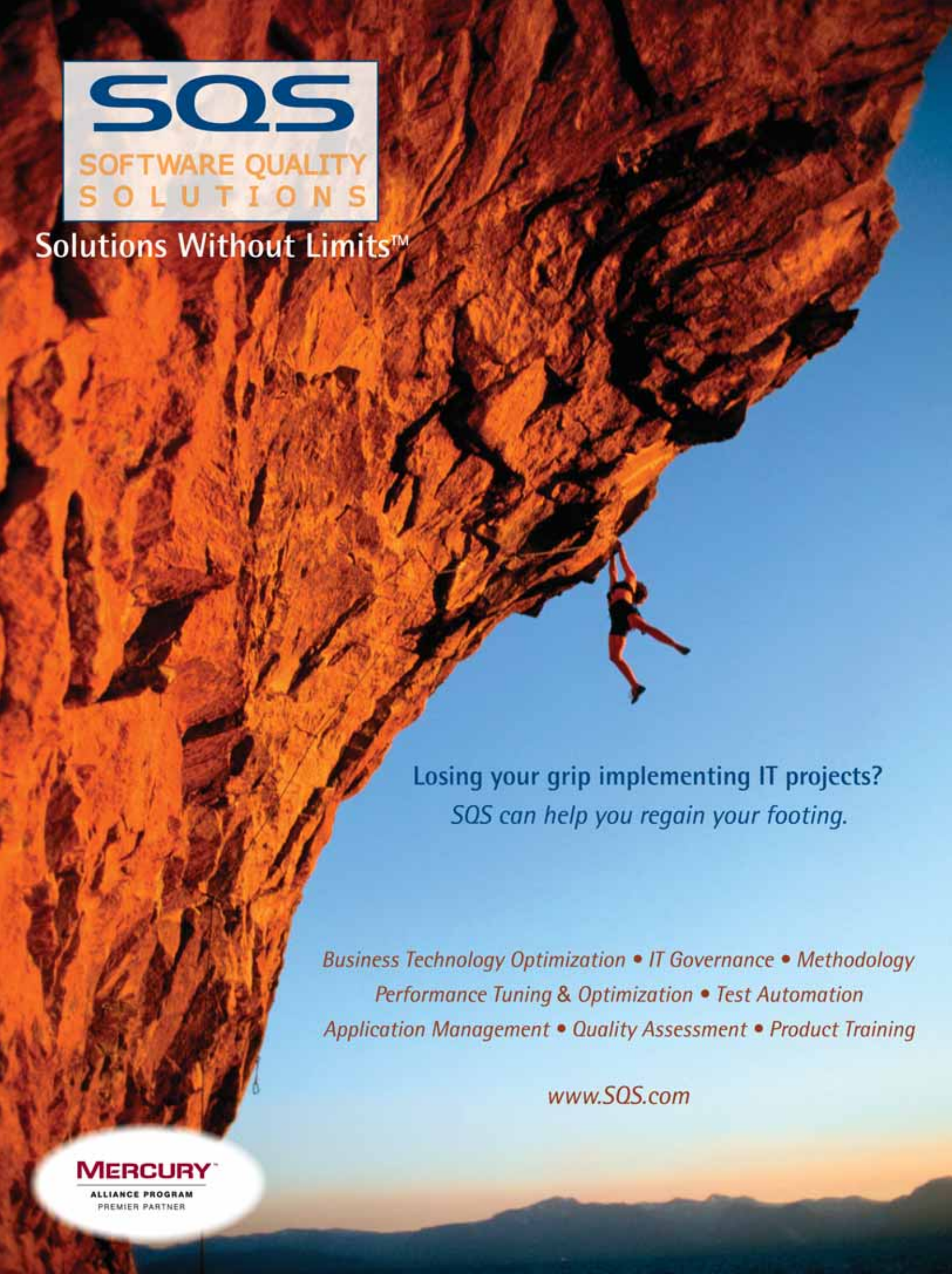
January 2006 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

Code with Character

BY TOD GOLDING


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

Seapine Software™
changing the world of software development

Bring harmony to your software quality assurance process.

Software quality assurance is a team effort with developers, testers, and management all working toward one goal – delivering the highest quality product on time.

Introducing Seapine SQA – a suite of tools that includes automated functional testing integrated with award-winning defect management and team-based script source control.

Consider...

Does your automated testing tool let you easily manage scripts in a source code repository?

Do test failures enter your workflow automatically, and are your developers notified quickly?

Is your automated testing tool forgiving of user interface changes, or do you have to modify your scripts?

Are you saving time and money using an automated tool, or do you still test everything manually?

Seapine SQA does all this and much more. Bring harmony to your team and work faster while improving software quality.

Features:

Fully automate functional testing of Windows-, Java-, and Web-based applications.

Coordinate team-based testing with integrated script source control—track versions, remotely access scripts, store scripts with your other development assets.

Take complete control of defects and your testing process—track defects, change requests, feature requests, and more.

Accelerate quality assurance by automatically adding defects found during testing to your defect management workflow.

Perform data-driven functional testing using data from relational databases, Excel files, text files, and other ODBC data sources.

Tailor the suite to your testing process with configurable workflows, customizable field names, field relationships, default values, and custom fields.

Extend the suite using triggers, email notifications, SOAP support, XML data exchange, and an ODBC client.

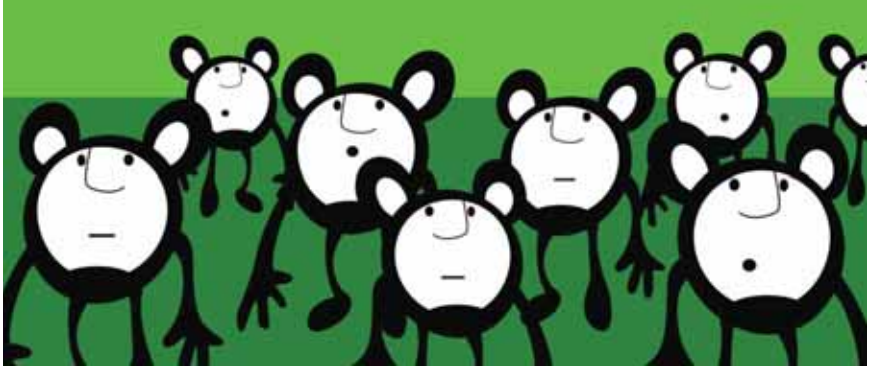
Achieve major improvements in software testing team performance through better tool integration and process automation. Quickly automate your testing process with QA Wizard, easily manage thousands of scripts with QA Wizard's seamless Surround SCM integration, and control your development and QA processes with award-winning TestTrack Pro. Seapine's integrated software testing lifecycle tools streamline your testing process, giving your team more time to script, test, debug, and verify the quality of your software products.

Learn more about Seapine SQA at
www.seapine.com
or call 1-888-683-6456

Surround SCM
Source Control

TestTrack Pro
Defect Management

QA Wizard
Automated Testing



BETTER SOFTWARE

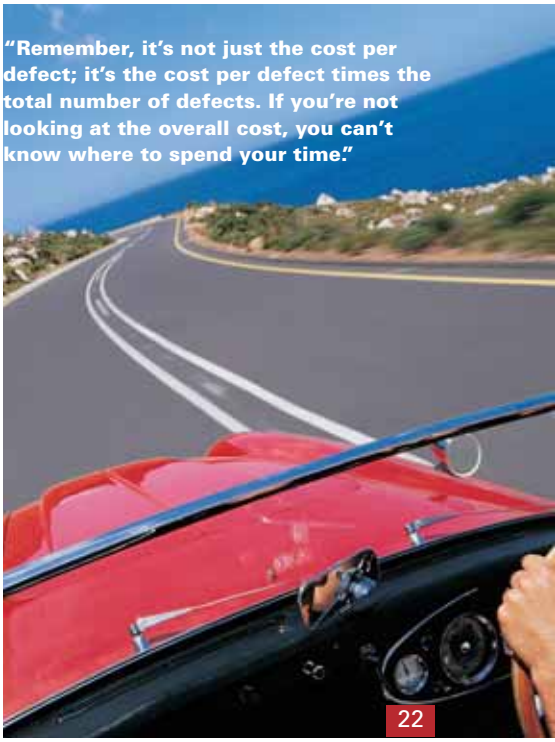
January 2006
Volume 8, Issue 1

Cover Story

Features

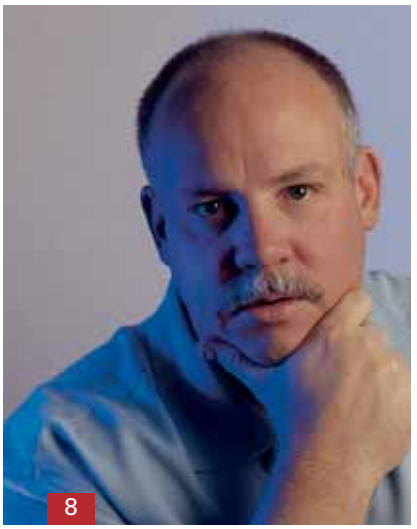
"As a tester you are an anomaly. The project's end product is a working software system, which most of the project team members are devoting all their efforts to building. **Then you come along and expose the flaws."**

FIONA CHARLES
PAGE 16



***Better Software* and StickyMinds.com**

We invite you to visit StickyMinds.com, the online companion to *Better Software* magazine. StickyMinds.com covers the same pertinent topics as the magazine, putting the power of information at the click of your mouse. Weekly columns, headline-making bugs, hundreds of technical papers, an online Tools Guide, discussion boards, and so much more make this your site for 24/7 brainfood to help you build better software.



"There are some issues that should be passed down the organization. This is the basis of empowerment. There is a fine line, however, between empowerment and shirking our own responsibilities."

MIKE COHN
PAGE 8

Columns and Departments

Featured Department

In Every Issue

RUNNING YOUR TESTING CENTER LIKE A BUSINESS

INTRODUCING MERCURY CENTER MANAGEMENT – AN OUT-OF-THE-BOX SOLUTION FOR MANAGING A PERFORMANCE TESTING CENTER OF EXCELLENCE

Today, many IT operations groups are centralizing performance testing resources into a Performance Testing Center of Excellence. A Center of Excellence, or CoE, is a team of people who use common solutions and methodologies to deliver services across the enterprise. However, running an effective CoE presents many challenges. The CoE must be able to:

- Handle all incoming demands in a consistent way.
- Make sure that customers are happy with the services they receive.
- Manage personnel so they are focused on the right tasks and have the right skill sets.
- Make intelligent staffing and training decisions.
- Provide feedback and updates to executives about the status of tests and projects.

The answer to these challenges is **Mercury Center Management™ for Performance Center**. Built on Mercury's IT Governance platform, Mercury Center Management delivers an out-of-the-box demand, project, and resource-management solution for running an effective Performance Testing Center. It ensures that current project status, health, and deliverables are available to all, in real-time, and without extra effort.

With Mercury Center Management, your performance testing project begins with a request for services from the LOB. Using a web-based form, the LOB manager submits all relevant project information – such as goals, priority, risk, and application business processes – to your Center team. The solution is **pre-configured with templates** for performance testing project management and execution, and includes a set of **digitized processes** to manage the work in a consistent fashion.

After your Center accepts the project, the LOB can track project progress via a **customized dashboard**. Updated in real-time, the LOB dashboard places project status, tasks lists, and any risks or red flags that exist at the manager's fingertips. It also provides an **automatic workflow** that routes project requests to the project management team within the Center. Notifications via email or dashboard alert the project manager that a new request has come in. The manager can request more information from the LOB, triage the timelines, and assign resources – all without ever picking up the phone.

As tasks are assigned to performance engineers, they automatically appear on each engineer's dashboard. Each time performance engineers run performance tests and surface bottlenecks, they can update their tasks' status locally. These updates are automatically rolled-up through the project hierarchy so that project managers and stakeholders see current project status in real time.

Mercury Center Management provides a **framework** and a concrete **set of best practices** for running and managing performance testing projects. While these practices can be modified, they are based on a set of steps that are essential for effective performance testing, including load test design, system baselining, and system optimization.

Mercury Center Management for Performance Center provides you with **consistency, control, and visibility** across all of your Center operations – allowing you to focus more of your time on delivering critical projects, versus drowning in documents, spreadsheets, and formatting reports.

To learn more about Mercury Center Management for Performance Center, visit **http://www.mercury.com/us/products/performance-center/.**

From the **Editors**

Thanks to everyone who attended the recent STAR*WEST* conference in Anaheim, California. It was our pleasure to have the opportunity to meet so many *Better Software* magazine readers and StickyMinds.com users.

If you didn't make it out to STAR*WEST* this year, don't miss us on the East Coast. Log on to www.StickyMinds.com to find out how you can register for STAR*EAST* 2006, May 15-19, in Orlando, Florida.

Here's what some of you had to say about STARWEST 2005. As always, we love the feedback.

"For anyone serious about software testing best practices, STAR*WEST* is the place to be—experts coming together with the testing community to share a wealth of ideas!"
Kenneth Hass

"STAR*WEST* is the best conference so far. It opens up new niches for ideas and opportunities to explore and create new concepts in software testing."
Mark Garnett

"I'm glad I came. I have learned a lot and I am excited about going back to implement some of these techniques."
Belva Porter

"This is my second time at STAR*WEST*—very impressive and informative. I recommend to all. It's a learning experience you'll never forget, and you'll always find new ideas to use and apply."
Laura Greer

"The conference has been great. I have many new techniques to make testing more efficient and better with less effort."
Lynne Register

We Take Our Mistakes Seriously

Due to an editor's error, incorrectly formatted tables appear on pages 28 and 29 of the November/December feature article "Brushing Up on Functional Testing."

Please visit http://www.stickyminds.com/brushingup to view the corrected listings and read the article in its entirety.

BETTER SOFTWARE

Publisher
Wayne Middleton

Director of Publishing
Holly N. Bourquin

Editorial
Editor
Heather Shanholtzer

StickyMinds.com Managing Editor
Francesca Matteu

Assistant Editor
Joseph McAllister

Copy Editor
Dayna Spear

Managing Technical Editor
Lee Copeland

Technical Editors
Mike Cohn, Brian Marick

Design
Creative Director
David Parrish

Art Director
Sarah Rice

Advertising
Sales Director
Alison Wade

Sales Executive
Shae Young

Administrative Manager
Heather Buckman

Sales Assistant
Ellen Mahoney

Production Coordinator
Julie Morgenstern

Circulation and Marketing
Circulation and Marketing Manager
Sommer Farrin

A PUBLICATION OF SOFTWARE QUALITY ENGINEERING

CONTACT US
Editors: editors@bettersoftware.com
Subscriber Services: info@bettersoftware.com
Phone: (904) 278-0524, (888) 268-8770
Fax: (904) 278-4380
Address: *Better Software,* SQE, Inc.
330 Corporate Way, Suite 300
Orange Park, FL 32073

Eliminate the finger pointing with WebLOAD Analyzer™.

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

Want to accelerate resolution of Web performance problems?

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

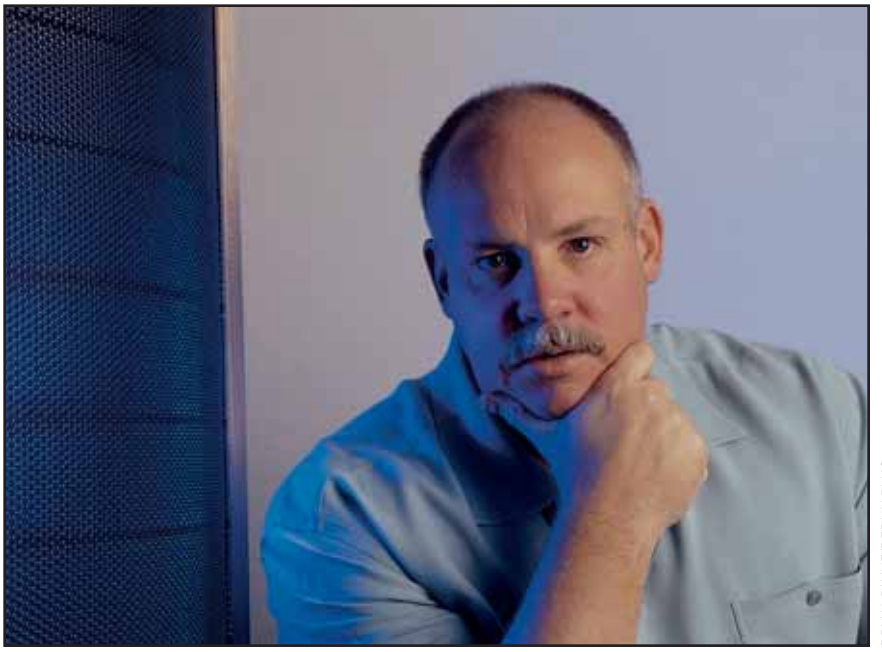
>> Learn more at www.radview.com/analyze

RADVIEW™

The Smart Choice in Web Application Testing

Put a Tough Decision in Its Place

by Mike Cohn



BRIAN PAYNE/REDUX PLUS

Technical Editor Mike Cohn

In a recent discussion with a project sponsor, I asked whether it was more important to deliver on schedule or to deliver all of the desired features. The sponsor's answer was "Both." She refused to clarify for her team the relative importance of scope, schedule, resources, and quality on that project. She was insistent that all features be delivered on time, with no loss of quality, and without going over budget by increasing the team size.

I was gradually able to help her understand that this puts the team in an impossible position. I told her that every time a programmer passes a module to testing, he has made a tradeoff decision between scope, schedule, and quality. He has made a micro-level decision that the code is good enough, and that further improvements in quality are not justified by their cost in either a longer schedule or in delivering fewer features. I told her that these types of decisions are made dozens of times each day by all team members. And here's the kicker—I told her that, without guidance from her, these micro-level decisions were being made inconsistently. At that point, I was able to get her to prioritize scope, schedule, resources, and quality against one another.

This particular project sponsor was doing something I see happen much too frequently—she was pushing a difficult decision down the organization. If we cannot make a particularly difficult decision, what we should do instead is push the decision up—not down—the organization.

Multitasking, which I've written about in a previous column (see the July/August 2005 issue of *Better Software* magazine), is another example of inappropriately pushing a decision down the organization. The manager who assigns a tester to be on three projects is saying, "I can't decide which is the most important project for you to work on, so I'll make you decide." If the manager, who presumably knows more about the relative priorities of the projects, cannot decide how this tester should spend her time, how can she reasonably expect the tester to decide?

Of course, there are some issues that should be passed down the organization. This is the basis of empowerment. There is a fine line, however, between empowerment and shirking our own responsibilities. Here are three questions I use to help decide whether a decision should remain with me, be pushed up, or be pushed down:

- **Who has the appropriate knowledge to make this decision?** Before passing a decision down, be sure that the employee has or can get all the necessary information to make an appropriate decision. If you pass it up, make sure your boss is in touch with the necessary day-to-day details.
- **Who owns the risk of a bad decision?** Is it my boss, my employee, or me? In the product sponsor's case, the risk of choosing incorrectly between scope, schedule, resources, and quality was hers. She needed to own that decision.
- **What would my boss or my employee think of my pushing the decision to the other person?** If I told my boss, "I just delegated this decision to my employee," would she agree? If I told my employee, "I can't make that decision; let me run it by my boss," would he agree?

Next time you are tempted to pass a difficult decision down the organization, stop and ask yourself these three questions. Similarly, next time a decision is inappropriately passed to you, see if you can get your boss to push the issue up instead of down. **{end}**

Mike Cohn is the founder of Mountain Goat Software, a process and project management consultancy that specializes in helping companies adopt and improve their use of Agile processes and techniques. He is the author of Agile Estimating and Planning *and* User Stories Applied for Agile Software Development. *Mike is a founding member of the Agile Alliance and serves on its board of directors. He is a technical editor for Better Software magazine, a regular columnist for the magazine and StickyMinds.com, and a frequent presenter at STAR and Better Software conferences. He can be reached at mike@mountaingoatsoftware.com.*

SOFTWARE TESTING
ANALYSIS & REVIEW

The World's Largest Software Testing Conference

STAR EAST

2006

ORLANDO, FLORIDA MAY 15–19, 2006

KEYNOTES BY INTERNATIONAL EXPERTS

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

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

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

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

Testing: The Big Picture
Brian Bryson, IBM Rational Software

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

SOFTWARE QUALITY ENGINEERING

www.sqe.com/stareast

REGISTER EARLY AND SAVE $200!

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

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

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

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

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

Support for Testing, Testing for Support

by Michael Bolton

Getty Images

In the last issue, I introduced the first part of the Quality Criteria dimension of James Bach's Heuristic Test Strategy Model. People often refer to these quality attributes as "the -ilities," properties of the product that customers might find desirable: capability, reliability (which under the HTSM includes security), usability, scalability, performance, installability, and compatibility.

The second part of the Quality Criteria list focuses on development, or producer-facing attributes. The HTSM identifies supportability, testability, maintenance, portability, and localizability. The first two of these attributes are so important that I'm going to dedicate this entire column to them.

Rapid Testers use Jerry Weinberg's definition of quality: "Quality is value to some person." The end-user is just one member of the project community whose values matter. We also attempt to produce value and reduce cost for the organization that is developing the software. We think about supportability and testability to remind us to look for problems that have a real impact on support people and the testers themselves; they are also customers of the testing effort. Both groups can and should ask for supportability and testability.

In the early '90s, I worked for a company called Quarterdeck. Its flagship products were DESQview and QEMM-386, multitasking and memory-management utilities for DOS running on Intel-based personal computers. The PC environment in those days was a mishmash of mostly compatible hardware, but because our products worked so closely to the metal, they were more vulnerable to compatibility problems than most other products.

I worked in technical support, then testing, and later in program management. Those departments were closely linked—everyone used the products in his daily work, so everyone tested to some degree. Ever since then I've been aware that good technical support people are natural allies for testers and can be highly valuable to the testing effort. They're experts with the product, they're direct conduits to the customers, and they're keenly aware of the kinds of problems that make telephones ring and support forums choke. As testers, we want to prevent those things. We reduce cost and add value when we find bugs or anything else that would cause problems in the field or extra work for the support staff.

Cultivate relationships with your support people. They can help you understand what supportability means in your context and what's important to customers. Support people may identify risks that you may not have considered, and they can help estimate the cost and impact of a bug. Most of them will be delighted to help you find problems and will advocate specific bug fixes. Support people may have access to tools, tricks, or tips that testers can use. Swap useful documents, diagrams, or scripts. Make sure that the support people have your number so they can call you directly if they see a problem in the product. Some support staff may aspire to become testers, in which case you have a farm team in-house.

A support person will be the first to tell you that coherent and consistent error messages make a big difference to a program's supportability. Testers should try to trigger all kinds of exceptional conditions while testing. We do that primarily to expose risk; unforeseen conditions that the program doesn't handle put the program in an unpredictable state. But even if we don't find bugs, we look closely at each error message and other feedback supplied by the program. Does the message clearly and accurately describe what's going on? Does it help the end-user solve the problem—or if that's not feasible, would the message assist support staff or developers? Does the error message uniquely identify the point of failure in the program? If the problem is a missing file or resource, does the error message *specifically* identify what's missing? A support person will identify a problematic error message for you right away ("A .DLL could not be found."—OK, but *which* .DLL?).

Many of the attributes that add to a product's supportability also add to its testability. By testability, Rapid Testers primarily mean *visibility* and *controllability*. We evaluate the program based on the means it provides for service people to support it, testers to test it, and developers to debug it. Does the program produce log files? Can we configure logging to provide varying levels of detail? Are the logs consistently structured? Can they be scanned quickly and easily, either by a human or by a program written in a scripting language? Is each event that is recorded in the file precisely time stamped and well structured so we can write scripts that parse, filter, and summarize? Log files are a feature of the product—are we testing and evaluating them, or are we taking them for granted and thereby possibly missing bugs? What other reports does the program produce? Is there information in them that might help in the testing or support effort? What information is missing that we would like to be able to see? Can we query the program on the fly?

We'd like to be able to automate certain functions of the program or our interaction with it, so we ask for controllability—scriptable interfaces, typically using languages like Perl, Python, or Ruby, to reduce the need for expensive front-end tools. When the program provides a means to control it, we can use automation to operate the program, to set up data and manipulate it, to probe the state of the product, or to install and configure it. Programs that can be controlled remotely might add to testability and to supportability if they can reveal useful information. The emphasis is on doing things efficiently—getting the machine to do the work—which leaves us more time for critical thinking, observation, and evaluation.

Note that when Rapid Testers talk about testability, they're referring to visibility and controllability of a program. In some places, "testability" has another meaning; it is sometimes equated with "falsifiability," or "decidability," which is the ability to make a true-or-false or yes-or-no statement about the program. That's a valuable notion, but there are two pitfalls to avoid. The first is that a falsifiable statement may not tell the whole story of an observation we could make. The statement might be too vague to be testable. For example, "The application shall exhibit responsiveness." Conversely, it might be precise in a way that may not really matter to anyone. For example, if an application, as specified, must return a result within one second, give or take five milliseconds, does that difference matter? It might, but if not, we could be tempted to create overly precise tests that distract us from more important things in the mission.

The second pitfall is that jargon words like "testability," "stress testing," or "functional testing" may take on different meanings in different organizations. Someone who claims that a program is testable (meaning falsifiable) may not understand when we assert that the program is not testable (meaning visible or controllable). We can choose to use whatever words are culturally feasible to ask for visibility and controllability, as long as we clearly express that we need them. Moreover, if we consider both possible meanings of testability, we spark our imaginations to create more diverse tests.

To get visibility and controllability, we may need to recruit developers to our cause, but there's something in it for them, too. The developers themselves benefit because a more testable program is almost always easier to debug. They may appreciate that, when a program is more testable, we testers need less time to achieve the same amount of test coverage, or we can achieve more coverage in the same amount of time. Either way, we have a better shot at discovering some problem that threatens the value of the product.

Testability fosters collaboration. At Quarterdeck, memory management and multitasking were tricky to understand and diagnose. To resolve problems and test the product, the development, support, and testing teams needed access to information about the system, DESQview, and QEMM, so the developers wrote a program called Manifest and included it in the package. They continued to refine the product based on ideas from support staff, testers, and customers.

Manifest added a lot of value by making invisible things visible. It allowed our support staff and testers to be more productive by allowing them to troubleshoot problems quickly. Manifest had easily understandable maps of memory that showed which program or device was using which addresses, so finding and resolving conflicts was a breeze. Manifest collected all of the relevant system information in simple tables that were easy to navigate, clearly presented, informative, and able to be printed, mailed, or faxed. Some customers and vendors came to use Manifest as a general troubleshooting tool. For other customers, the DESQview and QEMM packages were more valuable than their competitors, at least partly because they were better tested and more supportable. Did that make a difference? Well, for some time, QEMM was consistently the best-selling PC software package in the world. Testability and supportability count. **{end}**

To get visibility and controllability, we may need to recruit developers to our cause.

Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries as part of James Bach's Rapid Software Testing course. He is program chair for the Toronto Association of System and Software Quality and is a regular columnist for Better Software *magazine. You can contact Michael at mb@developsense.com.*

Don't Stop Now!

Log on to **StickyMinds.com** and join Michael Bolton and your peers in a conversation about this topic. At the end of the digital column, add your views or just read what others have to say.

Say It…Don't Stew in It

by Naomi Karten

GETTY IMAGES

"We don't like being treated like robots," said Syd with a sigh.

Chris jumped right in. "Right, we're doing good work, but it's like we're on an assembly line. We finish one project and plunk—there's the next one. Why are managers so clueless? Even Kent. He may be a senior VP, but even he doesn't seem to get that we're not machines!"

"I hear you," said Lynn, their manager. "So what is it you want?" Lynn had recently joined the company and was eager to hear her project managers' concerns.

"What do we want?" Syd asked, grinning. "Well, double the salary and triple the vacation time would be nice. But just getting some credit for what we accomplish would suffice. We work hard, and we want to be recognized. It's not just us. You'll see this is an IT-wide problem. People don't feel valued."

Lynn nodded. "Serious stuff, definitely. So when you say you want to be recognized, what do you mean?"

"Well," Chris said, "I guess we just want to be acknowledged once in a while."

Lynn looked puzzled. "I'm still not clear. What does it mean to be acknowledged?"

"Um, er, well, uh . . ." Syd and Chris hemmed and then hawed.

"See, here's the thing," Lynn explained. "If you're going to claim that managers are clueless, you need to offer clues so they understand what you want and why it matters. Even 'clueful' managers might misinterpret what you mean by 'recognized' and 'acknowledged.' These are vague terms that mean different things to different people."

She gave them a chance to take this in, and then added, "You need to do some thinking. What, specifically, do you want? How will it help? How will it make a difference to you and to the company? Focusing on what you *don't* want doesn't clarify what you *do* want. And concrete recommendations will get you a lot further than vague wish lists."

With head-slapping awareness, Syd and Chris realized that *they'd* been clueless too—both about what they wanted from management and how to frame what they wanted to ensure better-than-zero odds of getting it. But as a long-time manager, Lynn had seen this pattern repeatedly. Employees grumble about their grievances but give little thought to what would rectify the situation. Even worse, they expect management to intuit both what's troubling them and what will reverse the situation.

"Let's see if we can start to pin this down a little," Lynn said. "Tell me more about this issue."

Syd went first. "The senior team seems unaware that we even exist. When the IT top cheeses do their ceremonial march around the floor each December to wish us a happy new year, it's obvious from their lame comments that they don't even know what we're working on."

"OK," Lynn responded. "So what would you like instead?"

Syd reflected a moment. "Well, I guess an occasional communication—in person or even just by email—that indicates they're aware of what's going on down here in the trenches."

"Good. That's a little more specific. Now what else?" asked Lynn.

Chris exclaimed, "The Superstar Award they give each quarter to two people who've pulled off miracles—what an insult to the rest of us!"

"And what would you like instead?" asked Lynn.

"Hmmm," pondered Chris, thinking out loud. "Sara and Jeff certainly deserved the award, so that's not the issue. What bugs us is, what about everyone else—all the people who are just plain working hard and getting the job done? They deserve credit too. But how, exactly, I'm not sure."

"Fine. You've articulated the issue. Now you can do some thinking about what would help. Now, what else?"

Syd was on a roll. "Every time something goes wrong, Kent remembers who we are and he rubs our faces in it. "*What we would like*," he said, pleased with himself for having caught onto the formula, "is some attention paid to what we're doing well. Even an occasional thank you would help. Some sign that management appreciates our efforts would go a long way."

Lynn was pleased. "This is a great start in translating vague grievances into concrete recommendations. Talk with your teams to see what else you can come up with. Be specific, and be sure to include examples such as the ones you've

STORY LINES

- DON'T JUST VENT. DO SOMETHING. Venting may feel good, but it rarely generates change. Just the reverse, it can give you a reputation as a complainer.
- CLARIFY WHAT THE PROBLEM IS. What are your grievances? Why do they matter? How do they interfere with your ability to deliver results?
- EXPLAIN WHAT YOU WANT AND WHY IT MATTERS. Changes that don't help management achieve its goals won't get far. To get attention, explain how the changes you'd like will help management achieve its goals.
- OFFER SPECIFIC, CONCRETE RECOMMENDATIONS Identify the actions or steps needed to achieve the results you'd like to see.

just told me. When you're done, I'll go over it with you. If you can make a persuasive case, I'll bump it upstairs."

"One more thing," Lynn said as Syd and Chris headed for the door. "Don't forget: Managers like to be recognized, acknowledged, and appreciated, too." **{end}**

Naomi Karten has taught seminars and delivered presentations to more than 100,000 people internationally. Her training and consulting services help organizations improve customer and employee satisfaction, strengthen teamwork, and manage change. Her books, Managing Expectations *and* Communication Gaps and How to Close Them, *provide practical ideas and advice on carrying out projects, improving service, strengthening relationships, and managing change. She is a regular contributor to StickyMinds.com and* Better Software *magazine. Contact her at naomi@nkarten.com or via her Web site, www.nkarten.com.*

Don't Stop Now!

Log on to **StickyMinds.com** and join Naomi Karten and your peers in a conversation about this topic. At the end of the digital column, add your views or just read what others have to say.

Your Favorite Books

Thank you to everyone who participated in our recent survey. The following books topped your list of favorites. Don't forget to visit the Books Guide on **StickyMinds.com** to read reviews of many of the books you see here.

Testing Computer Software
Author: Cem Kaner/Jack Falk/Hung Nguyen

Lessons Learned in Software Testing
Author: Cem Kaner/James Bach/Bret Pettichord

Code Complete (2nd Edition)
Author: Steve McConnell

The Mythical Man-Month
Author: Fred Brooks

Software Testing Techniques: Finding the Defects that Matter
Author: S. Loveless/G. Miller/R. Prewitt Jr./M. Shannon

Peopleware: Productive Projects and Teams (2nd Edition)
Author: Tom DeMarco/Tim Lister

Systematic Software Testing
Author: Rick Craig/Stefan P. Jaskiel

Design Patterns
Author: Erich Gamma/Helm Johnson Vlissides

Software Requirements (2nd Edition)
Author: Karl Wiegers

The Art of Software Testing
Author: Glenford Myers

Software Testing in the Real World
Author: Edward Kit

A Practitioner's Guide to Software Test Design
Author: Lee Copeland

How to Break Software Security
Author: James Whittaker/Herbert Thompson

Refactoring
Author: Martin Fowler

Managing the Testing Process, (2nd Edition)
Author: Rex Black

Quality Software Management, Volumes 1-4
Author: Gerald Weinberg

The Pragmatic Programmer
Author: Andrew Hunt/David Thomas

Software Engineering: A Practitioner's Approach
Author: Roger Pressman

Software Project Survival Guide
Author: Steve McConnell

Waltzing with Bears
Author: Tom DeMarco/Tim Lister

Down to the (e)Letter

TOOL TIME

Coming to a mailbox near you, a new monthly eLetter covering tools and automation for the software development lifecycle. The Sticky ToolLook will feature an interview with an expert in the field—bringing you insights on tools and automation straight from the source. Don't miss what industry veterans like Danny Faught and Linda Hayes have to say about tools and automation.

WEEKLY UPDATE

The weekly *What's New Gram* highlights new book reviews, articles written by top industry professionals, contributions from your peers, conference and training notices, and so much more. It's really the best way to keep abreast of the latest content being published on StickyMinds.com.

OUR FAVORITE PICKS

The *StickyLetter* features hand-picked content from StickyMinds.com. Find out what topics and content we think you should read or re-read, and learn how "Our Take" on the industry can broaden your horizons and help you view life around your work differently. For past issues, visit **http://www.stickyminds.com/SLArchive**.

NEWS THE STICKYMINDED WAY

We know you're busy and don't have a lot of time to sort through the hundreds of computer industry news items published each month, so let us do the work for you. Each *Between the Lines* eLetter summarizes interesting software industry news that will inform and sometimes surprise you. For past issues, visit **http://www.stickyminds.com/BTLArchive**.

IT'S AS EASY AS 1, 2, 3!

If you're not already receiving our eLetters, subscribe today for free. It's as easy as **1:** sign on to StickyMinds.com; **2:** make sure your email is in the right field and that you've selected the eLetters you'd like to receive; and **3:** click Go! **http://www.stickyminds.com/eletters.asp**

At Your Service

Need an answer to a question? Try asking your peers. Post your queries on our Discussion Boards and join the conversation at **http://www.stickyminds.com/discussionboards**.

If you have some insight you'd like to share, why not post an article or paper on StickyMinds.com? You can upload your work for possible publication on StickyMinds.com at **http://www.stickyminds.com/submit**.

"This is the era of the soundbite. One of the bumper-sticker mottoes I drill into all of my project management students is: Adding more people to a late project always makes it later."

StickyMinds.com member GENE FELLNER commenting on Linda Hayes's article "Less is More" (accessible from the column archive on StickyMinds.com).

Read the article or see what other users have to say by visiting http://www.stickyminds.com/comments8-1.

EDITOR'S PICK

Out of the Frying Pan and Into the Melee

FRANCESCA MATTEU

Too many chefs in the kitchen can spoil a meal. And if you're a member of a big family that loves to cook, like mine, a kitchen full of self-proclaimed chefs can be more explosive than water in a pan of hot oil.

There's little method to our cooking madness. We jump right into the frying pan—at least that's what it feels like. My family packs into the kitchen and begins prepping meats, vegetables, and herbs.

As the smoke thickens, we get into excited discussions about how much mirepoix should be used in the risotto, whether there's too much salt or not enough white pepper in the stew, and if the meat in the oven needs to be basted more frequently. It might be hard to imagine we're capable of turning out a meal without burning everything, but we manage. That's just how we operate, and we love each and every intense minute we work together. We even joke about our raucous way of cooking, but I can't help but think that there's always one too many hands in the kitchen.

Linda Hayes' column "Less is More" paints a similar scenario. She discusses a study that concludes "the more people you add to a project, the lower your per-person productivity and the higher the defect rate." In the case of the holiday meals my family and I feverishly prepare, I bet we'd be able to sit down to eat at least an hour earlier if fewer people were involved in the cooking process. With our "staff," we tend to spend more time dodging hot pans, knives, and each other than actually cooking. And while we're adding the finishing touches to the dishes, we're also spending a little extra time picking out the charred pieces.

Next time you find yourself in a brawl of team members fighting to finish a project on time, step back and ask if your efforts are really needed. I'm not sure my family will ever cut down on the number of helpful hands in the kitchen—and I hope our mayhem never changes—but I must admit, doing so would serve us well in the long run.

Read Linda's column "Less is More" at **http://www.stickyminds.com/editorspick8-1**.

Francesca Matteu, managing editor of StickyMinds.com, brings a background of public relations to her position. Francesca's previous experience includes Web site development and design and print publication management.

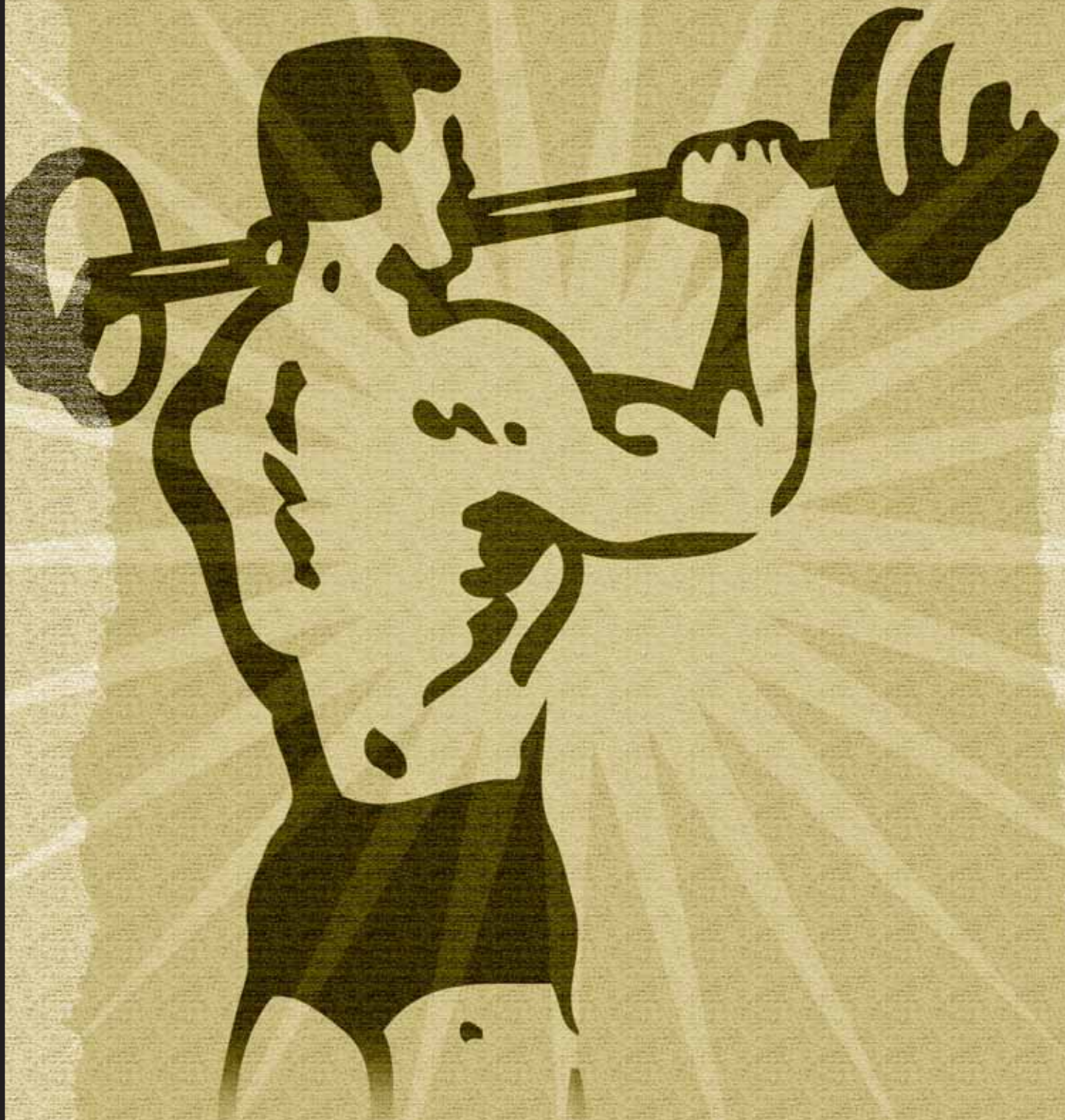
PowerPass Pointer

Conference Materials

Six Impossible Truths about Developing Software—All Before Breakfast

> Alice laughed. "There's no use trying," she said: "one can't believe impossible things."
>
> "I daresay you haven't had much practice," said the Queen. "When I was your age, I always did it for half-an-hour a day. Why, sometimes I've believed as many as six impossible things before breakfast." (Lewis Carroll, *Through the Looking Glass*)

Tim Lister discusses the impossible things that many developers believe—things that often make us look just flat stupid to the outside world. He believes that declaring things to be impossible is both therapeutic and the first step toward finding a better way to deal with our limits and frailties. Working to get the requirements right before beginning development is an example of believing an impossible thing. Learn about Tim's five other impossible truths and begin your personal recovery program at **http://www.stickyminds.com/powerpass8-1**.



Testers are almost always part of a team. As the only tester on a project team, or as a member of a test team that is part of a larger project team, we need to interact with our colleagues to get our jobs done. But for some testers, that interaction—the people skills, or soft skills—is their weak spot.

Bulking Up: Strengthening Your Soft Skills

By Fiona Charles

Testers need to strengthen their soft skills just like athletes need physical conditioning. A successful basketball player must have certain "technical" skills: dribbling, passing, and shooting. But without a foundation of general physical fitness and agility, a player will not make the best use of those skills and will never be a star. For an athlete, physical fitness is a basic requirement and an enabler for technical skills.

As a tester you are a knowledge worker, making your living by applying technical skills. Instead of the physical fitness skills needed by an athlete, you need soft skills to empower and put your technical skills to work. As you build up your people skills and enhance your ability to communicate, you are bulking up on useful skills that will shape you into a top-form tester.

Two specific people skills, teamwork and communication, are basic skills that every successful tester needs—even the most junior. Unfortunately, these people skills are often neglected in testers' education plans.

Teamwork Toning

While teamwork is critical for everyone in IT, testers have unique challenges as team members. A tester must be seen as a solid contributor to the collective effort but must also maintain a level of independence sufficient to do your job. Achieving this crucial balance is difficult, and soft skills are essential to help you do it.

At some point in their careers, most testers hear statements like these:

- "That's not a fair test! Nobody would do something like that."
- "You're asking too many questions. The developers are complaining it's taking too much of their time."
- "You're just not a team player."

As a tester you are an anomaly. The project's end product is a working software system, which most of the project team members are devoting all their efforts to building. Then you come along and expose the flaws.

Your job (perhaps your secret delight) is to break the software everyone else is working hard to create. If you test well, you will probably find many bugs that will take time to fix and retest. When you report those bugs, you are providing valuable information, but your work may make developers feel exposed and threatened. It is not a large leap for developers—particularly on an inexperienced team—to start seeing you as an obstacle rather than a helper. If the development lead and project manager feel the same way, your testing skills alone can't save you and you are at risk of becoming a pariah.

Alternatively, you could end up in a precarious situation in which someone is asking you to suppress information:

- "Look, you don't need to log these bugs. I'll just fix them right now."
- "All this stuff about so-called risks is just negative. That sort of talk is not helping the project."
- "We're all a team here. It doesn't make sense for you to report a red status on your part of the project when everyone else's is green."

It can be tempting—appealing to your sense of camaraderie and teamwork. Or it can be coercive, with someone's refusal to hear you or trying to shame you into silence. If you succumb, you will compromise your effectiveness as a tester.

You need people skills in order to

exercise your testing skills and combat these obstacles. Ideally, you will be able to interact with others so that your position on the team is unquestioned. When you cannot avoid "teamwork issues," your credibility will depend on your soft skills.

Communication Conditioning

Next to teamwork, good communication is the most important soft skill a tester can have.

WRITTEN COMMUNICATION

Like it or not, writing is a requirement of your job as a tester. Often, a defect report is your first and most challenging writing task. Entry after entry comes back with "Not enough information" or "Can't reproduce," yet you know the problem is readily reproducible.

Because defect reports are central to every tester's job, this is an excellent place to improve your writing skills. If you can write a good bug report, you are well on the way to writing anything. Focus on the three Cs:

> "One of the most difficult conversations you can have as a responsible tester occurs when you have to persuade managers that the software you have been testing is not ready..."

Clarity:

- Is this a clear, unambiguous report, written in a logical order?
- Write using simple terms.
- Do not leave room for misinterpretation.
- Start by summarizing the most important information, and then give supporting detail in logical order.

Completeness:

- Is there anything missing?
- Describe what happened and why it is a problem.
- Include the information required to explain how you found the problem, including the data used and the steps followed.
- Demonstrate the problem with screen shots or other evidence.

Concision:

- Does it contain extraneous infor-

Get a Game Plan

What can you do to become part of the team? The best strategy is to present yourself from the beginning as a dedicated team member with a specific job that adds value and contributes to the team's goals. It helps to remember a few simple tips:

Have a clear understanding of the tester's role.

- Your job is to provide information by surfacing facts and reporting them, not by suppressing information.
- You are neither the quality gatekeeper nor the quality judge and jury. Never be judgmental about bugs or other problems, even if—and especially when—the quality is poor.

Assess the culture of the team and find ways to fit in.

- Identify the principal objective and make it your objective. Although this may sound obvious, it's hard for most testers to put another objective ahead of quality and really mean it. If meeting the schedule is paramount, you need to focus on finding bugs that might threaten the schedule.
- Use the project objective as the frame of reference for your important conversations. If you talk in terms of absolute quality when the project driver is the schedule, your message will seem irrelevant. Instead, talk about project issues in relation to schedule risk.
- Make allies rather than enemies.
- Try to understand what motivates other team members. Are developers rewarded for "completing" development on time, regardless of whether they have tested their code?
- Offer to help where you can, but never neglect your own job.
- Don't drop public bombshells that could embarrass people or catch them by surprise.
- Don't try to score points in meetings. If your testing is blocked by bugs, talk to the developer or development lead privately and enlist her help. Then you can report that you are working together to resolve the problem.
- Stand up for what you believe in, but pick your battles. It helps to have a mental framework, such as project risk, for deciding which battles are crucial. If the schedule is the project driver, then evaluate each potential battle in relation to schedule risk. Is this issue important enough to delay the project?

Learn to negotiate.

- Propose a reasonable conclusion that is in the best interest of the project. People often begin negotiations with unreasonable demands in case they need to negotiate down, but this can foster bad feelings and injure your credibility.
- Avoid adversarial negotiation. Offer win-win propositions.

Ask how you will be measured, and ask for regular feedback on your performance.

mation, or is the report verbose?
- Focus on essentials.

Someone will read every project document you write. Improve your reports by asking developers for feedback and incorporating their suggestions. This will make the lives of the developers easier and encourage them to be helpful with your reports. Writing, like programming, requires debugging and testing.

Practice is essential. Write something every day, and use the three Cs to examine your work critically. Write about anything—from impressions of the day's testing to observations of the subway ride home. Set each piece aside for a few days and then review it. Does it say what you wanted to say? How could you make it more clear, complete, and concise?

Finally, read on any subject that interests you by authors whose writing you admire, and try to understand why their writing works. What makes a particular book or article clear, interesting, and easy to read?

ORAL COMMUNICATION

You can flex your communication skills while speaking, too. When you speak up in a status meeting or talk to a project manager about what you have not been able to cover in your testing, you need to be clear, complete, and concise.

Many everyday occasions require a tester to have good oral communication skills:

- Asking questions about the software and interpreting the answers
- Interviewing for jobs or project roles
- Presenting testing status in a project meeting

Each of these requires you to communicate your meaning accurately. You also need to hear what others are saying and respond appropriately.

STAY IN-BOUNDS

If you dive straight into the details or wander off the point, you will lose your audience. Start by summing up the critical information. That may lead to questions or a request for supporting details. Only when you are sure your audience has understood the main point—and is willing to hear more—should you move out into amplifying your message. It is essential to stick to the point and deliver the details in a logical sequence. A tester who can tell a compelling story in this way is more likely to be heard.

Practice by writing down the most important points ahead of a meeting. Summarize each point concisely without obscuring the message and rank the points in order of importance. After the meeting, review how well you did. Did the participants quickly grasp the essential information? Were there questions you could have anticipated? Were people impatient with your explanations? What could you improve next time?

Ask yourself the same questions about impromptu conversations. If someone requests information and seems dissatisfied with your answers, ask them how you can improve your delivery.

ENDURANCE TRAINING: Listening

Listening is an equally important part of communication. Anytime you talk—and even when you don't—you should be prepared to listen.

As in exploratory testing, the answer to one question will lead you to the next question. Even if you have brought a prepared set of questions to the interview, it's dangerous to stick to your agenda when a developer or architect is potentially telling you important things about the software. Ask clarifying questions, delve deeper on a particular thread, and broaden the inquiry—all these depend on your ability to hear what is really being said.

Listening to questions is as essential as listening to answers. This should be obvious as early as the job interview. Some testers come to interviews and don't really hear the questions. If I'm faced with a candidate's failing to answer my questions after a couple of tries, I terminate the interview. I know I will have a frustrating time with a poor listener on my team, even if that person is a wonderful tester. Solid contenders for the job will admit they don't understand a question or ask for clarification before answering thoughtfully.

The need to listen is perhaps less

Tips for Difficult Conversations

Be prepared.

- Have a clear sense of your vision of testing and the value it brings to the project.
- Keep track of your work, and be prepared with facts to demonstrate coverage and progress.
- Prepare and file a weekly status report, even if it's not a project requirement. At the very least, clearly outline what you did, what you found, and any issues impeding progress. If nobody wants it, do it for yourself.

Interact with the others in a conversation.

- Listen carefully to the other person's point of view, and ask for clarifications where needed.
- Deal with questions directly. If you don't know the answer to a particular question, say that you will investigate and come back with the answer.
- Acknowledge the importance of the other person's viewpoint. If you can't honestly do that, at least acknowledge the strength of his convictions or feelings, as in, "I understand that this is important to you."

Always try to sound cooperative and positive.

- Watch your tone. It can be difficult to avoid sounding impatient or abrupt when you see your point of view as a self-evident truth, but you will be more persuasive if you are open.
- Watch the other person's reactions. If you sense resistance, you may be coming on too strong. Make sure you explain your points, and don't patronize others if they take a little time to get there.

Try to maintain an even emotional keel, especially when a project becomes stressful.

- If it helps to think you wouldn't be employed without software bugs, then do that.

Ask for time is you need it.

- If you are caught by surprise, say something like, "This is an important conversation, and I want to do it justice. May I have a little time to prepare?"

obvious in a status meeting, but a tester cannot afford to ignore everyone else's status. Your effectiveness depends on knowing what is going on elsewhere in the project. Listening to others—observing their body language, hearing what they are saying and what they are not saying, and asking pertinent questions—gives you crucial clues about the state of the software. A project meeting provides an opportunity to pick up other people's perceptions about the testing or testers and possibly influence them for the better.

Informal conversation with other project members is another common situation where listening skills are important for a tester. Typically, developers' chitchat is far in advance of official news and can be invaluable in directing your testing. Listen carefully and apply a "reasonableness" filter.

You can practice listening in various ways. Build in a feedback step to verify that you heard correctly. Offer to take meeting minutes and request feedback from participants before final distribution. Summarize the key points of conversations and play them back. During a conversation, you can do this before moving to the next point. Say something like, "I'd like to play that back to make sure I understand what you're saying."

Playing back important conversations in writing will ensure that you and the other participants agree about what was said and prevent any later misunderstanding.

Summarize the key points in an email, and preface your message with something like, "The following summarizes my understanding of our conversation. Please let me know of any omissions, errors, etc."

Review the comments that come back. If you missed or misinterpreted important information, ask yourself how that happened. Were you too focused on thinking about your next point to really hear the other person? Were you trying to fit what you heard into a preconceived model? These are common causes of poor listening. Keep notes about what you learn, and review the notes regularly as a reminder of what to watch out for.

SWEAT IT OUT: Difficult Conversations

Sooner or later, every tester will have to deal with difficult conversations—for example, defending your test strategy to skeptical managers or developers, or explaining or justifying why testing is taking so long.

Challenging situations like these can happen on any size project. Project managers, architects, and developers sometimes hold firm opinions about the tester's place on a project and just how, or how much, you should test.

Occasions will arise when you have to deliver unwelcome news. One of the most difficult conversations you can have as a responsible tester occurs when you have to persuade managers that the software you have been testing is not ready, and it would be a better use of everyone's time to send it back to development.

Conclusion

Soft skills are not only skills in and of themselves but also are a critical part of your mental conditioning as a master of technical skills. Developing teamwork and communication skills takes practice and patience. As you develop and nurture all of your skills—technical and non-technical, hard and soft—you will find less and less distinction between them. Together, they are all skills that will make you a more effective tester. **{end}**

Fiona Charles specializes in managing large systems integration tests for clients in retail, banking, financial services, and telecommunications. With more than twenty-five years of experience in software and systems development projects, she has been a technical writer, tester, QA manager, consultant, and test manager, most recently with IBM Global Services in Toronto. Contact Fiona at fccharles@sympatico.ca.

Sticky Notes

For more on the following topic, go to www.StickyMinds.com/bettersoftware.

- Further Reading

Your Search for the Best is Over...

SOFTWARE QUALITY ENGINEERING SOFTWARE TRAINING

TESTING • DEVELOPMENT • MANAGEMENT • REQUIREMENTS

CELEBRATING 20 YEARS

Software Quality Engineering provides the widest selection of specialized training courses and we're improving our selection all the time. Developed and delivered by top industry consultants, all courses are based on the latest industry practices and updated regularly to reflect current technologies, trends, and issues. Pick the training method that works best for you and your organization—public training, on-site training, or eTraining.

For more information about Software Quality Engineering's training courses and the complete 2006 Training schedule, visit **www.sqe.com**.

SOFTWARE QUALITY ENGINEERING

www.sqe.com

SPRING 2006 TRAINING SCHEDULE

TESTING

2/13-2/15	Software Testing Certification—Foundation Level	Denver, CO
2/28-3/2	Software Testing Certification—Foundation Level	Tampa, FL
3/7-3/9	Software Testing Certification—Foundation Level	Toronto, Canada
3/20-3/22	Systematic Software Testing	San Diego, CA
3/20-3/22	Software Testing Certification—Foundation Level	San Diego, CA
3/21-3/23	Software Testing Certification—Foundation Level	Dallas, TX
3/23-3/24	Mastering Test Design	San Diego, CA
3/23-3/24	Performance, Load, and Stress, Testing	San Diego, CA
3/23-3/24	How to Break Software	San Diego, CA
3/23-3/24	Requirements Based Testing	San Diego, CA
4/4-4/6	Software Testing Certification—Foundation Level	Bethesda, MD
4/18-4/20	Software Testing Certification—Foundation Level	San Francisco, CA
4/24-4/26	Systematic Software Testing	Boston, MA
4/24-4/26	Software Testing Certification—Foundation Level	Boston, MA
4/25-4/27	Software Testing Certification—Foundation Level	St. Louis, MO
4/27-4/28	Mastering Test Design	Boston, MA
4/27-4/28	Performance, Load, and Stress, Testing	Boston, MA
4/27-4/28	How to Break Software	Boston, MA
4/27-4/28	Requirements Based Testing	Boston, MA
5/2-5/4	Software Testing Certification—Foundation Level	Seattle, WA
5/14-5/16	Software Testing Certification—Foundation Level	Orlando, FL

DEVELOPMENT

3/27-3/29	Essential Software Requirements	Seattle, WA
3/27-3/29	Immersion in Test Driven Development	Seattle, WA
3/30-3/31	Agile Development Practices	Seattle, WA
5/1-5/3	Essential Software Requirements	Chicago, IL
5/1-5/3	Immersion in Test Driven Development	Chicago, IL
5/4-5/5	Agile Development Practices	Chicago, IL

MANAGEMENT

3/20-3/22	Test Management	San Diego, CA
3/21-3/23	Managing the Test Process	Philadelphia area
3/23	Test Process & Performance Improvement	San Diego, CA
4/24-4/26	Test Management	Boston, MA
4/25-4/27	Managing the Test Process	Chicago, IL
4/27	Managing Test Outsourcing	Boston, MA

REQUIREMENTS

3/20-3/22	Writing Testable Requirements	San Diego, CA
4/24-4/26	Writing Testable Requirements	Boston, MA

Are We There Yet?

CREATING PROJECT DASHBOARDS TO DISPLAY PROJECT PROGRESS

BY JOHANNA ROTHMAN

When it comes to projects, there are as many questions to answer as there are project teams, but "Where are we?" is by far the most popular. The key to understanding a project is to make regular measurements—both quantitative and qualitative—and display the measurements publicly. When project managers display these measurements as part of the project status, teams are able to adjust their work and proceed more successfully.

I like to call these measurements a "project dashboard." You may not be able to show all the project measurements in one small area, but taken together, the project measurements display your velocity, distance, consumption, and location—much as a car dashboard does.

Use Multi-dimension Measurements

It's easy to measure some facets of a project, such as the project start date, the current date, and the desired release date, and say, "We're X percent of the way along," because the project team has used that percentage of time. But if all you measure is the schedule, you're almost guaranteed not to meet the desired deadline.

I like to measure a project along at least four out of six dimensions. I think of this as a project pyramid (see Figure 1).

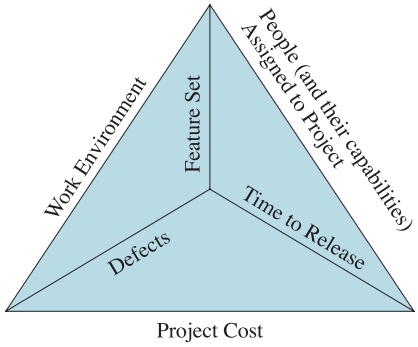


Figure 1: Project pyramid

The outside of the pyramid represents the constraints under which management has approved and funded the project. These project constraints are the work environment, the people and their capabilities, and the proposed project cost. I call them constraints because senior management tends to fix these attributes early in the project, and they tend to be difficult to change. Cost is the most likely to change, but in my experience it changes only when it becomes clear that the team can't meet the desired deadline.

The project requirements are what your customers care about. (And yes, if you work in an IT group, it is likely that your customers are the same people who constrained the project's cost, people, and environment.) Your customers care about what you're going to deliver (the feature set), when they'll receive it (time to release, the schedule), and how good that stuff is (defects). If you measure all sides of the pyramid, you will see a truer picture of your project than if you measure only one thing, such as schedule or defects, the two most common measurements I see.

Why Not Measure Earned Value?

So why not measure earned value? Earned value (taking credit for what you've been able to create in a specific amount of time) was developed to manage the tradeoff between measuring just the schedule and measuring what's been accomplished. Earned value makes sense for products that can be created in self-contained pieces. You create a piece and measure how long it took and how much it cost (in people and time) to create that piece.

But for many software projects, it's extremely difficult to calculate earned value. That's because all parts of a software project are interdependent. Even if you implement feature by feature—which is as independently organized as you can make a software project—when you work on a new feature, you may have to return to completed work and change it. When you change completed work, it's harder to determine the true earned value. Did you lose value because you changed something? Or, did you build on already-earned value? I have not worked on any project where it was possible to calculate earned value. In my experience, earned value is too often a fuzzy measurement for software projects.

Measure Project Completion

By using several measurements around the project pyramid, you can measure project completion. Project completion is a function of how accurate your original estimate was and how much progress you've made, but measuring only the schedule progress is not good enough. The only accurate way to measure progress for a software project is to measure how many features the project team has completed, how good those features are, and how many features are left to implement.

I once assessed a project in an organization where the developers had met every single date in the project schedules, but the testers were consistently late. Seems suspicious, doesn't it? The developers hadn't actually met any milestones at all. They checked in stubs and fixed the code when the testers reported defects, but because the project managers only looked at the dates—and never measured anything *but* the dates—the developers could say they had met the deadlines without actually meeting them.

As shown in the velocity chart in Figure 2, the number of features grew over the course of the project. The project team started with fifty features but released sixty-five features. If the team hadn't tracked its progress, including the number of features, team members would not have been able to explain to their management why things took "so long."

If I'm not implementing by feature, I like to use progress toward release criteria as a project completion measurement. Table 1 is an example of how I use release criteria to track project completion—or the lack thereof.

Release criteria are a late-in-the-project measurement. Even if you start assessing release criteria progress at the beginning of a project, most often, the release criteria data are available close to the end of the project.

No matter what lifecycle model you've selected for your project, to determine how good your initial estimate was, you can use the Estimation Quality Factor (EQF), originally described by Tom DeMarco in *Controlling Software Projects*. At periodic intervals during the project, the project team answers the question "When do you think we'll be done?" Each data point is the consensus agreement on when the project team believes the project will be finished. At the end of the project, draw a line backward from the release date to the beginning of the project. The area between the line you drew and the when-will-we-be-finished line is how far off your estimation was. This is a great technique for people to use as feedback on their individual estimates. But even if you don't use it for feedback, it's a great technique for the project manager to see what's going on.

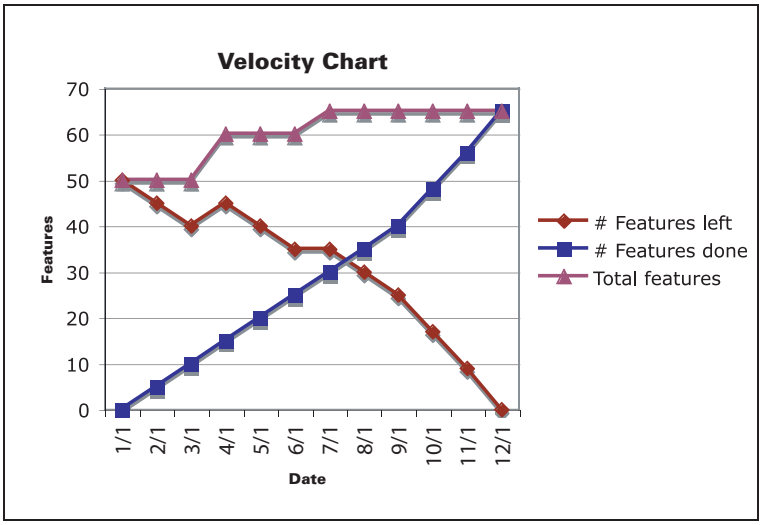


Figure 2: A velocity chart can be used to track schedule progress.

Criterion	Status	Status	Status	Status
Performance of Scenario 1 under 10 seconds	5/1, build 57: Performance < 30 seconds	5/8, build 70: Performance < 15 seconds	5/15, build 78: Performance < 12 seconds	5/22, build 85: Performance < 8 seconds
Number of defects found decreasing for at least 4 weeks	5/1, build 57: 22 defects found, same as last week	5/8, build 70: 15 defects found	5/15, build 78: 5 defects found	5/22, build 85: 2 defects found

Table 1: Use release criteria to track project completion.

Figure 3 is a chart of an EQF for a project that was originally supposed to be nine months long. For the first couple of months, when the project manager asked when people thought they'd finish, they said "September 1." And for a couple of months, they were optimistic, thinking that they might finish early. But during the fifth month, team members realized they didn't know enough about some of the requirements. What they discovered changed the architecture and pushed out the date. For the next few months, they still weren't sure of the date. They realized in the last three months of the project that, because of the changing architecture, they were encountering many defects they hadn't anticipated. So, evaluating EQF, a qualitative metric, was helpful to the project manager and the project team as a check against the progress charts.

Schedule estimates are just guesses, so anything you can do to show and then explain why your schedule varies from the initial plan will be helpful to anyone who wants to know "where are we?"

Collect a Variety of Project Measurements

Project completion measurements may be all your managers want to see, but if you're a project manager or a technical lead on a project team, I'm sure you'd like some early warning signs that the schedule may not be accurate. To keep my finger on the pulse of a project, I monitor several measurements:

- Schedule estimates and actuals, aside from EQF
- When people (with the appropriate capabilities) are assigned to the project versus when they are needed
- Requirements changes throughout the project
- Fault feedback ratio throughout the project
- Cost to fix a defect throughout the project
- Defect find/close/remaining open rates throughout the project

Measure the Schedule When It's All You've Got

Gantt charts aren't the only thing I use to measure a schedule. Instead, I look at when the project team expected to meet a particular milestone and when they actually met that milestone. If the project team starts the project late (no matter what the first milestone is), that project is not going to meet the desired end date. Time lost is never going to be regained. Figure 4 shows what a comparison of schedule estimates and reality looks like.

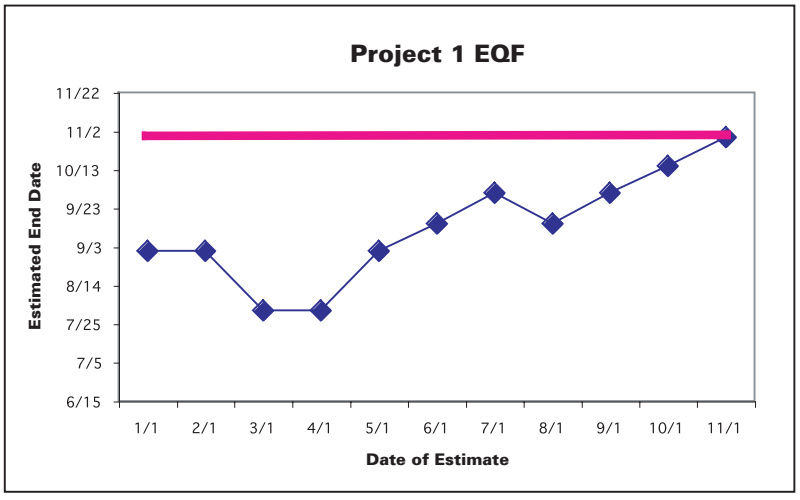


Figure 3: Example EQF for a project

This project is a modified waterfall lifecycle (the next phase can start without the previous phase being complete), but there are no iterations. Notice that the project started a full month late. When the project manager posted this chart, he said to senior management, "Don't expect us to pull in the schedule by a month. We started late; we can't make up the time." To the project team he said, "I'd like you to work as intensely as you can, without working overtime and getting tired. We don't have time for you to make mistakes. Do the best job as quickly as you can, and we'll keep tracking where we are."

See When Qualified People Actually Work on the Project

Too many projects start starved of resources. This can happen if some of the people are still working on a previous project, if people are yanked off partway through the project, or if your project is competing with several others for people's time. The problem with starving a project is that no matter how hard people work when they are working on the project, they can't make progress if they are assigned elsewhere or are attempting to multi-task on several projects. Figure 5 shows a project where the total number of planned person-months (666) was 75 percent of the actual person-months

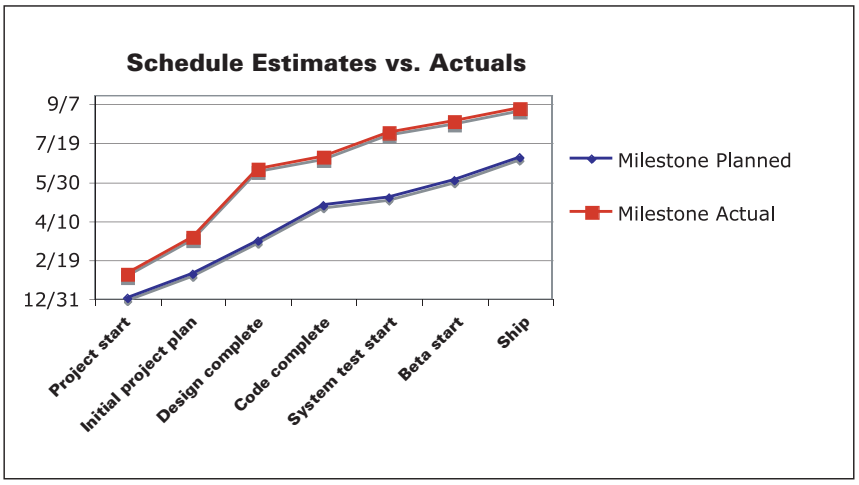


Figure 4: Schedule estimates vs. actuals

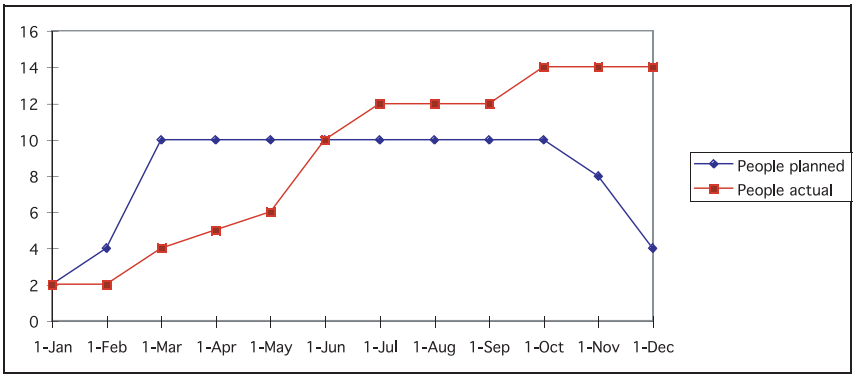


Figure 5: Tracking people's actual assignment to the project

(878). Unfortunately, the team's output was about 66 percent of the desired result.

For more expense (about 1.3 times the original planned cost), the team delivered fewer features (0.3 times the original feature set). You might be asking why the project team would deliver less than planned for more cost? This project used a staged delivery lifecycle, where requirements, analysis, and architecture were timeboxed. The project team could obtain a good idea about the requirements and their effect on the architecture, but not know all the requirements in detail or know the implications of those requirements on the architecture. The original plan was to spend the first two months obtaining a good idea and the next two months performing an initial iteration to make sure the rest of the project would succeed. In order to be successful, the project plan required all the people planned in those first four months to perform all the initial investigation and iteration work. Since the people were not available and the end date was non-negotiable, the project team needed more people to prototype and iterate in parallel.

As people prototyped and iterated, they found mistakes—work that had not been completed in the initial timeboxing and iteration. Team members decided they would rather release a product that worked a little rather than release a product that had all the features, most of which didn't work.

This chart supplied the project manager with opportunities throughout the project—and when planning for the next project—to explain to senior management the problem of starving a project.

If you ever start your projects starved of people who are capable of performing at 100 percent, this figure can help explain the consequences of that decision.

Determine the Rate of Change on Your Project

You may be working with people who are uncomfortable with velocity charts. Or, they may not believe the impact that some changes have on requirements. In that case, you can use a requirements change chart (see Figure 6).

In this case, I was the project manager. I had a simple criterion for deciding if the requirements change was major or minor. A minor change affected one module, and a major change required changes to more than one module. To make this decision, I used the principle that "interface changes between modules tend to create defects."

In this chart there are a lot of small changes—something most of us expect on projects. But we also encountered some major requirements changes late in the project (Week 22). When I saw these changes, I was able to explain to senior management that either the project would be later than we expected or the number of defects would rise. But with these changes, it was clear that the original date and the original feature set with the small number of expected defects was not possible.

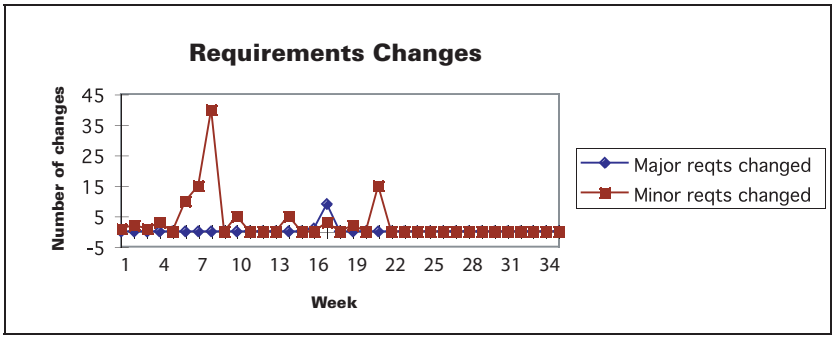


Figure 6: Requirements changes by week

See if the Developers Are Making Progress or Spinning Their Wheels

Once the project team is writing code, you can measure the fault feedback ratio (FFR). The FFR is the number of bad fixes to the total number of fixes. In my experience an FFR of 10 percent or more says that the developers are having trouble making progress.

I like to measure the FFR on a weekly basis. I use the FFR as data to initiate a discussion with the developers and testers. If I see a week where the FFR is high, I first check to see how many total problems were fixed that week. If only four problems were fixed and one was a bad fix, the developers and testers are probably OK. But if I see twenty defects fixed and five of them were bad fixes (25 percent), it's more likely that somebody or a few somebodies are having trouble. In Figure 7, notice that the FFR starts to get high around Week 6 and stays over 10 percent until Week 13. Once the project team hit the second week of high FFR, the project manager instituted peer review on all fixes. That helped, but there was a delay between the start of peer review and the reduction of FFR back to numbers where the fixes didn't interfere with progress.

To identify trouble areas, I first ask the developers if they are running into trouble with their fixes. I generally phrase the question this way: "When you fix something here, does a problem pop up over there?" I'll ask other questions, all leading to asking the developers if they need any resources to fix this problem altogether. If I hear that the developers want to redesign a module, we discuss the issues for that redesign.

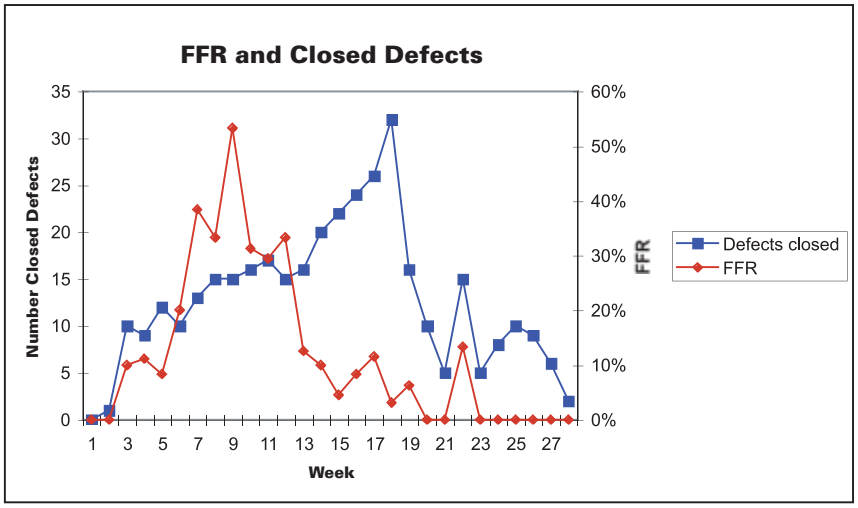


Figure 7: Fault feedback ratio compared to number of defects closed

My next question is for the testers: "Are you able to define all the conditions that create this problem?" I start with questions to see if the developers are fixing one piece of the problem at a time, or if the testers understand the system sufficiently to test thoroughly enough.

Measure How Much It Costs You to Find and Fix Problems

One key measure is how long it takes the project team to find and fix problems. You've probably seen "industry standard numbers" that show that it costs you one unit to fix a problem in the requirements phase, ten units to fix a problem in design, one hundred units in code, 1,000 units in test, and 10,000 units in post-release. We normally think of units as dollars or some other form of currency.

I've measured cost to fix a defect, and the numbers I find are different. Table 2 shows costs from a couple of projects.

Remember, it's not just the cost per defect; it's the cost per defect times the total number of defects. If you're not looking at the overall cost, you can't know where to spend your time. Based on cost to fix a defect from previous projects, you might decide to be proactive and use inspections of key project documents, test-driven development, or pair programming for the more challenging defects. Or you might decide to monitor cost to fix a defect and react as necessary, such as choosing when to institute peer review of fixes or inspection of all code.

If you haven't performed any proactive defect-finding activities, the cost to *find* a defect is fairly small. But the cost to *fix* can be high, and the overall cost to fix all the defects is very large. If you have been proactive using techniques such as test driven development, pair programming, inspections, or peer reviews, the cost to *find* a defect can be higher, because you've already looked for defects. But the cost to fix a defect tends to be lower when a project team has been proactive in trying to find defects early. And the overall number of defects is lower, lowering your total cost to fix defects for a particular release.

I monitor cost to find and fix so I can see if the developers or testers are surprised by what's in the code base. I have a couple of rules of thumb, assuming the developers have not been proactively looking for defects:

- The longer it takes developers to fix a problem, the more likely it is they are afraid of touching parts of the system or don't understand parts of the system.
- The longer it takes the testers to find problems, the less they know about the product or the less they

Project Phase/Cost	Requirements	Design	Code	Test	Post-release
Project 1 (reactive for defects)	Not measured	Not measured	0.5 person-day	1 person-day	18 person-days
Project 2 (proactive for defects)	0.25 person-day	0.25 person-day	0.5 person-day	0.5 person-day	8 person-days

Table 2: Measured cost to fix a defect for two projects

Three time zones, one solution

DevTrack

The Powerful, Configurable and Scalable Solution for Issue and Process Management

Powerful	Configurable	Scalable
Manage millions of issues	Point-and-click administration	Used by businesses of all sizes
Built-in indexed search engine	Graphic workflow editor	From small teams to 1000s of users
Windows and Web client	Fully configurable user interface	Distributed team support

TechExcel

www.techexcel.com | 1-800-439-7782

More than 1000 companies worldwide have chosen DevTrack for its power and ease-of-use. Visit www.techexcel.com to download a free, fully-functional 30-day trial.

know about multiple techniques to test the product.

The longer a defect takes to fix, the more careful we'll have to be when deciding what to fix just before release.

Understand if the Developers and the Testers are Making Progress with Defects

Almost everyone measures defect trends. I've seen some intricate defect trend charts, but my favorite chart shows just three things: the number of new defects found per week, the number of closed defects per week, and the number of remaining open defects per week (see Figure 8).

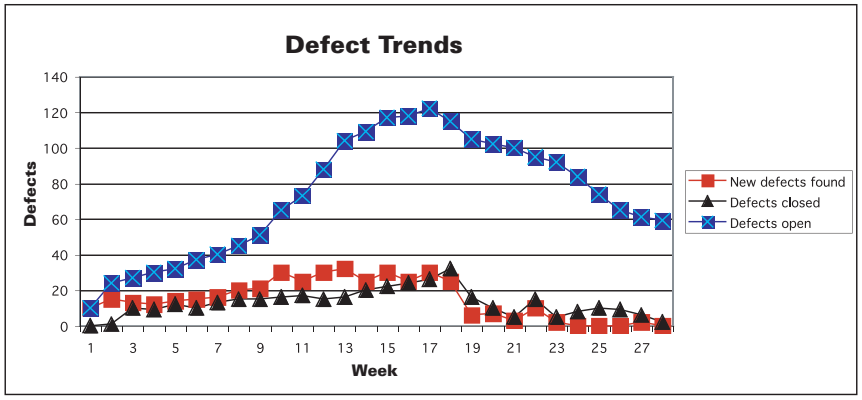


Figure 8: Defect trend chart

Area/Module/Feature	Last Test Date	State	Next Planned Test
Module A	1/12, Build 37	Blocked. See Ann for details.	Build 42
Module B	1/13, Build 40	Passed all regression tests.	Recheck with Build 42
Module C	1/12, Build 38	Passed all regression tests. Exploratory testing ongoing.	Ongoing Checking
Module D	1/13, Build 39	Vijay and Dan working together on regression tests. Major problems.	Build 41
Module E	1/13, Build 40	Passed all regression tests. All fixes verified. No more effort until final cycle.	Final Build
Overall Status	As of 1/13, 4 p.m.	Approximately half done with planned system testing.	Projected Last Build: 57

Table 3: Progress charts can be used to explain status.

I specifically do not chart defects by priority. I find that the project team and senior management become too willing to play the promotion/demotion game when I chart defects by priority. And the developers have to read through all the defects, even if they are supposedly a lower priority, so I just count all the defects.

I count the number of remaining open defects so I can see when the close rate passes the find rate, enough so that the number of remaining open defects starts to decrease. I look for the knee of that remaining open defects curve, knowing that as the slope of the remaining number of open defects goes negative, the risk of release lessens.

Display Qualitative Data

It would be easy if all the project data could be displayed on trend charts. But you need a different kind of chart, especially when you're trying to explain the status of something. I've used progress charts like the one in Table 3 when trying to explain the progress of algorithm development, performance scenarios, and testing.

Displaying Data

I try to be consistent in my charts. Red is bad; green is good. Up is bad; down is good. Bad and good are very judgmental words, especially when applied to data. After all, data just is. But you may not be able to explain your data to everyone all the time. In that case, it's helpful if people know how to read your charts. Tufte's *The Visual Display of Quantitative Information* is a great resource, especially when your charts will be read and interpreted by other people.

In Data We Trust

Without data, you're just another person with just another opinion. And while your opinion might be good, it's not the same as having data. When you measure around your project pyramid, you're gathering data that not only will explain where you are in this project but also will help you plan for the next project. **{end}**

Johanna Rothman, a regular StickyMinds.com and Better Software *magazine columnist, enables managers, teams, and organizations to become more effective by applying her pragmatic approaches to project management, risk management, and people management. You can contact Johanna at jr@jrothman.com or by visiting her Web site, www.jrothman.com.*

Sticky Notes

For more on the following topic, go to www.StickyMinds.com/bettersoftware.

- References

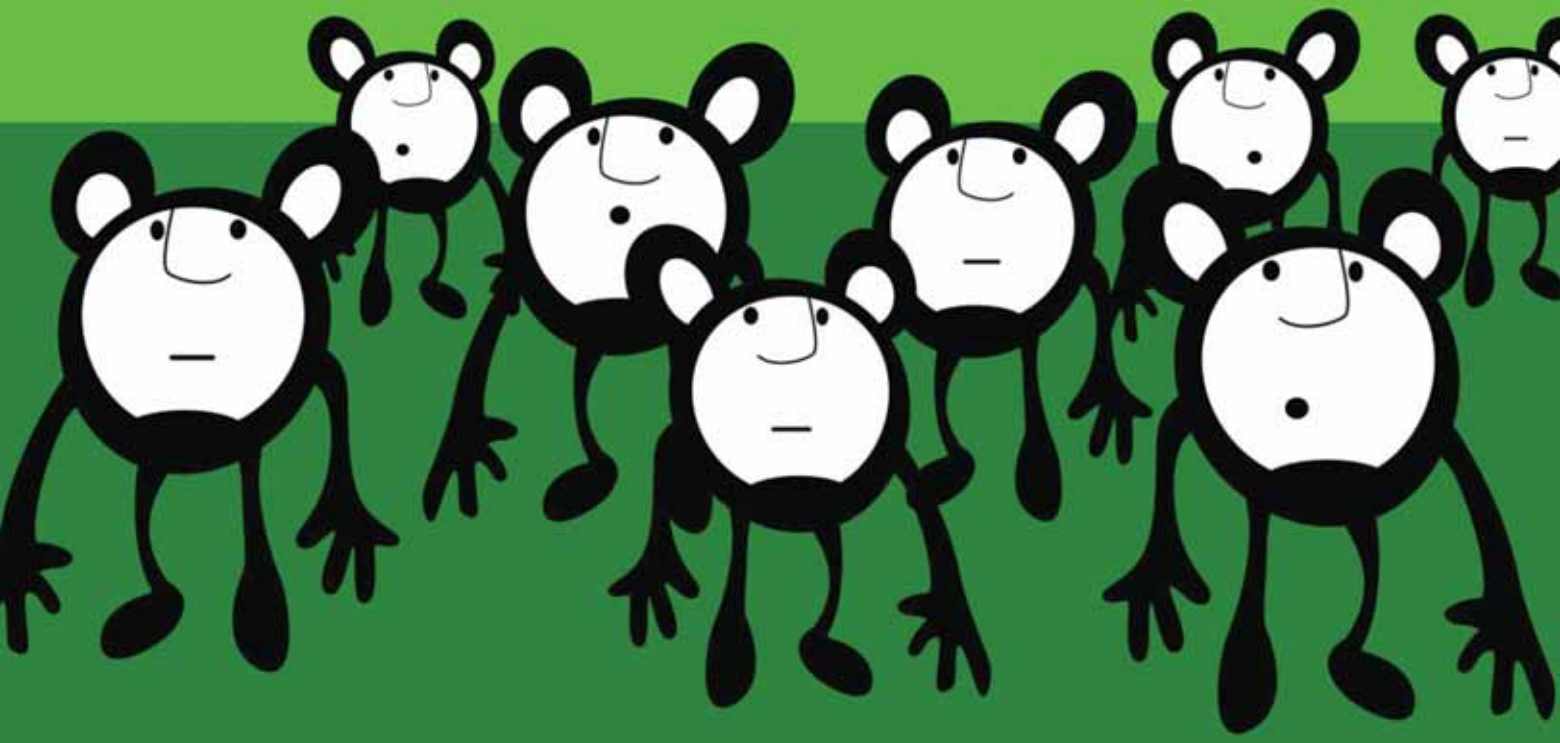
PowerPass

www.StickyMinds.com/PowerPass

Search no more. Access the information you need for your software projects with a StickyMinds.com PowerPass. Quickly search through the complete *Better Software* and *STQE* magazine archive, conference materials from every major Software Quality Engineering event, and online reference books and reports. Also tap into discounts on all Software Quality Engineering products and training. It's the ultimate information vehicle for software professionals who want to build and deliver better software. Seek us out at www.StickyMinds.com/PowerPass.

Code with Character

Getting to Know Your Data Types



When I'm writing code, I like my types to be straight shooters. I'm not into types that are complicated or mysterious. It's simply too much work to figure out who they are or what they might expect of me. That's why some of my favorite data types are those that are willing to be direct and honest with me.

Take my good buddy, Integer. An Integer data type doesn't pull any punches. He tells you he will hold signed numeric data, and that's exactly what he does. You try to give him a string or a date, and he'll laugh in your face. You'll get nothing unusual past him at compile time. He's purely a "what you see is what you get" type. Integer is about as trustworthy as they come.

The String type is a little harder to read. He tells me he's simply a handful of characters that are supposed to represent any old literal string and, for the most part, that is what he is. But he can be deceiving. One time I discovered him in an application where he was holding numbers and dates. It was very disappointing. You think you know a type, and then you find out he's been playing games with you. Still, String can be a reliable data type. I think he just strays from the path once in a while.

Some time ago, back in the era of COM, I knew a data type named Variant. This guy was unlike any other type I'd ever met. He was an integer, a string, an array, and almost any other type you'd ever met before. At first glance, you couldn't tell exactly what he was. Despite his confused nature, Variant was always good about telling you what mood he was in. If he was feeling like an Integer, he'd tell you outright. You had to ask, but he'd be very quick to let you know. So, even though you couldn't determine his state immediately, he gave

BY TOD GOLDING

you very reliable means of finding out. And for that I was grateful.

It wasn't until this Object data type rolled into town that I really started to question data types. Object was unlike any other type I had met before. He was the chameleon data type that could somehow be whatever he wanted to be. It was as if he had looked at all the other data types and asked if he could be something more, something that broke through all the type boundaries. This '60s, free-love spirit gave him a mystique that was extremely enticing.

Naturally, this disposition also made Object a very popular data type. Everyone seemed to want to have him around. Object was clearly the life of the party, and pretty soon he became the "in vogue" data type that developers just couldn't resist.

Object's appeal seemed rooted in something deeper than pure sex appeal. He really seemed to be adding value to applications. If you wanted to write a data container, for example, the Object data type allowed you to create a general container that could manage any type that could be represented as an object.

Imagine how different the .NET Base Class Library would look if there were no Object data type. `ArrayList`, `Hashtable`, and the like—all staples of the BCL—are made possible by the existence of this type.

Reaching Beyond Data Containers

Given its nature, it was only logical to expect that the popularity of Object would reach beyond collections. The allure of Object's charms soon started showing up in the signatures of everyday interfaces, delegates, and so on. The end result of all the Object euphoria was a population of classes that often seemed to make excessive use of this new type. The class shown in Figure 1 represents an example of objects gone wild.

```
public class DataAccessManager {
    public ArrayList GetDataObjects(DataType type) {}
    public Hashtable GetIndexedCollection(DataType type) {}
    public object GetDataObjectById(DataType type, object id) {}
    public object AddDataObject(object dataObject) {}
    public ArrayList DeleteDataObject(object id) {}
    public bool UpdateDataObject(object dataObject) {}
}
```

figure 1

After assembling this masterpiece, many might be quick to show it off to their friends. After all, we get excited by the generality of our solutions. It's our nature. Reusability gets us feeling like we're really building something more than just another simple, boring class. And after creating this new `DataAccessManager` class, some might think they've built something special.

`DataAccessManager` represents the Ginsu knife of data access managers. It provides all the common CRUD (Create, Read, Update, Delete) operations you'll need—in one clean, self-contained class that can be reused across a whole host of data object types. Of course, this was all made possible by your easy-going, type-friendly buddy, Object. Sure, you could have pulled off something like this with your own type hierarchy, but who needs all that polymorphism? Let's just throw some object types into this interface, and the `DataAccessManager` class will be ready to conquer the world.

If you were the author of this class, you'd probably be relatively comfortable with how it works. However, if you're only a client of this class, you may not be so enthusiastic about its interface. The truth is: The interface of this class is so general that it essentially tells you nothing about what it expects or returns. Consumers of this class would need a pack of tarot cards and a decoder ring to figure out the intent and type expectations of these methods.

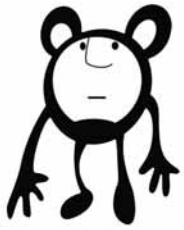
Let's start by looking at all the `ArrayLists` that show up here. What does an `ArrayList` really tell you when it's included in an interface? `ArrayList` is as mysterious and elusive as Object. All it really says is: "Hey, here's a collection of some objects. You go figure out what types I might contain." The same could be said for the other object types that show up in the signature of this interface.

So where does that leave you as a client of this class? Well, it basically turns you into a hunter and gatherer. It forces you to hunt through the existing code to gather information about what types are "really" valid for this interface. Does it accept any object type? Do the objects need to support any specific interfaces?

As you can see, all this generality simply pushes responsibility and type awareness out to the client. Consumers of this class will now be forced to litter their code with casts and dependencies that could easily be broken by some small change you might introduce into two indirectly related pieces of code. And, to top this off, you may not actually discover that you've violated this dependency until one of these casts fails at run time.

Imagine some relatively obscure, less frequently exercised path of execution

The truth is:
The interface of this class is so general that it essentially tells you nothing about what it expects or returns.



that appears in your code and happens to rely on one of these dependencies. It might easily slide through QA and find its way into production.

Even if you discount this entirely, there's still a fundamental problem with the expressiveness of this interface. I don't really want to be vague in my interfaces if I don't have to. However, `ArrayList` and Object and all these Object-related mechanisms take me down the slippery slope of vagueness; they all encourage me to be cryptic about what I'm conveying to my clients. That shouldn't be the default mode of operation for achieving generality.

Object is not exactly friendly to our old value types. To be represented in an `ArrayList`, for example, an integer data type must go through the utterly humiliating process of being boxed and treated like an object type. It's as if Integer's identity and purpose are being ignored in favor of the popularity of Object. Somehow, all types have to mimic Object's behavior to fit in. It's like high school all over again.

Of course, this whole act of boxing and un-boxing value types to make them behave like Object adds overhead. If you're going to put, say, 10,000 value types into an `ArrayList`, this overhead of boxing and un-boxing will not be insignificant.

As you can imagine, all this talk about casting, implied coupling, compromising of expressiveness, and boxing suddenly calls into question the true value of Object. Perhaps this type, which has represented himself as such a good friend, isn't really being so straightforward with you after all. Perhaps he's luring you into a lifestyle that has downstream impacts on the quality, maintainability, expressiveness, and performance of your code. Yeah, he's good to have as an encapsulating, polymorphic ally, but maybe we shouldn't be relying on him so heavily and pervasively.

To top things off, all this focus on Object is starting to make Integer, String, and Double downright depressed. After being there for you all those years, they are beginning to feel a bit neglected. They may not have been as glamorous as Object, but they had appeal and clarity that Object is simply not providing.

The key, then, is to find some way to strike a balance between the generality and reusability of objects while retaining the expressiveness and safety of these traditional, more specific types. It should be possible to be as open and flexible as Object, while remaining as practical and direct as Integer. I want all my types to simply get along and play nicely together. That shouldn't be too much to expect.

Mr. Generics to the Rescue

Fortunately, with the release of Visual Studio 2005, you have a new ally to combat these type obscurities. Your new-found friend, Mr. Generics, is going to step in and save the day. With generics you'll be able to abandon the idea of using objects as a least common denominator type and replace those objects with the *exact* types by allowing your types to be parameterized. So, much like you'd parameterize a method, your classes are now extended with the ability to accept type parameters that represent the exact type being managed.

As an example, I've modified our object-laden `DataAccessManager` class to accept a type parameter that will be the *exact* type being managed by the class (see Figure 2). This type parameter is then sprinkled throughout the methods of the class. I've made a few of the methods generic, adding parameters that will be used to determine the types of their incoming data .

Here, `DataAccessManager` has added a `<T>` to its declaration. This parameter should be viewed like any other parameter declaration, with one exception. You'll notice there is no type explicitly associated with the type parameter `T`. That's the key point here. `T` itself is not a placeholder for information—it is a placeholder for a type. So, when you declare this class, you'll now be required to supply a type as a parameter (referred to as the type argument). That type argument will be substituted for `T` everywhere it appears in the declaration of this class.

This same concept is applied to a few of the methods in this class. For example, `GetDataObjectById()` includes the declaration of a `<TId>` type parameter that will be used to define the type of the `Id` object that is supplied as an incoming parameter. This idea is included to illustrate how this flavor of parameterization extends beyond classes. Methods, delegates, interfaces, structs, and classes are all .NET constructs that have been extended to support parameterized types.

```
public class DataAccessManager <T> {
     public List<T> GetDataObjects() {}
     public Dictionary<long, T> GetIndexedCollection() {}
     public T GetDataObjectById<TId>(TId id) {}
     public T AddDataObject(T dataObject) {}
     public List<T> DeleteDataObject<TId>(TId id) {}
     public bool UpdateDataObject(T dataObject) {}
}
```

figure 2

Now, at first blush, you might be looking at this example and thinking: How the heck does this represent a more expressive interface? And, my response would be: At this level this interface isn't more expressive or even more readable. I could definitely see how some might think all this new syntax muddies the textual complexity of the class.

The real value shows up where it should be, in the code that constructs and consumes this class. Remember, the goal of expressiveness is to improve the client's experience. We're really trying to achieve clarity and safety for our clients. Making the internals of this class better is

```
DataAccessManager mgr = new DataAccessManager();

mgr.AddDataObject(new Customer(123));
mgr.AddDataObject(new Customer(321));

ArrayList customers = mgr.GetDataObjects(DataType.Customer);
for (int idx = 0; idx < customers.Count; idx++) {
     Customer cust1 = (Customer)customers[idx];
}

long id = 123;
Customer cust2 = (Customer)mgr.GetDataObjectById(DataType.Customer, id);
```

figure 3

a nice side effect, but our real gains in maintainability and type safety are going to be achieved via improvements in the client's experience.

In Figure 3 I've provided a very basic example of some client code that exercises the non-generic `DataAccessManager` class. This example adds some `Customers` to the database and follows this up with calls to `GetDataObjects()` and `GetDataObjectById()`. In the context of this simple example, this all seems perfectly harmless. The real problem is that, in the real world, the calls to add items to the database and the calls that retrieve, update, or delete data are typically separated in your code. In fact, it's possible they could live in very different areas of your code.

So, the cast and the general references to the `Customer` type that appear here are creating a series of implied contracts between the client who added the items and the code that is extracting them. It assumes, for example, that the cast required when fetching customers from the collection will, by default, succeed. To be honest, the code really has to make this assumption. At the same time, there's nothing about the interface of the `DataAccessManager` class that would have prevented me from adding `Dog` objects (or any other valid Object type) to this collection.

Overall, you may not find much wrong with this code. And, in the pre-generic era, we've all had to write code like this. However, now that we have generics, we should be much more critical of code of this nature. We shouldn't just assume that it's acceptable to make these compromises in type safety.

That's what's so nice about the generic version of the `DataAccessManager` class. There's nothing ambiguous about its interface. When I declare an instance of `DataAccessManager<Customer>` (in Figure 4), I've told my clients precisely what data type (in this case, `Customer`) will be acceptable for that instance, and the compiler will enforce

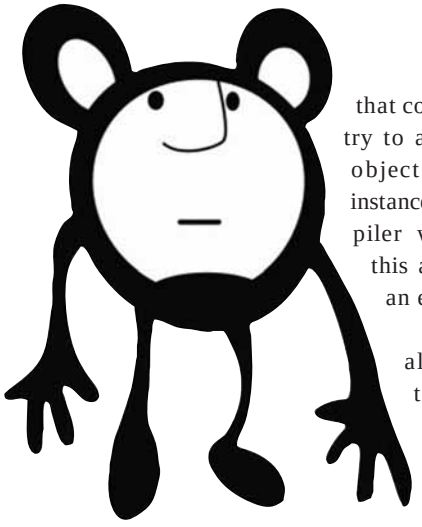
```
DataAccessManager<Customer> mgr = new DataAccessManager<Customer>();

mgr.AddDataObject(new Customer(123));
mgr.AddDataObject(new Customer(321));

List<Customer> customers = mgr.GetDataObjects();
for (int idx = 0; idx < customers.Count; idx++) {
     Customer cust1 = customers[idx];
}

long id = 123;
Customer cust2 = mgr.GetDataObjectById(id);
```

figure 4



that contract. If I try to add a `Dog` object to that instance, the compiler will catch this and throw an error.

You'll also notice that the casts are gone from my code. My call to `GetDataObjects()` now returns a `List<Customers>` type, which tells me that list can *only* contain `Customer` objects. And, as the code iterates over this collection, it's not forced to cast each item to a `Customer` type. The expressiveness and safety of this code are at the heart of what generics are trying to enable.

As an added bonus, you'll discover that the parameterization of your types allows you to overcome the boxing performance issues described earlier. You now can pass value type arguments to your generic types without fear that they'll be boxed behind the scenes. The .NET generics implementation went out of its way to be sure that its generic types would be able to support generics in the Common Language Runtime (CLR) and, as a result, the CLR can manage value type arguments without having to coerce them into being objects.

Where do generics leave us regarding our relationship with our data types? At a minimum, it seems that generics allow you to have more meaningful, more trusting, more type-safe interactions with your data types. The obscurity and mystery of the Object data type should no longer be an issue. This doesn't mean Object doesn't have its place. With generics, .NET programmers can contain some of Object's desire to find its way into places where it doesn't belong. It's just going to have to get used to the idea that it's no longer the center of attention. **{end}**

Tod Golding is the founder of Blue Puma Software, a technical consulting company that provides software training, mentoring, and development services. He has twenty years of experience as a software developer, lead architect, and development manager for organizations engaged in the delivery of large-scale commercial and internal solutions. Tod is the author of Professional .NET 2.0 Generics *and was a contributing author for the* XML Programming Bible. *Tod can be contacted at tod@bluepumasoftware.com.*

STARTING IN
THE FEBRUARY ISSUE,
TOD GOLDING
WILL BEGIN WRITING
OUR CODE CRAFT COLUMN.

A look at employment trends in 2005

The Better Software Magazine/StickyMinds.com Salary Survey

by Heather Shanholtzer

The beginning of a new year is a good time to take stock of career goals and accomplishments. It's a time when many people make a conscious decision to "Make this year the best one yet!" or move on to greener pastures. Whatever your workplace philosophy heading into 2006, you should find value in the results of our annual salary survey. How do your employment specifics compare to those of your peers? Are you ahead of the curve or trailing the pack? Living the dream or waking up in a cold sweat? Read on to find out how your job measures up to the industry average.

Getting to Know You

Demographically the survey results are very similar to last year's. The majority of our survey respondents (76%) work in the United States and Canada, but as always, it's evident that the industry is active globally. Forty percent of workers are in their thirties, and the gender ratio is still neck and neck.

One interesting change from last year is the increase in the years spent working in the industry. Obviously a year later we would expect more of our readers to report more experience, but the number of respondents who have been in the testing/QA industry for one to five years dropped from 36% in 2004 to 30% in 2005. This data seems to support recent studies that show fewer students are choosing to pursue careers in the software engineering industry.

Cracking the Books

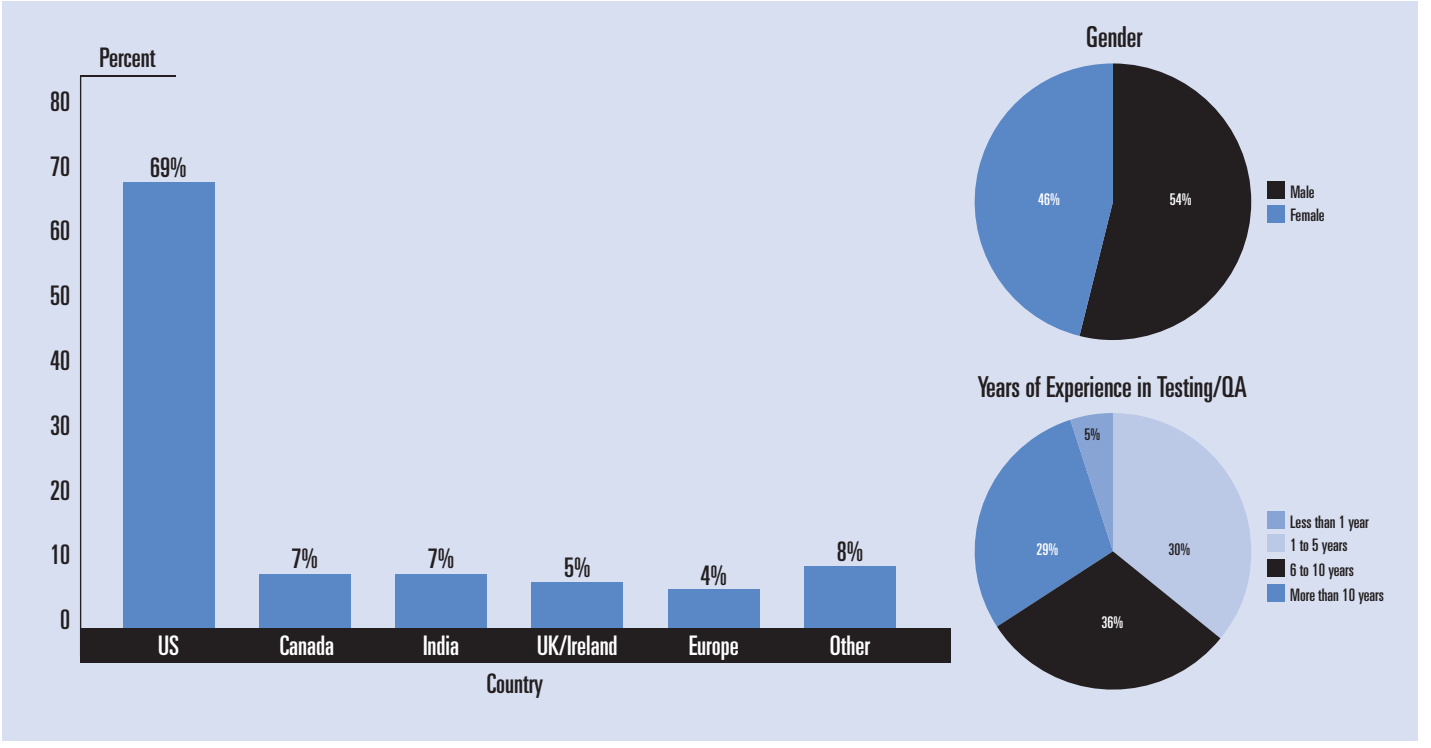
Education trends have remained constant. A bachelor's degree is still the most prevalent educational level (51%), and CS/IS remains the most popular field (31%). Other areas of study include engineering (20%), business (14%), liberal arts (10%), and mathematics (4%).

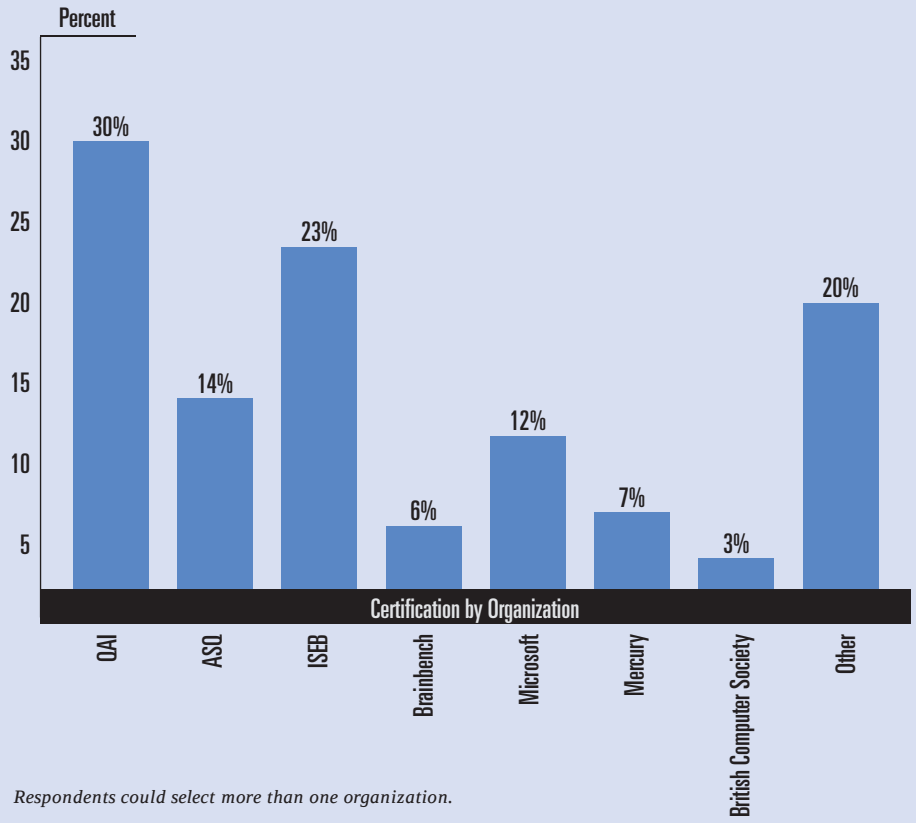
Putting Down Roots

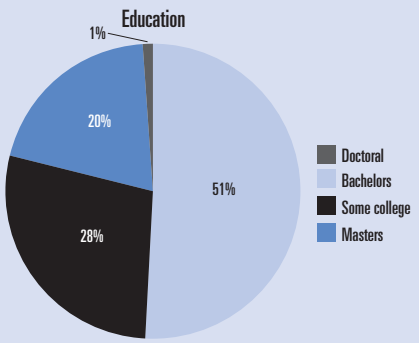
So what is it that you do? Well, an overwhelming majority (86%) of respondents describe their job function as test/QA, 6% are employed in development, and slightly more than 6% are working in IS/IT.

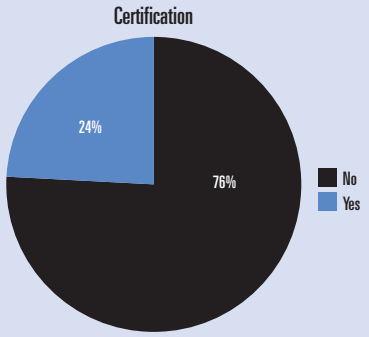
Nearly half of you (47%) have been with the same employer for one to five years, and 21% are fairly new to their jobs with less than a year under their belts. One to five years seems to be long enough for a lot of readers, as 60% report that as the length of time they worked for previous employers.

Surprisingly, almost all of you are working a standard, full time, forty-something hours a week job. Despite all of the communication devices and options available, very few people are working remotely or as part of a distributed team.

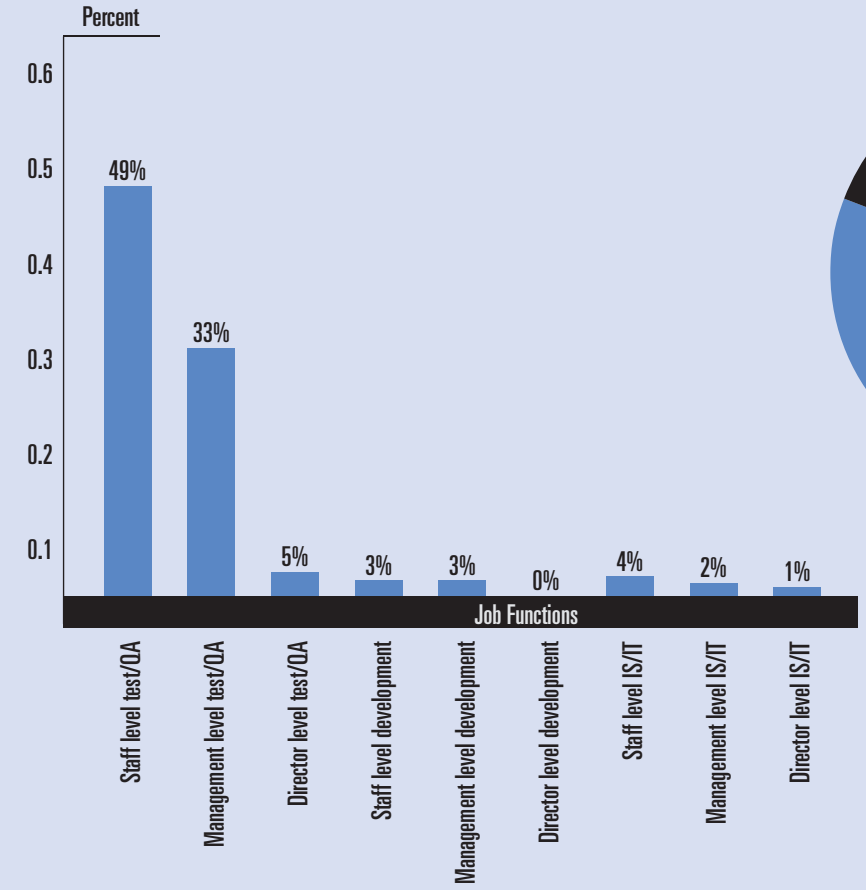


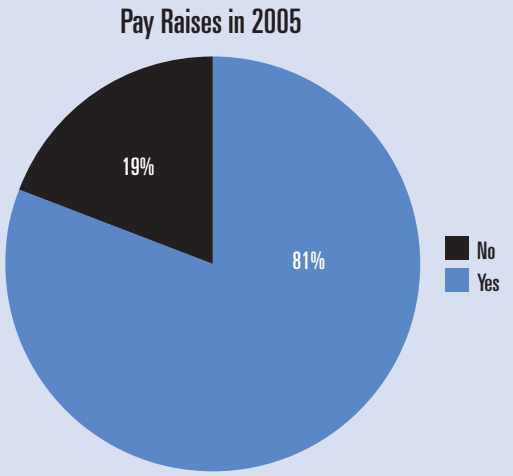






Respondents could select more than one organization.





Making It Worthwhile

As in most industries, job title and function determine the pay scale. In the world of software engineering, we can break it down to many levels. Forty-three percent of Test/QA staff report an annual salary of between $51,000 and $70,000. Twenty-five percent of the Test/QA managers and 70% of Test/QA directors who responded are bringing in $91,000 to more than $100,000 each year.

In the development realm, salaries for staff-level workers are pretty evenly distributed across the pay scale. The managers' salaries vary widely, with 23% claiming less than $30,000 per year and 20% bringing in more than $100,000. Incidentally, 100% of development directors report a more than $100,000 annual income.

As for IS/IT, 50% of staff members make between $41,000 and $60,000 each year. Managers report salaries across the pay scale, and all of the IS/IT directors are earning more than $51,000 annually.

While some might say money makes the world go 'round, others look for more than dollar signs when assessing the value of their jobs. Fifty-nine percent of you reported base pay as the most important thing about your jobs, but barely lagging behind came less tangible perks, such as job stability (48%), the challenge of the job (46%), and knowing that your opinions matter (43%).

More than half of respondents received a pay raise of up to 5%, and 21% of you received an increase of 5% to 10%. Eighty-nine percent of workers are optimistic that they will receive a pay increase of some sort in the twelve months following the survey.

In 2005 only 25% of you experienced layoffs in your departments; that's down from 32% in 2004. However, more people reported an increased uncertainty that layoffs will occur in the months following the survey.

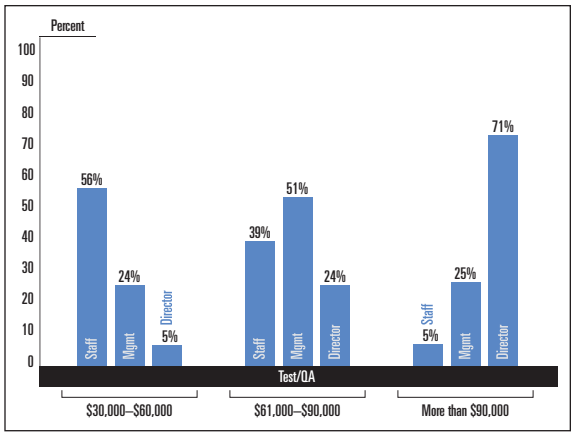
Fitting In

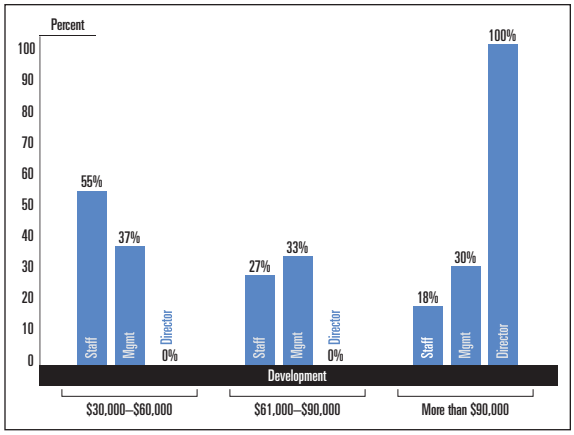
The bulk of survey respondents (21%) work in the commercial software arena, but many work in such varied areas as manufacturing, banking, health care, and utilities. Almost 50% of you work for companies with more than 1,000 employees.

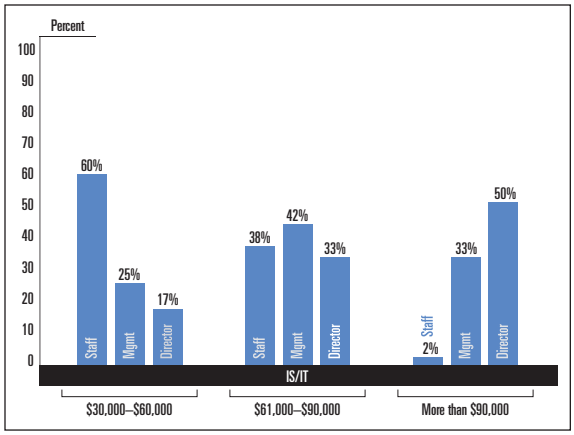
Tester-developer relationships seem to vary quite a bit throughout the industry with the exception of how testing and development functions are arranged. Seventy-three percent of you work for a company where testing and development are parallel, while 20% of responders report that testers report to development in their organizations. Forty percent report a tester-to-developer ratio of 1:2 to 1:4, and many of you work for companies with more than one hundred developers and/or fewer than ten testers.

Adding It Up

As always, it's hard to predict what all of this means. The industry appears stable; there were no wild fluctuations between the 2004 and 2005 surveys. But compared to the industry's heyday, salaries are still lagging and layoffs are more prevalent than we'd like. Maybe this stability is an indication of good things to come. Check back next year, when we'll revisit employment trends and share our readers' insight on the state of the technology job market. **{end}**







The industry appears stable; there were no wild fluctuations between the 2004 and 2005 surveys.

A Look at Administrator's Pak by Winternals

by Marnie Hutcheson

It's time that application testers got some good tools! I know, I know. You've got the "big automated Kahuna" on your shelf. I'm here to tell you about some less grand—but seriously useful—tools that you may want in your pocket.

Administrator's Pak by Winternals is a suite of ten utilities that allows you to repair unbootable or locked-out systems, restore lost data, and remove malware from infected systems while the system is safely offline. More important to software testers, though, Administrator's Pak will help debug and repair all sorts of less serious problems like configuration, application, and driver issues. You don't have to be a super tech to use these tools, but they definitely identify stuff that you can't find with the naked eye—and that makes you look really good.

Figure 1: Administrator's Pak Navigator

Some of the Administrator's Pak tools work only locally and others work both locally and remotely. For example, you can run Administrator's Pak on a healthy machine where it is installed, and you will see the Navigator shown in Figure 1. From here you can work on the local machine to restore files with FileRestore; monitor and modify Active Directory traffic, objects, and properties using Insight for AD and AD Explorer; and use TCPView Pro to view what applications are connected to which ports. All of these tools can be helpful in identifying the causes of application and service failures resulting from AD configuration, corruption, executable errors, and communication issues.

You can monitor and restore files and registry activity on both the local machine and on a remote machine. This is very useful in diagnosing compatibility issues, such as why a certain application isn't running on a certain system. And you can run the Crash Analyzer Wizard on either the local or remote machine.

The Crash Analyzer tool is unique. It uses Microsoft debugging tools and Microsoft's own symbol files to analyze both events and executables. Instead of spending endless hours debugging the problem yourself, the Crash Analyzer Wizard automatically debugs your system using the latest dump and the system's own environment; then it tells you "what" is probably causing your problem. The whole process takes only a few minutes.

I ran the Crash Analyzer Wizard on three different machines that had all experienced failures in the past twelve months. Crash Analyzer told us that one had a faulty driver, one had a bad hot patch, and one had been hacked. Crash Analyzer correctly identified the source of each crash. The first took six minutes to identify, and the others took only seconds. In the machine that had been hacked, Crash Analyzer caught a bad system file that we had missed. The Crash Analyzer Wizard more than paid for the Administrator's Pak by saving us time we would have spent manually debugging these crashes.

When our test PC took a dump last week, everybody wanted to be the one with the privilege of running Crash Analyzer. So all the testers gathered around the machine while we booted it up and ran Crash Analyzer. Crash Analyzer took only a few minutes to find the problem. Sure enough, it was a new .dll in the application we were testing that probably caused the problem. We attached the dump and the Crash Analyzer report to the bug log. The developers were very pleased. Instead of spending hours trying to recreate the problem, they just fixed it.

Restore a Locked Out, Damaged, Dead, or Dangerous System

When it comes to restoring a system, you have two main options. The first is to restore the system locally by booting the machine into the ERD Commander 2005 environment so you can work on it directly. The second option is to boot the

Figure 2: ERD Commander 2005

machine as a remote recover client and use the Remote Recover feature on the host (online) to recover it.

In either case, the CD Build Wizard will help ensure that you have everything you need on your Administrator's Pak recovery boot disk, including ERD Commander 2005 and any PC specific drivers you may need (i.e., SCSI, network adapter, etc.).

Since the Crash Analyzer Wizard uses the Microsoft debugging tools, you will want to have the latest version of them on the CD. The CD Build Wizard gives you a link to their Microsoft Download home and prompts you for their location on your system. You can also include any other programs you wish. Once you have selected everything you want to include on the CD, the wizard creates the image. If your CD burner is compatible, it will even burn the CD for you on the spot.

Local Recover

If you are performing a local recovery, you boot up to ERD Commander 2005 which looks a lot like the Windows desktop (see Figure 2). In fact, all the tools in the Administrator's Pak look and feel so much like Windows that it's easy to forget that they are not.

The Start menu is stocked with the tools that you need to figure out what happened and fix it while the crashing system is inert. The Administrative tools let you view the autorun list, Disk Management, Event logs, RegEdit, Service and Driver Manager, and system information including hot fix history. Networking tools give you file sharing, map network drive, and TCP/IP configuration information for the remote recover machine.

System Tools gives you the real heavy hitters: Crash Analyzer, Disk Commander, Disk Wipe, File Restore, Hotfix Uninstall, Locksmith, System Compare, System File Repair, and System Restore.

You get all the essential utilities: Explorer, a browser, Notepad, search, and a command line console. There is even a solution wizard that guides you through the most common types of recovery: system won't boot, data needs to be salvaged, lost password, and other problems.

Remote Recover

If you plan to use the Remote Recover capability, you have two options. You can either use a Remote Recover Bootable CD that you create using the CD Create Wizard or, if your system supports it, you can use a Pre-Boot Execution Environment (PXE) that allows a client to boot across the network from a host system. Either way, the machine does not need any installed operating system for the recovery to proceed.

Once the damaged system is booted using the Remote Recover CD or PXE, it will be a remote recover client waiting to connect with the host. The Remote Recover host broadcasts a query across the network or to a specific IP address. When the client responds, it appears in the Remote Recover host display. You simply connect to the client, and a list of its disks is displayed in the host console. Select the disks you want to work with and interact with them just as though they are on your local host machine. You can drag and drop files on and off the system, use the virus software on your local host machine to remove viruses, diagnose system and network issues, and use file restore.

Some Things to Know

Since your CD effectively becomes the C drive, you can't eject the Boot CD while the machine is running. You have to remove it either by pressing the eject button just as the system is starting up or by using a paper clip while the system is turned off—it's a small price to pay.

The Administrator's Pak is too large for a download except in extreme cases, so plan on buying the CD. The emergency version of ERD Commander 2005 is available online for immediate download, but it does not include the Crash Analyzer Wizard or Disk Commander. **{end}**

Price: Administrator's Pak: $1199 for a single user license plus $240 annual product assurance (free 30-day evaluation CD available)
ERD Commander 2005 Emergency Download: $299 for Servers and $149 for Workstations (http://www.winternals.com)

Marnie Hutcheson has been building Web-based business solutions by breaking software since 1987 when she helped launch online shopping at Prodigy Services Company. A specialist in user interface development and human factors, Marnie writes and speaks on a variety of technical topics. She can be reached at Marnie@ideva.com.

Borland Announces Enhanced Requirements Management System

CUPERTINO, CA—Borland Software Corporation announces Release 2.0 of its requirements management system, CaliberRM 2005. The updated release is designed to increase communication, productivity, and usability when managing requirements across the application lifecycle.

New capabilities include improved linking and traceability features to more extensively connect requirements with the rest of the application lifecycle. It also includes added secure sockets layer (SSL) support for enhanced security, as well as new enterprise-class licensing options to help improve internal tracking, reporting, and auditing across teams.

"Change is an inevitable fact that affects all phases of software delivery," said Marc Brown, director of product solutions at Borland. "A comprehensive requirements management system can help IT organizations assess the impact of change, communicate that impact across teams, and respond more efficiently to those changes."

For more detailed information, visit www.borland.com.

Quest Extends Toad Database Development Environment

DALLAS, TX—Quest Software announces version 1.0 of Toad for SQL Server, an easy-to-use development environment that increases the productivity of database developers and administrators.

"The complexity of enterprise-class databases, including SQL Server, requires advanced tools so that developers can quickly build optimized SQL code and DBAs can manage tasks across multiple servers with greater efficiency," said Carl Olofson, research director for marketing intelligence provider IDC. "Tools with a familiar interface can also be very useful for experienced DBAs who are new to SQL Server."

With Toad for SQL Server, database professionals can quickly create and execute queries, automate database object management, and develop highly optimized SQL code more efficiently. Administration capabilities allow users to manage large projects, integrate with version control software, manage users and database security, and import/export data.

To learn more, visit www.quest.com/sqlserver.

Solstice Software Expands Support for SOA and Integration Projects

WILMINGTON, DE—Solstice Software announces the availability of Integra Enterprise 5.0, the company's enterprise-class integration testing suite. Version 5.0 provides process-level visibility and validation, enhanced security testing, and expanded support for industry-leading enterprise protocols.

"We've automated the dirty work, so our clients can save time, reduce resources, and improve quality," said Lori Gipp, vice president of marketing at Solstice. "Integra Enterprise tells testers the path through the system, component by component, then helps testers pinpoint precisely where the problem is in the process and which data, field by field, is creating the problem—in minutes, not days."

Integration introduces layers of complexity in order to improve business processes. Integra Enterprise works through those layers in two simple steps: laying out message paths and correlating messages along workflows; and recording both the message wrapper and message content to diagnose the source of the problem in the flow.

By penetrating the integration layers, Solstice streamlines diagnosis and gives testers control and a repeatable integration validation environment. A central repository, user security, and an XP-like interface promote usability across the team and re-usability of assets.

To find out more about Integra Enterprise, visit www.solsticesoftware.com.

CollabNet Launches Enterprise Edition 4.0

BRISBANE, CA—CollabNet launches the next generation of the company's flagship software development platform, CollabNet Enterprise Edition 4.0. The new version features the CollabNet Application Lifecycle Manager (ALM), a codified yet flexible set of process templates containing content, artifacts, and tools that guide project members through the application lifecycle to help organizations overcome their distributed software development challenges.

The ALM enables project managers to select and customize a choice of pre-configured process templates and address a broad range of software

(Continued on page 42)

(Continued from page 41)

development styles and business processes. The Baseline Process Template can be utilized on a project-by-project basis or enforced company-wide to drive adherence to specific processes. Process templates enable CIOs and their software development teams to customize and define consistent processes for particular projects by allowing organizations to codify all lifecycle phases. The ALM also provides CIOs and their project managers global visibility to measure and subsequently improve their distributed software development processes.

For more information, visit us at www.collab.net.

McCabe Releases McCabe CM 3.1 with StreamCM

WARWICK, RI—McCabe & Associates announces the release of McCabe CM 3.1 with StreamCM. Its selective migration techniques enable the removal of specific changes or the promotion of a change from among those waiting, allowing the change to be fast-tracked through test to release.

Version 3.1 includes StreamCM, a Java client developed for ease of use. Its powerful visualization of the change-based application lifecycle allows development teams to drill down to the lowest level of the application history, so that information about change history is never lost.

"As with previous versions of McCabe CM, users will realize a significantly lower administrator-to-developer ratio than they would with other CM solutions, translating into a much lower cost of ownership and flexible, accelerated release schedules," said Barbara Dumas, McCabe director of SCM solutions.

Learn more at www.mccabe.com.

TechExcel Delivers Solution for Managing the Application Development Lifecycle

LAFAYETTE, CA—TechExcel, a leading developer of issue and process management solutions, announces the general availability of DevTrack 6.0. The latest version of DevTrack enhances the defect- and project-tracking tool's power to manage and automate the application development lifecycle through extended workflow functionality, robust scalability, and seamless integration with IT service and test management solutions.

"DevTrack has been an effective and powerful product for many years," said Tieren Zhou, president and CEO of TechExcel. "Some of our largest DevTrack customers are creating over 15,000 issues per week, with thousands of team members working twenty-four hours a day on our system. Providing these customers with fast performance and stability is the focus of the DevTrack 6.0 release. We have designed DevTrack 6.0 specifically to meet such scalability needs, while retaining the ease of configuration and ease of use that DevTrack has always been known for."

The DevTrack 6.0 graphical workflow allows users to draw shapes and connectors that represent states and transitions. Using the diagramming tools, users can define every aspect of a workflow rule—next actions for each status state, application owners for each status state, who can perform actions, and invisible/read-only/mandatory fields for each workflow action and status state—and immediately publish the new workflow settings to a live system. DevTrack 6.0 also offers users the ability to create multiple workflows to cover the full spectrum of development work items, defects, and change requests.

DevTrack 6.0 also introduces two major performance enhancements designed specifically for large development teams. For teams with thousands of users and millions of DevTrack issues, two operations could slow down performance: generating large reports and performing extensive keyword searching against multiple memo fields. DevTrack now supports Web server load balancing and dedicated reporting servers, which allow DevTrack customers to dedicate a single Web server or a group of servers for reporting purposes and thus remove possible performance impact to normal operations when generating large reports. DevTrack 6.0 also introduces the DevTrack Search Engine to deliver instantaneous keyword search results, even when DevTrack databases contain millions of issues.

For more complete information, visit www.techexcel.com.

(Continued from page 44)

crossed several departments—HR, IT, security—each of which serves many customers and evens out the ebb and flow of work during the day by holding work in queues. So, perhaps an hour was spent physically setting up the user ID, and the rest of the eighty hours was spent in queuing as the request moved between departments.

What can we do about this? The first step is to think about the dull knives in your organization. Think about how things flow—you may not even realize you have a problem. Ask staff members what is blocking the flow, ask them how to fix it, and then give people permission to fix it. Wherever you sit in the organization chart, have lunch with your internal and external customers and your suppliers. Ask them where it hurts and experiment with small changes to make things better. Invent a new role—the "flow master"—someone whose job it is to coach people on how to see the flow and the blockages. And don't forget to sharpen your tools. Often. **{end}**

Clarke Ching is a New Zealander who now lives in Scotland. He is a passionate advocate of Agile software development and is chairman of the special interest group Agile Scotland, which meets monthly in Edinburgh. Clarke has an MBA specializing in technology management and is a senior consultant with Vision Consulting.

Have the Last Word!

If you have a point to make or a side to take on issues and trends that affect the industry, we want to hear from you.

We are looking for insightful, thought-provoking commentary for possible use as a Last Word column.

Please send your submission to editors@bettersoftware.com. You will be notified if you are being considered for publication.

MARK YOUR CALENDAR

February 13-16, 2006
The 5th IEEE International Conference on COTS-Based Software Systems
Hilton in Walt Disney World Resort
Orlando, Florida
http://www.iccbss.org/2006

February 13-17, 2006
RSA Conference 2006
McEnery Convention Center
San Jose, California
http://2006.rsaconference.com

February 14-16, 2006
IASTED International Conference on Software Engineering (SE 2006)
Congress Innsbruck
Innsbruck, Austria
http://www.iasted.org/conferences/2006/Innsbruck/se.htm

Index to Advertisers

Display Advertising
Shae Young syoung@sqe.com

All Other Inquiries
info@bettersoftware.com

Better Software (USPS: 019-578, ISSN: 1532-3579) is published eleven times per year. Subscription rate is US $75 per year. A US $35 shipping charge is incurred for all non-US addresses. Payments to Software Quality Engineering must be made in US funds drawn from a US bank. For more information, contact info@bettersoftware.com or call (800) 450-7854. Back issues may be purchased for $15 per issue (plus shipping). Volume discounts available.

Periodicals Postage paid in Orange Park, FL, and other mailing offices. POSTMASTER: Send address changes to *Better Software*, 330 Corporate Way, Suite 300, Orange Park, FL 32073, info@bettersoftware.com.

Working with Dull Knives

by Clarke Ching

A few years ago I attended a sushi preparation demonstration given by a local Japanese chef who was promoting his new shop and restaurant. Sushi was uncommon where I lived, and the room was full of gastronomes eager to try the new delicacies. The chef swiftly and seemingly effortlessly prepared dish after stunning dish, deftly slicing fish and vegetables, then wrapping them with rice and seaweed or serving the fish on its own.

When he had finished, he asked if there were any questions. When was he opening his restaurant? Next month. Where do we buy the ingredients he'd been using? From his little shop, of course. Where did he buy his fish? He went to the local fish markets, but that meant getting out of bed at 5 a.m., so he recommended a good local fishmonger. I asked him how did he keep his knife so obviously sharp? He frowned, glanced down at his knife for a moment, then up again, before looking me directly in the eye. His face suggested that this was the stupidest question he had ever heard. "I sharpen it," he said. "Often."

Clarke Ching says you should sharpen your tools often.

I was reminded of his answer a few weeks ago when I started a new consulting gig. It took just over a week for my new user ID to be set up and another week to get it set up correctly, despite its taking, presumably, less than an hour's cumulative effort. I made do with my laptop, but I had no access to email or to the organization's software applications and data. I've asked around, and this is a common occurrence. In fact, during my career I've had only one job where I had the correct access set up to start working productively on my first day—but, sadly, they had no work for me to do for the first six weeks.

Contrast my experiences with those of the sushi chef. He not only brings his skill and experience to the job, but he also brings his own knives. His productivity, his safety, and his sense of professionalism depend on the quality of his tools, so he keeps them sharp and he doesn't allow anyone near them. Like the chef, our productivity and sense of professionalism are dependent on the skill and experience we bring to the job and the tools we use. But unlike the chef, we rarely have control over the quality of our tools.

For some reason, our employers and clients are content to pay us our high salaries but they give us dull knives with which to work. I have fond memories of my first job—we read the newspaper and did the crossword puzzle during the half-hour waits for mainframe compiles to finish. At another job, I was told off for sticking our data model on the wall for all to see and discuss—because it was against company policy. The organization seemed to value clear walls more than it valued productive workers.

On yet another job, the facilities department cleverly squeezed 30 percent more staff into the same office space by using flat screen monitors and smaller desks. Unfortunately, facilities didn't increase the number of meeting rooms—a basic tool for most knowledge workers. Instead of using the meeting rooms spontaneously as needed, we now had to book them two or three weeks in advance. We ended up having noisy meetings around our desks, which distracted others from their work. Other times we sneaked out to meet in local coffee shops—where we effectively paid for the company's meeting space out of our own pockets through the inflated prices of the coffee we felt obliged to drink.

When I was younger, I blamed these problems on stupid middle managers. I was wrong. The problem is that no one manages the flow of work through the organization at an operational level. True, senior management is trying to manage the flow from a high level, by setting up a hierarchy of specialized departments—the organization chart. Unfortunately, the work, money, and commitments flow horizontally—not hierarchically—across the organization. Each department tends to optimize its own performance against its own targets and within its own budget, and it's often done at the expense of other departments.

For example, my user ID request

(Continued on page 42)

You give up things when you buy TestComplete 4. Things like high price, steep learning curve and proprietary technology.


Automated application testing is a crucial piece of the software delivery puzzle. With the right automated testing system, you're far more likely to deliver solutions that behave as expected and consistently address the needs of your customers. The question is not whether to use test automation in your development process, but whether the solutions offered by the "market leaders" offer compelling value.

Overpriced?

Are you shocked by the price of most automated testing tools? Has your company spent tons of money on "market leading" products? Are you disappointed that the "market leaders" deliver so little when their price is so high?

If so, consider TestComplete. TestComplete is recognized as the most capable product of its kind. It's won a Jolt award two years in a row, yet it's a fraction of the cost of the competition.

So if we're able to offer TestComplete at such a reasonable price, why do our competitors charge an arm and a leg? That's a good question, we don't know why our competitors charge so much, but like our Fortune 500 customers have discovered, superior capabilities don't require massive licensing fees.

Locked-In?

At AutomatedQA we take standards seriously. We take them so seriously that we took the time to engineer our products with features like open file formats, standard scripting languages, full access to system functions and an open plug-in API.

- ✓ Test Windows, .NET, Java and Web apps
- ✓ Functional, regression, unit, distributed, HTTP performance testing and more
- ✓ Easy point-and-click test recording
- ✓ Advanced scripting in VBScript, JavaScript, C++ Script, DelphiScript and C#Script
- ✓ Data-driven tests and full DB access
- ✓ Part of the AutomatedQA family of tools

- ✓ **New:** Improved user interface
- ✓ **New:** Issue tracking integration
- ✓ **New:** Source control integration
- ✓ **New:** Manual test manager
- ✓ **New:** Visually create web load tests
- ✓ **New:** Unicode support
- ✓ **New:** Built-in screen capture OCR
- ✓ **New:** Visual Studio 2005 integration

Why should you put up with testing tools that lock you into their proprietary file formats or arcane scripting languages? Open standards are the future and TestComplete is based on open standards. Don't waste time on outdated tools.

Hard To Use?

When you buy TestComplete, you get access to all of our help and support. It will save you time and make learning TestComplete as easy as possible. More than a thousand pages of online help and manuals, extensive sets of sample applications and clear tutorials will guide you step-by-step through the learning process. And we pride ourselves on providing the most comprehensive technical support – unlimited free email support and free community support forums.

Hard To Buy?

We work hard to make sure that TestComplete is easy to get to know. That's why you can download a free evaluation copy of TestComplete and try it out at your own pace.

You don't have to give up your personal information just to download a demo. You don't have to worry about the dreaded sales call to "close" you. And you don't have to beg for a confusing price quote. TestComplete's prices are on our website. They're clear, easy to understand and the same for everyone. There are no hidden costs. There's no haggling and deception. We treat our customers with respect.

Visit automatedqa.com and download TestComplete today.

TestComplete™
automate your tests

TestComplete Standard	$499.99
TestComplete Enterprise	$899.99

Named user license / Concurrent licenses available

60 Day Money Back Guarantee
Unlimited Online Support
Free Evaluation

AutomatedQA
test, debug, deliver!
702·891·9424
www.automatedqa.com

SORRY.

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

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

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

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

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

segue®