of the product, but go a little in-depth as well.

Basic Acceptance Tests: This subset of the full test suite takes less than 90 minutes to run. The tests can be run on any platform, and the setup time is under five minutes. These tests examine in depth all most-supported features and most-supported configurations. They are used by the quality engineering team and the sustaining team (the latter team maintains the product after it has been released, mostly by fixing customer escalations and releasing product patches).

Full: This is the entire test base, run by the quality engineering team on all supported configurations. It is the most expensive and time-consuming to run. It takes about a day to run, and at least two to three days for failure analysis.

Along with the above-mentioned test types, we have two other sources of tests.

Developer Unit Tests: The automation framework for developer unit tests is provided by the quality engineering team. The development team designs, owns, creates, enhances, maintains and executes the unit tests.

Compatibility Test Suite (CTS): Sun Java System Application Server is built on a J2EE standard. To ensure that the J2EE standards are followed, and that the application server is compliant with those standards, another round of engineers provides the compatibility test suite, a group of tests that check all APIs used in the product.

A Tiered Approach to Testing Product Builds

With the product builds promoted incrementally, and tests designed to check all such increments, we came up with the following strategy to catch

defects whenever and wherever possible.

Catch defects with every change: Quick-look tests are used by development engineers before they make any change to the product. The engineers make a local change to the product, run the tests, and if all pass, check the change into the main server that holds the product code. Coupled with tinderbox, any basic defect that will render the product unfit to compile or be built is caught almost immediately.

Sometimes, the person monitoring tinderbox also may need to make sure that all engineers are executing the quick-look tests before making changes to the product. However, once the management buys into it, and once the engineers themselves have realized the importance of doing this, it can easily become a matter of habit for the engineers to run the tests with every change they make.

For small, simple changes and defect fixes, quick-look tests are a sufficient check. However for major integrations, the probability of inducing a fault is greater, and so is the pressure on the developers to get it right, since rolling back would then require a major effort. So in addition to running the quick-look tests, developers run unit tests designed by the developers themselves. This increases the likelihood of finding functional defects before major feature integrations.

Catch defects every night: The release engineering group has automation scripts that compile and build the product, and then execute the quick-look tests. Even if a developer has failed to run the tests and checks in code that breaks the functionality of the product even though it compiles, it is caught and reported the first thing next morning. The nightly build promotions are useful for getting fixes for those defects that prevent the quality engineering team from proceeding with test execution.

Catch defects every week: The release engineering group executes the quick-look tests, smoke tests and compatibility tests every week to ensure that the weekly promotion of the product build is of reasonable quality. This build is then used by the quality engineering team to run the full tests on every week. Unless the quick-look tests, smoke tests and the compatibility test suite pass successfully, the release engineer does not promote the build, and therefore, prevents the quality engineering team from embarking on a costly cycle of setting up machines and configurations that need to be tested.

Sometimes, when the promoted build does not have many changes, or when the promotion is delayed and now demands a quick turnaround by the quality engineering team, the basic acceptance tests are used to make up for lost time. This test suite is good for improving the turnaround time without compromising much on the test coverage. It is also great for verifying patches.

Before the basic acceptance test suite was introduced, the smoke test used to do its job. The basic acceptance test suite was introduced to fill the gap between smoke tests and the entire test base. The smoke test, since it was used by the release engineering group, needed to be very stable. Smoke tests were not enhanced as regularly as tests being added to the entire test base. This led to a situation in which smoke tests ran in 30 to 45 minutes, and the entire test base took more than 24 hours, and the gap between test coverage of both of these tests was not a comfortable one—that is, a successful smoke test run no longer guaranteed a successful run of the full test suite. The basic acceptance test suite is a reasonably big test suite and is regularly enhanced, since it is not handed over to the release team or the development team.

Catch defects with every milestone: The quality metrics on the milestone build are important for the entire project team as well as the executive management team so that they can decide to go forth with the release or not. The quality engineering team uses the milestone build (which is promoted by the release engineer just like a weekly build), and runs the entire test base on the build, using the expanded test matrix that includes the operating system, database and software packages. By this time, most of the major defects have been found and fixed. Therefore, the quality team can proceed with confidence that their efforts in setting up machines and running the full test suite on the product will not be in vain.

Using the tiered model for a small product that is easily tested may be overkill.

Benefits of Using the Tiered Model

By far the biggest advantage of the tiered model is that at every step of the application development life cycle, some check is in place that will ensure that defects are caught as early in the cycle as possible. Not all test execution resources need to be mobilized if the smoke tests have not passed on the build. Only when the build is of reasonable quality (as certified by passing quick-look and smoke tests) will the quality engineering team come into play. This consideration will save wasted time and effort on builds that are unfit to be tested against the full test suite.

Using the tiered model for a small product that is easily tested may be overkill. For many products, not all the tiers need to be used at once and not all tiers are always required. Besides, tiers in themselves are not a cure-all. However, if the product you're working on is growing, you will probably face a moment in time when the test base you have is no longer sufficient and the one-size-fits-all strategy is no longer effective. If you have witnessed situations in which you and your team spent considerable time setting up the tests, only to find in the first 10 minutes of the test run that the product being tested was dead on arrival, then the tiered model is for you.

The classic challenge to catch bugs early in the cycle is still tough in today's quality engineering environment. With enterprise software, the challenge becomes even more difficult, with dozens of modules getting integrated at different times in the development cycle. The delays raise the cost of finding bugs, sometimes by as much as a factor of 10. Investment in a tiered model is a one-time expense, but in the long run, when properly used, it will bring down the cost of finding bugs and pay for itself many times over.

Best Practices

There's Gold in Your Own Hills–And You Can Find It

Esther Schindler

For most developers and testers, a software change management (SCM) system is a lot like an accounting program. Like an accounting program, the SCM tool's role is to keep a record of what happened and to provide a snapshot of your current status. You record what you changed, rely on the SCM tool to track who has which module checked out, and keep a tab on the bugs to be fixed.

But, also like an accounting program, it's important to remember to use the tools to help you see the big picture. There's no reason to get stuck in the day-to-day details. Every so often, whether you're a manager or a techie, it's useful to step back and use your SCM tool to get a wider view. After all, you already have the data. Why not use it to learn more about the state of your project, the efficiency of your staff and likely sources of design problems?

For instance, Jeff Grigg, senior consultant at Daugherty Business Solutions in Saint Louis, looks for the answer to the question, "Who's had files checked out (locked) for a long time?" The definition of "a long time" can vary from project to project, Grigg says, but typically his eyebrows go up if he sees files checked out for more than a day on agile projects, and for more than a week on less agile projects. As a result, says Grigg, he appreciates SCM tools that can automatically send out "nag" e-mail messages to people with long check-outs.

Says Grigg: "I've been on projects where we were locked out of files checked out by people who had left the company months or even years earlier. The SCM administrator can clear such locks, but it's better to stay on top of these things, rather than letting them accumulate like land mines."

Jeffrey Fredrick, director of engineering at Mountain View, Calif.-based Agitar Software, says that at several organizations he has looked at reports on branches that are close to shipping. Doing so ensures that the organization knows why each file was changed. "I've found enough accidental check-ins that way to make it worth doing," he says.

William Pietri, a software process consultant in San Francisco, looks for a few things using his SCM tools when he evaluates codebases and teams. "One big one is size of check-in by developer. Adding or changing a lot of lines in one go is a sign of danger. Long intervals between check-ins are another. A positive sign is somebody who at least occasionally removes more lines than they add." According to Pietri, frequently changed files are often design hot spots that merit careful inspection. Files that are large and frequently changed are great candidates for refactoring.

Ilja Preuss, a software developer at disy Informationssysteme GmbH in Karlsruhe, Germany, also looks at code size—but in a larger sense. When working on a legacy system, says Preuss, "we typically expect code size to decrease while refactoring the system to incorporate new features."

Beyond Size and Time

There are other hints that your SCM tool can give you about the nature and status of your development project and its quality health. Says Pietri: "I also look carefully at check-in comments to give me an idea of what people are like, but that's more a qualitative thing."

Preuss uses his SCM tools to watch the frequency of build failure. "We are using Cruise Control to start, monitor and report on automatic builds. It has a nice chart showing the chronological succession of build successes and failures. The patterns in that chart mostly tell us interesting things about our process."

One "obvious" action is to keep track of your project using the reporting tools included with every application. However, some developers and testers find they aren't always the best way to see what's going on.

According to Pietri, exploring the data generates more value than consistently viewing the same report. Otherwise, he says, "people are prone to making a bad behavior look OK from the perspective of whatever the standard report is." Which isn't to say that an SCM application doesn't provide useful features—Pietri uses IntelliJ's IDEA, and is impressed by a plug-in that, he says, lets you easily browse recent changes in a variety of ways. "I find it handy for keeping up with a team's activity," he adds.

Jeanne Boyarsky, a Java developer for a bank in New York City, uses the information she's gathered about past tests to plan out new techniques. For example, she says, the results are examined to find out if certain types of code haven't been tested. Over time, her team adds new things to monitor and to report on. "I think this is because we have a tendency to focus on what is being measured. Once it is

Contributing editor **Esther Schindler** manages her own changes from her bitranch in Scottsdale, Arizona.

measured, we master that, get used to it, and go on to the next thing."

You can also look beyond what are typically considered SCM tools.

Mark Waite, R&D manager at CoCreate Software in Fort Collins, Colo., doesn't rely on an SCM system, per se, but on something very like it. "Our continuous integration machine present hyperlinks to the diffs which composed each submit to the master. I use those links all the time to get a quick view of the changes that happened, after reading the submit comment that is presented on the same page. It is not fancy, but it easily meets 80 percent of my needs for reports from an SCM system."

According to Waite, if the submit message includes text that is recognizable as a reference to a defect report, the continuous integration scripts convert that to a hyperlink into the bug-tracking system. "It isn't SCM (unless you consider bug tracking, like Perforce job tracking, as a part of SCM), but it meets our needs to know what is happening to our source code."

Whether you're a manager or a techie, it's useful to step back and use your SCM tool to get a wider view.

CoCreate has considered and discussed other measures (lines of diff, number of submits, "turmoil", etc.) and decided that none of them would encourage the behavior they want. "We want better software, and measures that mislead us don't help us get better software," Waite says. "If we measure the number of lines in a diff, then we will reduce the programmer's willingness to refactor. We want them to refactor when they find the code is in bad condition. If we measure the number of submits in a day, we will reduce their tendency to submit small test-driven changes instead of larger chunks of less-tested code. We want them to be test-driven and to submit small chunks so we can watch each other through continuous integration."

While the primary reason for using SCM tools is to manage the process of developing and testing software, there's no reason not to use them in unique ways to perform your own kind of data mining.

Index to Advertisers

Software Test & Performance

Advertiser	URL	Page
Eclipse Review	www.EclipseReview.com	28
IBM	www.ibm.com/middlewear/tools	2
iTKO	www.itko.com	23
Instantiations	www.instantiations.com/codepro	8
LogiGear	www.logigear.com	37
Parasoft	www.parasoft.com/STPmag	4
RadView	www.radview.com/analyze	3
Rally	www.rallydev.com/stp	31
Seapine	www.seapine.com/st&p	6
Segue	www.segue.com/lh?sorry	13
Software Security Summit	www.S-3con.com	19
Software Testing Analysis & Review	www.sqe.com/stareast	32
ST&P Conference	www.stpcon.com	39
SQS	www.sqs.com	40

The Changing Face of Software Test Teams

Niel Robertson

In contrast to the general technology space, software testing has stayed relatively free of TLAs (Three Letter Acronyms). Occasionally someone will sneak a "SUT" into the conversation or talk about the ins and outs of "FIT." But for the most part, testing types like to stick to more expansive descriptions. While keyword-driven testing has never turned into "KDT" and black-box testing has escaped a "BBT" acronym, in the past year, software testing has been under a TLA siege. Today's software tester hears less about ASQ and more about SOX, SOA, BPT and UAT. Where have all these new terms come from, and what can we expect the future of testing to look like?

Both software and software testing are undergoing a renaissance. The applications of tomorrow are being built as modular components, pieced together into complex business processes using BPEL, WSS or some other BPM standard. IT departments are adopting middleware frameworks such BEA, WebSphere or Microsoft's Sharepoint, and packaged application frameworks like Oracle's Fusion and SAP's NetWeaver are expanding their domain to cover next-generation composite applications. The mantra is clear: Focus on the business process, not on the application.

Much of this is driven by changes in IT, and IT is becoming an increasingly process-centric exercise. Changes in IT such as those introduced by Sarbanes-Oxley (SOX) requirements are realigning teams and initiatives around business processes instead of applications. And if there is any indication software testing is being swept along with this tide, look around the room at the new faces on the software testing team.

For many traditional software testing teams, the business user is a new and unpredictable variable in the testing process. Usually co-opted against their will into testing, business users generally don't want to test, and definitely don't want to learn new tools to do so. But they are a necessary and (you'll come to find) valuable part of your test team. As the focus of testing shifts from software quality to business process success, business users have become critical to testing. They know the business.

As a software tester, it is valuable to realize the symbiosis that exists between you and the business user. While you understand the applications and technology, you probably don't deeply understand the business process and associated data. What are the budget limits for expense reports in the marketing group? What should happen when a journal entry is reversed from the financials management software system? What are the four conditions that must exist to cancel an order heading to the warehouse? The business user knows these processes, plus five common ways users get around them. But the business user does not know the underlying application and does not understand test automation. You do.

Here are three practical ways you can learn to work more effectively with these new faces on the team:

Change What You Measure: In a world of business processes, no longer is software quality measured by close ratios, reopen rates, code coverage or P1-free release candidates. The new quality metric is successful business process execution. Take what you have learned from context-driven testing and use-case requirements and recast them in a business light. Do you know the right processes to test? Do you know what data will effectively test the process?

Change What You Test: Software testing is no longer about breaking the application, but about testing the business process path through the application. As more applications are built on third-party frameworks (such as Oracle Fusion), you have to assume the underlying application will work. Focus less on getting the application to fail (for example, testing boundary conditions for a textbox) and more on getting the business process to fail (for example, can someone alter a quarterly bonus or approve a purchase order greater than the allowed amount?).

Adopt New Technology: As business users have become involved in the testing process, automated testing tools have evolved to accommodate our new testing team members. New approaches such a scriptless testing and business process testing (BPT) are attempting to repackage keyword-driven testing into a user-friendly way to entice business users into using testing tools. By providing two different interfaces, one designed for the business user and another for the technical automator, you can work together to define the business process steps, required data and underlying technical implementation.

Whether you like it or not, software and software testing teams are headed in the business direction. Look on the bright side. Quality assurance is finally getting the corporate support it has always deserved. So when the corporate spotlight does eventually shine on you, make sure you're TLA-ready, and rest assured knowing one thing—there are only 17,576 possible combinations to learn.

Niel Robertson is the CTO of Newmerix, a software testing and change management company based in Superior, Colo. He holds a B.S. in computer science from M.I.T.

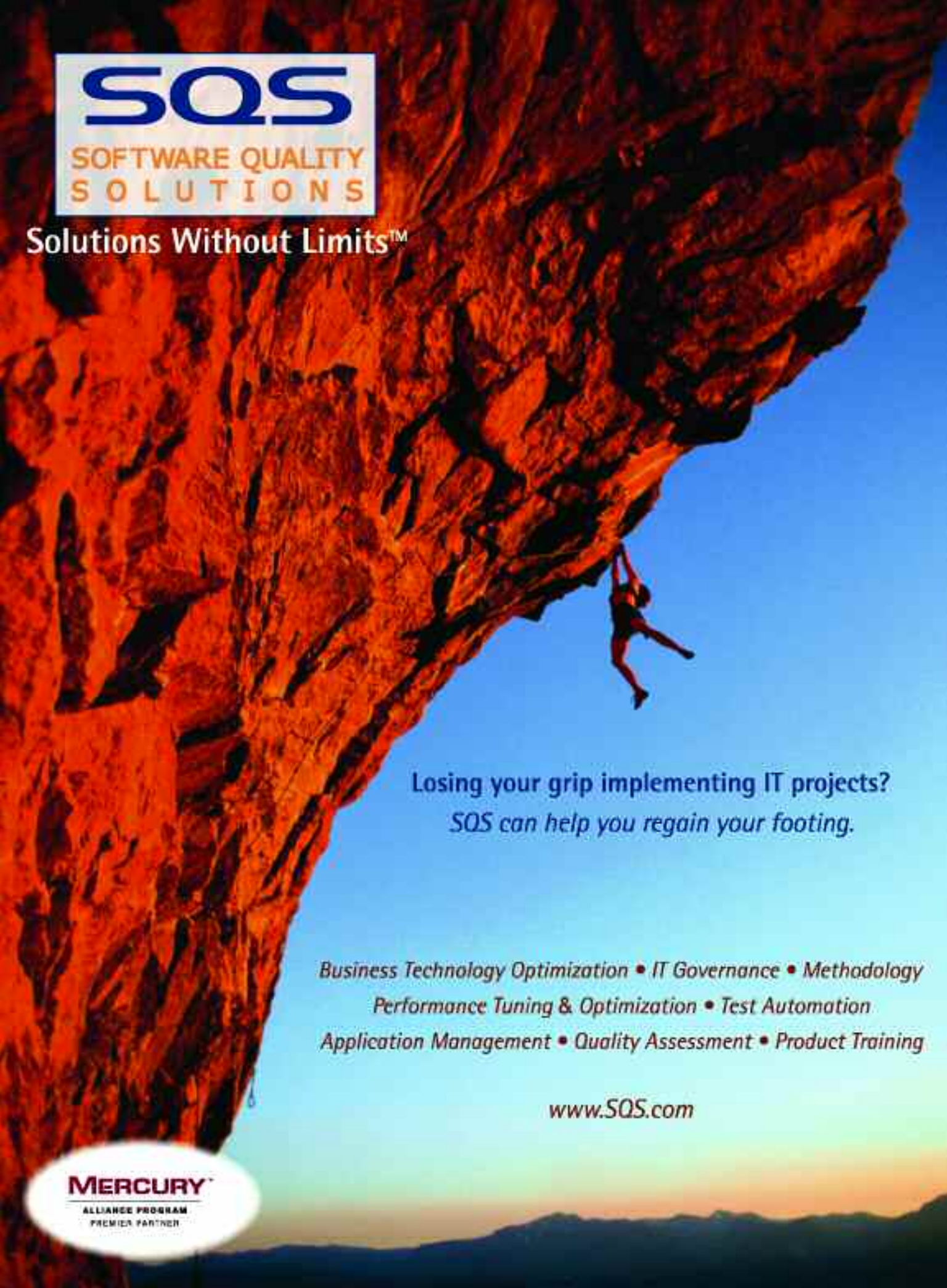
Mark your calendar!

Software Test & Performance CONFERENCE

www.stpcon.com

IS COMING TO BOSTON IN 2006!

November 7-9, 2006
The Hyatt Regency Cambridge


SQS
SOFTWARE QUALITY
SOLUTIONS
Solutions Without Limits™
Losing your grip implementing IT projects?
SQS can help you regain your footing.
Business Technology Optimization • IT Governance • Methodology
Performance Tuning & Optimization • Test Automation
Application Management • Quality Assessment • Product Training
www.SQS.com
MERCURY
ALLIANCE PROGRAM
PREMIER PARTNER