A BZ Media Publication

Software Test & Perfor ce

VOLUME 5 • ISSUE 2 • FEBRUARY 2008 • $8.95 • www.stp

BEST PRACTICES: Java Testing For the Enterprise

Multi-User Testing Can Be Challenging and Fun

Don't Drown in Chaos, Just Pull the Plug

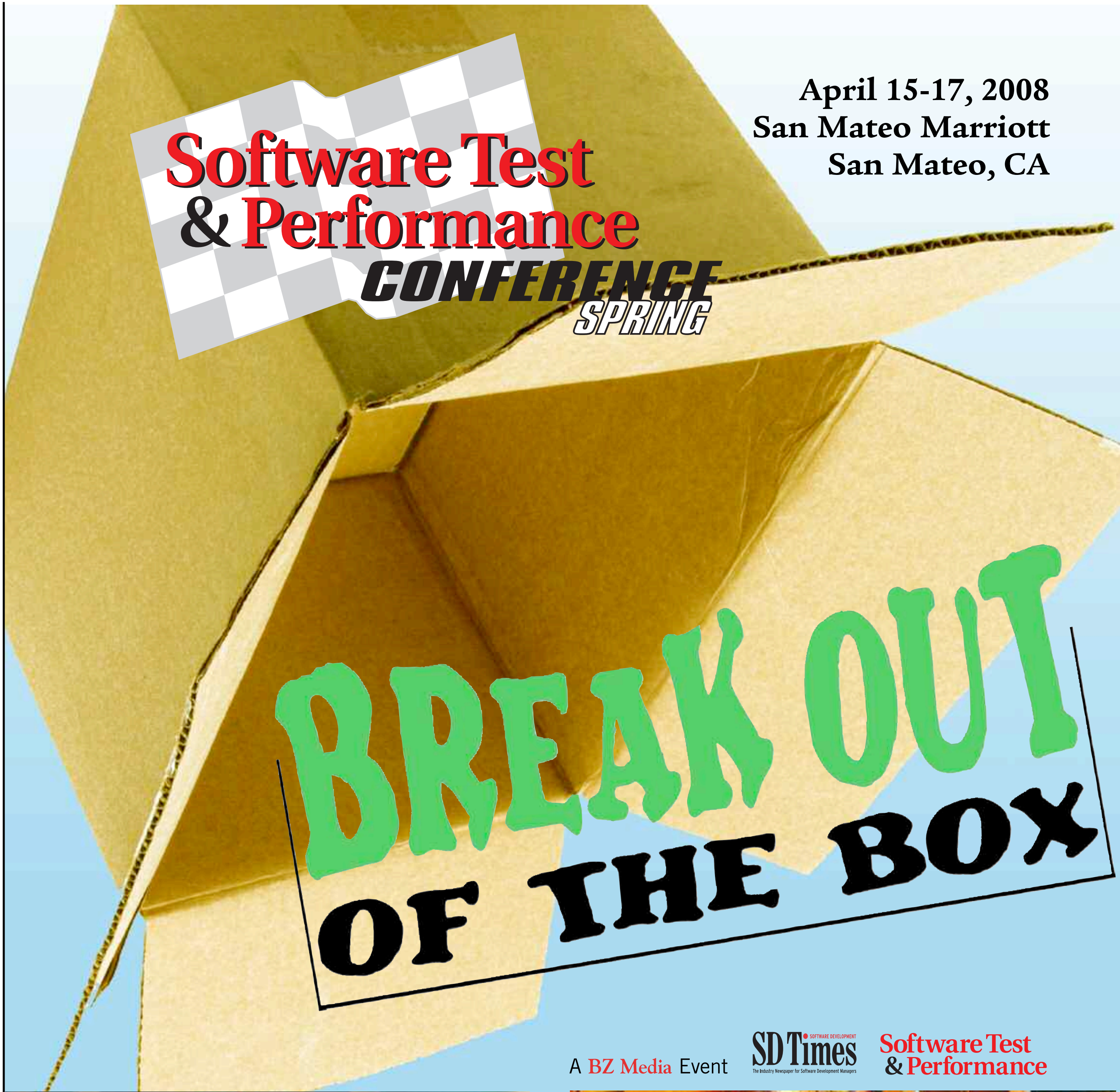
When, Oh When Will You Profit From SOA?

BUGZILLA VS. TRAC: KILLER APPS COMPARED!

A Battle of Open-Source Defect Trackers

Software Test & Performance CONFERENCE SPRING

April 15-17, 2008
San Mateo Marriott
San Mateo, CA



A BZ Media Event SD Times Software Test & Performance

Platinum Sponsors

empirix

RTTS

Gold Sponsors

LogiGear

Mu Security Mutate. Monitor. Manage.

Silver Sponsor

CODENOMICON

www.stpcon.com

SUPERB SPEAKERS!

Michael Bolton · Jeff Feldstein
Michael Hackett · Jeff Johnson
Bj Rollison · Rob Sabourin
Mary Sweeney · Robert Walsh
AND DOZENS MORE!

STPCon is THE best place to learn the newest techniques for improving software quality, but don't take our word for it—*just listen to your colleagues:*

"You'll find information outside of your daily activities, and options/alternatives to think about new approaches to testing."
—Alex Kang
Staff Engineer, Tellabs

"It solidifies the total testing experience and opens your eyes to alternative approaches and methods that you simply cannot get from books."
—John Croft
QA Manager, I4Commerce

"Great, informative conference for software testers, leads and managers alike. Useful tutorials and technical classes of wide variety—A must-attend for all serious QA/SQE professionals!"
—Alan Abar
Software Quality Engineering Manager, Covad Communications

TERRIFIC TOPICS!

- Agile Testing
- Testing Techniques
- UI Evaluation
- Java Testing
- Security Testing
- Test Automation
- Improving Web Application Performance
- Optimizing the Software Quality Process
- Developing Quality Metrics
- Testing SOA Applications
- Charting Performance Results
- Managing Test Teams

Register by Feb. 22 To Get The Super Early-Bird Rate and SAVE OVER $400!

Take the

handcuffs off

quality assurance

Empirix gives you the freedom to test *your way.*

Tired of being held captive by proprietary scripting? Empirix offers a suite of testing solutions that allow you to take your QA initiatives wherever you like.

Download our white paper, "Lowering Switching Costs for Load Testing Software," and let Empirix set you free.

www.empirix.com/freedom

empirix
Be Confident.

Software Test & Performance VOLUME 5 • ISSUE 2 • FEBRUARY 2008

Contents

A BZ Media Publication

Don’t Miss Out

On Another Issue of The

Test & QA Report eNewsletter!

Each FREE weekly issue includes original articles that interview top thought leaders in software testing and quality trends, best practices and Test/QA methodologies. Get must-read articles that appear only in this eNewsletter!

Subscribe today at www.stpmag.com/tqa

To advertise in the Test & QA Report eNewsletter, please call +631-421-4158 x102

Software Test & Performance

VOLUME 5 • ISSUE 2 • FEBRUARY 2008

EDITORIAL

Editor	***Editorial Director***
Edward J. Correia	Alan Zeichick
+1-631-421-4158 x100	+1-650-359-4763
ecorreia@bzmedia.com	alan@bzmedia.com
Copy Editor	***Contributing Editor***
Laurie O'Connell	Geoff Koch
loconnell@bzmedia.com	koch.geoff@gmail.com

ART & PRODUCTION

Art Director	***Art/Production Assistant***
LuAnn T. Palazzo	Erin Broadhurst
lpalazzo@bzmedia.com	ebroadhurst@bzmedia.com

SALES & MARKETING

Publisher
Ted Bahr
+1-631-421-4158 x101
ted@bzmedia.com

Associate Publisher	***List Services***
David Karp	Lisa Fiske
+1-631-421-4158 x102	+1-631-479-2977
dkarp@bzmedia.com	lfiske@bzmedia.com
Advertising Traffic	***Reprints***
Phyllis Oakes	Lisa Abelson
+1-631-421-4158 x115	+1-516-379-7097
poakes@bzmedia.com	labelson@bzmedia.com
Director of Marketing	***Accounting***
Marilyn Daly	Viena Ludewig
+1-631-421-4158 x118	+1-631-421-4158 x110
mdaly@bzmedia.com	vludewig@bzmedia.com

READER SERVICE

Director of Circulation	***Customer Service/ Subscriptions***
Agnes Vanek	
+1-631-443-4158	+1-847-763-9692
avanek@bzmedia.com	stpmag@halldata.com

Cover Photograph by LuAnn T. Palazzo
Animatronic Godzilla Appears Courtesy of the Daniel J. and Naomi Pagano Collection, NY

BZ Media

President	**BZ Media LLC**
Ted Bahr	7 High Street, Suite 407
Executive Vice President	Huntington, NY 11743
Alan Zeichick	+1-631-421-4158
	fax +1-631-421-4130
	www.bzmedia.com
	info@bzmedia.com

Software Test & Performance (ISSN- #1548-3460) is published monthly by BZ Media LLC, 7 High Street, Suite 407, Huntington, NY, 11743. Periodicals postage paid at Huntington, NY and additional offices.

Software Test & Performance is a registered trademark of BZ Media LLC. All contents copyrighted 2008 BZ Media LLC. All rights reserved. The price of a one year subscription is US $49.95, $69.95 in Canada, $99.95 elsewhere.

POSTMASTER: Send changes of address to Software Test & Performance, PO Box 2169, Skokie, IL 60076. Software Test & Performance Subscribers Services may be reached at stpmag@halldata.com or by calling 1-847-763-9692.

Ed Notes

Defect Tracker For Politicians

Edward J. Correia

By the time you read this, Super-Duper-Kalamazoop-er Tuesday will likely be in the history books. I'm referring of course to what we in the U.S. call "Super Tuesday," the day in February on which people of the major political parties from about 20 states vote in the presidential primary. Primary votes prior to Super Tuesday take place one state at a time. It's all part of the American process of deciding who gets to run for president, the nation's highest office.

As someone who follows national politics to a flaw (just ask my daughter), I sometimes have a tough time keeping track of candidates' positions (which are often numerous). Where does each one stand on the economy, national security and other important issues of the day? And how does their current position differ from things they've said and done in the past?

As I edited this month's cover feature, it occurred to me that the same tools we use for tracking software defects could also be applied to tracking politicians. Enter FlakTrak, a new defect-tracking system I just invented to help me keep abreast of where our leaders stand.

Here's how it works. The first time a politician takes a position on an issue, it's entered into FlakTrak and assigned to that politician. If it's a position I agree with, it's resolved immediately and becomes a feature. Promises to reduce taxes, build a border fence and win the war in Iraq fall into this category.

If a candidate takes a position I disagree with, a bug is created and assigned to that politician. Raising taxes, granting rights to illegal aliens or calling for surrender, for example. These all would be classified as defects. If a candidate has too many unresolved defects, I wouldn't vote to deploy that product into the Oval Office.

As in software, political defects can be minor or severe. Minor defects might include crying on TV, having an extramarital affair or experimentation with drugs early in life. While these things might offer commentary on one's character, they alone would not stop a project from being released.

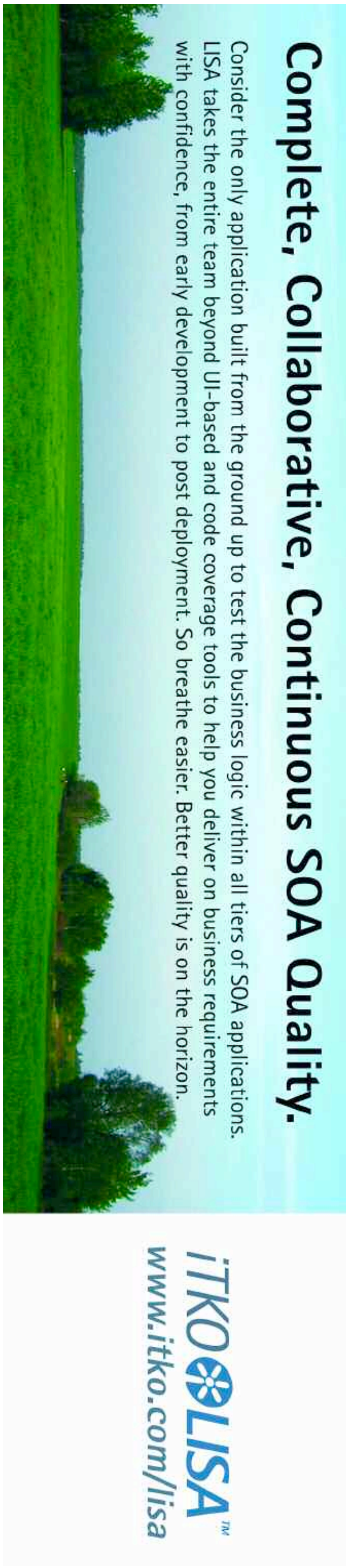
But severe bugs might, and would get top priority when discussing and questioning the candidate. Such things might include being caught in a lie (particularly while under oath), any type of fiscal malfeasance or too many flip-flops on important issues.

Just as desired software features change over time, candidates too have been known to change their positions on the issues, particularly as shifts in political climate affect public opinion. In the days and months after the attacks of 9/11, most Democrats and Republicans were in agreement that the U.S. should invade Afghanistan and Iraq. Now most disagree.

Similar flip-flops can be seen on abortion, POW detention, illegal immigration, taxes, global climate change and "corporate greed." When a candidate switches position, either a bug is resolved or a feature fails a regression test, and a new defect is logged.

Historically Significant

For the first time in 80 years, the field of candidates does not include an incumbent president or vice president. Interestingly, there's one candidate who in 1928 was only eight years from being born. The first few issues he would find in his queue would be called McCain-Feingold, McCain-Kennedy and McCain-Lieberman.


Complete, Collaborative, Continuous SOA Quality.
Consider the only application built from the ground up to test the business logic within all tiers of SOA applications. LISA takes the entire team beyond UI-based and code coverage tools to help you deliver on business requirements with confidence, from early development to post deployment. So breathe easier. Better quality is on the horizon.
iTKO LISA™
www.itko.com/lisa

Contributors

ELFRIEDE DUSTIN is a testing consultant currently employed by Innovative Defense Technologies (IDT), a Virginia-based software testing consulting company specializing in automated testing.

In her role as consultant, Elfriede has evaluated scores of software testing tools. In this month's cover story, she tackles Bugzilla and Trac, two of the industry's leading issue-tracking systems. If you're about to select a bug tracker or thinking about switching, you won't want to miss her analysis, which begins on page 14.

This month we're pleased to welcome **KAREN N. JOHNSON** to our pages. Karen is an independent software testing consultant working in the Chicago area, is a frequent conference speaker and has authored numerous articles and papers. She is active in numerous testing associations.

In the first of what we hope will be many contributions to this magazine, Karen explores how and why multi-user testing typically requires a shorter cycle than other types of testing. Learn how to do multi-user testing right, beginning on page 20.

ROBIN GOLDSMITH has been president of Go Pro Management, a software testing and training consultancy, since 1982. He specializes in business engineering, requirements analysis, software acquisition, project management and quality assurance.

Beginning on page 25, Robin explores the real cause of chaos, and why most IT projects are late, over budget and wrong. Learn to avoid these problems by developing objective factual project measures in an environment where everyone involved takes responsibility for their results.

FRANK GROSSMAN is co-founder and president of Mindreef, which offers testing solutions for Web services and SOA-based applications. Frank, along with Jim Moskun, developed SoftICE, BoundsChecker and DevPartner, popular tools that would ultimately be acquired by Compuware.

Drawing from more than 20 years of technical expertise in software tool development, Frank delivers a thoughtful examination of what companies need to do to ensure a positive return on their investment in SOA. Turn to page 29.

As a quality control specialist with a global IT services organization, **PRAKASH SODHANI** is involved in a multitude of testing and quality assurance activities. He has served as a quality professional in numerous capacities with several top IT organizations.

Prakash holds a master's degree in computer science and many testing certificates. Beginning on page 33, Prakash converts some of his experiences into a hypothetical case study on the dos and don'ts of software testing.

TO CONTACT AN AUTHOR, please send e-mail to feedback@bzmedia.com.

CHANGE: THE ONLY CONSTANT

"Traceability: Still a Fact of Life in 2008" (Test & QA Report, Jan. 8, 2008) is still an interesting read. Having started in an industry where regulatory compliance was required, it has been interesting to watch how things are going back toward compliance. The culture of an organization needs to change and adapt to compliance. It is easy to see why some not-so-good software is produced, as requirements and changes are not tracked or traced, let alone tested.

Compliance does not mean adding cost but being smarter about the processes that are used. If a process is not reviewed and modified to incorporate the changes required for compliance, then the bottom-line cost for software increases, as stated in the article. Change is the only constant in life, and we all need to adapt and change.

Gretchen Henrich

LATE-TERM BUG FIX

Edward J. Correia's article "Defect Tracker, Heal Thyself" (T&QA Report, Dec. 15, 2007) states the same basic thing found in many articles: Look at the defects and evaluate them early in the testing process to determine what can be changed in requirements or further developed requirements to reduce future defects.

But what if you are past that stage? How do you evaluate the situation when there are over 2,000 critical defects and development team is in system test? How do you look at these defects, determine which ones are the most important to correct with limited resources, and which ones can be put aside so the application can continue forward toward going live? How do you salvage the situation to achieve the most impact on making the application work and the least impact on data and the users when the application goes live?

Jim Watson

AUTOMATION FOR FLASH

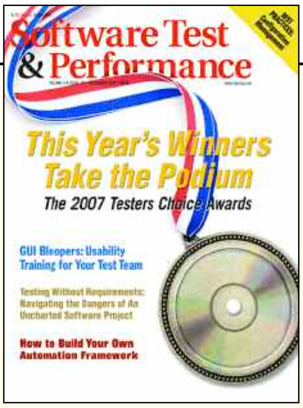
Regarding "What's the Best Way to Test Flash Apps?" (T&QA Report, March 27, 2007), my biggest pain point is regression testing for Flash applications. Manual testing is time intensive. An automated solution would be helpful.

Scott Schimanski

From the editor:
Thanks for writing, Scott. I suggest that you take a look at this month's Out of the Box section (page 11) for coverage of how a tool called Eggplant is being used to automate Flash testing.

According to its creator, Redstone Software, this was an unexpected development.

KNOWLEDGE FIRST, THEN TOOLS


Software Test & Performance
This Year's Winners Take the Podium
The 2007 Testers Choice Awards
GUI Bleepers: Usability Training for Your Test Team
Testing Without Requirements: Navigating the Dangers of An Uncharted Software Project
How to Build Your Own Automation Framework

Regarding Geoff Koch's question about what resonates with his readers ("A Code-Free Look at Change and Management," Best Practices, Software Test and Performance, Dec. 2007), I agree that tools are secondary to knowing what to do.

People seem to forget this when it comes to CM and process-related issues. For example, I was working with a dev manager who wanted me to "enable the teams to test more effectively," but he didn't want me to spend too much time working on current issues or even training people one-to-one, and a new framework wasn't going to solve the problem. People needed new habits (and some coaching).

Ironically, this same manager was a big advocate of use cases and architecture (which are tool independent). Testing, CM and deployment are often treated as secondary issues. Testing, CM, deployment and architecture are closely related; your testing, configuration and architecture can greatly affect how easy it is to deploy and upgrade. Maybe that's obvious, but...

Another of my common rants is that a team should have a deployment process from about day one, so that they can tune the configuration and deployment process. I remember working at places where "build the installer" was scheduled for a few days before ship date.

CM Crossroads at www.cmcrossroads.com and the CM Journal are all about issues related to what Mr. Koch is writing about.

Thanks for the article.

Steve Berczuk

Geoff Koch responds:

Hi, Steve,

I really appreciate you taking the time to correspond. The substance of your comment does validate a conclusion I'm reaching after writing this column for more than two years—namely, there's a healthy collection of good tools and processes for doing test, CM and deployment. Uber-geeks will argue over the finer points of these tools and processes, but in the bigger picture, these arguments are almost always irrelevant. That's because the real stumbling block to sound test & QA is almost always human behavior, from the individual developer who finds it too much of a hassle to really embrace test-driven development to organizations that perennially shortchange test and QA when it comes to resources, schedules, etc. And it's not as if there's a shortage of information about the cost to developer organizations of fixing issues downstream or even the macro cost to society of buggy, unreliable software.

I know that contemporary economic research is making much hay out of finding examples of how in fact people most often don't behave rationally; testing in general seems to be a case in point of this phenomenon.

Best regards, and thanks again for reading,

Geoff

FEEDBACK: Letters should include the writer's name, city, state, company affiliation, e-mail address and daytime phone number. Send your thoughts to feedback@bzmedia.com. Letters become the property of BZ Media and may be edited for space and style.

SICK AND TIRED

OF MANUAL TESTING?

TestComplete™

Sensible price. Superior automated testing.

Test **.NET** Delphi **Java**

Web Automated **Windows** Test Desktop

Load Easy **Vista** Test Synchronized

Fast Distributed Powerful Client/Server

✓ NEW! Record Test Checkpoints

✓ NEW! Better & Faster Web Tests

✓ NEW! Test Your 64-bit Apps

✓ NEW! Test Your Web Services

Try TestComplete™ free for 30 days and enter to win an **Apple iPod Touch**

www.testcomplete.com/stp

FREE TRIAL - DOWNLOAD NOW

www.testcomplete.com/stp

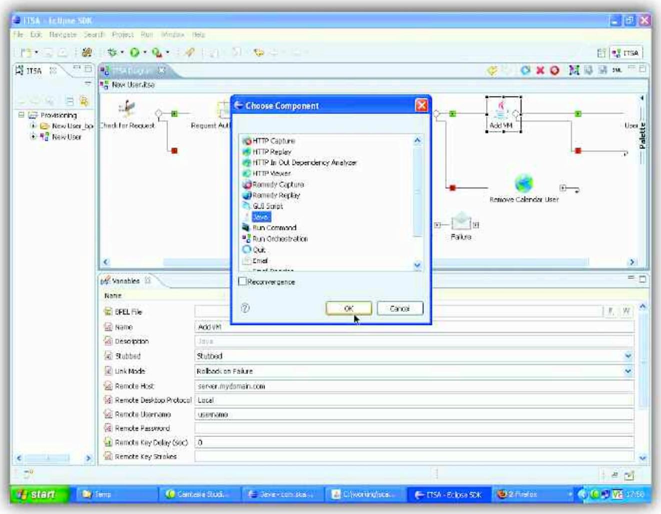
AutomatedQA

test, debug, deliver!

(978) 236-7900

Scapa Scraps IT Repeats

Tired of all those user directory moves, adds and changes whenever someone's hired or fired? Scrap them with Scapa—Scapa ITSA, that is. That's short for IT Service Automation, an automation tool introduced in December that the company says can automate many of the IT operations tasks currently being performed manually by people.



Scapa ITSA presents a GUI-based environment for automating repetitive tasks now done by people.

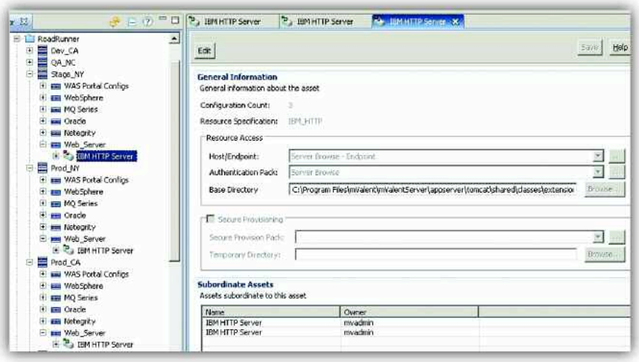
Using an Eclipse-based GUI, IT operations staff can create process flows that handle service requests coming in through e-mail or Web-based forms or other applications. Tasks appear as graphical icons that can be set to perform operations selected from a drop-down list. Flows can be run periodically or set to trigger automatically.

ITSA also includes record and playback capability. "If you can do something through the GUI, Scapa ITSA can capture and automate it," according to documents on the company's Web site. The solution also can be controlled remotely using Citrix, VMware/VDI or RDP.

mValent Integrity Gets Pushy With Provisioning

Configuration management tools maker mValent last month released Integrity 5, the latest version of its flagship solution that it claims now enables companies to push configuration settings to large, distributed systems and detect and manage configuration changes across an enterprise.

With the addition of automated deployment tools, Integrity 5 now can provision single or multiple hosts in unlimited numbers and locations and across firewalls. Provisioning can consist of single components or packages. The new version can also analyze local or remote configurations to pinpoint differences and inconsistencies and help prevent drift. Reporting is simplified through new automated change dashboards.



mValent Integrity 5 can now send configurations in whole or in part to far-flung systems.

An SDK for Every Issue Tracker

Perforce in December released an SDK for its Defect Tracking Gateway, a set of components for integrating the company's Perforce 2007.3 Fast SCM with third-party issue-tracking systems. First released in February 2007, Defect Tracking Gateway integrated only with Hewlett Packard's Quality Center 9.0. The SDK enables the creation of integration with any issue tracker.

Defect Tracking Gateway includes a graphical editor for creating and managing field maps between Perforce 2007.3 and a defect management system. A replication engine moves data between the two systems and keeps everything in sync. Synchronization can be either one-way or bidirectional. Changelists, created and used by the Perforce system, describe all changes made to files, and also can be replicated.

The SDK and Defect Tracking Gateway are included with Perforce Server 2007.3 for Windows XP, available now; pricing starts at US$800 per seat.

It's Springtime For Integration

Open source and Java solutions provider SpringSource in December unveiled Spring Integration, an extension of the Spring Framework that the company claims can simplify the creation of message-driven integration systems. SpringSource, which changed its name from Interface21 in November, created and maintains the Spring application framework.

According to the company, Spring Integration simplifies development by "handling the message-listening and service-invoking aspects, [and] applies inversion of control principles" to the runtime. The tool also handles common input and output sources through included adapters, and adds to Spring's core functionality with support for JMS, remoting, e-mail and scheduling. Spring Integration also takes care of task execution, life cycle management, dynamic languages, aspect-oriented programming, event-oriented publishing, and subscription and transaction management, the company said.

The extensible framework, which is set to reach version 1.0 by June, allows for the creation of custom input and output adapters, content-based routers, content "enrichers," and message filters and translators.

Klocwork Offers Insight Into Desktop Code Analysis

According to Klocwork, maker of automated source code analysis tools, a tool introduced in January can prevent buggy code from finding its way into an organization's main code tree.

Insight, which began shipping in late January, is a desktop analysis tool with collaboration features that ties in with the team's larger system and identifies defects in the developer's local build. Defect reports can be broken down by component, team or geography.

Although desktop-based source code analysis tools have been available for years, Klocwork CEO Mike Laginski claims their accuracy was limited since they lacked the context of the overall system. "Conversely, a system that runs only at the system level is viewed as an audit tool by developers and doesn't give them the ability to find and fix problems before they check in their code." Insight provides both, Laginski claims.

Along with Insight, Klocwork introduces a declarative language that can be used to extend or customize it and other Klocwork products. Insight is available now in a C/C++/Java version as well as one just for Java.

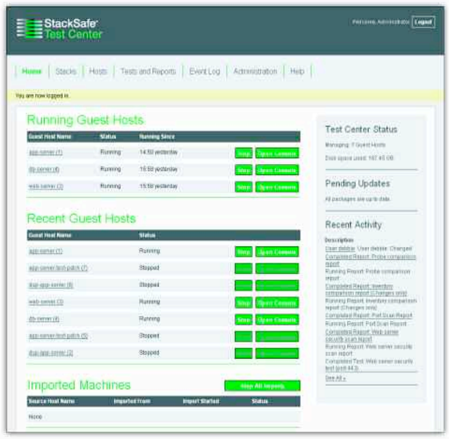
Solution for Your Flash Testing Might Include Eggplant

Hungry for tools to automate your Flash application testing? Perhaps you should try Eggplant. That's the word from Redstone Software, which reports an increase in usage of the multi-platform UI automation tool for testing apps written in Adobe's popular environment.

According to Redstone managing director Christopher Young, Eggplant was never intended to test Flash. "We designed it to test or automate anything, but especially the user experience," he says. "Anyone who has an issue testing their Flash applications or tests them manually now" has an easier way. Eggplant works across multiple browsers and operating systems, including even Symbian and Windows CE, the company says.

StackSafe Virtualizes The Data Center

A new solution from StackSafe uses virtualization technology to give IT operations teams an easy way to simulate multi-tiered systems and applications for testing, performance tuning, upgrades and other activities that might pose risks to production systems, according to company claims.



You are now logged in.

Running Guest Hosts

Guest Host Name	Status	Running Since
app-server (1)	Running	14:50 yesterday
db-server (4)	Running	15:59 yesterday
web-server (3)	Running	15:59 yesterday

Recent Guest Hosts

Guest Host Name	Status
app-server (1)	Running
app-server-test-patch (7)	Stopped
dup-app-server (6)	Stopped
web-server (3)	Running
db-server (4)	Running
app-server-test-patch (5)	Stopped
dup-app-server (2)	Stopped

Imported Machines

Source Host Name	Imported from	Import Started	Status
None			

Test Center Status

Managing: 7 Guest Hosts

Disk space used: 167.45 GB

Pending Updates

All packages are up to date.

Recent Activity

Description

User debbie: User debbie: Changed
Completed Report: Probe comparison report
Running Report: Probe comparison report
Completed Report: Inventory comparison report (Changes only)
Running Report: Inventory comparison report (Changes only)
Completed Report: Port Scan Report
Running Report: Port Scan Report
Completed Report: Web server security scan report
Running Report: Web server security scan report
Completed Test: Web server security test (port 443)
See All »

StackSafe's Test Center, through its browser-based interface, permits change testing on virtual systems prior to deployment to production.

Dubbed Test Center, the new tool began shipping late last month and is claimed to enable test teams to "import [virtual copies of] production systems to conduct staging, testing, analysis and reporting." Imported copies are networked into a working infrastructure stack that simulates the production configuration, enabling production-safe changes, regression testing, patch testing, security and risk assessment, diagnostics and root-cause analysis, emergency change testing, application assembly and validation and compliance reporting, the company says.

Faster, Better Automation Is The Claim From VMware

Virtualization technology maker VMware in December released Infrastructure 3, which updates its ESX Server to version 3.5 and VirtualCenter to version 2.5, and delivers "new capabilities for increased levels of automation, improved overall infrastructure availability and higher performance for mission-critical workloads," according to a company document.

"This new release delivers the nonstop virtual infrastructure… and better management," says Raghu Raghuram, VMware's vice president of products and solutions. Performance in ESX Server 3.5 is enhanced through support of paravirtualized Linux and large memory pages, read company documents, bringing "significant performance gains," particularly to Java applications and Oracle database workloads.

Performance for Citrix and Windows Terminal Services workloads is improved thanks to support for TCP segmentation offload and jumbo frames, which the company claims reduces the CPU overhead associated with network I/O processing.

Live migration of virtual machine disks between storage systems is now possible with "little or no disruption or downtime," using VMware's Storage VMotion module. The tool also supports dynamic balancing of workloads and can help head off performance bottlenecks. An Update Manager automates the deployment of patches and updates for ESX Server hosts and virtual machines.

A new Guided Consolidation module assists in the initial setup of VMware, automatically discovering physical servers. It identifies candidates for consolidation, converts them into virtual machines and "intelligently places them into the optimal VMware server host."

Also included in the release is an experimental feature that balances workload to help reduce power consumption in the data center, automatically powering servers up and down based on demand.

Send product announcements to
stpnews@bzmedia.com

Improve **Test Productivity** and **Product Quality** through **Integrated Testing** and **Issue Tracking**

- Communicate critical testing information using real-time metrics
- Enable rapid development through process standardization
- Promote the creation of reusable test cases
- Leverage previous results in test planning
- Enforce test case standardization
- Respond to issues more quickly

DevTest
KnowledgeWise
DevTrack
Knowledge
DevSpec
VersionLink
DevPlan

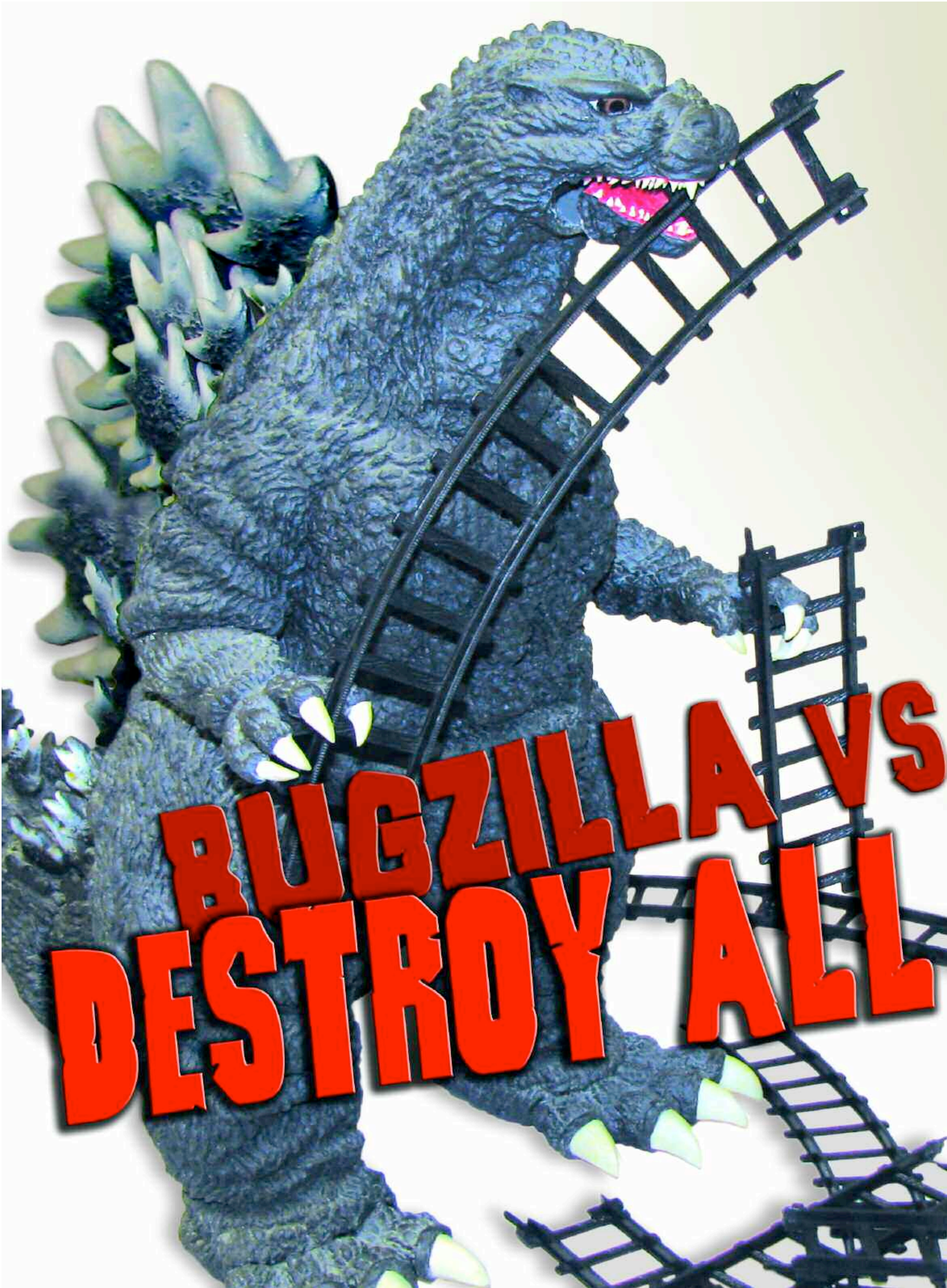
TechExcel **DevSuite**

Whether you adopt just one DevSuite product, or implement the entire integrated suite, you will benefit from a powerful, knowledge-centric solution that is easily tailored to your organization's needs and development processes.

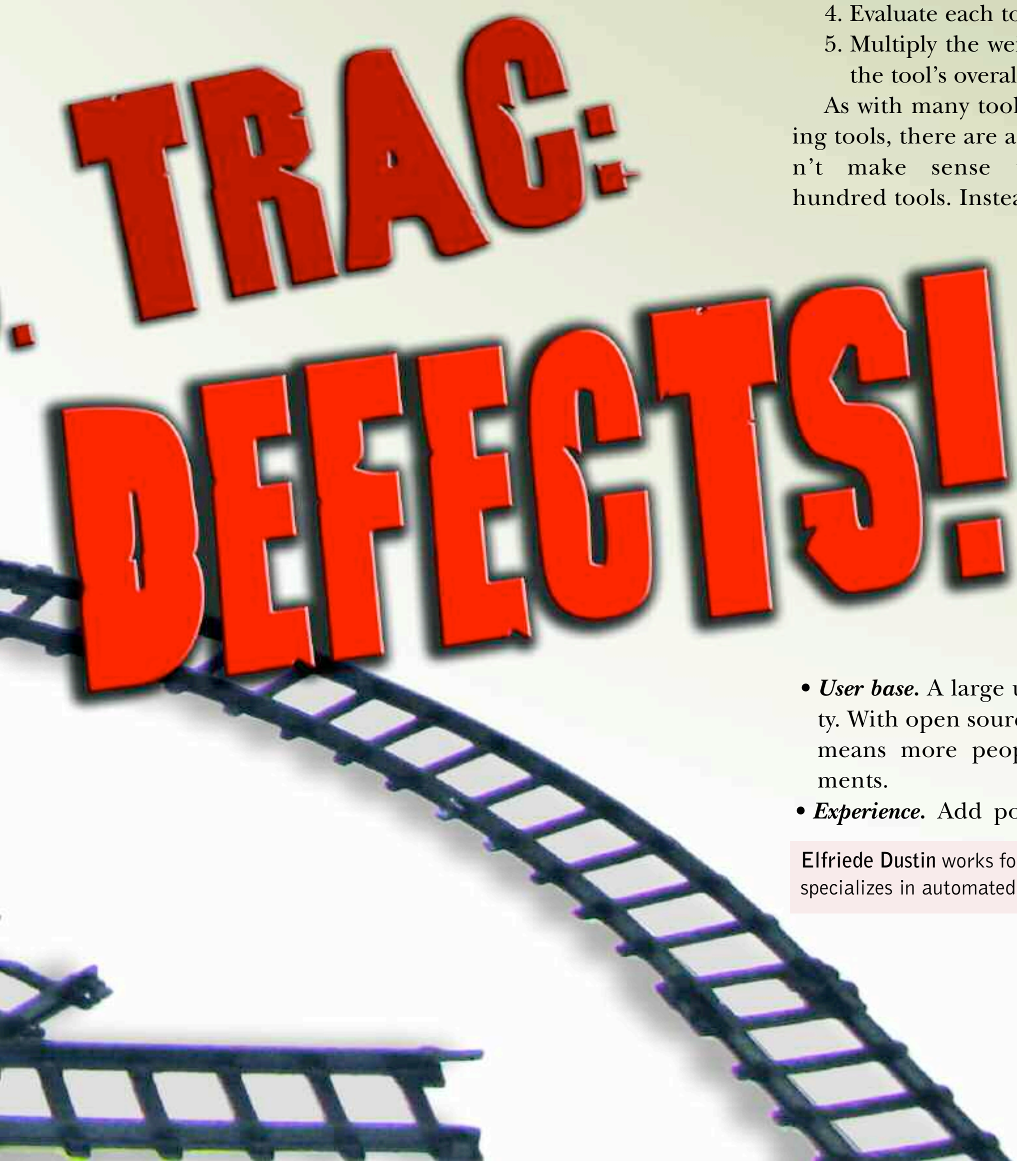
- Supports agile, iterative, waterfall and other methodologies
- Scalable from small teams to thousands of users
- Facilitates close collaboration of globally distributed teams
- Windows Smart Client and Web interface

TechExcel
1-800-439-7782

Download the whitepaper at www.techexcel.com/stp


BUGZILLA VS
DESTROY ALL

Today! An Epic Battle Between Two Top Open Source Defect-Tracking Tools—Don't Miss It!



By Elfriede Dustin

Given any online testing user group on any given day, you'll see this query—"Which is the best tool for *xyz*?"—where *xyz* equals any testing category, such as automated software testing, performance testing, defect tracking, etc. No matter which testing category this specific query is related to, the answer will generally be "It depends."

Finding the best tool for your organization almost always depends on your specific needs and requirements. In this article, I'll explore how to evaluate and choose a tool using a defect-tracking tool as an example, enumerate reasons why open source tools can be a viable solution, and describe an example comparison of two of the "top" open source defect tracking tools: Bugzilla and Trac.

Recently, I was tasked with evaluating defect-tracking tools for a client. We generally approach any type of tool evaluation strategically, as follows:

1. Identify the tool requirements and criteria (in this case, defect-tracking tool requirements/criteria)
2. Identify a list of tools that meet the criteria
3. Assign a weight to each tool criteria based on importance or priority
4. Evaluate each tool candidate and assign a score
5. Multiply the weight by each tool candidate score to get the tool's overall score for comparison

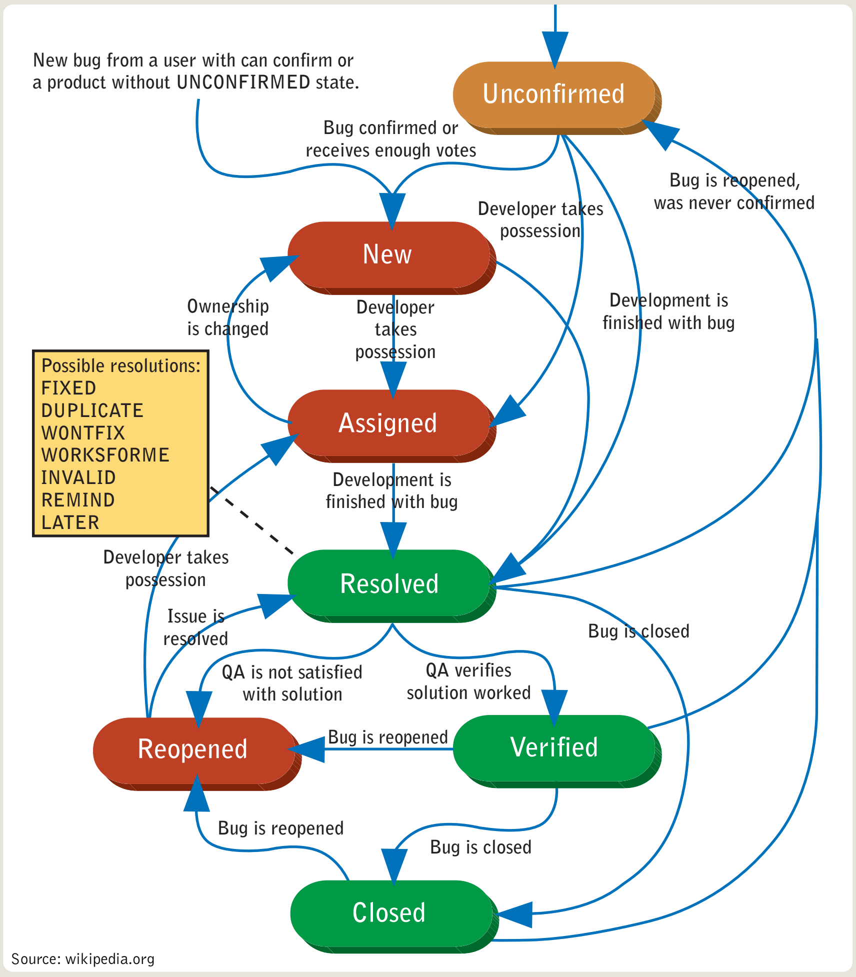
As with many tool categories, in the case of defect-tracking tools, there are a good many choices. Of course, it doesn't make sense to implement these steps for a hundred tools. Instead, it's a good idea to narrow the broad list to a select few. This can be done using such criteria as:

- ***Requirements.*** Does the tool meet the high-level requirements? For example, if you're looking for Web-based tools and several work only as client/server, those would not be considered.
- ***Longevity.*** Is the tool brand new or has it been around a while? Darwin's principle of "survival of the fittest" applies to the tool market. Decide on a number of years of existence and eliminate the rest.
- ***User base.*** A large user base generally indicates good utility. With open source, a busy development community also means more people providing feedback and improvements.
- ***Experience.*** Add points if you or your client have had a

Elfriede Dustin works for Innovative Defense Technologies (IDT), which specializes in automated testing.

Photograph by LuAnn T. Palazzo

FIG. 1: BUGZILLA DEFAULT WORKFLOW (v.3.0)


New bug from a user with can confirm or a product without UNCONFIRMED state.
Unconfirmed
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
New
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions: FIXED DUPLICATE WONTFIX WORKSFORME INVALID REMIND LATER
Assigned
Development is finished with bug
Developer takes possession
Resolved
Issue is resolved
Bug is closed
QA is not satisfied with solution
QA verifies solution worked
Reopened
Bug is reopened
Verified
Bug is reopened
Bug is closed
Closed

Source: wikipedia.org

good experience with a specific tool that meets high-level requirements, longevity and user base criteria described.

Using these criteria, I narrowed the field from the scores of defect-tracking tools to four: two commercial and two open source. This article compares and contrasts the latter two, Bugzilla and Trac. Coincidentally, Bugzilla is widely used by the open source community (in Mozilla, Apache and Eclipse) and was selected "Testers Choice" by the readers of this magazine, announced in the December issue (see "Bugzilla: The 2007 Testers Choice" sidebar for more).

COTS vs. Open Source

Commercial off-the-shelf solutions certainly have their advantages, including published feature road maps, institutionalized support and stability (whether real or perceived). But buying from a software vendor also has its downsides, such as vendor lock-in, lack of interoperability with other products, lack of control over improvements, and licensing costs and restrictions.

And while a few of those downsides might be applied to some open source projects, the advantages of leveraging the open source community and its efforts are holding sway with more and more companies. Advantages of open source include:

- No licensing fees, maintenance or restrictions
- Free and efficient support (though varied)
- Portable across platforms
- Modifiable and adaptable to suit your needs
- Comparatively lightweight
- Not tied to a single vendor

Licensing. Trac is distributed open source under the modified BSD License (trac.edgewall.org/wiki/TracLicense), a "commercially friendly" license sometimes known as a "copycenter" license (take it to the copy center and copy it). It permits changes to be made to source code and kept or distributed commercially with few restrictions.

Bugzilla is covered by the Mozilla Public License (www.mozilla.org/MPL), which is sometimes described as a BSD/GPL hybrid. This so-called "weak copyleft" license requires copies and changes to source code to remain under the MPL, but permits such changes to be bundled with proprietary components.

Support. During my Bugzilla and Trac evaluations, my questions were often answered within minutes, solving any issues I ran into from the simple to the complex.

In my experience working with commercial tool vendors, support is usually more complicated. It can take days before a support specialist addresses an issue, and there is sometimes an extra cost or annual mainte-

BUGZILLA: THE 2007 TESTERS CHOICE

Bugzilla was at the top of the 2007 Testers Choice Awards, as noted in the December 2007 edition of this magazine. "Bugzilla, competing in a category with commercial products developed by companies with infinitely more money and resources than the open source community from whence it comes," writes Edward J. Correia of the product. "Originally written in Tcl by Terry Weissman, Bugzilla began its life in 1998 as a replacement for the defect tracker used by Netscape for the Communicator suite (it surely must have been quite loaded).

"Thinking another language might get more traction with the community, Weissman decided to port it to Perl, resulting in Bugzilla 2.0. As the Wikipedia story goes, Weissman in April 2000 handed the project off to Tara Hernandez, who succeeded in gaining more participation from the development community. She handed it off to its current custodian, Dave Miller, and the rest is history. Bugzilla won our top spot in the free test/performance tools category."

Download back issues of ST&P, including the December 2007 issue, at www.stpmag.com /backissues2007.htm nance contract to be kept current.

Adaptability. Trac is written in Python (www.python.org). First released in 1991 by Guido van Rossum, this dynamic object-oriented programming language can be used for all types of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned relatively quickly by the average developer.

Versions of Python are available for Windows, Linux/Unix, Mac OS X, OS/2 and Amiga, as well as for Palm and Nokia (Symbian) mobile-phone operating systems. Python has also been ported to run on Java and .NET virtual machines, and is used at organizations as diverse as NASA, Rackspace, Industrial Light and Magic, AstraZeneca, Honeywell, Symantec and many others.

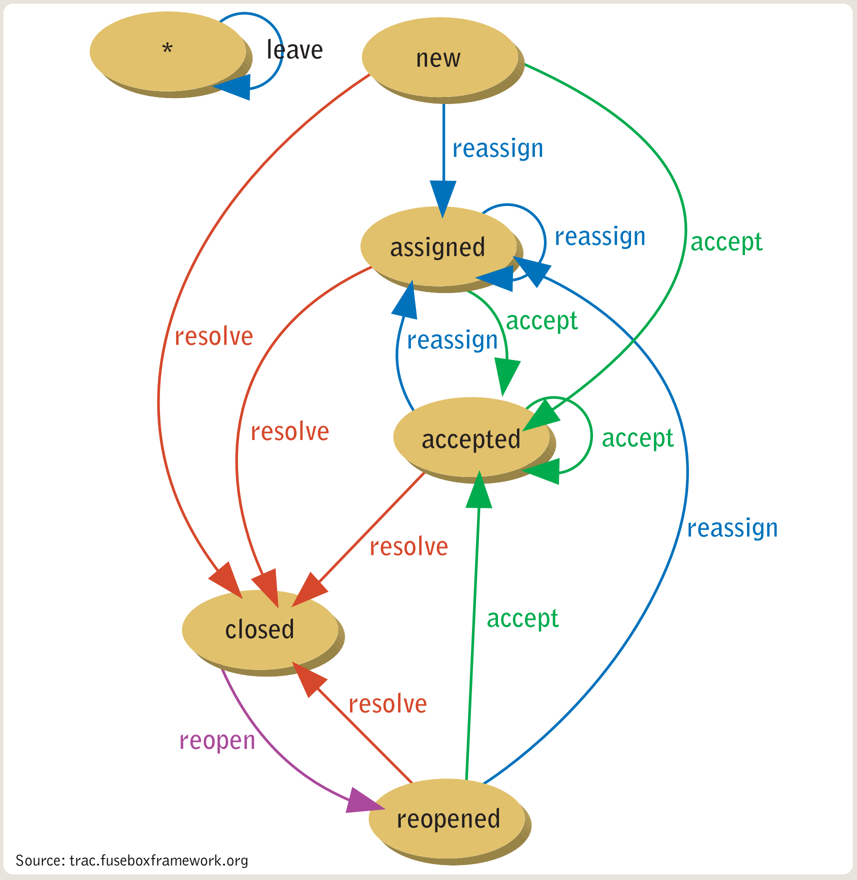
Bugzilla (www.bugzilla.org), in its current iteration, is written in Perl. Perl (www.perl.org) is a stable cross-platform programming language first released by Larry Wall in 1987 that borrows from C, shell scripting, AWK, sed and Lisp, according to Wikipedia. It's employed for many projects in the public and private sectors, and is widely used to program Web applications of all stripes.

Perl is a high-level programming language with an eclectic heritage. Perl's process, file and text manipulation facilities make it particularly well suited for tasks involving quick prototyping, system utilities, software tools, system management tasks, database access, graphical programming, networking and Web programming. These strengths make it especially popular with system administrators and CGI script authors, but mathematicians, geneticists, journalists and even managers also use Perl, according to a history posted at trac.edgewall.org.

TABLE 1: DEFECT-TRACKING TOOLS COMPARED

Defect-Tracking Tool Evaluation Criteria	Bugzilla 3.0	Trac 0.10
Installation on Windows	–	+
Ease of use	+	+
Extra features	+	+
Customizable workflow	+	+
Security	+	–

FIG. 2: TRAC DEFAULT WORKFLOW (v.0.11)



And with the release of Bugzilla 3.0, Trac and Bugzilla both now allow for modification and customized fields.

Community. Trac is hosted at edgewall.org and maintained by a community of developers who collaborate on projects based on Python. Edgewall.org is perhaps best known for Trac.

Bugzilla is hosted at bugzilla.org and is among the many projects administered and maintained by the Mozilla Foundation, which is probably best known for its Firefox browser.

By Definition

As defined by their respective Web sites, Bugzilla is "server software designed to help you manage software development." Trac is "an enhanced wiki and issue-tracking system used for software development projects."

Trac isn't an original idea. It owes much to the many project management and issue-tracking systems that came before. In particular, it borrows from CVSTrac. The project started as a reimplementation of CVSTrac in Python and an entertaining exercise, as well as toying with the SQLite embeddable database. Over time, the scope of the endeavor broadened, a goal formed, and it was brought on its current course.

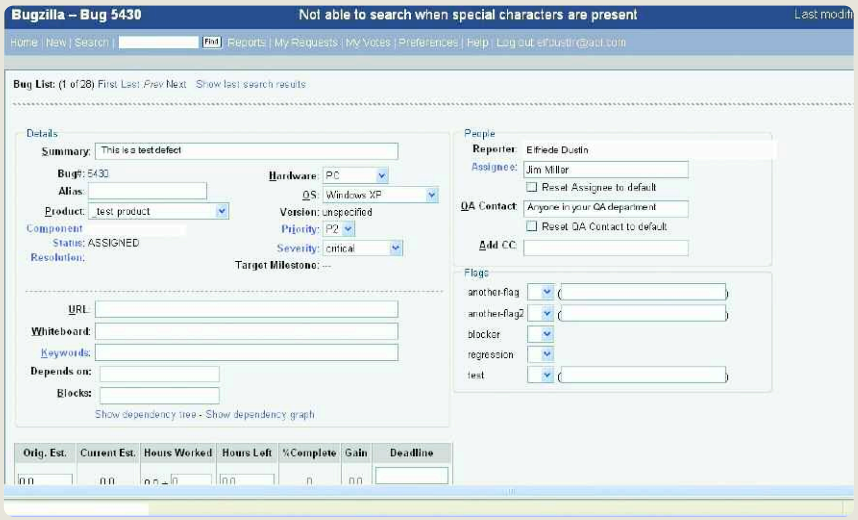
Both Trac and Bugzilla offer browser-based defect tracking with multi-user access, attachments, e-mail integration, import/export capability, custom fields, authentication and authorization, reporting and audit trails.

Once I had identified the two tools I wished to zero in on, I started comparing features beyond their basic capabilities. Some of their major pluses and minuses are shown in Table 1.

Installation on Windows. One of my client requirements was that the tool installs on Windows servers. While most open source defect-tracking tools are cross-platform compatible, Bugzilla (which also runs on Linux) focused less on the Windows environment until its later releases.

Windows installation is a Trac strength. It offers bundled installation and simple setup, installing on Windows without a hitch. Trac also can run as a stand-alone Web server; Bugzilla can't. There is currently no Bugzilla bundle installation package for

FIG. 3: ADD A BUG


Bugzilla – Bug 5430 Not able to search when special characters are present Last modifi

Bug List: (1 of 28) First Last Prev Next Show last search results

Details

Summary: This is a test defect

Bug#: 5430

Alias:

Product: _test product

Component

Status: ASSIGNED

Resolution:

Hardware: PC

OS: Windows XP

Version: unspecified

Priority: P2

Severity: critical

Target Milestone: ---

URL:

Whiteboard:

Keywords:

Depends on:

Blocks:

Show dependency tree - Show dependency graph

People

Reporter: Elfriede Dustin

Assignee: Jim Miller

☐ Reset Assignee to default

QA Contact: Anyone in your QA department

☐ Reset QA Contact to default

Add CC:

Flags

another-flag

another-flag2

blocker

regression

test

Orig. Est.	Current Est.	Hours Worked	Hours Left	%Complete	Gain	Deadline
0.0	0.0	0.0 + 0	0.0	0	0.0	

Home | New | Search | Find | Reports | My Requests | My Votes | Preferences | Help | Log out elfdustin@aol.com

Windows; components must be collected and installed separately.

Ease of use. Once installed, both products are easy to use through Web interfaces that are straight forward and intuitive. Of the two, Trac is more lightweight and requires little overhead, while Bugzilla includes more components and options.

Both tools received pluses in this category.

Extra features. In addition to their main role in tracking defects, both Trac and Bugzilla offer numerous additional capabilities.

Trac also includes a wiki, allowing users to add, remove or edit Web content. This tool can be used for project management or documentation, to provide project status to management and testers at log-in and be leveraged by other departments for their own uses. The environment itself is implemented as a wiki, making every Trac page editable, if the organization so chooses. Trac also is compatible out of the box with the Subversion as well as other SCM systems via numerous plugins.

Bugzilla also integrates with Subversion, but requires help from one of several plugins for the job. Plugins and its extensible architecture are among Bugzilla's main strengths. Scores of plugins have been developed, including several desktop and handheld clients, integration with Eclipse, Mylyn and about a dozen SCM systems. Other free plugins offer dashboards and integration with project and test case managers, IRC, Web apps and data harvesting.

Customizable workflow. The goal of any defect management process is to solve as many defects as possible as quickly as possible. Independent of the defect-tracking tool, a process should be followed to ensure consistency and accuracy in the way defects are logged and tracked. A sound workflow is necessary to ensure that a defect follows a consistent path and doesn't get lost at any point. Therefore, the ability to customize the workflow is an important consideration when evaluating a defect tracking tool.

Both Trac and Bugzilla permit extensive customization. Trac's ticket workflow customization is implemented through plugins; Bugzilla's is done by editing templates written in HTML, CSS and JavaScript. Their default workflows are illustrated in Figures 1 and 2.

Personally, I preferred Trac's option to Bugzilla's, but if your team is skilled in Web-based languages, your choice might be different. Both tools received a plus in this category.

Security is job #1. According to Bugzilla's maintainers: "The current developer community is very much concerned with the security of your site and your Bugzilla data. As such, we make every attempt to seal up any security holes as soon as possible after they are found."

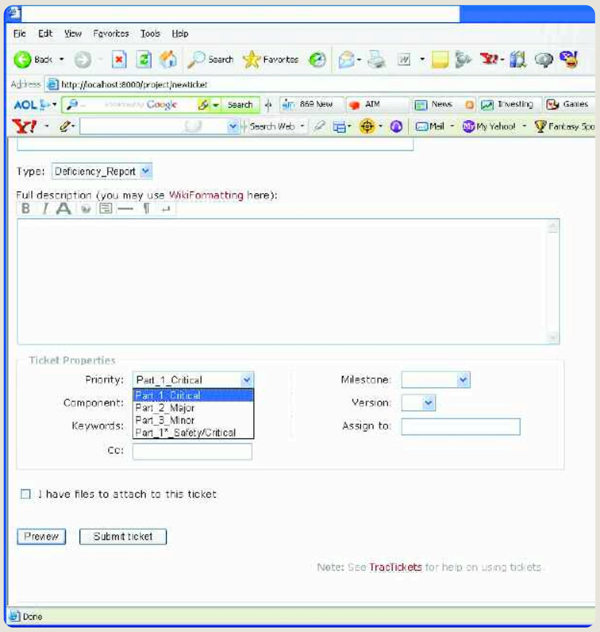
As such, a list of the security advisories issued with each release that included security-related fixes is provided on the Bugzilla homepage. "This is almost every version we've ever released since 2.10," read a statement, indicative of the recent attention being paid to security matters.

When I asked the Trac development team about its attention to security, I got this response: "I cannot give a complete answer, but what I know is that we actively look at code we have from multiple angles to see if there's a potential abuse." I am concerned about such a lax position toward security. In our case, lax security was a deal breaker. And because of the attention to security paid by Bugzilla developers of late, the project's longevity also played a major part; they've had more time to fix security flaws.

After all, Darwin's theory of survival of the fittest also plays a role in the open source arena. And the more active the community and the longer an open source tool has been around, the bigger the user space and the better the chances that security issues have been addressed.

While Trac is lightweight and its plugins and available components integrate seamlessly, Bugzilla offers more add-ons and utilities. If you're looking for a large number of integrations, Bugzilla should get the nod. On the other hand, if you're seeking a lightweight system that's quick and easy to install, Trac is the one. ☒

FIG. 4: TRAC A TICKET


Address http://localhost:8000/project/newticket

Type: Deficiency_Report

Full description (you may use WikiFormatting here):

Ticket Properties

Priority: Part_1_Critical

Part_1_Critical

Part_2_Major

Part_3_Minor

Part_1*_Safety/Critical

Component:

Keywords:

Cc:

Milestone:

Version:

Assign to:

☐ I have files to attach to this ticket

Note: See TracTickets for help on using tickets eclipseCON™ 2008

5th Annual Gathering of the Eclipse Community

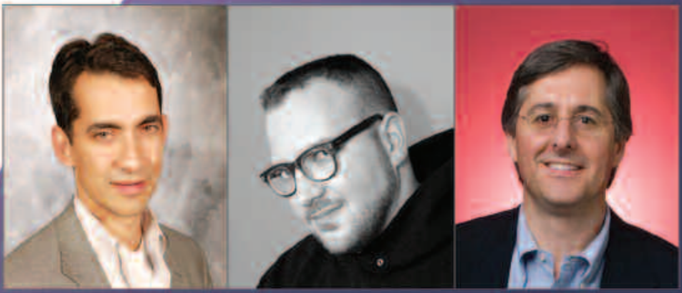
Attend Eclipsecon 2008

EclipseCon is the premier technical and user conference focusing on the power of the Eclipse platform. From implementers to users, and everyone in between, if you are using, building, or considering Eclipse, EclipseCon is the conference you need to attend.

Over 150 Sessions and Tutorials Including:

- Business
- C/C++ Development
- Database Development
- Industry Vertical
- Java Development
- Mobile and Embedded
- Modeling
- OSGi
- Project Mashups
- Reporting
- Rich Client Platform
- SOA Development
- Test and Performance
- Technology and Scripting
- Tools
- Web Development

Keynotes from:



Sam Ramji   Cory Doctorow   Dan Lyons

This is your opportunity to get in-depth technical information from the Eclipse experts, learn the latest tips and techniques for using the tools, network with fellow enthusiasts and experience the breadth and depth of the Eclipse community. Attending EclipseCon will expand your knowledge and make Eclipse work better for you.

March 17th - 20th
Santa Clara, California

Register at:
www.eclipsecon.org

Multi-User Methods Every Field of Dandy

By Karen N. Johnson

Multi-user testing can be fun. That's true because multi-user apps are straightforward to test. Bugs in this category appear either dramatically with splashy database errors, or quietly as the application and database handle the test conditions gracefully and the test cycle ends without incident.

In either case, multi-user testing typically involves relatively short test cycles because the number of objects that need to be tested in multiple user scenarios has, in my experience, not been large. Also, the errors tend to be less debatable than, say, errors uncovered during functional testing. For these, opinions can vary about what the application should do or what the requirements truly meant. Conversely, there are no arguments that a deadlock error is unacceptable.

Overlooked, but Essential

Multi-user testing involves testing an application while simulating two different users executing the same transaction at the same time for the purpose of discovering application, data and database errors. Multi-user testing is a form of testing frequently not talked about and often overlooked. One reason this cycle gets forgotten is that over the past decade, relational database products have matured, and errors in this category may be less likely to occur than they did several years ago. But database software such as MySQL has come to market, and new releases of databases require testing, just as new software releases require testing. Clearly, multi-user testing remains necessary.

As many applications have moved to the Web, focus has shifted to performance testing, for good reason. We've been focused on dozens, hundreds and even thousands of users, not just two. The perceived likelihood that two users would be accessing and updating the same object at the same time is low, low enough to drop multi-user testing off the list of testing to accomplish. But errors from this test cycle reveal that the impact of errors remains high; we don't need to think about dozens of users, we just need two users to create the dreaded deadlock.

Multi-user testing is often mistaken for inexpensive performance testing. Since performance testing and multi-user testing both (sometimes) focus on high-frequency, high-volume objects, the confusion about multi-user testing persists. But looking at the same

Karen N. Johnson is a software testing consultant in the Chicago area.

Photograph by Anna Sirotina

Should Be In Test Designs

Identifying Race Conditions And Deadlocks In Your Apps Can Leave You Smelling Like A Rose

objects doesn't mean the testing focus is the same; multi-user testing is focused on the concurrency of transactions and how concurrency and locking are handled.

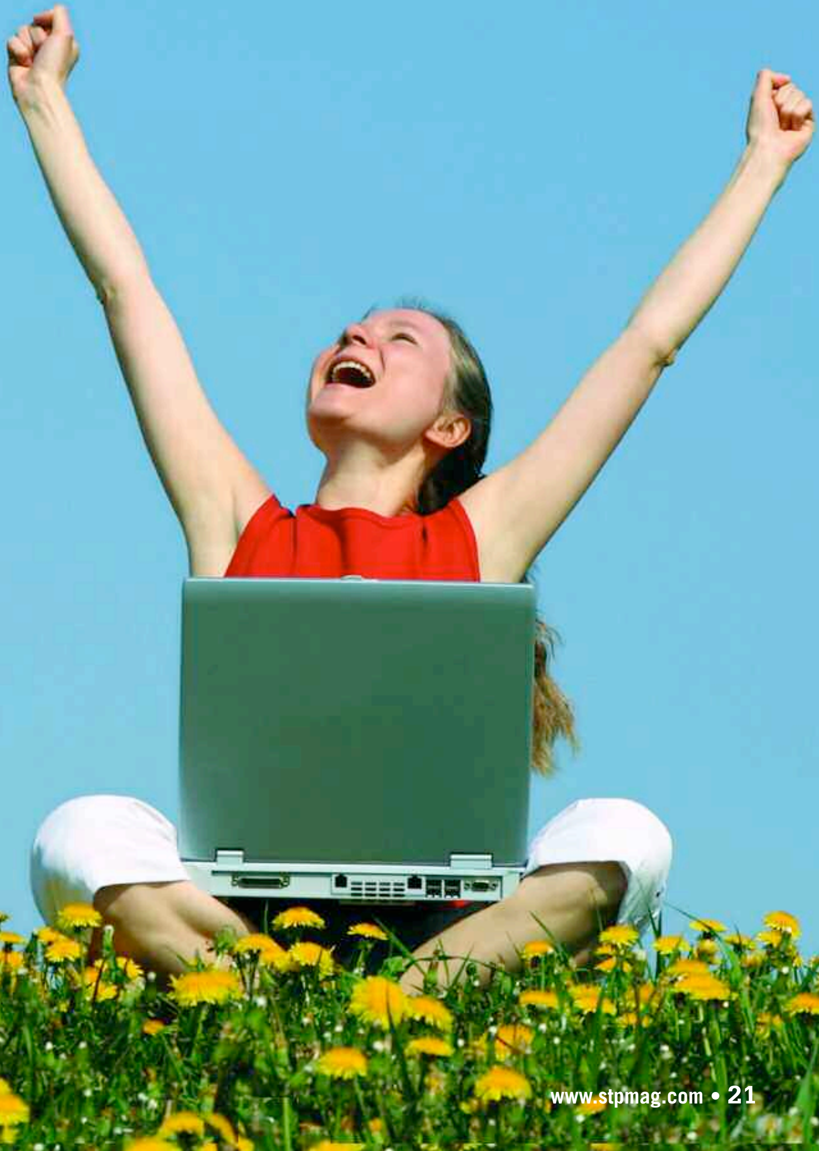
Staggered Timing

Tests in the multi-user cycle involve adding, updating or deleting an object at the same time or with staggered timing. Let's break this description down with an some example. Imagine a Web application that allows users to manage internal documentation for a company: a library management system. This system allows internal users to access company documents and, depending on their permissions, enables them to add, update or delete documents. The application includes functionality to allow administrative users to add users and user groups. This gives us multiple transactions to work with.

Now imagine in the course of a workday, two administrative users attempt to create a new user group at the same time. One user adds the new user group with no error and continues on. The second user encounters a database error referencing something about a unique constraint. Unlikely to happen? Perhaps. But it's not unlikely that two users would be adding or editing documents; in fact, at a large company with a heavily used library management system, dozens of users are likely hitting the same transactions at almost exactly the same time all day long.

Identifying Tests

You can choose from a couple of ways to plan what objects to test. First, look at the database in your production environment when you make this



assessment—which means you might need a database administrator who has access to production. Unless development and test environments contain a recent copy of production data, you won't get the same assessment as production. Even with a production copy in test, you can't be sure the DBA setting up your dev or test environment didn't trim any tables to save space.

A practical way to plan testing is to use your knowledge of the application. What objects are users likely to be "touching" all day long with high frequency? What are the fastest-growing tables in the database? What objects do those tables contain?

When planning your testing program, remember that you don't need to test every object. Instead, you're looking for high frequency and high volume; high frequency because these objects are being used the most and are therefore more likely to encounter errors. High volume is a likely target because these are the fastest-growing objects, which also likely makes them high frequency. Timestamps and version numbers can serve as reference points to determine frequency. In the case of volume, you're looking for high table counts.

What is *high*? Compared to other objects in the database, these are the objects being added and updated more often. If you're conducting performance testing, you might already be acutely aware of what objects generate the most traffic. Use Table 1 to plan multi-user testing.

Once you identify the objects, think about what action is being used the most often. Are the objects being added, updated or deleted? A simple point I've learned in executing this testing is that once I've added an object, I test edit and then delete. This makes the testing move quickly since there's no additional work in setting up objects; I cycle through add, then edit, and my final test even cleans up my test data as I delete as the last test.

Pair Up or Go Solo?

You can test with two people pairing up to cycle through various add, update and delete transactions. Alternately, I execute this type of testing alone, preferring to arrange two fairly equal class PCs as I manage two keyboards and execute transactions. If you choose to go it alone, don't forget to log in to each PC as a different user. After all, the purpose is to simulate two users at the same time—not the same user on two different workstations (which, by the way, is another form of testing.) For equal-class PCs, the same timing is easier to accomplish with PCs of equivalent processing speeds.

If you compare a deadlock to traveling down a highway that uses a tunnel that allows only one car at a time, you can envision the lock.

What to Watch For

Deadlocks. Unique index constraints. Lost edits. Application errors. If multi-user testing didn't sound exciting at first blush, consider these errors in production and you might be willing to allocate a test cycle to multi-user testing. If your testing has been more black-box focused or if you haven't included database considerations in your testing previously, some of these errors might be new to you. Let's examine each error type one at a time.

A *deadlock* occurs when two processes are locked and neither transaction completes. Deadlocks in production can wreak havoc if two users lock a table. If you compare a deadlock to traveling down a highway that uses a tunnel that allows only one car at a time, you can envision the lock. As two cars compete to pass through the entrance first, neither allowing the other to pass, the lock is set. Add a few more cars coming along, like transactions continuing on a Web site, and you can envision a queue growing with frustrated users (or drivers). Deadlocks are ugly.

There are several locking schemas available to prevent deadlocks, and more than one database vendor on the relational database market so there are different locking schemas and concurrency controls. In fact, there are several fascinating problems outlined as stories you can find on Wikipedia, beginning with the entry on deadlocks. Some of the stories are well known, the most popular and the start of the collection is the dining philosophers' problem (see Edsger W. Dijkstra's work). One type of problem and its related teaching story is referred to as the producer-consumer problem, which also brings up the point of race conditions.

Race conditions are a core consideration in deadlocks. Like the tunnel analogy, many traffic issues wouldn't take place without a race condition. Rush hour is a race condition. The same takes place on the database as the timing of transactions becomes an essential factor.

This is one reason I test both same-

time and staggered timings. Staggered timing can catch errors when a process or lock hasn't been released but a user can't view the lock from the application front end. Testing add, update and delete transactions with slightly staggered timings can catch these errors. If the lock hasn't been released, the next transaction will encounter an error.

In my experience in a decade of multi-user testing, I'm more likely to encounter deadlocks with the same precise timing on the creation of an object. This is why I'd rather operate two keyboards than perform pair testing; I can get to the exact same precise moment by my own two hands better than any other way. Plus, I have the patience to execute tests multiple times until I can find the timestamps that make me convinced I've covered the test.

The second most frequent error I encounter is deleting the same object with slightly staggered timing.

In terms of practical knowledge and more immediately tangible ideas for testing, you might look to know more information about the specific database you're working with. Are you working with Oracle, Sybase, SQL Server, Informix, MySQL or another database? Each has different implementations available, so it's worthwhile to talk with your DBA about the concurrency controls that have been implemented.

If you can't get the information you need, test to find a deadlock and then you'll likely get the support and information needed—a harsh but effective approach. As most of the database vendor products have matured, I haven't uncovered as many issues as I did years ago, but multi-user testing still is a test cycle likely to harvest bugs, and since the impact can be significant, multi-user testing remains a risk area worthy of investigation.

Unique index constraints are database errors that occur when two users attempt to add the same information at the same time. One user should be able to add the record, and the second user should be notified of an existing entry of the same value. If the timing is sequential, the user who attempts to add the same record receives an error stating a record of the same value already exists. In some cases, such as with MySQL, unless the database has been defined as a transactional database, all inserts for the table may be halted. These issues are sometimes referred to as *primary key* or *unique key errors.*

TABLE 1: MULTI-USER TEST PLANNING FORM

	Same Timing	Staggered Timing
Add		
Change		
Delete		

Knowing exactly which edit is being made by each user helps to verify that both edits made it into the database.

A challenge with lost edits is whether or not the user is informed. Consider this example: Two users access the same record at the same time, with admin users accessing a user record, and each user updating the user record. For the first user to access the record, the edits will be saved, but the second user might not obtain the necessary lock on the record for their edits to be saved. In the worst case, the user's edits are lost and the user isn't informed. Essentially, the transaction is lost.

This is why, in practice, when I test multi-user editing, I make a point to know what edit each user makes, and the edits made are not the same. In the case of the user record, I might edit the last name field, adding a 1 to the end of the existing name as admin user 1, and a 2 to the end of the existing name as admin user 2. In short, knowing exactly which edit is being made by each user helps to verify that both edits made it into the database.

Too Much Information?

Another test idea to keep in mind while executing multi-user tests is security. Here's a test you can pick up at the same time as multi-user testing.

Review the database errors displayed to find the behind-the-scenes information of an application. Look for database table names, admin account information or directory path information being given away on error messages that share too much information. If your application can trap for database errors, a design decision needs to be made about how much information should be revealed through error messages.

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

This Wikipedia entry relating to deadlocks, which quotes a statute passed by the Kansas state legislature early in the 20th century, is an excellent way to visualize the importance of multi-user testing. And now you have a few techniques to help you implement it.

The Future of Software Testing...

FUTURETEST 2008

Save $200 With Our Early Bird Discount Register Online by February 8

February 26–27, 2008
New York Hilton
New York City, NY

Stretch your mind at FutureTest 2008 — an intense two-day conference for executive and senior-level managers involved with software testing and quality assurance. Our nine visionary keynotes and two hard-hitting panel discussions will inform you, challenge you and inspire you.

Brian Behlendorf on Open Source

Rex Black on ROI

Jeff Feldstein on Test Teams

Robert Martin on Craftsmanship

Gary McGraw on Security

Alan Page on Centers of Excellence

Robert Sabourin on Just-in-Time Testing

Joel Spolsky on Software Testing

Tony Wasserman on the Future

Alan Zeichick moderates two panel discussions on Test Tools and on the Application Life Cycle

A BZ Media Event

SD Times

Software Test & Performance

Gold Sponsors

COLLABNET

empirix

hp invent

www.futuretest.net

Drowning in Chaos?

Pull Yourself Out!

Make Sense Of the Standish Reports And Take Control Of Your Projects

By Robin Goldsmith

Most IT projects are late, over budget and deliver something other than what was expected. Such outcomes are the consequence of arbitrarily mandated budgets and schedules, inadequately defined business requirements and too-little/too-late reactive attention to quality and testing. Learn to avoid these problems by developing objective factual project measures and an environment in which everyone involved takes responsibility for their results.

Looking Into a Train Wreck

Like a train wreck, the Standish Group's periodic CHAOS reports (www.standishgroup.com) about IT project success rates evoke a certain ghoulish fascination. Despite, or perhaps partly because most readers don't recognize the reports' questionable measures and analysis—including overlooking what undoubtedly is the most common real proximate cause of project failures—there's wide acceptance of the reports' basic findings that IT projects seldom are on time, on budget and what the stakeholders want.

While such an unflattering depiction of project effectiveness obviously

Robin Goldsmith is an author and testing consultant specializing in business engineering and requirements.

Photographs by Kristian Peetz

reflects a somewhat contrarian view of IT, there's plenty of room for even more contrarianism. In fact, the seeds of this admittedly contrarian article were born in a letter to the editor and subsequent correspondence with (I believe he'd accept the characterization) contrarian Bob Glass, who may have been the first to publicly question the seemingly sacrosanct inviolable CHAOS findings that the great majority of IT projects fail (see his "Loyal Opposition" article "IT Failure Rates—70% or 10-15%?" in the May-June 2005 IEEE Software).

For more than a decade, the Standish Group has published a series of CHAOS reports that describe embarrassingly low IT project success rates, starting with an abysmal 16 percent in 1994 and improving to 34 percent in 2006 (a number that looks good only in comparison to preceding years' reports—or for a baseball batter). Even though it's now well over a decade old, the 1994 report seems to continue to be the one cited (and read) most, by me and others, primarily I assume because it's the only one available in its entirety for free (find it at www.standishgroup.com /sample_research/register.php). Since subsequent reports are priced prohibitively (for me), I and presumably most people know only snippets of them reported in the trade literature, such as the current 34 percent figure (which was described in SQE's 2/22/2007 Between the Lines e-mail newsletter).

So far as I can tell, though, the Standish Group's methodology, findings and analysis have remained fairly consistent over the years, with changes essentially only in the absolute figures. Consequently, concerns raised by the 1994 report are reasonably likely to remain current.

Is IT Really That Bad?

Measurements must be both reliable and valid. The reports' year-to-year consistency indicates that the measures were made reliably. Glass questioned whether the CHAOS reports are valid and said his personal experience indicated a much lower IT project failure rate.

He may be right, both about his personal experiences and their being representative for the broader IT project population. We can't really tell either for sure, though, because the fact is that nobody else seems to have sufficient suitable objective IT project measures that could validate whether or not the CHAOS reports' main finding is accurate.

Survey responses in general tend to be shaky in both reliability and validity.

In fact, this lack of objectively measured IT project data isn't just limited to formal study reports. It's highly unlikely that (m)any of the organizations surveyed for CHAOS based their answers on reliable and valid facts, which raises serious methodological reasons to question the CHAOS figures' validity. The CHAOS data comes from surveys of IT executives' perceptions regarding project measures. I think we all recognize that survey responses in general tend to be shaky with regard to both reliability and validity.

Even when well intentioned, survey responses often are guesses or, at best, broad approximations. I don't know about you, but I'm asked to respond all the time to surveys that ask questions I don't have good answers to, either because I don't know or because none of the answers actually fits my situation. Nonetheless, it doesn't stop me from answering something, probably with vague impressions or answers that I know have no basis.

Illusion of Precision

Although I'm sure it's unconscious, statistical manipulations can both distort reality and imbue an illusory appearance of precision. For instance, "About 90 percent" sounds much less precise than "89.51 percent." Surveys often ask people to pick ranges of values; say 100-150. Having to give a single answer fails to take into account the pattern of variation in the source's actual project data, which I'm sure most of the CHAOS sources were highly unlikely to consider, let alone have quantified. Thus, the respondent may have felt the 100-150 range was most typical, even though perhaps a few instances are in the 25-50 and 400-500 ranges. It's like the old story of a person whose left foot is in a bucket of boiling water and whose right foot is in a bucket of ice. The average of the two temperatures might be a comfortable 90 degrees, but neither foot feels at all comfortable.

While CHAOS groups within project size categories, it also consolidates across categories, and it's unclear whether its calculations accurately reflect the varying sizes of reported projects. Should a one-month one-person project be given the same weight as a one-year 100-person project? Should a project's size be defined based on the original estimate or the bloated size the report tells us the project eventually reached?

Regardless, to come up with averages, it's necessary to convert each of those 100-150 range responses to an average score of 125. Although the reports do show distributions of responses for the various ranges, they focus on single average overrun percents, which take on an appearance of authority and scientific precision.

Moreover, even if the CHAOS-reported average overruns of 189 percent and 222 percent are absolutely accurate, it's unclear exactly how to interpret these reported overruns. If the budget was $100, does an overrun of 189 percent mean that the project actually cost $189 or $289?

On the other hand, despite such serious methodological issues, a lot of subjective data, including my own, does support the general tenor of the CHAOS conclusions.

For example, when I describe the 1994 CHAOS numbers in my seminars/speeches, I usually ask the participants whether the report reflects their own project experience. Over the years, thousands of attendees regularly have given what seems to be almost always unanimous affirmation. Furthermore, the reports wouldn't be so widely cited and accepted unless people do find the conclusions consistent with their own experiences.

It's All in Your Head

However, Glass also has a point. IT projects ordinarily *do* deliver systems that people can and do use. Both Glass and CHAOS are right, to an extent, and focusing on either's conclusion alone may itself obscure other significantly important issues. To understand this balance more fully, certain interesting psychological factors also need to be taken into account.

CHAOS is correct that IT projects routinely are late, over budget and not entirely what is expected. The aberration is large, and it's really not necessary to quibble over CHAOS' specific numbers. Glass is right that usually projects produce some kind of working system. Once people can start using the system, they get busy with it and tend to forget that it was late, over budget and perhaps only a portion of what they'd expected.

That's fine; life must go on—but this forgetting can involve some bigger ramifications. First, the same scenario recurs, project after project. As an industry, IT tends not to learn, or perhaps tends to learn, but not necessarily the *right* lessons, from our experience. Instead, we've learned to become quite proficient at a development process that reliably repeats failure—late, over budget and not what's expected—and then through denial essentially accepts that failure as adequate.

Second, by failing to relate our own actions to our results, we prevent the personal initiative needed to make meaningful improvement. Consider a perceptual paradox phenomenon that's evident in the worlds of politics and culture. For example, survey after survey finds low approval ratings of Congress, yet congressional re-election rates historically are close to 100 percent, sometimes not even impeded by incarceration or death. Similarly, surveys repeatedly find that people say the American education system does a terrible job, but their own local school is just fine.

People often have an understandable disconnect relating broad-scale external results to their personal sphere. That's a major reason why people seldom change their individual behaviors in response to well-known big problems. Annual reports of lung cancer death statistics tend not to cause individuals to stop smoking. The scientific community's agreement that global warming will deplete animal life in the oceans and submerge coastal cities doesn't cause individuals to cut their gas guzzling. Statistics on the damaging health impacts of obesity don't make individuals eat better or less. And CHAOS's reported 84 percent IT project failure rate doesn't cause individuals to change how they do projects.

Perhaps the problem seems overwhelming, or one's own power to affect it seems so insignificant, but ultimately it comes down to the normal psychological defense mechanisms people enlist unconsciously to protect their self-images. We've gotten so good at denying anything that reflects poorly on us, so unwilling to recognize, let alone take responsibility for our results, and so willing to shoot the messenger, that we not only fail to take appropriate corrective actions but also sometimes intentionally engage in additional self-defeating behaviors. For example, consider the dysfunctional behaviors that get acted out even more excessively every afternoon on the television scream shows.

The Real Cause of Failure

It's not only pathetic TV wanna-be celebrities who respond to dysfunction with greater dysfunction. Many, if not most IT projects are destined from the start for failure because management has arbitrarily dictated a project budget and schedule that bears no relationship to the work to be done.

But that's just the beginning of a downward spiral. When budgets and schedules are nonsense, overrunning them becomes nonsense too. So what if nonsense budgets and schedules are overrun 189 percent and 222 percent of the time? People doing the projects don't take a personal stake in the outcome because they go into the projects "knowing" the nonsense budgets and schedules are impossible to meet, which becomes a self-fulfilling prophecy, regardless of how "objectively" feasible the budget/schedule may be.

The more the worker bees grumble and miss their targets, the more the managers feel compelled to dictate yet-even-more nonsensical budgets and schedules, thereby ensuring failure and confirming to them that they were right in having to hold the troops' feet to the fire. Dysfunction begets more dysfunction.

The business users don't know about these internal dynamics. They only know that as an industry, IT almost always blows project budgets and schedules. They perceive that IT doesn't know what it's doing, and thus they may not believe what IT says, which further impedes IT's ability to keep its promises.

Project managers' psychological defense mechanisms prevent them from becoming aware, let alone believing, that they may have credibility issues. So they attribute their difficulties to other, often-irrelevant factors and mistakenly divert attention to these perhaps non-issues instead of addressing their real credibility problems. This further reduces their likelihood of success and their already-diminished credibility. The business puts more pressure on IT management, which leads to even more nonsense dictates and more overruns; and the cycle perpetuates.

Real business requirements tend not to change nearly so much as people's awareness of them.

Close, But Missing Critical Distinctions

Ultimately, project budgets can't help being nonsense unless they're based on adequately defined real requirements. At first glance, this seems very much in line with the project success and failure factors that CHAOS analysis identifies.

Requirements and user involvement issues certainly dominate the tops of the factors lists. They indeed are important, but I fear the report simply parrots overly simplistic, widespread conventional beliefs that continue to miss the distinctions critical for making meaningful improvement.

The report focuses only on amount of user involvement. While a certain *quantity* of user involvement is necessary for discovering the real requirements, it's not sufficient. Rather, it's the *quality* of that involvement that really matters. Merely subjecting users to more of the same ineffective current practices won't produce better requirements.

It's understandable that the report fails to realize this distinction, because the industry is unaware that such a distinction exists. Perhaps one reason why user involvement is low is that managers may sense that just giving more time by itself often may not pay off.

Similarly, the report mirrors the industry's general lack of awareness of the important distinction between real, business requirements and product/system/software requirements.

The common use of the term *requirements* refers to the requirements of the product, system or software that is expected to be created. Said product, system or software actually is the high-level design of one of the possible ways how to accomplish the presumed real, business-requirements deliverable *whats* that provide value when delivered/accomplished/met/satisfied.

The CHAOS report identified incomplete and changing requirements as two separate issues. In fact, the main reason that product/system/software requirements change is because they aren't defined accurately and completely enough in the first place, which in turn is mainly due to the failure to adequately discover the *real* business requirements. Designs (including their product, system and/or software requirements) can change frequently and rapidly. Real business requirements tend not to change nearly so much as people's awareness of them.

Missing Resources And Unrealistic Expectations

The main remaining failure factors identified in the report include lack of resources, unrealistic time frames, unrealistic expectations, and lack of executive support. The first three are all largely attributable to not adequately defining the real business requirements and the product/system/software requirements to satisfy them, which of course contributes to management's propensity for arbitrarily establishing budgets and schedules.

Once a project is set to fail, albeit by management's actions or absence thereof, managers quickly distance themselves from the project.

One More Little Oversight

Roughly half the time spent in IT projects is taken up by testing, yet the CHAOS report's failure factors don't mention quality or testing. Again, this probably reflects a lack of awareness among the surveyed industry executives, which in turn translates into the project failures described. Inadequate quality clearly causes users not to receive what they expect, and the unplanned time needed to fix defects is a major contributor to project budget and schedule overruns.

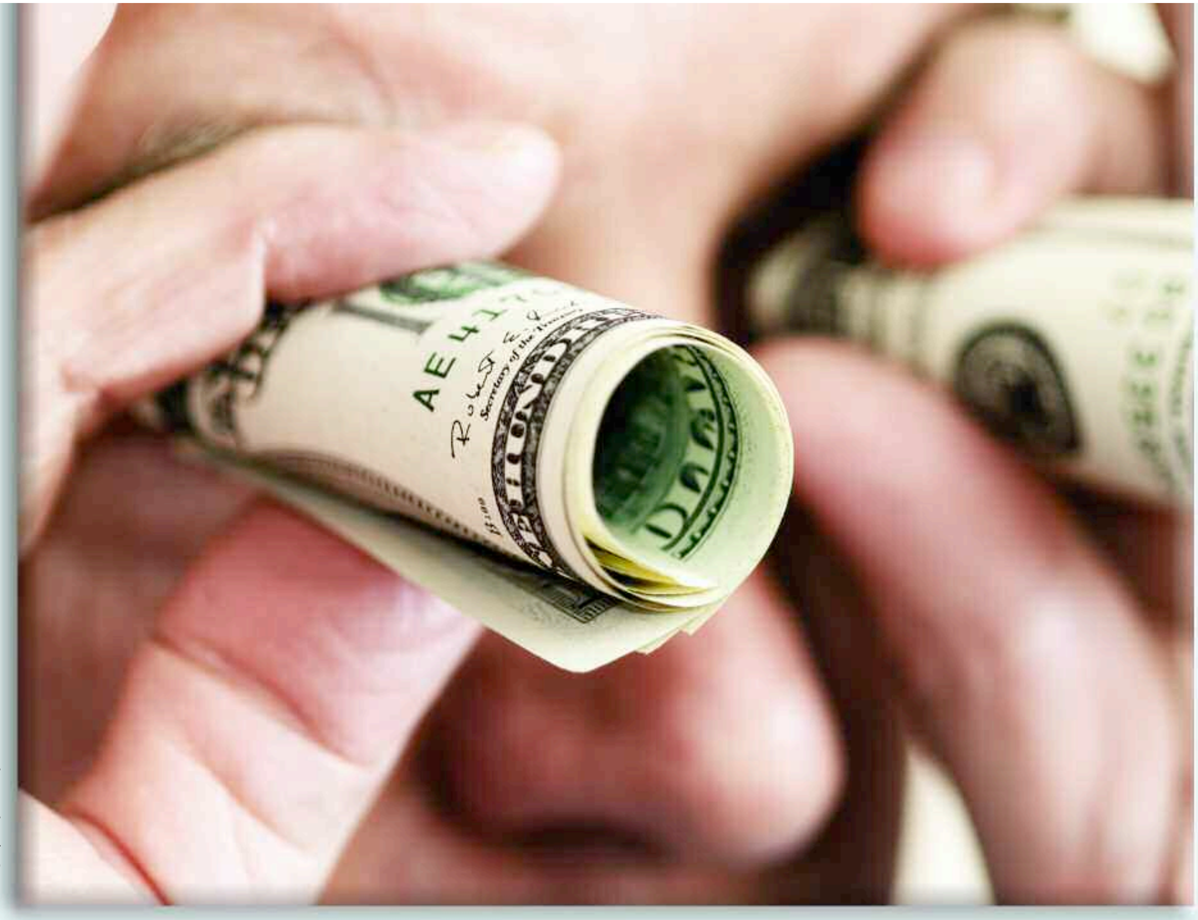
Without awareness of—and informed attention to—quality and testing, quality and testing activities are too little and too late. Too many defects escape to be found by the users. Those defects that are found during development tend to be discovered so late that they're very expensive to fix. Proactive testing can turn this situation around, actually helping projects deliver quicker and cheaper by catching and preventing more of the errors earlier, when they're easiest to fix.

All in all, in spite of significant questions about their measures and analysis, the Standish Group's CHAOS reports seem generally accurate in finding that most IT projects are late, over budget and not entirely what is expected.

To a considerable extent, such outcomes are the inevitable consequence of arbitrarily mandated project budgets and schedules, inadequately defined business requirements and too-little/too-late reactive attention to quality and testing.

Such problems persist in part because few organizations have suitable or sufficient objective factual measures of their projects and because those responsible for IT haven't created an environment in which they and other involved individuals take enough personal responsibility for their results.

The solution? Know, document and adhere to the *real* requirements.


Photograph by Alexey Klementiev/Fotolia.com

On the Lookout For SOA Profits

The Key to ROI's in The Quest for Quality

By Frank Grossman

Quality—you know it when you see it. Yet the concept of quality has little meaning unless it's related to a specific service, object, experience or desired result. As the use of service-oriented architecture moves from early adopters to mainstream corporate initiatives, companies are struggling to achieve their intended goals of business agility and cost savings from service reuse. Some industry watchers and media surveys report that many, if not most companies are engaged in some form of SOA initiatives—but very few are at the point of realizing a positive return on their investments.

It should come as no surprise that the companies seeing positive returns from their SOA initiatives are the early adopters in vertical industries such as financial services, telecommunications, energy, insurance and healthcare. These global leaders in technology optimization have embraced initiatives such as SOA as a core business strategy and are realizing positive returns. They also take a pragmatic approach, knowing that a successful SOA initiative depends on SOA quality.

Frank Grossman is president and cofounder of SOA test-tool maker Mindreef.

Web Services vs. SOA

What exactly is SOA quality? First, let me tell you what it's not. Software quality initiatives are usually associated with testing. And more recently, testing often means testing a single Web service using a waterfall approach. This might work for a Web service that has a distinct function with a specific life cycle, and can be designed, developed, debugged and deployed. It then exists in production until being revised, updated and replaced with a new service, which is subjected to the same cyclical process to ensure quality.

On the other hand, an SOA doesn't have a life cycle. SOA is an architecture made up of infrastructure and services that must constantly interoperate. It doesn't go offline and it can't be replaced. It can, however, evolve and expand. It can also improve or degrade. Therefore, the quality of an SOA is reflected by the amount of use and reuse of the services within it, and by how well its implementation meets the needs of the business, even as those needs evolve. See "SOA Quality Defined" for Wikipedia's take on SOA quality.

A common misconception among SOA project leaders is that quality concerns can be addressed with a governance solution alone, typically in the form of a registry/repository. Vendors have positioned governance solutions as the "law enforcement" for an SOA by controlling service implementations with the process, procedures and standards deemed necessary for quality.

While SOA governance solutions vary in functionality, governance must be applied to design time, change time and runtime to be truly effective. The challenge is that each phase requires a different approach to maximize the enablement and adoption of governance controls. In other words, developers require education more than enforcement when it comes to building compliant services (that is, don't just tell them a service is not accepted, show them why and offer guidance on how to fix it).

Photograph by Amanda Rohde

Architects and analysts require more than just a listing of available services; they need translation of the XML-formatted descriptions into something human-readable that they can easily understand.

Most registry/repository systems focus on design-time governance in the same way they address runtime. They're designed to regulate access, security and performance by consumers. Without varying approaches, the strict enforcement becomes detrimental to the SOA's intended goals. If developers grow frustrated and unconvinced of the standards and policies applied by the registry, it often leads to development of rogue, redundant services for specific applications in order to meet their project dates. Not only is quality at risk, the fundamental value of the SOA initiative breaks down.

One approach to achieving SOA quality is to deploy an SOA quality gateway along with governance solutions. This can complement the control aspect with interfaces designed for various roles within the process and different approaches for design time, change time and runtime. The SOA quality gateway then becomes the quality enablement point for registries.

Collaborative Quality

In any SOA, the more loosely coupled services are, the more communication is needed among every team and member involved in the design, implementation and support processes. Agility and interaction are constantly at risk in these loosely coupled environments because of the temptation for each group to be autonomous.

Collaboration begins at design time. Architects and business analysts need to be aware of existing services, what they do and if they're being used by other applications. Knowing that a Web service is already in use provides a level of trust that is essential to service reuse and the overall success of an SOA. At the same time, developers need to know that quality services exist before building a redundant service.

To ensure SOA quality, collaboration must continue during runtime and change time. Once services are identified and understood, architects can begin prototyping an application. Additionally, to obtain SOA quality, business analysts, developers, QA and support must find an easy way to work together, regardless of language or platform. Traditionally there hasn't been a way for them to effectively communicate, since their functions are so different. A collaborative approach for SOA implementations will help increase the agility and reuse of Web services by ensuring quality and building trust with service consumers.

The Five Components

SOA quality doesn't exist in any single part of a system, nor is it the sole responsibility of an individual or team. The architecture must have SOA quality throughout. To do this, you must build a solid foundation for quality consisting of five core traits:

- Compliance
- Prototyping
- Testing
- Diagnostics
- Support

As with any foundation, if one of the components is weak or missing, the entire structure is at risk.

Compliance. SOA quality starts with compliance with standards. Non-compliant services pose the highest risk for SOA quality and positive business returns, and simply can't exist if an SOA is to be successful. Even well-written services can't guarantee broad interoperability unless standards and best practices are well designed and adhered to throughout an organization.

Developers must embrace standards, rules and policies by seeing value in what they provide. Architects and governance teams must educate others as to what the standards are, why they're in place and most importantly, how developers can improve their projects by becoming compliant. Finally, analysts and architects must embrace compliant services and leverage them in SOA applications to ensure trust and reuse.

Prototyping. Seeing is believing. As business and IT groups work together on SOA initiatives, prototyping is one of the best ways to reach agreement on a WSDL contract before any code is written, and to deliver SOA quality early.

Prototyping lets business analysts, architects and developers design and develop very usable interfaces early in the process, thereby creating services designed for reuse. It also allows consumers and testers to get involved much earlier in the design process, reducing the overall development cycle.

Strong diagnostics are essential to providing and maintaining SOA quality

Testing. SOA has many moving parts. It's virtually impossible to test every Web service and its interaction with its dependencies within an SOA. And unlike traditional software applications, SOA testing occurs while many services are in various stages of development or production. Yet with service-oriented architectures, testing is unavoidable.

Therefore, teams need tools that can provide the necessary user interface, simulate unavailable services and ensure that all team members—even ones without programming or XML language knowledge—can test services. It's also important that test scripts are accessible so that retesting can be easily performed when a service policy changes or if a new service consumer uses the service in a different way.

Diagnostics. Remember, this is still software. So no matter how well Web services and business processes are tested, problems will still occur. With an SOA, problems often need to be solved in real time, and may involve disparate teams and systems. This requires collaborative diagnostics. Determining if a Web service can perform its intended function is often a time-sensitive issue that may require fast identification of the root cause of a problem. This also means that all members of the team are responsible for helping to diagnose problems. For this to be effective in an SOA, strong diagnostics are essential to providing and maintaining SOA quality during runtime, and for preventing runtime issues by catching them in design time and change time.

Support. The final component in the foundation of SOA quality is support. It is absolutely essential to have a mechanism for supporting the disparate groups that use services, without overwhelming the development teams. Also, support is fundamentally different in an SOA because it involves two phases that span design time and runtime simultaneously.

As services are exposed for use and reuse, consumers will seek support to assist in the development of their applications. In production, support teams need to understand and resolve problems quickly and often need to reproduce scenarios when a failure occurred. When service developers need to get involved, complete problem data needs to be shared with other team members who can simulate different scenarios to more effectively diagnose problems.

SOA QUALITY DEFINED

In Wikipedia, SOA quality is defined as "a service-oriented architecture that meets or exceeds business and technical expectations by consistently yielding value in the form of cost savings, productivity and time to market."

This is achieved through continual optimization of all components within the SOA environment to ensure maximum adoption, business agility and service reuse. Therefore, SOA quality is a key component of reaping the intended benefits of an SOA—it's the strategy needed to achieve maximum business benefit.

Communication, Trust and Control

Communication. The ability to communicate effectively is an essential core element in a successful SOA. It's as important to SOA technologies as it is to the people involved with the environment. Communication in a technology domain is often referred to as *interoperation* and occurs across different platforms, languages, locations, policies and standards.

Communication must take place between people, because individual team members depend on each other to do their jobs, complete a task, solve a problem or contribute to a project. The communication between business and IT is a perfect example. This is often associated with collaboration.

But communication is more than just interoperability and collaboration—it involves the interaction between people and technologies, which is often where quality breaks down in an SOA.

Visibility is an important attribute offered by a registry, providing a catalog of services that are available for use. While this helps architects and developers to communicate, visibility alone is not enough. Services need to be visible, accessible and easily understood to ensure SOA quality.

Accessibility and understanding help make visibility a form of communication in an SOA. Together, they allow team members to easily view and comprehend what a service can and can't do, which can be a challenge when interacting with a technical registry that lists a myriad of XML files and WSDL contracts.

An SOA quality gateway can provide the accessibility and understanding attributes to an existing registry and address the requirements for effective communication within an SOA.

Trust. The primary characteristics of SOA quality for any business are agility and reuse. Trust is what drives service reuse. It's critical for SOA teams to be aware of existing services in the SOA, to understand what they can and can't do, and most importantly, to have trust and confidence that the services will execute as intended.

Trust is relevant in nearly every aspect of an SOA. Architects need to trust the services they choose to use. Developers need to trust that an existing service will be appropriate for an application versus building a new one. They must also trust that the implemented policies and standards are meaningful and add value. SOA leaders must trust that services are being added to the registry and reused. And QA teams must trust that policies and standards will enable compliance and interoperability at runtime.

Without trust, an SOA will never achieve reuse, which leads to redundant or rogue services. Trusted services, however, will lead to "socialized quality." As services gain trust, users will share their positive experience with peers—the trust factor grows and reuse increases. This is what we mean by socialized quality. SOA quality optimization depends on creating trusted services and then socializing that quality throughout the SOA.

Control. Governance in an SOA is about control. Often what is lacking, however, is the ability to enforce policies. Control without enforcement really isn't control at all, and will not yield SOA quality. Although governance alone isn't the answer to SOA quality, it's a critical element, and comes into play at design time, change time and runtime.

BUILDING BLOCKS OF SOA

When building a foundation for SOA quality, consider the following:

- How will the individual or team responsible for the overall quality of your SOA manage all the facets of SOA quality?
- How do you ensure that best practices are actually followed, once they're defined?
- How do you enable team members to communicate effectively, given diverse development environments and skill levels?
- How do you enable team members, especially those are remote or who don't have advanced coding or XML skills, to test effectively?
- How do you efficiently reproduce and solve problems that span disparate systems and disparate teams without finger pointing?
- How do you achieve the reuse of your SOA service assets across your team?

Look for platforms and tools that allow team members to:

1. Define executable standards and specifications early in development that team members must develop to. These help ensure that your Web services will be of high quality.
2. Create and run analysis profiles to verify that your services meet the best practices defined by your organization.
3. Save services in reusable workspaces that all team members can run, regardless of skill level. This promotes team communication.
4. Save workspaces, simulations and test scripts that all members of your team can run to test your services.
5. Run previously saved simulations to effectively reproduce, diagnose and solve problems.
6. Save services simulations for reuse of your existing SOA service assets in new implementations.

Governance involves policy enforcement and sometimes requires changes in human behavior, development concepts and processes. Control also involves reducing the number of production issues during runtime and resolving those issues quickly.

Further, control involves understanding the dependencies applications have on multiple, disparate services, and prompts the need to test services that depend on other services that might not be directly accessible.

Quality In = Quality Out

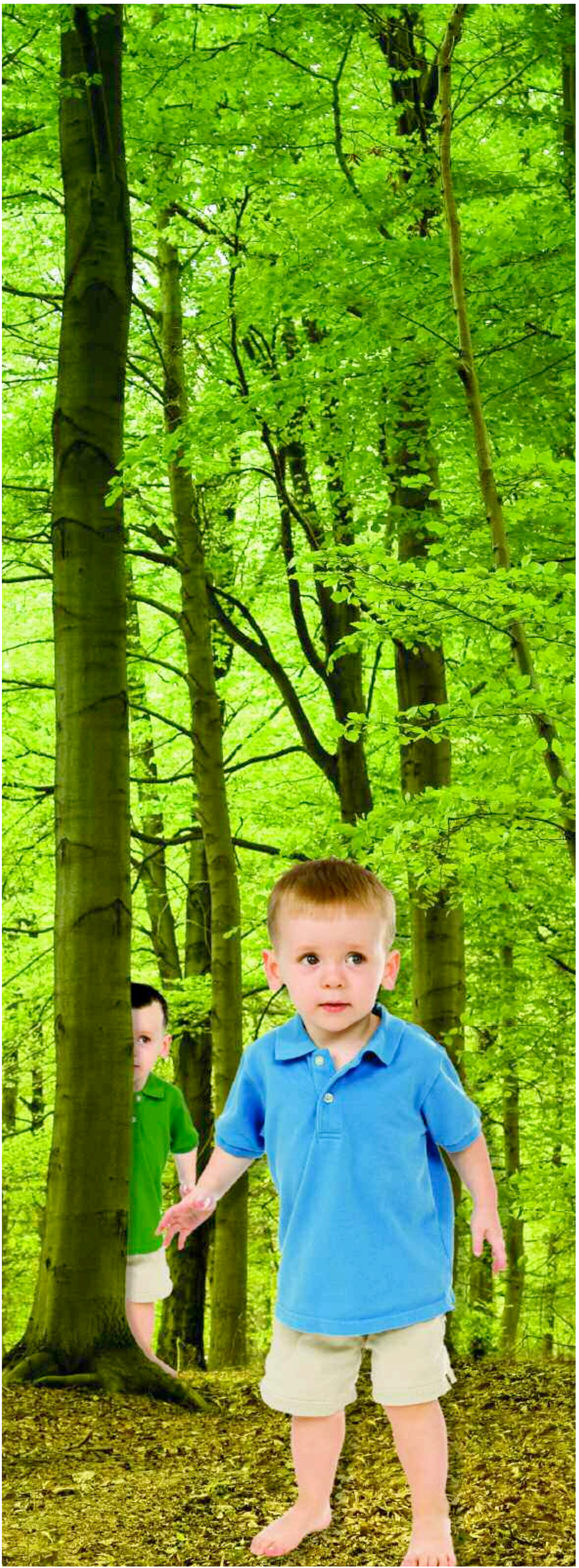
This variation of the old "garbage in, garbage out" adage from the computer science field is relevant to the SOA registry/repository concept. If quality isn't enforced when services are entered into a registry, quality is at risk when services are used from that registry.

While registries are an essential element of SOA governance and governance is about control, it goes well beyond simple governance to ensure SOA quality.

Achieving Business Objectives

As with any business application, the SOA's ultimate goal is to help an organization achieve its business objectives. And SOA quality is required to ensure that the service-oriented architecture meets or exceeds business and technical expectations, whether in the form of cost savings, productivity, better customer service or shorter time-to-market. Achieving this quality means understanding the overall goals of the SOA strategy and optimizing the environment for successful execution.

SOA quality also requires a top-down approach to mapping out those objectives and a bottom-up strategy for maintaining it. By laying a foundation for quality at the outset of any new project, companies can establish a process for success that will become a standard practice as new services are added—ensuring consistent trust and reuse, and a high-quality SOA at all times. ☒



Newbie Tester: A Babe In The Woods

A Case Study That Shows How Not To Staff a Project

By Prakash Sodhani

I've long wanted to write an article about manual testing. As an automation specialist, I don't think much of testing efforts that include only manual testing. But I firmly believe that a prerequisite for automation is a somewhat stable application, which can be ensured only by doing at least some manual testing.

Let's look in on a manual testing project that went awry. It involves someone we'll call Peter working in a company that sells merchandise over the Internet.

Recently, Peter worked on a project as the lead tester. The project, referred to internally as "the Rewrite," involved an existing application that was being rewritten using newer technologies. The app was designed to allow users to come to a Web site and purchase products. Nothing very special so far; an e-commerce app with lots of process flows and state combinations to be tested. But since it was being rewritten, it carried the potential for lots of new defects in the system and lots of time to test all functionality from end to end.

Most of the testing was of the manual variety, with a bit of load testing thrown in for flavor. What follows is a summary of Peter's experiences. Each topic is divided into three parts: What was done, what went wrong, and what I believe *should* have been done.

Prakash Sodhani is a quality control specialist at a global IT services company based in Texas.

Resource Allocation and Planning

What was done?

The development team followed Scrum, an agile methodology. Development was divided into different sprints, each one month long. Each tester was assigned to different sprints based on a percentage determined by the testing manger.

Prior to the initial launch, the original application development team included about 14 developers. The Rewrite project was assigned one of the original testers, who'd been testing this application for the past couple of years. Unfortunately, the manager approved a month-long vacation for him during this time.

So we were assigned only two testers for this project; one, a new hire who joined the team a couple of weeks back, was allocated 100 percent to this project. The other was Peter, who was allocated 50 percent. So in essence, the project had 1.5 testers.

What went wrong?

Since we were just starting the Rewrite project, it was an educated guess as to how many testers and how much of their time would be needed. When this application was first developed, seven testers were assigned, with 100 percent of their time allocated to testing. So it made no sense to us to assign only two testers, both of whom were new to the project.

Also, no planning was done. Management simply assumed that this sprint would be like any other and follow the existing plan. But as the project progressed, the number of application features increased by about 50 percent. To make matters worse, no one realized that we'd lose three days of testing due to holidays.

What should have been done?

First and foremost, at least one tester with experience with the application or a similar project should have been present at all times. No logic can justify the decision to assign 1.5 testers to a project that originally required seven. Managers also should have sought input from at least one tester familiar with the original application to determine how much work was involved.

Test planning also should have been made a priority for a project of such importance to the business. Scrum Masters and leads should be in the position to know and have exactly what they need for such a project rather than just attending morning scrums. If test planning had been done correctly, a viable plan to compensate for time lost during holidays would have been taken care of, and testers wouldn't have to spend late nights getting the job done.

Prioritizing Features

What was done?

In the absence of a plan, testers went about testing the application as they thought best. With countless flows to be tested, there was no input from the Scrum Master, development leads or the test manager as to which should be a priority. And while experienced testers might be expected make these decisions, none were assigned to this project.

What went wrong?

A lack of guidance and prioritization led minor functionalities being tested first, leaving little time for major ones, and end-to-end flows were also inadequately tested. Most of the time was spent on individual features rather than on the complete solution.

What should have been done?

Scrum Master and leads should have made sure that testers knew what was expected of them, particularly in their shortened time frame. Testers should have been asked regularly what they were testing and how it was going. Since the testers were new, chances were good that many common test scenarios would be left out. Trust is good, but blind trust can lead to disaster. It's better to "trust but verify."

Load Testing

What was done?

In one of his earlier jobs, Peter worked as a load test specialist. He worked on a variety of projects and learned many of the minute details of load testing. However, working on this project showed him a different side of the practice.

Peter was asked to perform load testing just a few days before the application was scheduled to go live. He was literally instructed to "write some scripts and run them." No one had any idea how many users to simulate, what kind of scenarios to run, and what was expected from the tests. It seemed that all they wanted was to hear that "x number" of users ran successfully without any errors.

What went wrong?

What was done is just about the exact opposite of what correct load testing is supposed to be. The goal of load testing is to find the breaking point of the application, using varying real-time scenarios. Making the matter worse was the assumption that test scripts take little time to write and can be created and executed the same day.

This total lack of knowledge in the area of load testing, and the fact that it was done as a formality, left Peter unmotivated. It became worse and when he tried to explain how useful load testing can be when done correctly; no even responded. And because the order to run load tests was so close to launch, it was clear to Peter that even though flaws were found, the application would go into production anyway. Peter's outlook changed from seeking the breaking point to running the load tests without any errors.

What should have been done?

Load testing is a specialized form of testing that requires time and planning. Just asking someone to write test scripts shows a lack of knowledge. Also, load testing should have been included in test plan as a priority item, and resource allocation should have been done accordingly. Doing these kinds of activities just for formality's sake is a total waste of time and resources, and should be avoided, especially when everyone is hard pressed for time.

Teamwork

Anyone reading this should be familiar with the concept of teamwork, which, if improperly managed, can have a dramatic effect. In this case, a small teamwork incident led Peter to stop testing the application and give it a green light for release to production.

It's essential to keep people motivated during the project.

What was done?

Even with Peter's 50 percent involvement, he spent around 14 hours each day during a two-week period. He was, after all, the more experienced tester and was therefore obliged to work on the more complex functionalities. Most evenings, no developers were available to support his issues. In every morning scrum, it was repeatedly mentioned how busy everyone was and that they couldn't deliver some of the requested functionality, even halfway into regression week.

What went wrong?

Here's the funny part: In one of the morning scrums, Peter initiated the notion of working weekends to get things done. The development lead asked his developers who would be available to work. All that could be heard was crickets. These are same developers who weren't able to deliver the required functionalities on time and were always "busy." "Busy doing what?" Peter wondered. It was an awkward moment, and I would have expected more concern to get things done from the leaders and people who built the product than from a tester with less of a stake in the project's success.

In another meeting just a couple of days before the production release, Peter was asked if he would be able to work during weekends. His question was, "Is everyone coming to work so that we can get a lot done? After all, it's a UI-based application, and everyone can play around with it to see if they see something wrong." As soon as the question came out, the faces of the Scrum Master and leads became indignant, appearing to say, "Are you kidding? Why should we work weekends?"

That was the final blow, and Peter lost all interest and concern about the project. From that moment on, all of his efforts were toward going to production as soon as possible, after which came the wait for the inevitable production defects to come in.

What should have been done?

The team should have pulled together. Even a short presence or small show of effort and willingness to pitch in on the part of leads and managers during crunch time goes a long way and can motivate the team toward the common goal.

While it's a good idea to circulate e-mails or memos of praise when everything is said and done, it's essential to keep people motivated during the project. When true effort and commitment have been maintained throughout the project's life cycle, after-project commendation is all the more sincere and long lasting.

Too Little, Too Late

What was done?

Everyone needs help at one time or another. This project involved many extra hours, and the test manager correctly asked Peter if he needed any help to get his testing done.

What went wrong?

Even though Peter finally got some help, it came too late, right before the application was to go live. It's difficult for someone to walk in to a project where so much has happened and be expected to instantly perform at a veteran testing level.

Granted, some of the testers who were assigned to help had previously worked on the same application, but all they could do was to go through the basic flows. The end result? They spent only a few hours reviewing the application and concluded that everything looked great. Everyone was happy to hear that, and the launch remained on schedule. No one had any idea of what had happened during the previous month, and Peter was so disgusted that he just let it go.

What should have been done?

For help to be effective, it has to be offered at the right time, or you can foster a false impression that everything is great. Peter knew that the system still had many defects, but was totally unmotivated to point them out. Had he noted this early enough in the life cycle, the project could have been rerouted toward an effective course.

There are lessons to be learned in everything we do. Testing is a subjective practice, and carries no hard-core formulas for success. It's important to learn from experience; in this case, from our fictional friend. Peter's mistakes encapsulate many of the pitfalls I've witnessed throughout my career in quality control. Keep him in mind when you face some of your own.

Web Services Burden Java Developers With Testing

Frank Cohen is a 30-year veteran of the software industry and CEO of PushToTest, a company that earns its bread selling testing tools and services. So he's an altogether unlikely candidate to be fazed by much in the way of testing Java-based Web applications. Yet even the venerable Cohen had something of an epiphany during a recent upgrade of the software that runs PushToTest.com.

The site runs on Zope, an open source application server, and Plone, an open source content management system. Among the many recent enhancements to the latest version of Plone that Cohen's team installed is a slick AJAX interface to facilitate the process of searching the site. Start typing what you're looking for in the search box and up pops a floating window with a list of suggested results. Type a little bit more and the list changes, with any luck converging on your search term.

Say you're looking for the slides Cohen presented at the Practical Software Quality Testing 2007 conference in September. Type "Prac" in the PushToTest.com search box, and the first item in the floating window jumps to an entry on Cohen's blog where the slides are posted.

This functionality comes from a wee bit of JavaScript associated with that search box. One data model allows the JavaScript program to send queries as they're entered to the back-end server. Another data model formats the query results in HTML and displays the results live. From a user's point of view—at least that of an impressionable columnist—the whole experience is slick and yet another sign that the rich Web will eventually carry all our browsing blues away. But there's a small problem, at least for those in charge of the PushTo Test Web site.

"How do you test any of that?" asks Cohen. "There are no standards bodies standing behind JavaScript or behind these two data models that this JavaScript program is using.

"We never set out to add AJAX capability to our Web site; it just happened to show up with the upgrade," continues Cohen, author of the book "Java Testing and Design: From Unit Testing to Automated Web Tests" (Prentice Hall PTR, 2004). "That's the epiphany. The world is marching forward because developers love the rich Internet that AJAX brings. The world is marching forward because enterprises know they can communicate with other enterprises using things like SOAP, Web services and XML-RPC—and there's no gatekeeper to any of the message formats that exist."

Beyond Test-Driven Development

Cohen's story is evidence of the tectonic shifts occurring in Java testing. Java 1.0 was released in 1995, when client-server computing was entrenched and the Web was more or less in its infancy. A well-established division of labor had emerged in the world of software. Roughly speaking, developer organizations wrote programs, QA organizations tested programs, and IT organizations supported programs. Indeed, most early Java programs were accordingly handed off from team to team on their way from requirements document to finished product.

Fast forward to today. Java technology, now conspicuously open source, is used by millions of developers, runs on hundreds of millions of desktops and, according to the official history at www.java.com/en/javahistory/timeline.jsp, has even been to Mars by way of the Java-powered Mars Rover, Spirit. Back on Earth, the Web has exploded and is fast on its way to joining electricity and phone service as basic utilities in the developed world. Internet protocols have matured to the point where Web services and SOA are now possible.

Despite these dramatic changes, it's still true that, depending on the circumstances, code should be subjected to some combination of unit, integration, functional, system, performance and acceptance testing. However, the rise of Java Web applications has changed the notion of what it means to test software and even the idea of what exactly comprises an application, says Kishore Kumar, director for the UST Global Java Center of Excellence in India. "Now there are other considerations, including how to test with SOA, how to make sure that exposed services are consistent with terms of any relevant contracts, and that changes to such services comply with the broader regulatory environment," says Kumar, author of "Pro Apache Geronimo" (Apress, 2006).

Kumar, who also serves as a senior technology evangelist for the Aliso Viejo, Calif.-based UST Global, suggests that what's needed is an altogether new mindset about what constitutes successful Java programming. The aim, he says, should be to move beyond even relatively ambitious practices such as test-driven development, where test cases are written before the production code that implements new functionality. Soon, he insists, developers will be measured more by their

Best Practices columnist **Geoff Koch** can be reached at gkoch@stanfordalumni.org.

ability to write testable code than to implement new functionality. The fact that the latest Java versions allow developers to easily annotate their code is a foreshadowing of this future, he believes. In fact, Kumar insists that each software class might soon come with notes from the author about just which combinations of functions can be used to test it, a change that would require considerably more skill than that used simply to cobble the function together in the first place.

Granted, Kumar speaks at conferences, writes for a slew of trade publications and leads a team of 400 technology specialists, while I'm just a casual industry observer marooned in geographically challenged Michigan. Still, I'm more than a little skeptical of his assertion for much the same reason that I looked askance at the sunniest unit testing claims in last month's column. Here's the thing: Behavior change is difficult, and people rarely do what's best for them on the basis of sound, logical argument. Rather, proper incentive is needed, such as fear (do this or get fired) or greed (do this and make twice as much money as the poor sap in the cube next to yours).

Unless I'm missing something, neither such incentive looms on the tech horizon, and as Cohen sees it, there's little evidence that the overall level of coding competence is on the rise. So how to deal with Java testing in the age of the rich Web, when you wake up one morning and all of a sudden have an AJAX-enabled search box on your home page?

Rx: Repurposing

The only approach, Cohen says, is to try to combine the efforts of development, test, QA and IT. For example, presumably the developer behind the search function in the Plone site has already created a unit test for the JavaScript program. Perhaps that unit test could be repurposed for test and QA into a functional testing framework, which in turn might be repurposed for IT to use and run automatically as a business service monitor.

Cohen is right, and his story only proves that in technology, eventually everything old is new again. In the early days of computing, at least through the mid-1950s, there was essentially no distinction between testing and debugging. Developers were expected—required, in fact—to both test and debug. Testing and debugging were decoupled by 1960, and later testing was further subdivided into the current array of bang-on-the-code activities, most of which aren't done by the developers who wrote the code in the first place.

Now, the pendulum may be swinging back. Developers, especially in the open-source, share-and-share-alike haven of Java programming, should be expected to promptly debug their newfangled Web application as soon as it's unveiled as a service—or at least to share their secrets for testing it with their downstream colleagues. ☒

Having Performance Issues?
We can help...
Software Test & Performance
Subscribe Online at www.stpmag.com

Index to Advertisers

Software Test & Performance

Advertiser	URL	Page
Automated QA	www.testcomplete.com/stp	10
EclipseCon	www.eclipsecon.org	19
Empirix	www.empirix.com/freedom	4
FutureTest 2008	www.futuretest.net	24
Hewlett-Packard	www.hp.com/go/securitysoftware	40
iTKO	www.itko.com/lisa	8
Software Test & Performance	www.stpmag.com	37
Software Test & Performance Conference Spring 2008	www.stpcon.com	2-3
Software Test & Performance White Papers	www.stpmag.com	39
TechExcel	www.techexcel.com/stp	13
Test & QA Newsletter	www.stpmag.com/tqa	6

Who's *Really* Responsible For Code Quality?

Bugs are easily mass populated. In our brave new Web-connected world, software vulnerability is a concern of a whole new magnitude. Who's to blame when that super fridge, which automatically orders your groceries as they run out, is hacked to run a password-cracking algorithm or file-sharing relay? Who pays when that life-saving defibrillator can't power up its paddles because it's busy looking for extraterrestrial life?

Gwyn Fisher

Although we depend on software from morning until night, it's riddled with bugs—many of which are unearthed only after the software is deployed in the appliances we purchase and the services to which we subscribe. The fact is that this brilliant multitasking software is developed by people. And people aren't perfect, especially when armed with inherently dangerous weapons.

Bug Inevitability

The creation of bugs is inevitable, from typing errors to thinking errors, to not-so-simple design errors and insanely complex architectural flaws. Throw in some garden-variety security nightmares and you've got a recipe for disaster, or at least a recipe for patches, costly recalls and product churn, because the unfortunate reality is that software development encourages features and release dates over bug-free code.

It's time for a new approach to software testing and debugging. Today's mission-critical, ubiquitous software must be defect-free long before it hits the streets.

One of the flaws of traditional software testing in today's markets is that it tends to be limited to testing what the software is supposed to do. Today, software is exposed to all manner of unexpected assaults—from the benign (unexpected use by unexpected users) to the malicious (a hacker stealing personal information). Usage-driven testing (i.e., testing the bits that you expect will be used the most) is a popular model, but it's increasingly outmoded as the primary approach to testing software.

Expecting the Unexpected

What's needed is an approach that rigorously tests the code under expected and unexpected conditions—one that ensures all code receives the same test coverage regardless of whether it is *expected* to be executed or not. The approach must also be thorough, and that's difficult to achieve when the testing approach requires a decision about which paths in the software will be investigated and which will be overlooked.

A better process would see developers themselves testing their own code for "simple" bugs. At the developer's desk, agnostic test coverage (typified by pairwise programming and peer review) is possible because the code is available to be examined regardless of runtime state. And, since the code isn't being executed at this stage, thorough coverage of all code paths is theoretically possible.

The challenge with such approaches is obvious: It's far too costly to have developers testing their own code, and it would interfere with time-to-market. While we may expect a rocket guidance system author to apply 100 percent provability to his test approach, for the majority of software developers, competitive market dynamics will dictate when software is released.

Given these realities, it's sensible to equip people as early in the development process as possible with tools and processes to identify bugs that should never even make it to testing.

Automated source code analysis provides one such mechanism by allowing developers to scrub code before check-in and to test for code failures, such as:

- Memory and resource leaks
- Buffer overflows and code injection vulnerabilities
- Invalid pointer or object reference
- Tainted data propagation and use of unsafe libraries or APIs
- Phishing vulnerabilities such as cross-site scripting, request forging

These aren't design flaws and they're not baffling architectural failings. They are bugs. As such, they should never make it off the desk of the developer creating them.

I believe that automated source-code analysis at the point of development is the next evolution of software development and testing. Ten years ago, the number of developers who routinely used runtime profiling tools was small. Today there are few professional developers who haven't used them at some point. The smart ones carry these tools with them from project to project as a vital part of their work practices.

Likewise, five years from now, every developer will use source code analysis as a matter of course, and QA departments will be freed from the tyranny of bug-finding missions to get to the real heart of the matter: testing whether products are any good and acting as full-time customer advocates.

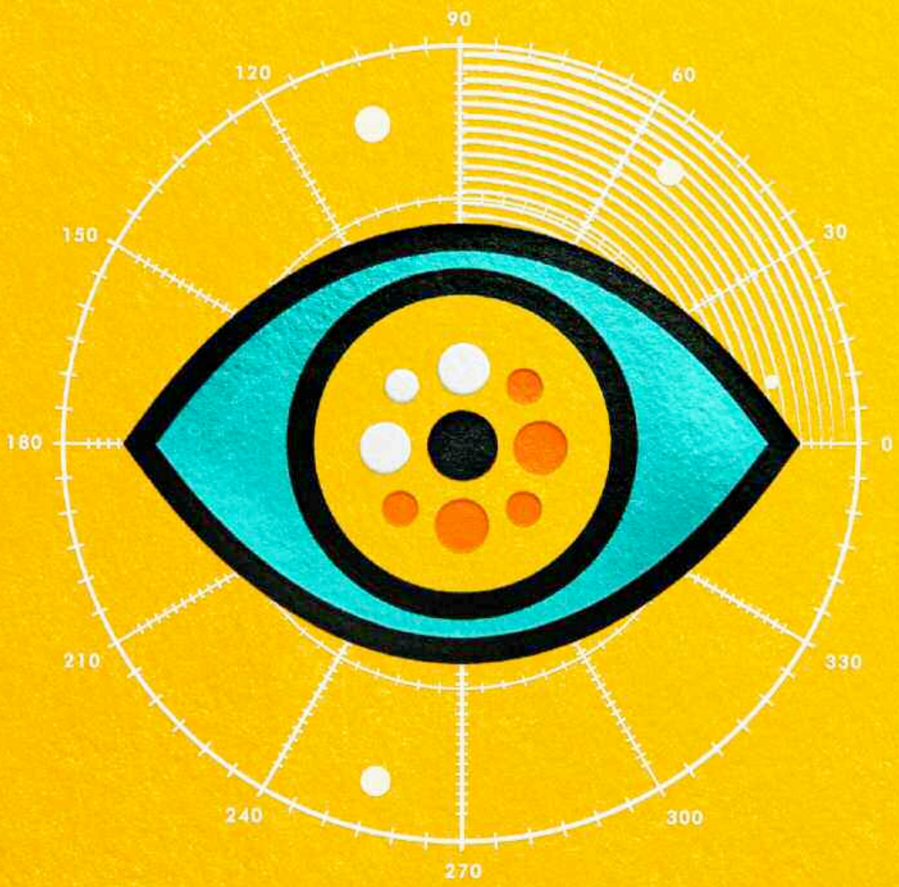
This approach has another positive outcome. By making the developers who created the bugs responsible for finding and addressing them, the creation of those bugs is likely to decline over time. ☒

Gwyn Fisher is CTO of Klocwork, which makes source code analysis tools.

Learn Some New Tricks!

Get Your Free White Papers At www.stpmag.com

Discover all the best software practices, gain an in-depth understanding of the products and services of leading software vendors, and educate yourself on a variety of topics directly related to your industry.



ALTERNATIVE THINKING ABOUT APPLICATION SECURITY:

Hone Your Threat Detection (To A Telepathic Level).

Alternative thinking is attacking your own Web applications, finding vulnerabilities and destroying them with precision and vengeance—throughout the life of the application.

It's looking at application security through the eyes of a hacker to identify threats to your system and risks to your business.

It's harnessing the power of SPI Dynamics, recently acquired by HP, to redefine and expand your security abilities. (Please note: positive effects on your bottom line.)

It's assessing security the right way, from development to QA to operations—without slowing down the business. (Cue elated cheers.)

Technology for better business outcomes. hp.com/go/securitysoftware