Software Test & Quality Assurance

STP SOFTWARE TESTPROFESSIONALS

ST&QA MAGAZINE

VOLUME 9
ISSUE 2
2012

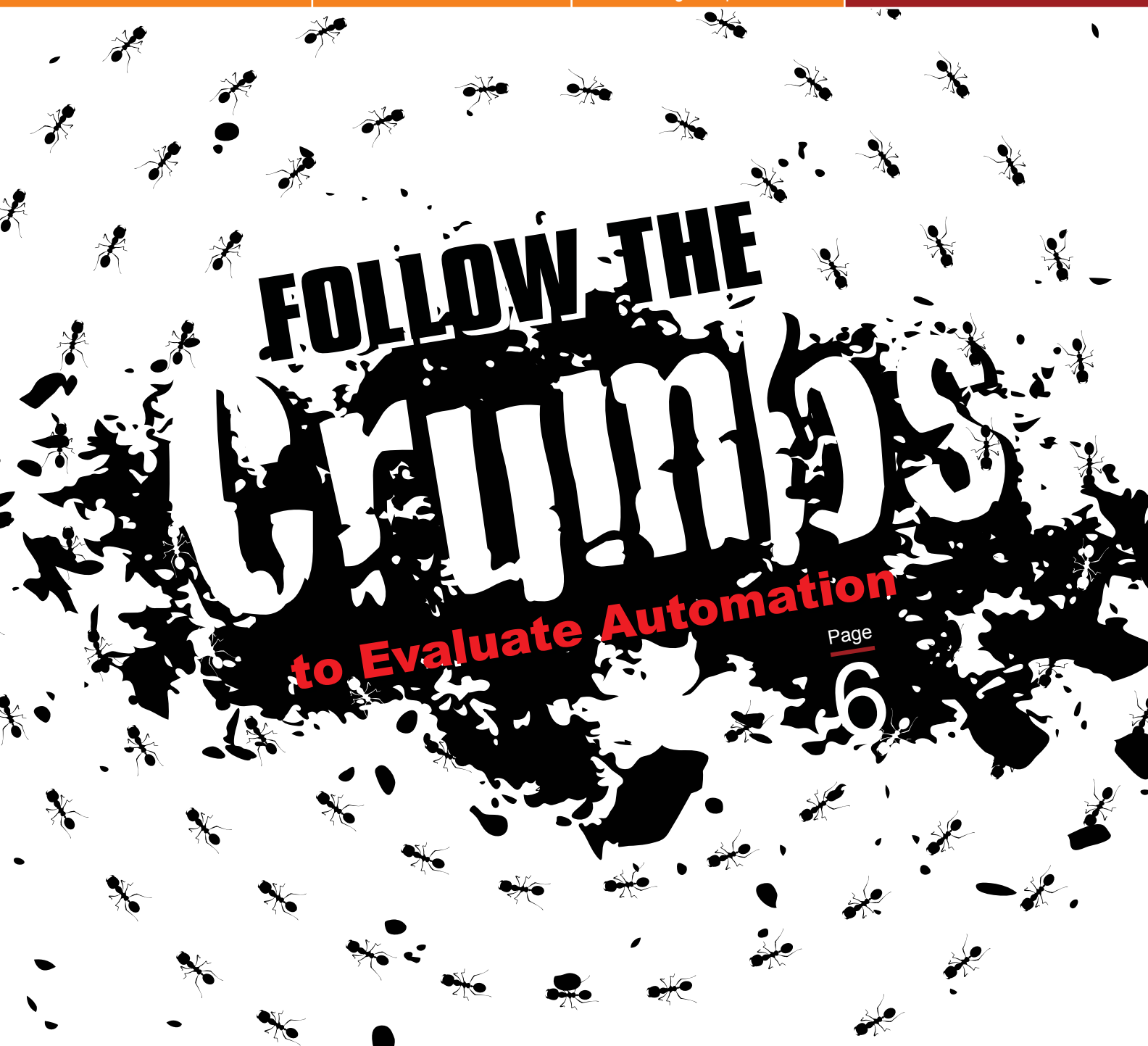
www.softwaretestpro.com

Database Virtualization – For Development and Test 14

Never Be Satisfied – Take time to identify where you can improve 16

Hiring Testers From Outside IT – Look for something unexpected 38

ASK THE TESTER Interview with Mike Lyles



FOLLOW THE Crumbs

to Evaluate Automation

Page 6

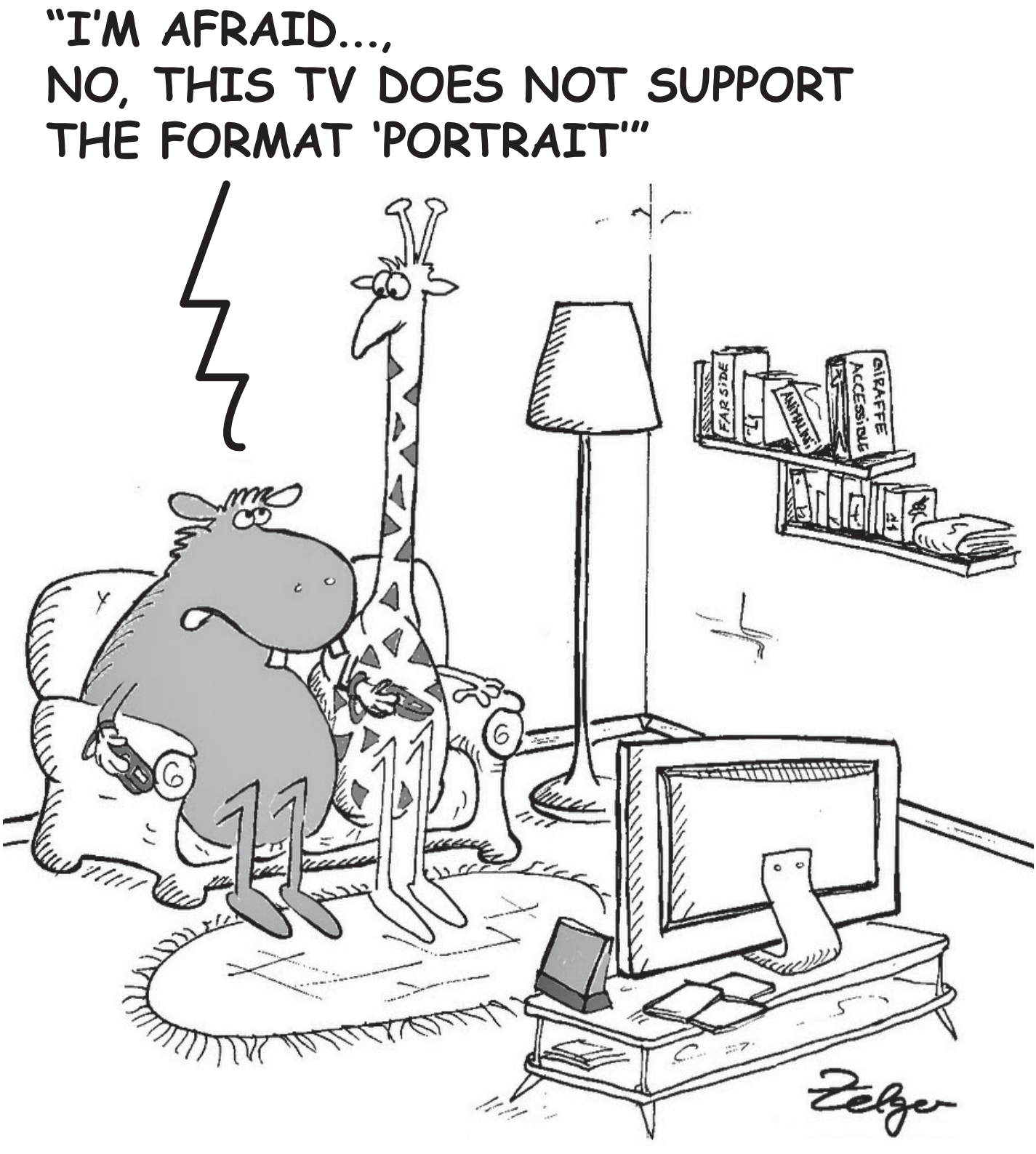
The world's best applications pass the test of real-world success.

End-to-end application testing. In the wild. On demand. Today's best applications are put through their paces with real-world testing by uTest.

uTest®

Where the best test

In-The-Wild Testing for Functional + Security + Load + Localization + Usability



Another purchase that just wasn't giraffe accessible.

CONTENTS

Volume 9 / Issue 2

06 Follow the CRUMBS to Evaluate Automation: A Follow Up Dialogue Between Michael Larsen and Albert Gareev
Michael Larsen and Albert Gareev

11 A Tale of Two Testers: Far, Far Better Stress Tests Than You Have Ever Known
David Borcherding

14 Database Virtualization for Development and Test
Wayne Ariola

16 Never Be Satisfied
Matt Angerer

19 Intentionally Avoiding Unintended Side Effects Using Observation Driven Testing with Test Driven Development
Dale Brenneman

25 Ask the Tester
Matt Heusser, An in-depth interview with *Mike Lyles*

32 Think You're Done?
Jim Hazen

38 Cast Your Eyes Afield: Hiring Testers From Outside The IT Industry
Brian J. Noggle

Editor-in Chief: Rich Hand
rhand@softwaretestpro.com

Creative Director: Scott Hanson
shanson@softwaretestpro.com

How to reach ST&QA
Letters to the editor may be sent to:
editor@softwaretestpro.com

About Software Test Professionals (STP)
STP provides more than 50,000 software professionals with information, education, training and professional networking opportunities.

724 S Tejon Suite C • Colorado Springs Colorado, 80903 • 719.476.0301

About ST&QA
ST&QA is published bi-monthly by STP (Jan, Mar, May, July, Sept, Nov). It is a digital magazine geared towards the software test and quality assurance professional. Subscription to ST&QA is an STP Pro Membership benefit. If you would like to subscribe to ST&QA, visit softwaretestpro.com and join the Pro Membership.

Back Issues of ST&P and ST&QA
Pro Members may access electronic back issues at www.stqamag.com.

Editorial Submissions
Visit softwaretestpro.com to learn more about submitting editorial to ST&QA.

ST&QA Magazine (ISSN #1548-3460) (USPS #23771) is published bi-monthly by Redwood Collaborative Media (724 S. Tejon Street, Suite C, Colorado Springs, CO 80903). Periodicals postage paid at Colorado Springs and additional mailing offices. The price of a one year subscription is US $69 for postal addresses within North America; $119 elsewhere. ST&QA Magazine is a registered trademark of Redwood Collaborative Media. POSTMASTER: Send changes of address to Software Test Professionals, 724 S, Tejon Street, Suite C, Colorado Springs, CO 80903. To contact ST&QA Magazine subscriber services, please send an e-mail to info@softwaretestpro.com or call 719.476.0301.

Contributors


MattANGERER
Retail Software QA Leader – METTLER TOLEDO


WayneARIOLA
VP of Strategy & Corporate Development – Parasoft


DavidBORCHERDING
Marketing Communications Specialist – Seapine Software


DaleBRENNEMAN
Vice President of Technology – Agitar Technologies, Inc.


AlbertGAREEV
Dedicated Testing and Test Automation Professional


JimHAZEN
Consultant – Connected Testing


MichaelLARSEN
Lone Tester – SideReel.com


Mike LYLES
QA Program Manager – Lowe's Companies Inc.


Brian J. NOGGLE
Freelance Software Testing Consultant – Jeracor Group LLC.

Preparing for the conference in New Orleans...

As we prepare for the conference in New Orleans, it is exciting to welcome some new sponsors to our event. It is also very exciting to have our veteran sponsors coming back which to us is a sign that we are providing an event that fits into their objectives of sharing valuable information with a valuable audience. You will find a number of those sponsors throughout the pages of this edition. I hope you will support them as they help us to deliver a quality conference experience.

At the end of 2011 I sent out a request for content submissions to the community and received hundreds of responses! These individuals have made it clear they want to share their opinions and expertise with you. It is a testament to the quality of individual in our community.

Some of the quality individuals featured in this edition are **Michael Larson** and **Albert Gareev** with "*Using CRUMBS to Evaluate Automation*"; **Dale Brenneman** authored "*Intentionally Avoiding Unintended Side Effects Using Observation Driven Testing with Test Driven Development*"; **Brian Noggle** discusses the pros of hiring Testers from outside IT in "*Cast Your Eyes Afield*"; **David Borcherding** tells us "*A Tale of Two Testers*"; **Jim Hazen** lays out the things Tester's should consider when utilizing automation in "*Think You're Done?*"; **Wayne Ariola** shares some great advice in "*Database Virtualization*"; **Matt Angerer** tells us why we should "*Never Be Satisfied*"; and last but not least **Matt Heusser** interviews **Mike Lyles**, QA Program Manager for Lowe's Companies Inc. in "*Ask The Tester.*" Thanks go out to every one of these industry professionals.

RichHAND

"There is no secret to our success. Our success is helping you to succeed." RichHAND

Content is King in any industry, and we have been doing our best to bring it to you not just in our publications but at our conferences and in our Online Summit education series. You have really embraced the Online Summit series where we have taken on some timely and relevant topics like Automation Testing, Performance Testing, and Agile Transistions to name a few. Again, we are using highly regarded members of the STP community to deliver these online educational presentations to you. These presenters are some of the most respected in the industry and we thank them for trusting us to deliver their content to the testing community.

Software Test Professionals is a launching ground and delivery organization for the best talent in the testing community. There is no secret to our success. Our success is helping you to succeed.

The Software Testing community gets more important with every new technological advance in the market place. Together we can continue to share the latest and greatest with each other. This edition of ST&QA is an example of that commitment. I hope you agree!

I also hope to see you in New Orleans!

RichHAND
Director of Membership & Publications

FOLLOW THE CRUMBS

to Evaluate Automation

A Follow Up Dialogue Between Michael Larsen and Albert Gareev

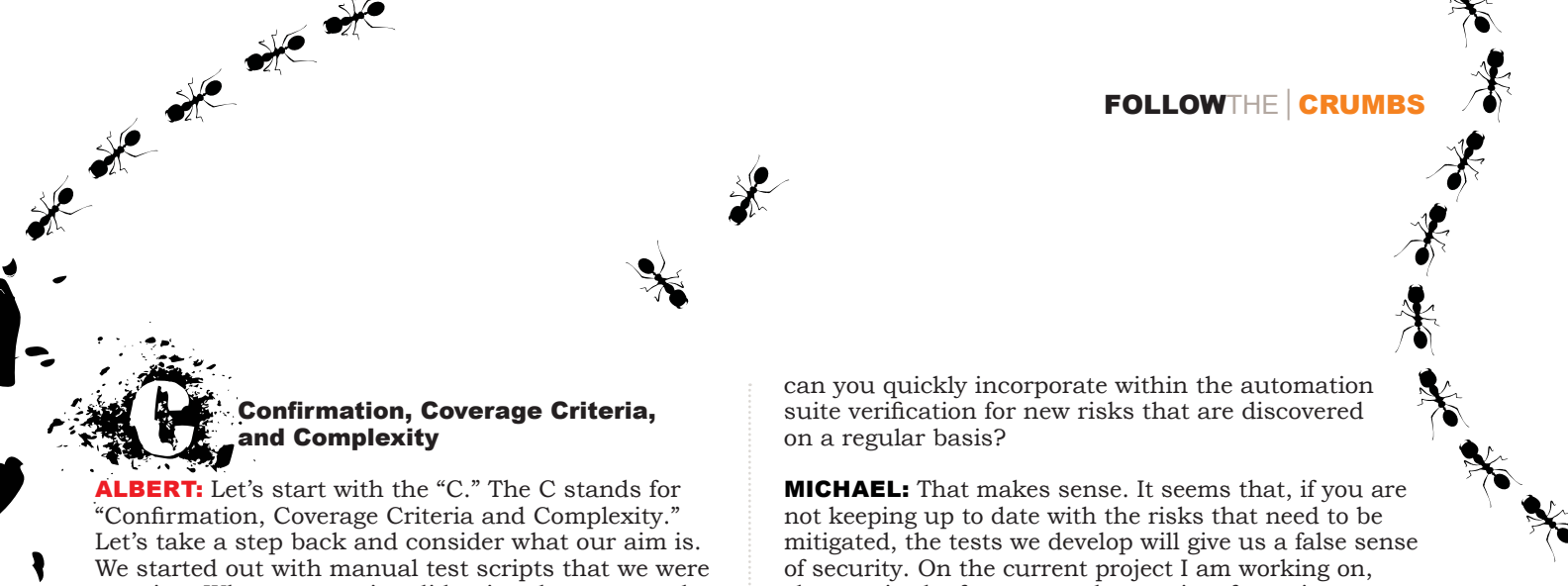
When we were last together, we took on the topic of deciding if Automation was something that we wanted to pursue, and what we might want to consider before we made that death-defying leap. We decided to come to TERMS with it all [see ST&QA Issue January 2012 (Vol. 8 No. 6) for previous discussion], and yes, we have committed to taking on an Automation project for our respective hypothetical company.

So let's rejoin the dialogue. It is a few months later. We've been actively plugging away. Screens fly by. APIs get exercised. Logs get made, parsed, and analyzed. Reports are sent to the important parties. Everything's great... well, at least we think it is. In the back of our minds though there's that nagging worry. Are we addressing what we set out to do? We had the best of intentions going in, and we certainly used the TERMS heuristic to decide if it was worth doing, but what now? How do we know that we are applying the best efforts in the most important areas? Am I actively pursuing a winning strategy, or have I succumbed to the latest in "automation snake oil"(1)? To help shine a light on this, I decided to see if Albert would be interested in helping me examine this once again.

MICHAEL: So Albert, here we are a few months later, and we've spent several weeks, a fair chunk of money, and a lot of sweat equity to try to get into an automation groove. Some things feel really solid, but others feel, well, less focused. I know that the last time we talked we discussed a mnemonic called TERMS that helped us know if we were ready to actually get started with an automation project. How do we gauge our progress after we are underway?

ALBERT: Well, a lot of this comes down to our expectations and what we hope to have our automation efforts do. TERMS gives us a good idea as to what we need to do to get the process started and make sure we have identified a good candidate for automation. Is that everything we need to know? Of course not, it's just a list of characteristics to help us. However, there are a number of questions and thoughts we should be applying after we have gotten our automation project underway. As you might expect, I have an acronym for it as well. I call it "following the CRUMBS," where CRUMBS makes up the attributes and questions we want to ask. Another purpose for the acronym is to remind us of the secondary role of automation in testing. If you rely on automation and abandon testing – "crumbs" is all you will get.

MICHAEL: All right, once again, I'm intrigued. What does CRUMBS stand for?



C – Confirmation, Coverage Criteria, and Complexity

ALBERT: Let's start with the "C." The C stands for "Confirmation, Coverage Criteria and Complexity." Let's take a step back and consider what our aim is. We started out with manual test scripts that we were running. What expectation did using those manual test scripts intend to confirm? Were our automated scripts able to confirm them? If the answer is Yes, then we are already off to a good start. If not, we might want to step back and evaluate what we are doing.

MICHAEL: You mentioned Coverage in that list. I'm assuming that the goal of having coverage as one of the criteria is to have the machines run a larger number of scripts in a faster manner than we could ever run manually. Likewise, I would assume that our goal is to make sure we are gaining coverage with our automated test scripts.

ALBERT: Well, that's part of it. We have to take into account that human beings, even with manual test scripts will vary their approach from time to time. They might get bored, or distracted, or some other factor comes into play, but humans are not good at doing the exact same thing over and over. Thus, the odds of getting interesting behavior from a human tester will be higher than automated test steps, which really do copy the exact same steps over and over as many times as you tell them to. With human testers, we may learn new details because of these variations. With automated scripts, they are not likely to tell us anything new.

MICHAEL: It also seems to me that we have to take into account whether or not automation simplifies our testing process, or if it adds complexity to our testing. We commonly consider having automation to do the "tedious" part of testing and leave us open to focus on the more "interesting" aspects. That's the idea in any event, but it begs the question; is our automation making our testing process simpler or more complex?

R – Risk, Robustness, and Reliability

ALBERT: The truth is, automation adds complexity to any testing project, whether it is intended or not. Because of this, we turn to the next letter in our acronym, R, which stands for "Risk, Robustness and Reliability." Let's take a look at Risk first. What are the product's risks that automated test execution addressed well? At the same time, what risks were not addressed well? Once you have identified them, can you quickly incorporate within the automation suite verification for new risks that are discovered on a regular basis?

MICHAEL: That makes sense. It seems that, if you are not keeping up to date with the risks that need to be mitigated, the tests we develop will give us a false sense of security. On the current project I am working on, changes in the features and a rewrite of certain aspects of the user interaction means that the flow of the application has changed, yet the suite of scripts that I have still runs without issue. Were I to take that at face value, I might believe that everything was fine, while missing the changes that were recently implemented.

ALBERT: That is a real problem with many test suites. We often make tests simple so that we can focus on key areas, but we also run the risk of having tests that underperform and give us what would be termed "near misses." While we have passing tests, there are areas that have changed that we are not covering. This leads us to the Robust part of the R; How robust are the automated testing scripts? Do the automation tools provide reliable object recognition and interaction? Also, think about what happens when we do have changes and everything stops working. Are you familiar with that?

MICHAEL: Oh, definitely! I have had plenty of experiences where I've had to babysit scripts as they have gone through feature enhancements and story submission, and whereas just a week ago, everything worked, now I have a 20% failure rate. Sometimes it feels like I have to look for workarounds, or I find myself putting together a dedicated and simplified environment so I can take out as many conflicting variables as possible.

ALBERT: Sure, that's a logical thing to do, but ask yourself this: If you have simplified your environment, and taken out a lot of the aspects that are similar to what a user might have on their system, how valid are your tests? If all of your tests are running a simplified, let's say "sanitized" environment, sure, you eliminate a lot of the complexity that might interfere with your test scripts completing successfully. You also reduce your chances of actually catching problems that might impact your customers.

MICHAEL: A very good point. This is often an area I point to and say, "If we have to run on such sanitized and overly clean environments, is it really worth it to run in this state? Wouldn't it be much more valuable to run these tests manually on more realistic customer environments?" I think, if I have the time to do it manually, under these circumstances, I might actually choose to.

Usefulness and Usability

ALBERT: Of course, that's totally natural. This brings us to the "U" part of the acronym, which is "Usefulness and Usability." Automation is not as helpful if it makes us feel that we need to run the tests over again or manually to "make sure." Yes, there will be times when we will need to do that, so we want to make sure to focus our efforts towards areas that are the most useful to our testers.

MICHAEL: OK, so let me shift the conversation a little bit here. So we are able to run a large number of tests. That's great, but we also ultimately have to share the results of this information. That's just as much a part of the automation puzzle as is coding up the steps necessary to automate the process, right?

ALBERT: It certainly is. While the test automation itself is important, and can be a tremendous time saver, if we then have to spend the time we saved in massaging data into reports for our stakeholders, we are missing an important part of the benefit of automation. We always want to consider what is most important to the stakeholders. If the fact that the tests were run and that they passed is enough, that's one thing. If they want to have a specific report showing what was tested and what passed and failed, then that makes for a much greater need for developing a report to show the test results. That has to be worked into the overall automation efforts.

MICHAEL: It seems to me that there are also aspects of how easy it is to run and use the automated scripts. Are we talking about running a simple command and triggering the whole process, or do we need to create very specific suite files with explicit parameters and then run them individually? Also, is it OK to be "close to the metal" with our scripts, or do we want to be able to easily represent to our stakeholders what is being performed with each test run. For example, a report for a tool like xUnit is going to look very different compared to the output from a Cucumber script.

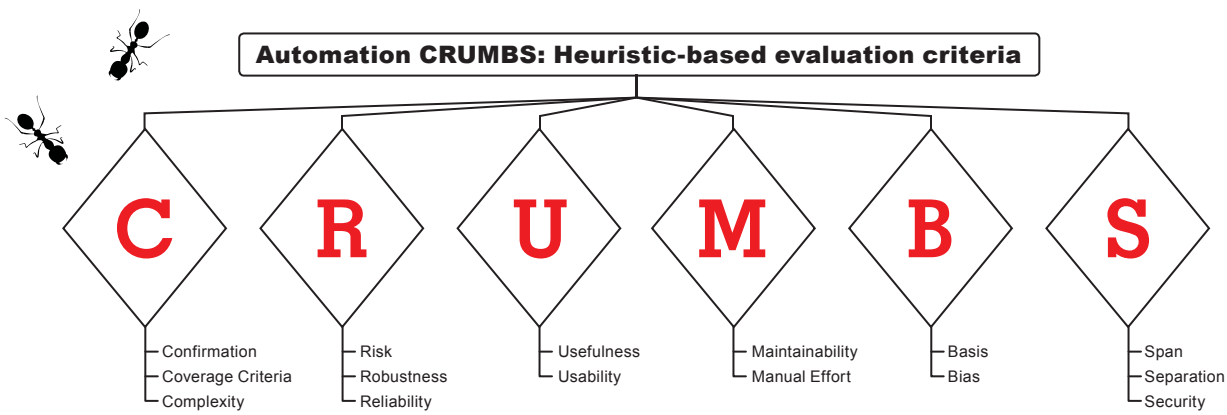
Maintainability and Manual Effort

ALBERT: These are all challenges that every tester who works with automation faces when they are knee deep in a testing project. It seems that much of what we think is good enough at first gets expanded upon and requires ever more effort to keep under control. This is normal and expected, but it still takes a significant amount of time. This brings us to M, which stands for "Maintainability and Manual Effort." Think about each time we need to run a test suite. Do changes in the test setup require code changes? How about changes in the application's GUI? Do we need to modify our tests? If we do, are these changes code change related? Are there data changes involved?

MICHAEL: In my experience, it seems to be that "both" would be the right answer? Changes in page layout or in displayed items on the page are relatively easy. When I have to actually rewrite tests because we have changed business logic or our workflow, that's when things get really interesting. There is often a big push needed to be able to change scripts because the developers have changed behavior of the application in a significant enough manner.

ALBERT: That's not as frequent an issue, but no doubt, it does happen. We also have to think about how soon our automation scripts will need to be run once the build is deployed. This is common in Continuous Integration environments. Your scripts may not be run as often or as regularly, but overall, that should be the goal, to be able to run our scripts as soon as the code is deployed on our respective machines.

MICHAEL: My biggest fear is that I'll be caught in the middle of a test cycle and there's some big change, and in that process, there's a number of modifications that would be necessary for the testing framework to effectively maintain our automation efforts. We'd have to consider what it takes to bring everything back to running as it was prior to the changes. It could make for a considerable manual effort to bring everything



back into line. If we have a dedicated team that handles debugging/fixing/updating automation scripts on a regular basis, I feel a lot more comfortable than if these changes are left entirely up to me and only me.

Basis and Bias

ALBERT: These are all important considerations, and they need to be evaluated from time to time. This brings us to our next part of the Acronym, the B, for "Basis and Bias." When we examine how well an automation project is working, we do so with certain "lenses" that will often color the results that we see. We need to ask who and what are the sources of information for answering questions of these projects. What's more, do we have the input of all the stakeholders involved in reviewing and commenting on this study? Who prepared the analysis? Was it the testers? Was it the stakeholders? Was it an outsourced team? All of these will, again, color our outlook.

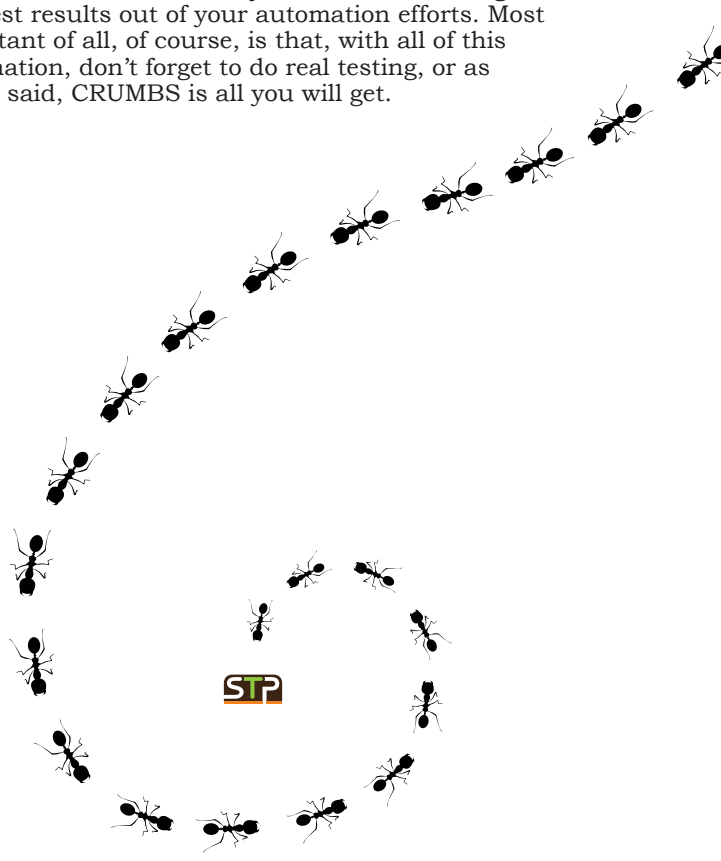
Span, Separation, and Security

MICHAEL: This feels like a lot to have to take in. I get that there are many aspects that we need to focus on, but my concern is for projects like this to "have legs," so to speak. It's one thing to develop an automation scheme, but most of us want to make sure that we are making something that will last for the long haul. What can we do to help us see to it that we have that ability in these projects?

ALBERT: Yes, you're not the only one thinking about this. The S in CRUMBS stands for "Span, Separation and Security." The questions we need to ask fit along these lines. How long do we plan to be using these automation scripts? Will this framework be reused for other projects? If we do use these for other projects, how much of the underlying verification logic will be transferable, or will we need to modify this to work for other projects?

Completely outside of the framework is the test data. It may also have a "shelf life" that we need to be aware of. We also need to consider how well the automation was integrated within our existing testing process. Did automation remove a bottleneck, or did it become a bottleneck unto itself? Finally, we have to consider Security whenever looking at automation. What do we do when we have to ask to have our company's security policy modified to allow us to run these tests? That opens up the organization to new risks, and we have to mitigate those risks.

Taking on an automation project is a lot of work, and there are a lot of areas that we have to consider when we approach these projects. Each of the steps described will give us more information and insights into our efforts and allow us to adjust and change course if necessary. It's our goal to help give you, the tester, the ability to look at your projects and determine which CRUMBS you should follow to get the best results out of your automation efforts. Most important of all, of course, is that, with all of this automation, don't forget to do real testing, or as Albert said, CRUMBS is all you will get.



About The Authors

Michael Larsen *is a lone tester with SideReel.com in San Francisco, CA. He is a brown belt in the Miagi-do School of Software, an instructor with the Association for Software Testing, facilitator for Weekend Testing Americas and the producer of Software Test Professionals "This Week in Software Testing" podcast. He can be found on Twitter at @mkltesthead and blogs at http://mkl-testhead.blogspot.com.*

Albert Gareev *is a dedicated testing and test automation professional, technical lead and a hands-on consultant. He combines passion for testing as investigation and learning with an automation approach helping to reach observation levels beyond "naked eye and hand" in complex, large-scale systems. Albert frequently writes in his blog (http://automation-beyond.com/) and occasionally tweets as @AGareev.*

"With QA Wizard Pro, I don't need virtual machines for stress testing."

Running stress and boundary tests used to be such a pain. I had to set up countless virtual machines to test for every possible strain the app might encounter. As soon as the next version came out, I'd have to set them all up again with new parameters.

With **QA Wizard Pro**, virtual machines are a thing of the past. I can make the app think there's no disk space, or the connectivity has failed, or any other condition I need to simulate. Because there's no virtual environment limiting disk or memory space, I can also use other tools while the test is running.

QA Wizard Pro not only takes the pain out of stress testing, but it also makes it easy for me to test more of an application in less time. What's not to love?

For a free trial of **QA Wizard Pro**, visit **www.seapine.com/stpstress** today.

Seapine Software™ | Be Quality Ready.

A Tale of Two Testers:

Far, Far Better Stress Tests Than You Have Ever Known

by David **BORCHERDING**

When it comes to stress and boundary testing, there are the best of practices and the worst of practices. Take Chuck and Sid, for example. Chuck is a QA tester for Evremonde Softworks. Sid does QA testing for Stryverware, one of Evremonde's competitors.

Recently, both companies received support calls from irate customers who had lost critical data. The Evremonde product crashed when it ran out of disk space while trying to save, while the Stryverware product failed after a user accidentally entered an astronomically large number in a field.

Management demanded improvements to stress and boundary testing for the next release. For Sid, this meant long hours of setting up dozens of virtual machines. Chuck, on the other hand, got to try a new tool that made the task much easier.

The Worst of Times

Whether you're in development or QA, stress and boundary tests probably aren't things you enjoy. Nevertheless, they are critical to the development process, because stress and boundary errors often occur at the worst possible times—when the application is trying to save data.

Contrary to popular belief, stress and boundary tests aren't just for web applications. The evolution of computing from desktop machines to ultrabooks, tablets, and smart phones makes stress and boundary testing even more important, because developers often program with a workstation mentality, thinking the end user will use hardware similar to their own. Unless smaller, less powerful computers are specifically documented in the requirements, these devices may not be on development's radar.

Often, then, stress and boundary tests fill the gap between what's in the requirements documentation and how the application needs to perform under real-life stresses. But setting up all of those virtual environments to test every fault condition and boundary error can be such a pain, right? Not necessarily. Let's compare Sid's traditional approach to Chuck's new testing tool.

> "The evolution of computing from desktop machines to ultrabooks, tablets, and smart phones makes stress and boundary testing even more important."
>
> David **BORCHERDING**

Virtual Machines Ad Infinitum

Sid began by identifying all the stresses he needed to test. The first and obvious stress was the boundary error that caused the customer's crash. He also listed boundary values for all other fields, as well as low memory, low disk space, and sudden connection loss.

Unfortunately, no one knew exactly what the boundaries were or how the application should behave when those boundaries were hit. The product manager said, "Everything should degrade gracefully with limited resources." The lead developer said, "You need enough memory and disk space for all the data. And maybe some temp files. And the cache. I think that's everything." So Sid made his best guesses and moved on.

With a list of tests in hand, Sid began setting up virtual machines for each test. For the low disk space test, he set up a virtual machine with three megabytes left, one with two megabytes remaining, and one with one megabyte left. When he tried to install the application on the one-megabyte machine, he discovered that he needed more free space to complete the installation.

Likewise, when Sid created the limited memory test, he found he didn't have enough memory in the virtual environment to run both the app and his profiling and leak tool simultaneously. As a result, he had no idea where the application was crashing, which resources weren't being released, or any other useful data to help development. This led to several rounds of everyone's favorite testing game, "Well, works on *my* machine."

Sid had just about completed all the testing when he got word from Development that they had added new functionality to the app. Forced to perform all of the stress and boundary tests again, Sid muttered curses under his breath and began to set up a whole new bank of virtual machines.

BECAUSE CHUCK DIDN'T HAVE TO CREATE VIRTUAL MACHINES, HE HAD ENOUGH TIME TO TEST MORE OF THE APPLICATION BEFORE THE RELEASE DATE, RESULTING IN A HIGHER QUALITY PRODUCT AND HAPPIER CUSTOMERS.

Chuck vs. the Stress Test

Over at Evremonde, Chuck faced a similar situation. Like Sid, he began by listing all the stresses he needed to test, including the disk space problem that caused the failure for the customer. Although his list looked similar to Sid's, Chuck approached the testing much differently.

Instead of creating dozens of virtual environments, Chuck set up the tests using QA Wizard Pro, an automated testing tool. When testing the disk space limits, for example, Chuck made the test application think it only had one megabyte of disk space free by setting that as a limitation in the tool. When the app worked, Chuck reset the limit to half a megabyte and ran the test again. That also worked, but when he set the limit at 250k, the app failed. Chuck was then able to give the developer the exact file size and free disk space that caused a crash.

Because Chuck ran the tests in a simulated environment and not in a virtual machine, plenty of memory remained to run a disk-logging tool that recorded which files were (and were not) created. Chuck sent this report to Development, too.

The tool also made it possible for Chuck to reuse existing functional tests as stress and boundary tests. Likewise, as the app grew in later iterations, Chuck easily re-ran the tests by just changing the settings in his tool.

A Far, Far Better Rest

Because he didn't have to create virtual machines, Chuck had enough time to test more of the application before the release date, resulting in a higher quality product and happier customers.

Unfortunately, Sid wasted so much time setting up virtual machines that he couldn't complete all of the testing needed before the release date. As a result, the application crashed again, and the fed-up customer decided to abandon Stryverware.

While Sid's troubles with stress testing keep him up at night, QA Wizard Pro gives Chuck a far, far better rest than he has ever known.

About The Author

David Borcherding *is the Marketing Communications Specialist for Seapine Software. He helps spread the word about Seapine Software's products and services. Seapine develops application lifecycle management software for a variety of industries, including health care, financial services, utilities and game development. Our products include TestTrack, Surround SCM, and QA Wizard Pro. Check us out at www.seapine.com.*

DATABASE VIRTUALIZATION FOR DEVELOPMENT AND TEST

Standing up a development or test environment for incrementally exercising the latest code modifications is a common roadblock to developers & testers. Application-Behavior Virtualization (ABV) offers a new alternative.

Development and testing teams can capture the desired transactions (or behavior) of dependent systems and deploy a virtualized instance of the transaction that mimics the interaction of the constrained component—removing the traditional roadblocks associated with shared or inaccessible test environments.

by Wayne**ARIOLA**

What is Application-Behavior Virtualization (ABV)?

Application-Behavior Virtualization is a focused and efficient strategy for eliminating the system and environment constraints that impede the team's ability to test their heterogeneous component-based applications. Instead of trying to virtualize the complete dependent component, you virtualize only the specific transactions that developers and testers actually need to exercise as they work on their particular applications, components, or scenarios.

For instance, instead of virtualizing an entire database (and performing all associated test data management as well as setting up the database for each test session), you monitor how the application interacts with the database, then you virtualize the related database behavior (the queries that are passed to the database, the corresponding result sets that are returned, and so forth). This can then be accessed and adjusted as needed for different development and test scenarios.

Whether you need to rapidly exercise a new/evolving software component or performance test an application, you'll likely be dealing with interconnected systems that are constrained by the need to stand-up a database instance. A properly-configured database is a valuable and sometimes scarce resource. It slows down under load. It needs to be continuously available. Moreover, multiple systems may depend on the same database, thus limiting flexibility and increasing the time needed to configure it. To enable the team to efficiently develop and test without the constraints associated with a dependent database, you can leverage Application-Behavior Virtualization to capture the necessary scope of database calls and deploy a virtualized instance of those calls—totally cutting your dependency on the actual database.

ABV for Databases

Application-Behavior Virtualization allows for traffic to be recorded over a real database connection so that the same data will be available to be replayed later. This type of virtualization is usually achieved using a proxy in the interface between software system and database (you can also use a modeling tool to achieve the same result).

The proxy starts by recording requests and corresponding data responses. That recorded data will be used when the real database needs to be isolated from the load and data of the software system. At that point, the proxy stops forwarding the requests to the real database and instead looks up a virtual response. The software system is unable to tell the difference as long as sufficient sample data was recorded and some dynamic logic was implemented. The real database becomes isolated from this software system under development or test. At any time when development work and testing is finished, the proxy can switch back to forwarding traffic to the real database instance.

The overhead of database virtualization is much less than setting up a second database for testing and development. There is no need to install and maintain another database. No two databases need to be synchronized. And the proxy can even switch between virtualized responses and forwarding to the real database without requiring any software systems to be restarted.

Leveraging Application-Behavior Virtualization to emulate database transactions removes the bottleneck of development, QA, and production teams scheduling their work around the other teams' activities. Being able to disconnect from a staged (or even a production) database without the system working with different data or even restarting is a huge boost to productivity.

How Does ABV Work?

To start, you designate which database calls you want to virtualize, then—as the application is exercised—the behavior of the associated transactions, messages, services, etc. is captured in what we call a "virtual asset." You can then configure this virtual asset by configuring its conditional behavior, performance criteria, and test data. This virtual asset can then emulate the actual behavior of the dependent system from that point forward—even if the "real" database is no longer accessible for development and testing.

The Application-Behavior Virtualization tool can record database queries, along with their corresponding results data, and use that to virtualize the data connection. This enables you to decouple your application from database dependencies while working on testing and development activities. Such decoupling alleviates challenges associated with test data management challenges and multiple teams competing over the same database assets—which can often introduce complexity and instability into the environment. Furthermore, being able to virtualize database connections allows load tests to scale better in such environments—without requiring you to allocate or license additional database instances for such purposes.

There are three main steps involved in virtualizing database behavior:

- Capturing queries that are passed to the database and the result sets returned.
- Creating a virtual asset emulates the desired behavior.
- Deploying the virtual asset.

The test data that is associated with these virtual assets eliminates the need for the dependent database and the need to configure and manage the dependent database that, if shared, usually gets corrupted.

Hardware and OS Virtualization Lowers Cost & Increases Access – Yet Does not Solve the Problem

In an attempt to provide all of the necessary team members ubiquitous access to realistic dev/test environments, many organizations have turned to hardware and OS virtualization. Virtualizing the core test foundations—specific operating systems, configurations, platforms, etc.— has been a tremendous step forward for dev/test environment management. This virtualization provides considerable freedom from the live system, simultaneously reducing infrastructure costs and increasing access to certain types of systems. Moreover, leveraging the cloud in concert with virtualization provides a nearly unlimited bandwidth for scaling dependent systems.

Nevertheless, in terms of development or test environments, some significant gaps remain. First of all, some assets cannot be easily virtualized. For example, it is often unfeasible to leverage hardware or OS virtualization technology for large mainframe applications, third-party applications, or large ERPs.

Moreover, even when virtualization can be completed, you still need to configure and manage each one of those applications on top of the virtualized stack. Managing and maintaining the appropriate configuration and data integrity for all the dependent systems remains an ominous and time-consuming task. It is also a task that you will need some outside help with—you will inevitably be relying on other groups, such as operations or DevOps, to assist with at least certain aspects of the environment configuration and management.

Conclusion

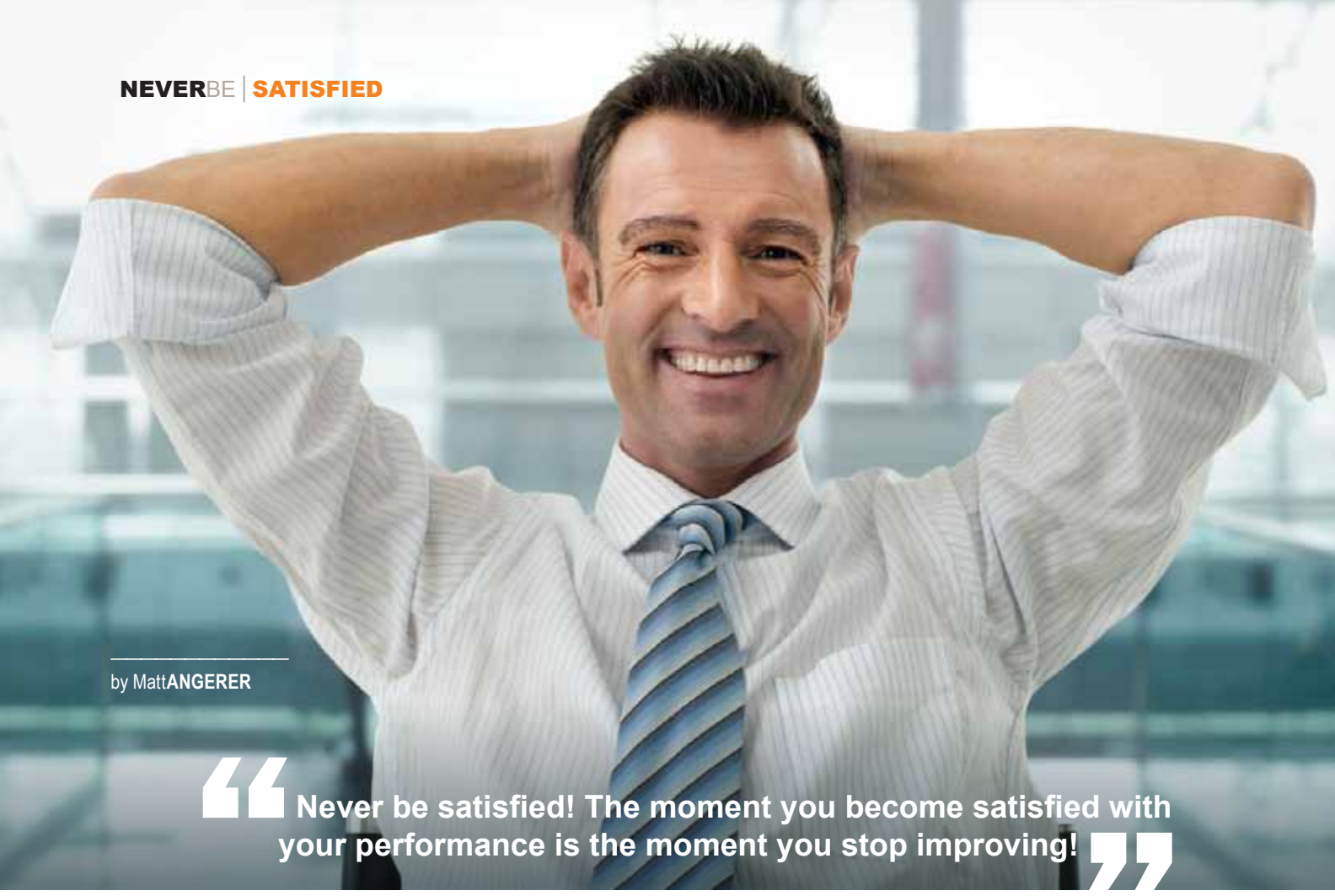
Application-Behavior Virtualization reduces this configuration and data management overhead by enabling the developer or tester to rapidly isolate and virtualize just the behavior of the specific dependent components that they need to exercise in order to complete their end-to-end transactions. Rather than virtualizing entire systems, you virtualize only specific slices of dependent behavior critical to the execution of development and testing tasks.

It is completely feasible to use the cloud for scalability with Application-Behavior Virtualization. Nevertheless, since you're virtualizing only the specific behavior involved in dev/test transactions (not entire systems), the scope of what's being virtualized is diminished… and so is the need for significant incremental scalability.

Even after the initial provisioning, these virtual assets are still easily modifiable and reusable to assist you in various dev/test scenarios. For instance, one of your test scenarios might access a particular virtual asset that applies a certain set of conditional responses. You can instantly construct an additional virtual asset that inherits those original conditions, then you can adjust them as needed to meet the needs of a similar test scenario.

About The Author

Wayne Ariola *leads the development and execution of Parasoft's long-term strategy. He leverages customer input and fosters partnerships with industry leaders to ensure that Parasoft solutions continuously evolve to support the ever-changing complexities of real-world business processes and systems. Ariola has contributed to the design of core Parasoft technologies and has been awarded several patents for his inventions.*



by MattANGERER

> "Never be satisfied! The moment you become satisfied with your performance is the moment you stop improving!"
>
> CoachBALLARD

Never Be Satisfied

It seems like yesterday when I heard my high school football coach bellow out, "Never be satisfied men! The moment you become satisfied with your performance on the field is the moment you stop improving." That was over 15 years ago and the echoes of that statement are forever etched into my central nervous system. In my humble opinion, I was lucky enough to be influenced both on the grid iron and off by some of the best mentors and coaches that the game of life can bring. With that said, I am fairly confident that Coach Ballard didn't realize the impact that his statement would have on me or the organizational processes I would touch in years to come. On that 100 degree August afternoon, he was trying to do more than simply motivate his football players to perform better. Although motivational techniques are useful in getting your Software Quality Team over the last hump of testing before go-live — it won't ensure long-term organizational success. My football coach understood that in order to achieve success on the football field, his statements had to provoke much deeper consideration among his more mindful players. His statements had to truly shake each of his players' central nervous system to the point where improvement was a *"must have" requirement* every single day we stepped foot onto that football practice field. We didn't have to consciously think about improving – we sought out ways to improve our game and we naturally identified opportunities to improve the smallest things in how we stepped up for a particular play or caught the ball for that matter. As a result, we went from 2 and 7 my junior year to 6 and 3 my senior year. Even though we experienced loss, the team culture was fundamentally altered by changing our focal point of ***never being satisfied.***

As an SQA Leader, Test Manager, or Tester within your organization – have you seriously taken the time to identify areas you can improve upon? Perhaps the test management tools you've been using the last 5 years need a massive re-haul. Or maybe you need to shake things up a little bit by moving people around within your team. Just because you've done something one way for so long, does not mean there isn't room for improvement. Don't get stuck in the mindset of: "If it ain't broke, don't fix it." No matter where you look, no matter what process you think is working, there is always room to improve and tweak an existing process. If your brain is conditioned to identify opportunities of improvement, you will find them. It's called the reticular activating system (RAS) and it's an area of the brain responsible for regulating arousal and sleep-wake transitions. Don't believe me? Has anyone ever painted a picture of someone's personality characteristics before you met them? As you listen to someone describe somebody else, your RAS is collecting this information – the more colorful the story that this person tells of the other and the more impressionable you are – the more alive your RAS becomes. When you meet this "person" finally, you will automatically (yes, without conscious thought) pull out cues that align with what you've heard about this individual. The same concept applies to your ability as a software quality professional to uncover opportunities for improvement. If you listen to the "wrong people" within your organization or "follow the crowd" in terms of thought, action, and process – you could be focused on the "wrong stuff" that leads you away from software quality objectives.

A real-world example that will help crystallize this concept for an SQA professional comes from my recent experience of leading a team of testers that are operationally responsible for regression testing a Java application installed on a PC-based device. The application was architected, designed, developed and rolled out in the late 90s. The functionality of the software is stellar as it relates to its pragmatic use by the customers. Customers are very satisfied and the company continues to grow its market share with the device. The software is highly stable, easy to use, and continues to evolve with new features and functionality. All is good – the customer is happy – why should we look for ways to improve existing SQA processes? Without belaboring the point, we know why: **never be satisfied.**

As an SQA Leader, my mind directed me to the team's use of tools to manage the regression testing cycle of this Java application prior to field testing. I asked myself the right questions, which I think led me to some good answers:

Q *Although our customers are happy today, is there anything I can do as an SQA Leader to inject business value into the systems development lifecycle of this Java application to make the customers even "happier"?*

A Since the number of software defects found by the customer is very low and customer satisfaction is already sky high – what else could we do? Perhaps the timeframe of having the product in SQA could be critiqued and improved upon.

Q *Is the team fully utilizing the set of tools made available to them by the organization: HP Quality Center and Microsoft SharePoint to name a few.*

A The team is utilizing HP Quality Center as a glorified bug tracking tool – but they are not taking advantage of its full range of capabilities. Defects are not linked to test cases, and test cases are not linked to requirements. Requirements traceability is non-existent and needs to be addressed. Additionally, Microsoft SharePoint was being utilized, but only at a surface level. By expanding the use of SharePoint to granulate a test schedule across the decentralized testing team, productivity among the testers would improve and less downtime would occur as a result of setup activities.

Q *Are there other "test management tools" available that could improve software quality while simultaneously shortening the test execution time table?*

A With so many product platforms & device configurations to test against on this team, a tool that allowed for simultaneous test execution across various configurations would immensely shorten the test execution time table.

By evaluating product sheets, sitting through demos, and talking to sales representatives – we were able to pilot and prove out a test management product that dramatically shortened the regression testing cycle timetable. The new tool helped us to improve manual testing by giving us the ability to simultaneously execute test scripts across 5 different product configurations at once – known as mirroring. The SQA testers were no longer limited to testing against one PC-based product at a time — they could test the Java application from one product while it simultaneously (in real-time) mimicked the UI actions across the 5 other product configurations. Specific paths that did not require manual intervention were easily automated with the use of macros. Our pilot program proved to be successful – increasing the velocity of our regression test suite by nearly 500% per SQA Tester. The improvement in this area allowed our offshore team to focus more of their efforts around exploratory testing to uncover those pesky bugs that the customer usually never finds, but should be fixed because it's the right thing to do.

The moral of the story is simple: don't be a wallflower. Just because the customer is satisfied today, you shouldn't rest on your laurels. The competitive landscape is shifting by the day – we have a responsibility as QA professionals to continually evaluate and implement tools that work best in our organization. Regardless of your vendor preference, you should find tools that add business value to your organization. Reducing the time it takes to get a software product shipped is not what many QA professionals want to hear – it's the unpopular thing to speak or write about because we could test an application until we're blue in the face. But if you can find a test management tool that does just that while also expanding test coverage – go for it!

Here's to my high school football coach. Sometimes a mindset is more important than knowledge because it will lead you to opportunities.

About The Author

Matt Angerer *spent nearly 10 years as a traveling ERP (enterprise resource planning) and APS (advanced planning systems) consultant. Matt has managed and consulted in a broad range of areas, including: supply chain planning, IV&V, software training, test management, and systems integration.*

GET A TESTING **RESULT** YOUR CEO WILL **LOVE**.

Your CEO will love you when you show them how you can achieve an outstanding testing ROI with ISTQB Software Tester Certification.

With the average cost of a software defect in the range of $4,000 – $5,000[1], if ISTQB Certification helps your tester eliminate even just one defect, the result is nothing less than, well, loveable: an ROI of up to 2000%.

ISTQB Software Tester Certification is the most widely recognized and fastest-growing software tester certification in both the U.S. and the world. Discover how ISTQB certification can pay for itself in a matter of days: That's a testing result any CEO will love.

Want an even better ROI?

Take advantage of our new **Volume Purchase Program**.

Learn more now at

www.astqb.org

ASTQB
American Software Testing Qualifications Board, Inc.
ISTQB Certification in the U.S.

[1] Capers Jones, "A Short History Of The Cost Per Defect Metric", Randall Rice, "The Value of ISTQB Certification"

by DaleBRENNEMAN

INTENTIONALLY Avoiding Unintended Side Effects Using Observation Driven Testing with Test Driven Development

Executive Summary

Observation-Driven Testing (ODT) is a development methodology for organizations intent on maximizing the effectiveness of development teams while also maximizing code quality.

About The Author

Dale Brenneman *is Vice President of Technology at Agitar Technologies, Inc. and has been in the software development industry for over 30 years. He has served as a manager of development, testing, services, and support organizations, and as a programmer, tester, quality assurance specialist, trainer, and consultant. Mr. Brenneman has worked for several federal government agencies, for a government contractor at NASA, and for commercial software vendors. He has taught training courses and presented at software industry events.*

ODT is complementary to, yet not dependent on, the widely adopted Test-Driven Development (TDD). Ron Jeffries observed that TDD is an excellent methodology for developing "clean code that works." Its many strengths include the ability to create code that does only what you want and to create a thorough set of automated tests. However, by itself, TDD is incomplete as a coding and testing methodology because of the fact that it creates code that does only what you want without taking into account unintended side effects. ODT fills that void, using automated tools to test the behavior of code and providing developers with actionable observations about possible unintended side effects.

Test-Driven Development (TDD)

TDD is a very good technique for developing new code and maintaining existing code. It has been called "test-first programming" and "code unit tests first" development. It results in not only a complete set of automated tests at the "unit" level (with high code coverage) that protect your investment when each change is made, but also a code base that has been thoroughly tested step-by-step as it was being developed. These results are in contrast to the results of the all too common "code-first test-as-long-as-you-have-time-left-before-your-deadline" development techniques, which can result in incomplete test sets and limited testing cycles.

Let us assume that requirements for implementation are in a form that we will generically call Use Cases, whether formal or informal, and even if details are negotiable or TBD. Then, a simplified view of TDD is a process that repeatedly, in small increments, performs the following steps:

- **Red Bar –** Implement test(s) based on a Use Case or a developer observation of a requirement, and execute the set of tests, which will result in test failure(s), i.e., a Red Bar test result indicator.
- **Green Bar –** Implement code until the tests execute correctly, i.e., a Green Bar test result indicator.
- **Refactor –** Change code to eliminate the duplication created by focusing on getting tests to work quickly, which also reduces dependencies.

TDD performs implementation from a different angle than architecture-driven development. Architecture-driven development first focuses on defining designs that are clean, and often later making changes when specific cases reveal that changes are needed to make the code work. Alternatively, TDD first focuses on implementing code that works, and sometimes later making changes when general cases reveal that changes are needed to make the code meet requirements. Note that TDD can be used successfully within various methodologies, including but not limited to those based on Agile methods such as eXtreme Programming (XP) and Scrum.

TDD is successful due to the human brain power that repeatedly performs its very focused steps, and that brings in ideas that add thoroughness and reduce duplication. There is limited automation in the process other than automated test execution/reporting and coding tools/IDEs. There is no automation to generate tests for code that doesn't exist.

TDD's strengths and weaknesses relative to unintended code behavior are:

Strengths:

- Defines code requirements more accurately than writing code-first.
- Allows fast code writing through the use of refactoring tools after tests are written.
- Avoids code bloat and feature creep by requiring testability for all implemented features.

Weaknesses:

- Tests/requirements developed before product implementation tend to emphasize only sunny day/success scenarios.
- Tests cannot be automatically generated.
- Tests do not cover cases that developers do not think of, which may include some general cases encompassing the interactions of many code units.

Observation-Driven Testing (ODT)

ODT is a good technique for identifying code behavior that is unexpected. An automated tool can test code by executing it in many ways based on the contents of the code units (e.g., Java methods) and their relationships, and then provide its observations about code behavior to the developer for review. This exploratory testing of code has been termed "agitation." In order to find these observations, each code unit is executed multiple times with various input values using exploratory techniques. Then, the developer is presented with normal and exception outcomes and observations in the form of code expressions, and with code coverage obtained during this testing. This gives several different perspectives on the code's actual behavior that can be used for review and validation. After adjusting code so it works as expected and has the desired number of constraints to be highly maintainable in the future, the developer can promote remaining observation expressions to test points.

In addition to executing available unit tests, the ODT process should be applied to small increments of development. A simplified view of ODT is a process that repeatedly, for small increments of code that might be checked in after a small amount of work (maybe an hour to a day's worth of effort), performs the following steps:

- Agitate Code – Request an automated tool to perform exploratory testing and provide its observations; and apply techniques to improve code coverage where code constrains coverage.
- Adjust Code – Until no Unexpected Observations exist, adjust code to correct the behavior and rerun your automated tests, then re-agitate.
- Add Test Points – For Expected Observations, promote them to test points to help constrain the code behavior as known by tests, creating additional expressions, if desired, to include as test points.

ODT's strengths and weaknesses relative to unintended code behavior are:

Strengths:

- Points out unintended behavior for which a developer would usually not write tests.
- Promotes good code construction by pointing out code that is difficult to test, prompting early refactoring.
- Provides an easy mechanism to add test points from observations of concrete behaviors.

Weaknesses:

- Code must exist before observations are made.
- Automated testing is for actual rather than intended code behavior.

How Are TDD and ODT Complementary?

TDD has great value as a software development process but can be incomplete in terms of the creation of code that does what you want without unintended side effects. ODT uses automated tools to test the behavior of code and provide actionable observations about possible unintended side effects. Therefore, because ODT is strong where TDD is weak, ODT is an excellent complement to TDD as part of your development processes. Using ODT with your TDD will reduce defects caused by unintended code behavior. The following table summarizes how ODT complements TDD.

	TDD	ODT
Goal	Make sure code does only what you want	Make sure code does what you want without unintended side effects
What is performed during development and testing?	After requirements in some form exist (formal or informal), automated tests are created. Then code is created to cause failing tests to pass. Then code is refactored to reduce duplication (improving testability and maintainability).	After code exists, code is analyzed by automation to identify observations about behavior that may point out either defects or needs to specify further tests; then code may be changed, further tests created, assertions made, or code refactored.
Focus in reducing defects	Reduces opportunities for untested behaviors, relative to requirements and code coverage	Reduces opportunities for untested behaviors, relative to code base's behavior
Strengths in avoiding code defects from unintended side effects	Specific cases, based on known requirement	General cases, based on code behavior

ODT Tools for Java Projects

A new generation of automated tools can support an ODT process for Java. These tools use exploratory testing techniques to execute code and provide actionable behavior information to the developer. The developer then can review each observation and determine whether to change code (to fix unintended behavior), create additional tests, promote observations to assertions, or refactor code for improved testability. These tools are meant to be used during code development to reduce defects due to unintended code behavior. Only an automated tool can perform such a thorough analysis of behaviors.

ODT tools are also available to support legacy code development. Some projects, such as those that have not used TDD, may not currently have a complete set of unit tests (i.e., with high level of code coverage across the project). Most organizations cannot find resources to go back and manually create a complete set of unit tests, even though they know these tests would have great value in reducing defects going forward. Having a complete set of characterization tests gives confidence to make changes to existing code with minimized fear of regressions. ODT tools can be used to create such a set of tests. They provide a set of automatically-generated, passing tests with 80% or better coverage, using exploratory testing techniques plus significant automatic mocking technologies. Easy to use techniques can be used to improve coverage levels from there if desired. The process is simple: automatically generate passing tests, make code changes, run tests, inspect test failures, account for unexpected failures and automatically re-generate tests so they are available for the next code change.

One of the significant obstacles to successful TDD implementation is the amount of effort required to create harnesses for testing. On the one hand, this encourages development designs that make components more easily testable in isolation, because introducing more dependencies increases the testing effort. However, since the test is written before much of the system is implemented, there are inevitable cases where the developer will have to implement interfaces, stubs, mocks, etc. for the integration points as part of writing the test. This can be a substantial effort. Exploratory testing features automatically create mock objects to use when executing the code under test. Thus, they can provide feedback about the behavior of the unit of code, even when the external integration points are incomplete.

Sample ODT Observations of Possible Unintended Code Behaviors

This section provides an example of first using a TDD process to create tests and code, then using an ODT process to identify and resolve unintended code behavior.

Using TDD

First, let's say our requirements call for a method to set a model number String for a product. In the Product class, the setter method should check the input parameter for the following:

- String cannot be null
- String cannot be blank
- String cannot contain newline or carriage return characters
- String cannot be greater than 40 characters long

Now we will write the tests to fit our requirements:

```
public void testSetModelWithNewline() throws Throwable {
    Product product =
        new Product("M-6249-56-Y", "testProductName", 100.0);
    try {
        product.setModel("XXX\nXXX");
        fail("Expected IllegalArgumentException to be thrown");
    } catch (IllegalArgumentException ex) {
        assertEquals(
            "ex.getMessage()",
            "Model cannot contain newline or return characters",
            ex.getMessage());
    }
}

public void testSetModelWithReturn() throws Throwable {
    Product product =
        new Product("M-6249-56-Y", "testProductName", 100.0);
    try {
        product.setModel("XXX\rXXX");
        fail("Expected IllegalArgumentException to be thrown");
    } catch (IllegalArgumentException ex) {
        assertEquals(
            "ex.getMessage()",
            "Model cannot contain newline or return characters",
            ex.getMessage());
    }
}

public void testSetModelEmpty() throws Throwable {
    Product product =
        new Product("M-6249-56-Y", "testProductName", 100.0);
    try {
        product.setModel("");
        fail("Expected IllegalArgumentException to be thrown");
    } catch (IllegalArgumentException ex) {
        assertEquals(
            "ex.getMessage()",
            "Model cannot be null or blank",
            ex.getMessage());
    }
}

public void testSetModel41Chars() throws Throwable {
    Product product =
            new Product("M-6249-56-Y", "testProductName", 100.0);
        try {

             product.setModel("41CharactersXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
            fail("Expected IllegalArgumentException to be thrown");
        } catch (IllegalArgumentException ex) {
            assertEquals(
                "ex.getMessage()",
                "Model cannot be longer than 40 characters",
                ex.getMessage());
        }
    }
        public void testSetModelNull() throws Throwable {
        Product product =
            new Product("M-6249-56-Y", "testProductName", 100.0);
        try {
            product.setModel(null);
            fail("Expected IllegalArgumentException to be thrown");
        } catch (IllegalArgumentException ex) {
            assertEquals(
                "ex.getMessage()",
                "Model cannot be null or blank",
                 ex.getMessage());
        }
    }
}
```

Next we will create the code that will make our tests pass. To improve readability, we will factor out the input checking from the setter into a separate validateModel() method:

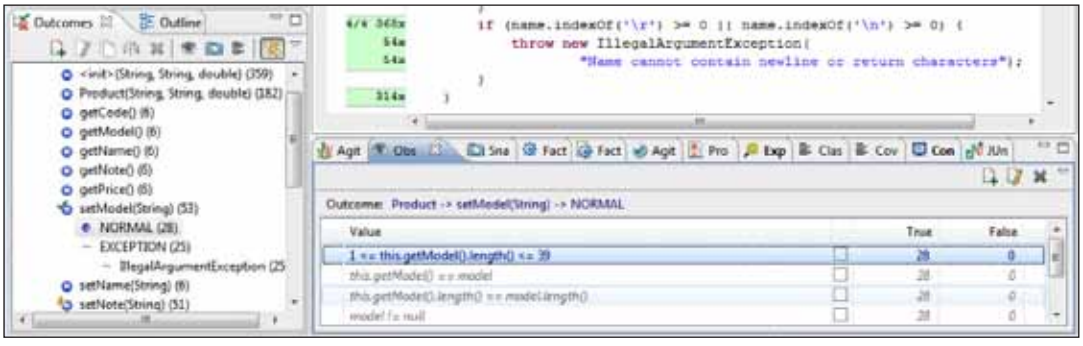
```
private String model;

public void setModel(String model) {
          validateModel(model);
          this.model = model;
}

public String getModel() {
    return model;
}
private void validateModel(String model) {
    if (model == null || model.length() == 0) {
        throw new IllegalArgumentException(
            "Model cannot be null or blank");
    }
    if (model.length() >= 40) {
        throw new IllegalArgumentException(
            "Model cannot be longer than 40 characters");
    }
    if (model.indexOf('\r') >= 0 || model.indexOf('\n') >= 0) {
        throw new IllegalArgumentException(
            "Model cannot contain newline or return characters");
    }
}
```

Using ODT

The new code created by a TDD process seems to fit our requirements just fine, indicated by our passing tests, but is it really doing what we want? Let's agitate it (that is, apply the ODT process) and see if we can spot any bad behavior:



One glaring observation we can make here is that if we look at our NORMAL outcome from setModel(), it looks like agitation found that the string can be from 1 to 39 characters long but the requirement stated "not over 40." A simple off-by-one error here could be a problem in the future, so we should fix the code, re-run the tests and see if we can make any agitation assertions.

```
    if (model.length() > 40) {
        throw new IllegalArgumentException(
                "Model cannot be longer than 40 characters");
}
```

In the code above we have changed the '>=' symbol to a '>' symbol which should fix the problem. If we re-run the tests they all still pass, and if we re-agitate the code we have an observation that reads:

```
<= model.length() <= 40
```

This is now consistent with the specified requirements. At this point, we should promote that observation to an assertion. This is a quick alternative to adding another JUnit test and the ODT tool integrates the results into the same dashboard report along with the outcomes of the JUnit tests. Thus, the ODT process improved the code quality.

Recommendations on When to Apply ODT

ODT can provide value regardless of whether you have been using TDD. This section provides recommendations on when to apply ODT, relative to your project's process history, development plans, and existing unit test suite completeness. Additionally, we will discuss when automated unit test generation can be used to shore up test deficiencies if you have not been using TDD or otherwise do not have a complete set of unit tests (i.e., low code coverage).

TDD can be a very good practice as part of a high quality software development process. But TDD is not totally thorough by itself. Even if you have 100% code coverage with your TDD-generated tests, your process can be enhanced using ODT. Furthermore, if TDD has been implemented only partially by not applying it to all portions of the project, or has not been used at all, your unit tests may be incomplete. And therefore automatic test generation should also be employed as you proceed from this point. Your process should be enhanced to use ODT for at least all newly developed and changed code, and possibly also for code that has a history of defects.

> "Even if you have 100% code coverage with your TDD-generated tests, your process can be enhanced using ODT."
>
> Dale BRENNEMAN

The following table summarizes the recommended processes.

TDD	ODT	Automated JUnit Test Generation
After requirements exist, tests are created, then code is created to cause failing tests to pass; then code is refactored to reduce duplication (improving testability and maintainability)	After code exists, code is analyzed by automation to identify observations about behavior that may point out either defects or needs to specify further tests or assertions; then code may be changed, further tests created, assertions made, or code refactored	After code exists and is tested and stable, additional unit tests are created by automation; then after next code change tests are executed and failed tests analyzed for potential unintended defects (regressions); then code may be fixed or refactored, or other tests created

The following table recommends when to use ODT, and also automatic test generation, based on your project's process history, future development plans, and state of existing unit tests.

	Process History with Respect to TDD	Future Development Plans	State of Existing Unit Tests	Recommended Process Enhancements
1	Just starting development, and will be using TDD	All new code to be developed (may have reuse from prior projects)	None	Add ODT to TDD
2	All parts of existing code base have been developed using TDD	Much new code still to be developed	Fairly Complete (very high coverage)	Add ODT to TDD
3	Some parts of existing code base have been developed using TDD, others not	Much new code still to be developed	Fairly Complete (very high coverage)	Add ODT to TDD; even if you plan to proceed without using TDD for new code, at least use ODT
4	Some parts of existing code base have been developed using TDD, others not [same as 3]	Much new code still to be developed	Incomplete (much less than 100% code coverage)	Add ODT to TDD; even if you plan to proceed without using TDD for new code, at least use ODT Add automatic JUnit generation to create a more complete safety net of unit tests
5	No parts of existing code base have been developed using TDD	Much new code still to be developed	Fairly Complete (very high coverage)	Add ODT to TDD; even if you plan to proceed without using TDD for new code, at least use ODT
6	No parts of existing code base have been developed using TDD [same as 5]	Much new code still to be developed	Incomplete (much less than 100% code coverage)	Add ODT to TDD; even if you plan to proceed without using TDD for new code, at least use ODT Add automatic JUnit generation to create a more complete safety net of unit tests
7	**All, some, or no parts of existing code base have been developed using TDD** **[same as some above]**	**Little or no new code is to be developed; only minor maintenance changes planned**	**Fairly Complete (very high coverage)**	**Apply ODT to areas of code that have a significant history of having defects, to help find potential remaining defects**
8	**All, some, or no parts of existing code base have been developed using TDD** **[same as some above]**	**Little or no new code is to be developed; rather only minor maintenance changes planned**	**Incomplete (much less than 100% code coverage)**	**Add automatic JUnit generation to create a more complete safety net of unit tests** **Apply ODT to areas of code that have a significant history of having defects, to help find potential remaining defects**

Steps to Use TDD and ODT Together for Maximized Defect Avoidance

This section provides a suggested set of steps by which TDD and ODT can be used together for maximized defect avoidance. Note that your processes and terminology may be slightly different, but the basic concept is that you would perform TDD then ODT in small, incremental steps during development. Reference earlier sections of this document for more detail on the processes.

Perform the following in small, incremental steps:

1. Apply TDD to develop a small set of tests and clean, working code, to the point that you are ready for checkin; possibly one or several hours of work up to a day's work.
   - Red Bar
   - Green Bar
   - Refactor
2. Apply ODT to improve that code, to the point that you have no remaining unintended observations.
   - Agitate Code
   - Adjust Code
   - Add Test Points

Conclusion

It is never a good idea to put all of your eggs in one basket. No matter what type of development and testing philosophies you adapt, including TDD, there can be holes in your attempts to reduce defects and improve maintainability. ODT can help fill such holes with its automated code behavior observation. A result of successfully applying multiple techniques is that your code will be more constrained, more maintainable, and better tested.

SOASTA

The Leader in Cloud Testing

CloudTest supports functional and performance testing for today's web and mobile applications.

- Fast Time-to-Test
- Scalable and Global
- Real-time Intelligence
- Affordable

Download the FREE edition today at www.soasta.com

ASK THE TESTER

Compiled By
Matthew **HEUSSER**

Matt contributes to the STP community blog at http://www.softwaretestpro.com/blog or follow him on Twitter @mheusser

Many of the folks we feature on ask the tester are consultants with big names and public faces, and that's good. Still, we try to achieve a balance; we also want to hear from practitioners.

If we're going to put on a practitioner, how do we know they are good?

Well, how about someone from a company so flawless in its IT execution that you never hear their name in the news?

Here's one: Lowe's companies. You know, the big buildings that sell lumber, hardware, and lots of other things.

You could probably also guess that they have a corporate office, HR department, have to pay a lot of salaries, have a complex supply chain, keep a website running, and have an IT department with more than a few testers.

Meet Mike Lyles, QA Program Manager for Lowe's Companies Inc.

Like many folks in QA, Mike has been at one company for over a decade (nearly two), quietly doing the work while slowly gaining promotion from help desk to system analyst, project manager, QA Manager, and now QA Program Manager.

For this month's theme on test management, we thought he was the ideal fit.

MICHAEL LYLES
will now answer your questions.

QUESTION *As a test manager, how do we get our testers to actually work on improving their personal test approaches and knowledge within whatever test context the organization has?*

– Jon Hagar, *Mountain tester; Hot Sulphur Springs, CO, USA*

Want More?

Go to softwaretestpro.com and join the conversation – 'Crew' members can ask your questions and interact with Michael directly!

Next issue we'll flip things up a bit – Marlena Compton will interview yours truly, Matt Heusser, on new directions in software testing.

Please email your questions, name and location with 'Ask the tester' as subject line to ***matt.heusser@gmail.com***

MIKE LYLES This is always a struggle. What I have found in my experience is that it's even more difficult to get a testing organization to follow the same methodologies and practices. We have guidelines, templates, processes, and standards, yet teams get caught up in the consistent drive of the projects and lose focus on following those standards. With any occupation, the key is to keep yourself in continuous improvement. There is a quote "HR is not looking out for your career growth – it's up to you." This is so true, especially in testing organizations. I encourage my team to continually read up on new processes and methodologies, as well as look for opportunities to build upon their skillsets by studying testing materials (books, online repositories such as www.softwaretestpro.com magazines, articles, etc). When you cease studies, you cease to improve. And finally, I encourage my team to think outside the box and to come prepared to suggest new testing approaches – we don't guarantee that all suggestions will get accepted, but we will surely discuss and determine if it's an improvement opportunity. Certifications are an awesome way for team members to stay in sync. Our organization went through training courses in early 2011, and each of our team members became ISTQB certified as a result. This investment by our company was instrumental to ensuring that everyone 'spoke the same language'

QUESTION *What is unique about testing at your org compared with others in the industry? How is testing changing? Is it changing? Are your concerns as a manager different than they were as a test practitioner? If so, how are they different?*

– Lanette Creamer, *Seattle, Washington, USA*

MIKE LYLES Our company started our testing group in late 2008. Before then, testing was conducted by two developers – the one that developed the code, and another developer that would conduct a 'second test' on the changes. We have grown so much in three short years yet we are still evolving as a world class testing organization. I am not sure I would characterize our organization as unique compared to others in the industry. In fact, I feel we are still learning – trying to establish ourselves among the other groups in IT. However, I do feel we are making tremendous strides currently, with plans to improve our processes in 2012 in a way that will make us very effective.

As far as changing, the world changes so fast, you either follow it and stay ahead or you become irrelevant. Stephen Covey once said "Nothing fails like success." Things you do today that are above par and surpassing others in the industry will be below average and irrelevant in a very short time. Therefore, it's our responsibility to keep relevant and ensure we move with the best practices that are being presented by so many of the testing experts in the field. I am fortunate to be involved in major initiatives right now, as a Test Environments and SCM manager, to do just that for my organization. Our first focus is to ensure our testing environments are stood up, set up accurately, and ready for the first day of testing. In concert with this initiative is our focus on Test Data Management – ensuring that not only our data is accurate and precise, but that it is in sync among all the various test environments we work on today. Environments and Test Data are so critical to the success of testing, and if an organization is not focusing on these to streamline the processes and move to steady state in these two areas, they will always have issues at test execution time. Lastly, we are making a significant step this year in Software Configuration Management. While this is not typically a testing organization role, it is critical to ensure that the code promoted from development to QA and eventually to production is 100% accurate. Incorrect code, data, or environments are three elements that can make or break a testing organization's execution.

To your last question – I have held many roles in the organization. I was a developer & tester for a long time. I was fortunate to be part of many testing efforts even before we had the testing group. And I feel that the major difference in concerns of a test manager verses a test practitioner is a focus on commitments. In a well-run team, the test practitioners should feel confident focusing on the preparation of test cases, preparing for and conducting test execution. A test manager should be focused on orchestrating all the external factors that could keep the test practitioner from achieving those results – such as coordinating with the Test Environment, Test Data Management, and SCM teams to ensure that everything is available and ready for the test practitioners to do their job. Additionally, while the test manager should empower the team to monitor and govern the entry & exit criteria to move between development testing to QA testing to UAT and production, the accountability for auditing and governing these practices must belong to the test manager. Any role where a person is a manager should be taken seriously. Your team is looking to you for answers and guidance, and it's critical to be responsive and supportive at all times.

QUESTION *What would you advise a tester to do now if they wanted to become a test manager in the future? How did you become a test manager?*

– Lanette Creamer, *Seattle, Washington, USA*

MIKE LYLES This is a great question and one I get asked by my senior test engineers often. I strongly feel that each step we take in our careers can prepare us for the next phase. There is so much to be learned from being in the trenches as a test engineer. You get the opportunity to have hands on, "front row," visibility to the things that work and the things that do not work in the testing lifecycle.

The thing I challenge my team to always focus on is to become evangelists for the testing practices that our organization has established. And I suggest that they submit them to memory so vividly that they don't need to refer to a standards document or reference book to speak it. The team should be in sync and speaking the same language, and MOST IMPORTANTLY, they should collaborate to ensure that there are no contradictions in the practices by one team or test engineer verses another.

The last thing you want to hear in a major project or initiative is "these rules, standards, practices, entry / exit criteria are not the same as the last project I worked on." If you can get the team to speak consistently, the positive perception and respect of the organization will grow exponentially in a very short time.

Lastly, most teams expect their test engineers to be heads down, running the tests, reporting the results. I always suggest to team members wanting to move toward management that they begin building the relationships with the key stakeholders of the projects they are working on – because as they move into a management role, they will need the support from each of these teams to accomplish their goals fully. Therefore, I always suggest to my up and rising test managers that they focus on not only the methodologies, but their approach on how they will enforce them with minimal to no friction among the teams.

QUESTION *Do you experience "communication gaps" to other managers or execs/stakeholders – i.e. a simplified perception of what testing can and can't do (is and isn't)? If so, how do you work with that perception?*

– *Simon Morley,* *Stockholm, Sweden*

MIKE LYLES We all know that it is difficult to enforce standards and practices. We are the 'Highway Patrol' of IT. We set the rules for testing, we post them, and if the teams are not following them, we step in and work with the teams to correct the mistakes. The goal of the test manager is to ensure you do this with tact, respect, and a focus on collaboration – and that we maintain respect among the organization in the process. None of us like to see a Highway Patrol, in our rearview mirror, pulling us over for a violation – but we most always respect them for the job they are doing to make the roads a safe place to travel. The same applies to the role of the test manager (and ultimately the testing organization). You have to be prepared to show the value of the testing organization and the benefit of the methodologies being enforced.

QUESTION *And a follow-up, if I may: How do you discuss or present your "testing story/message/report" to stakeholders (or other non-testers)? Do you (or your stakeholders) separate this from 'feelings' about the product being tested?*

– *Simon Morley,* *Stockholm, Sweden*

MIKE LYLES I like your statement of "perception of what testing can and can't do." This is critical to be clear and concise early in the project for the expected roles of the testing organization, the development teams, and any other external teams to the testing group. And it is important that the key stakeholders are aware and agree with the set roles and responsibilities. The way our organization has ensured this collaboration is to schedule and conduct a kickoff meeting early in the project and to walk through what our expectations are for entry and exit criteria and what conditions would cause us to not be able to move from one phase of the project to the next.

Once testing has started, the focus should be on the defects and how to resolve them – the key is to ensure this is not "we vs. they" – we drive for a collaborative team approach to determine the reason for the defects, the resolution, the timing of that resolution, and the plan for resolving those issues effectively. The earlier an organization can detect and recognize a defect, the lower the costs involved in the resolution. Tracking defect leakage and where the defect was detected (i.e. in Unit Testing, Component, System Integration, or UAT) will be critical to helping the team to look for ways to identify defects earlier in the process.

QUESTION *As a QA Program Manager, how do you think you can best impact a tester's role? (Speaking of which, how do you differentiate between test program management and test management?)*

– *Janet Gregory,* *Calgary, Canada*

MIKE LYLES In our organization it has proven critical, as a test manager, to provide immediate assistance and support to the testers to ensure they have everything easily accessible to conduct their jobs. Our focus for this year has been to provide fully functional test environments, concise and accurate test data that is mapped to the requirements for the project, and to react quickly to any escalations or needs that the team has along the way. Another thing we focus on is to ensure that the testers do not have to debate the methodology or standards for the testing organization. They should be able to focus on their role in creating, executing, and reporting on the testing efforts and the discussions between the testing organization and the other teams should be made at the management level.

To your question on test manager vs test program manager – in my situation, my title didn't change, only my role, as a QA Manager and QA Program Manager. However, my responsibilities grew. Instead of being a direct Test Manager for a specific project or projects, I was in position to lead Test Managers who were working with major program initiatives for the company. We had many testers across these major strategic programs (over a hundred). And in this situation, these programs were all inter-related. Therefore, I spent a lot of my time bridging the gap between the programs, recognizing the interdependencies, monitoring the risk – especially if issues or delays in program A impacted program B, C, or D.

QUESTION *Your management asks you to come up with metrics that will show the value your team brings to the organization. What metrics do you use?*

– *Yvette Francino,* *Colorado Springs, Colorado*

MIKE LYLES This is one of my favorite subjects when it comes to testing. "What gets measured gets done" is so true when it comes to testing organizations. I had the privilege of meeting Michael Bolton (the testing guru not the singer) earlier this year at a conference and he was kind enough to sit with me over dinner and share his thoughts on metrics.

Michael made a statement that has stuck with me throughout – he said "First we must agree that what we are talking about is a Measurement Program. Metrics is the raw data that feeds a Measurement Program. You wouldn't characterize a book as 'words.'" I agreed and since that time I have referred to our reporting as a Measurement Program.

Building a world class Measurement Program is a monumental task. Anyone who says it is "just data" has not been involved closely enough to realize the importance of measurement. And you have to be prepared to present your measurement program in various ways to different audiences. For instance, there are things a test manager, development manager, PMO, or business stakeholder will want to see that an IT VP or Senior VP will want to see at a more summarized level.

Regardless, I will share with you some of the things at each level that I feel are important for any measurement program:

- **Project Level:** Need to show the defect leakage, time to resolve defects, and the overall metrics on the effectiveness of the testing at all phases of the project. Also, if your entry and exit criteria require that certain levels of defects (i.e. critical defects) cannot exist to move to another phase of the project, it is critical to be able to report the status of the critical and high defects and the timing for when they will be resolved.

- **Management Level:** For this reporting, you should focus on a testing dashboard which will display the status of the testing efforts, any impacts to timelines, and any mitigation to the risks of not meeting those timelines.

- **Executive Level:** This is where your reporting cannot have the raw data. Executives will want to see the health of the project, but also to recognize the areas where testing was effective and saved time and resource dollars along the way. Also, this is an opportunity for the team to showcase areas where major issues were diverted and to give praise to the teams that were responsible for assisting in the mitigation.

- **Business Level:** For this reporting, it is critical to first sit with your business stakeholders and discuss what is important to them in measuring the project success. Different business stakeholders may see value in various ways. But once you have this information, the reporting here should showcase the benefits of the testing, and a focus on how we saved time and money in the process.

At the end of the day – the goal of the measurement program is to provide every one of the recipients of the reports the information they need to fully understand the status of the project and to be aware of any risks. You will know that you have successfully implemented an effective Measurement Program when the selling of the benefits of the testing organization is coming from the stakeholders and not the testing organization. And it all hinges on the accuracy and presentation of a measurement program.

QUESTION *As a manager, what challenges have you faced to keep close and accessible to your team(s)? What have you done to overcome these challenges?"*

***– Abigail Buell,** South Bend, Indiana*

MIKE LYLES I assume my challenges would be the same as any manager in similar environments. The more you are responsible for, the more meetings and emails you will be accountable to participate in. Being accessible is always a challenge but every minute spent with your key players is critical to the success of your team. I say "key players" because as a manager, your focus should be on building a team where you have your star players supporting you in areas that are the most critical to your deliverables and your company. Without this, you will spend all of your time in the details and the time you need for strategy, planning, and coaching the team will diminish. My response to this challenge is to first ensure those key players are in position to support the team with me. And the one thing you should always coach the team on is the information that is being shared with their peers and management. A VP once said to me during a status update "Mike, tell me what I NEED to know, not what you WANT me to know." I have never forgotten this advice, and when you and your team follow this mantra, you will find that meetings are more efficient, status updates are more concise and clear, and unexpected surprises will cease to exist.

QUESTION *I'm a QA Manager, wondering where to climb next. What does a QA Program Manager do and how often do testing skills actually come in to play in that role?*

***– Eric Jacobson,** Atlana, Georgia*

MIKE LYLES This seems to be common for most all professionals. Reaching a management position means preparing for executive management positions to move upward. And as we know, the number of positions decrease as you make the climb. Opportunities are so open, however, as a QA Manager. In my company, we are still growing as a QA team, and I'm fortunate to be part of many opportunities to build out something new and innovative for the organization. My suggestion to you would be to not wait until you take a step above QA Manager. Begin today looking for opportunities to be innovative. Look for areas where the organization could see a high ROI if a new way of thinking or process change could be instituted. Volunteer for things you are passionate about, and give every ounce of hard work and dedication to making it successful. Words are cheap... deeds are dear. Let your work speak for you, and you will be noticed for this in the end. Did I mention you should look for INNOVATIVE opportunities?

But I will stress again, if you don't have the appropriate skillsets and leadership in the Test Manager positions, you will find yourself drowning in the details and unable to function as a Program Manager. Team member selection is critical!

Quality

XBOSoft is a software quality services company dedicated to improving software quality. *It's all we do.* From our software QA consulting services to hands-on testing, our only objective is to increase the quality of your software.

XBOSOFT
Software Quality Improvement

QUESTION *As a leader, I find a lot of value and joy in encouraging team members to bring their individuality to the table. I think it's very important to rely on the unique strengths that each person contributes, rather than forcing people to "do as I say." How do you tap into the more personal side of the members of your team when managing such a large group of people? Is it possible?*

– Michele McCubbins, *Buchanan, Michigan*

MIKE LYLES It has been said that it's much easier to pull a string than push it. And I can't agree with you more on the value of coaching and mentoring your team. People join companies for the company and opportunity, and they most always leave one because of their boss and how they feel they are valued.

You have to strike the balance between folks who are capable and willing to take on strategic roles verses those who want to be given clear direction on the tactical deliverables. You need both of these personalities to survive today.

> **"We spend at least one third of our days working with our teams, and it's critical to ensure they are happy, sure of your expectations for them, working together collaboratively, and delivering to the expectations."**
>
> Mike LYLES

Nothing satisfies me more within the team than a team member approaching me to say they'd like to offer a suggestion for a better way to do something. And I was fortunate to work with one employee who brought me problems, but always brought me multiple solutions to them. I felt like I was handed a menu, and all I had to do was make the best selection.

Also, if you gain nothing else from my responses, the one advice I can give that I hope you take to heart is that being a manager does not mean you are always the mentor or coach. I have learned that you can learn a lot from your team, and I have had situations where I felt I was going to be the mentor, and to my surprise, I was the one that was mentored.

I like your note about tapping into the personal side of the team. We spend at least one third of our days working with our teams, and it's critical to ensure they are happy, sure of your expectations for them, working together collaboratively, and delivering to the expectations. I learned early on that being someone your team can depend on is very important. And making them feel important and valuable to the team is your only hope for survival. It has been said "people don't care how much you know until they know how much you care." If your team feels valued, respected, and important, they will move mountains for you. I take pride in making sure I talk to my team members about their feelings regarding the efforts we are responsible for, and even if it's a very short time due to the size of the team, I make an effort to spend time with each of them as much as possible. The challenge is finding the balance, and how you can work this in to your daily rush of emails, meetings, and emergencies. However, the more time you find to spend with your team and appreciate them for their hard work, the higher the quality and timeliness of their deliverables.

QUESTION *What aisle are the hammers in?*

– David Hoppe, *Grand Rapids, Michigan*

MIKE LYLES At least one question I know I will be 100% right on – that would be aisle 64 in Tool World. And my painful attention to detail forced me to call 5 stores in our company and ask them this very question to validate it was true. I'll be honest, I expected it to be different in various stores, but the resounding response from our helpful Lowe's store employees was "aisle 64."

Your question first made me laugh – thanks, I needed that – but then it made me think of something motivational I could say to close out this set of questions. Abraham Maslow said "It is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail." How many times have we, as testing organizations settled with a tool that we know is not the best choice? How many times have we known the best solution but decided we had what we needed to get the job done.

My challenge to each of you is to think differently. Look for ways to improve your organization. Technology changes so fast today – and when you see something that is successful for another organization, learn from this and look for opportunities to do the same with yours. Inform your management team of new ideas and ways of thinking, and keep yourself fresh with the latest tools, methodologies, and best practices. You don't have to start the next Apple, Google, or Facebook. You can change the world we live in one day at a time by reaching beyond your limits.

My very special thanks to Matt and STQA for allowing me to be part of this article. And I wish each of you the best in your future!

Connect to me on linkedin at **http://www.linkedin.com/in/mikewlyles**. Or on Twitter: **@mikelyles**

STP SOFTWARE TEST PROFESSIONALS

877.257.9531

Live Interactive Professional Development Without Ever Leaving Your Home or Office

STP Online Summits are live topic-specific events that are delivered entirely online for three and a half hours per day over three consecutive days. The speakers present their sessions live in real time, right on your computer screen. You can listen to and watch their presentations, as well as ask questions and get instant answers. You will also be able to network with your fellow participants through an STP Crew that will be facilitated by our summit host and panel of speakers.

Why Attend?

- Summit programs are created for testers, by testers.
- No travel required.
- Network and interact live with speakers and participants.
- Sessions are spread out over three days in order to minimize the impact on your daily schedule.

Date	Time	Topic
4/10/12 – 4/12/12	10:00AM – 1:30PM PT	**Test Management:** Bridging the Gap Between the Tests and the Stakeholders
6/5/12 – 6/7/12	10:00AM – 1:30PM PT	**Mobile Devices and Applications:** What's the Same, What's Different, and What That Means for Testing

For more information email info@softwaretestpro.com or call 877.257.9531

www.softwaretestpro.com/onlinesummit

THINK You're DONE?

CREATE A PLAN! YOU CAN'T RUN EVERYTHING ALL AT THE SAME TIME. IT PROVIDES THE STRUCTURE FOR GROUPING TESTS FOR EXECUTION.

You've completed your automation framework and script development work. *Now what are you going to do with it?*

byJim**HAZEN**

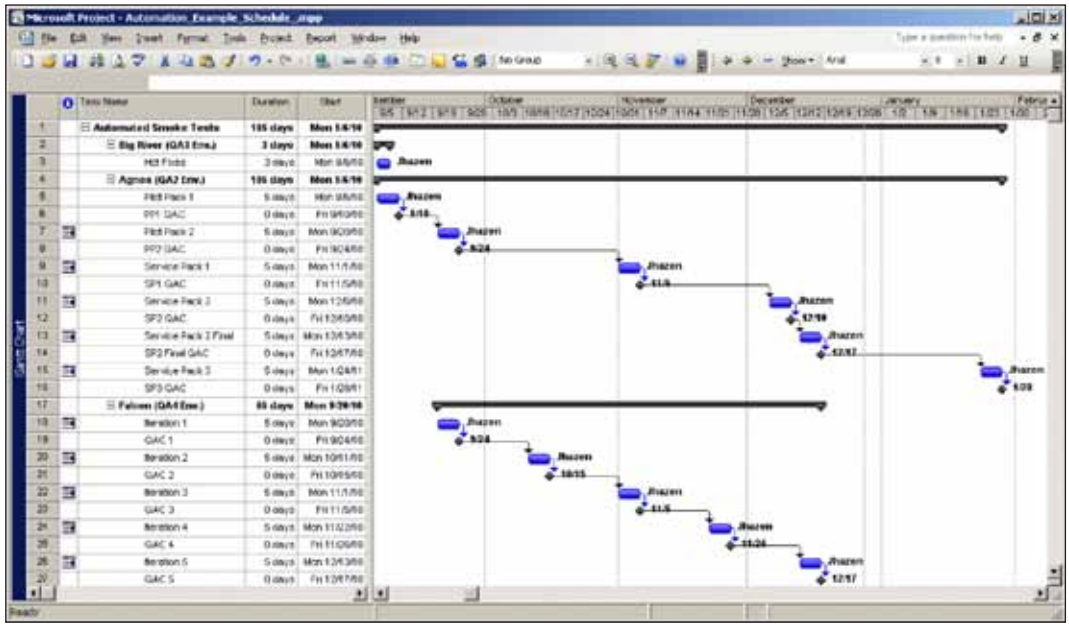
Do you have a plan to run it and report on it? Are you going to group your Scripts into suites by test type or prioritize them by functionality to cover? Do you have the capability to run scripts unattended? Can you distribute the workload to different machines and run them in parallel? Is your equipment and test environment ready to run? Is the data setup and a database ready to work with? Do you have a plan for maintaining the framework and scripts? Are your people trained on how to use and maintain it all?

Think you're done? Think again, because if you haven't taken care of all of these items, you've only completed half of the process of implementing automation.

Test Execution Plan

What is the Execution Plan? The game plan that explains the Who, What, When, Where and How for execution of tests. It can be a detailed document, outline or schedule. Or it can be a sketch on a whiteboard (with the "do not erase" warning and a digital picture of, it just in case). I like to use a simple MS Project schedule. For me MS Project is a quick and easy way to build a plan, and make it available to other teams.

Figure 1



Why create a plan? To better organize and manage the execution of tests. You need a road map because you can't run everything all at the same time. It provides the structure for grouping tests for execution.

Who uses the plan? The entire project team needs it as a communications tool, because you don't want to go 'dark'.

About The Author

Jim Hazen *has over 20 years of experience testing applications on the PC and web platforms. He has been involved with the startup of testing groups at multiple companies and has served as a consultant for the last 10 years.*

Group Tests

Why do we group tests into suites to be executed? To better manage the tests and organize them according to need and focus. This enables the team to determine what to run, in what order and when.

There are multiple criteria to use when grouping tests. For example, the types of tests, the functionality to be covered, and the interaction/dependency of the tests. On my projects I typically use the following criteria:

Common Test Types:

- **Smoke –** Tests that exercise the functionality 'across' (width) the system, with little or no depth. For example; startup, login, pop a few menu items/ dialogs. This checks readiness/ state of application for in-depth testing by both manual and automated means. Done as a first pass on a new build.
- **Regression (Critical Path) –** Tests that exercise the system using 80/20 rule (aka the Pareto Principle), 20% used 80% of the time. These are basic usage scenarios to validate necessary functionality is working correctly. They are done to ensure the application is usable and testable.
- **Regression (In-Depth) –** Tests that exercise the system in detail. These can be Equivalence Class/Boundary Value Analysis, Combinatorial, etc.
- **Business Scenario –** Tests that exercise the system as a user would to complete a business task.
- **Fault/Error Handling –** Negative condition tests to validate fault/error detection and correct handling.
- **End-to-End –** Tests that exercise the system as a whole. This tests the interaction of the system across the middle and back-end layers.

Functionality Coverage is grouping according to functionality and coverage. I find the types listed to be the most valuable:

- User Interface
- Reporting
- Database
- Middle and/or Back-end layer
- Interaction with other systems

Dependency Levels are used to determine if a test can be run by itself, priority of the tests, or if there are other pre-conditions needed for a test to run. I've found the following to be useful on my own automation projects:

- **None –** Test can be run independently; creates and cleans up data as needed, no dependency upon prior tests being run, no dependency upon configuration of system, no pre-conditions before execution.
- **Data –** Test can only be run with correct data or data condition present.
- **Environmental –** Test can only be run with correct configuration of system.
- **Previous Test –** Test can only be run if a prior test has been executed.
- **Combination –** Test is dependent upon a combination of data or environment or prior condition (due to previous test).

The benefits of grouping tests include improving the team's ability to classify, organize and execute tests according to focus/purpose of the testing effort. Grouping improves flexibility of test execution and the ability to estimate how long the tests will take to actually run. This leads to a more efficient use of the tests and allows for targeted test execution. It also improves the ability to understand "ripple effect" of changes/fixes in scripts, and reduces rework.

Unattended Execution

One of the original goals of automation is the unattended execution of tests. It typically gets lost in the process of creating the scripts themselves during a project. Why does this happen? There are many reasons, but for now let's focus on what it is and its benefits to the automation project.

Unattended Execution is the ability to execute a set of tests on a machine, or set of machines, without tester intervention or oversight. The machine does the work and the tester only needs to review/evaluate the results of the execution. Unattended execution allows testers to focus on the creation of testing efforts and eliminate running the scripts "by hand." It increases the efficiency of the execution of automated tests.

A methodology I learned, and still use is called "SEARCH." The acronym translates to:

Setup
Execute
Analyze
Report
Clean up
Help

First each test, or suite of tests, should **Setup** the conditions of the test. Setup includes test condition checks and tasks to ensure the system under test (SUT) is in the correct state to run the tests. Second, the test(s) are **Executed** against the SUT. Third, the tests should have the ability to **Analyze** (verify) the results of its actions as part of execution. Fourth, the tests should be able to **Report** on their progress and findings during execution. Fifth, tests should **Clean up** after themselves. Temporary data or files need to be deleted, or settings need to be reset if changed. Sixth and final point is **Help**, where the documentation of the system is maintained.

For me the sixth and final point instead of Help is **Home**. Tests should return to a common starting point (base state) so that the next test can run without problems. This is separate from the Cleanup step which emphasizes that the tests must return the SUT so other tests can execute without a lot of additional setup.

Using SEARCH along with grouping criteria can help in determining the run order. You can organize your automation so that it runs efficiently, and if there are dependencies you can order tests correctly to avoid problems. For example, when tests try to get to common data and collide.

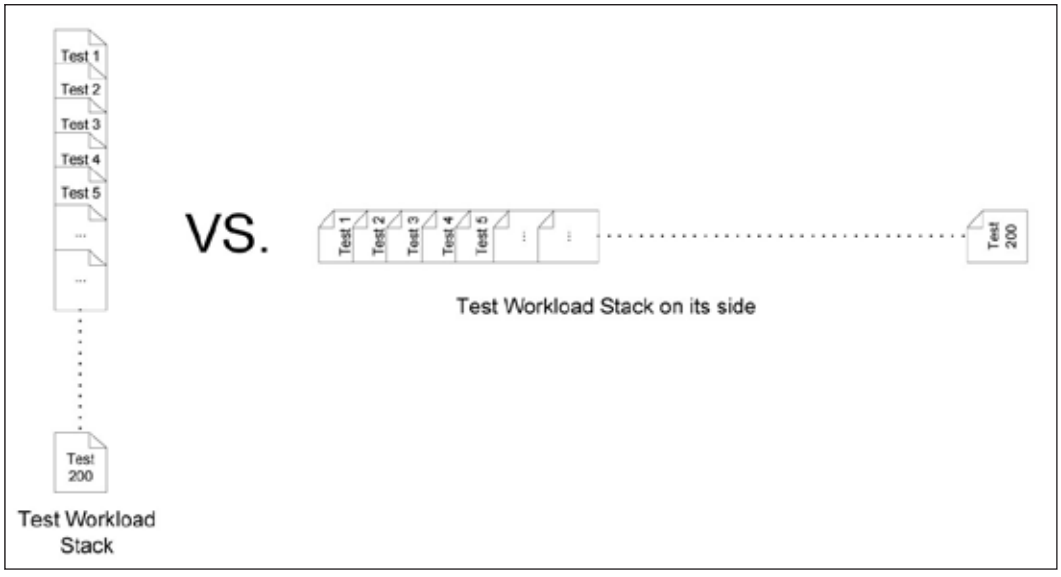
This can eventually lead to tests running as part of a Continuous Integration (CI) process, and running 24 hours a day with fewer tester resources needed to monitor it.

You'll get more "bang for the buck" by being able to run tests repeatedly in a consistent and reproducible way.

Distribute Workload

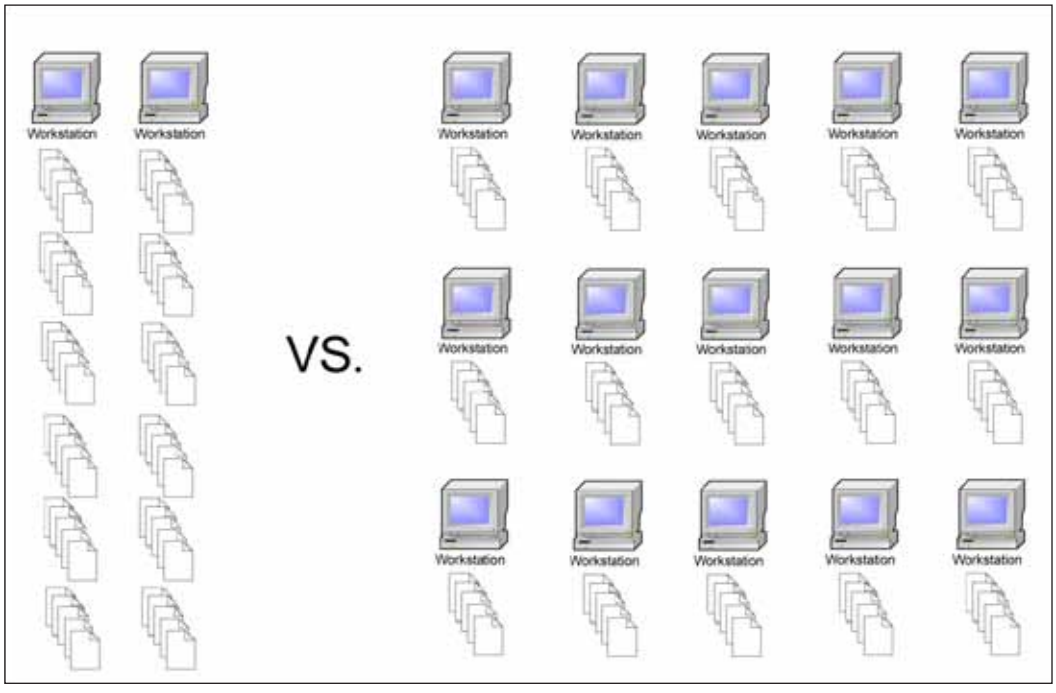
Now you have all your tests organized and ready to run unattended. You have this stack of tests to run, but are you going to run it all on one machine? How do you run it all? The best way to go is to use a distributed model. Take the workload stack and flip it on its side.

Figure 2



Divide up the workload and run tests using multiple machines.

Figure 3



This allows you to spread the workload (tests) out across multiple test execution machines and run them in parallel. You leverage the economy of scale and gain time efficiencies. You magnify the "illusion of speed" of automation.

You get more bang for the buck by being able to run automated tests repeatedly in a consistent and reproducible way.

Equipment and Test Environments

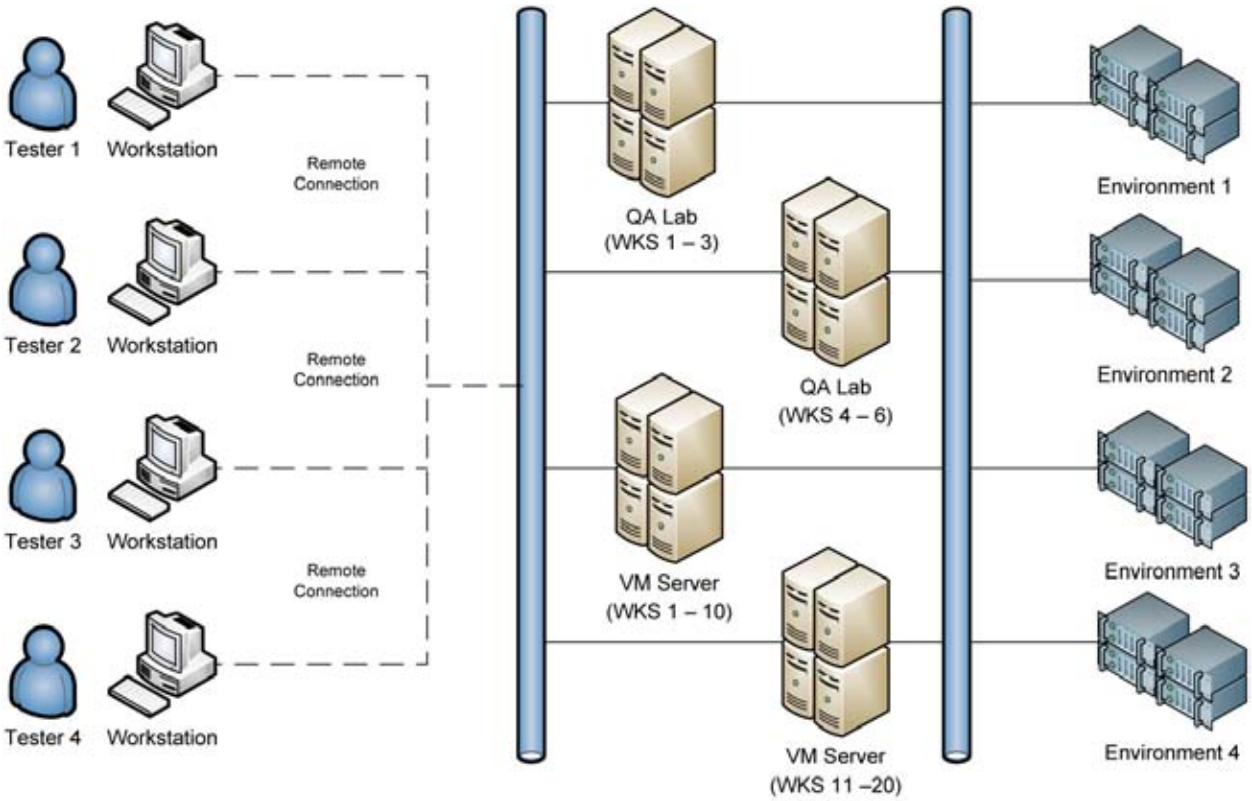
So far so good. The next step is to figure out where you're going to run the tests. What equipment do you have available? Should you use physical workstations or virtual machines? If you only need a few machines or a small budget for test machines then physical ones are best. You can reuse old systems, or buy inexpensive new ones. If you need a large farm of test machines it is better to go with Virtualization. Spend the money on a powerful server, setup a template for your test machine and run the automation suite(s) from there. In the long run virtualization gives you more flexibility for configurations and maintenance. It's also cheaper in situations where a large farm of test systems are needed and to be maintained. This can be done for both client systems and servers in your test environment.

Test environments need to replicate the production system as closely as is possible. If you're not able to replicate the production system, scale it down and run the tests accordingly. If needed, double duty your staging environment for automated and manual testing. Just be sure to setup test data that is only used by automation. This eliminates problems during execution because of data that may have been changed during a manual test.

If you can, separate the automation environment. As mentioned before, do this to avoid conflicts with development and other manual testers. This way you don't use the same data/accounts, or allow other people to change setup/configuration settings during test runs.

Now it is just a question of justifying the cost and getting buy-in. Machines are cheap, people are not. As mentioned before, leveraging more machines can spread the workload out for the automation. You can run things in parallel and get more bang for the buck. This is the illusion of speed. Also, you can run the test suites in parallel with manual testing and set it up to run off hours. You now are optimizing your work effort and becoming more efficient. That means savings for the company. This is a cost containment and savings method. Management will appreciate your efforts to contain costs.

Figure 4



Data and Databases

Automation needs the correct data to run successfully. Without the correct data your scripts will not run successfully. If possible, pull data from production and clean it for use. The test data needs to be mapped so you know which scripts depend upon it. You will have to mine for it. This takes time and effort. If you are using a shared a database you will need to separate the data used by automation and the data used by manual testing. You don't want to step on each other's toes.

Otherwise, the databases will need to be separated and a baseline backup made for automation. This way you can restore the database as needed to begin a new cycle of testing. A baseline backup will save a lot of time during a project if you do the upfront work needed to save you time at the end.

Utility Scripts

One thing that is often overlooked in automation projects is the use of tools to help with other tasks related to running the tests. Create a set of utility scripts. Use these scripts to setup and configure the system to some known base state. This can be done for both pre-test conditions and post-test restoration. For data and databases have scripts that create data, or mine for it. If test data is time sensitive have scripts that age it properly. For the test environment have scripts that cleanup or restore the test system itself. For example, have a script for any temp files created during a test run that need to be removed. Other things utility scripts can help with are FTP of files or data from other systems. These can be used for anything else outside of the execution of the tests themselves. This can add value to the use of the automation tool(s).

Maintenance of Framework and Scripts

Automation is a development project unto itself. Treat automation as such, just as you would have the developers maintain and archive the source code and components for the system itself. The framework and scripts ARE code, period. All of it has to be maintained to continue to receive the benefit of the original effort. Time and resources have to be pre-allocated to do this task. Plan maintenance as part of an iteration cycle, don't put it off. Be sure to update libraries, components, object definition/repositories and data sets. After making updates be sure to retest your automation code, don't be sloppy.

If you don't maintain the code, you run the risk of tests failing for the wrong reasons. Being lazy will cause your team to incur technical debt which you will never repay, and it will prevent you from recouping any of the original investment for the project. This is the first step to becoming shelf-ware, and the automation project failing. Don't waste the company's time and money; management will be reluctant to support future automation projects.

Remind management about the automation investment, and how maintenance is an important task to protect the investment. This is an asset to the company and we need to communicate it effectively to the management team.

Training of Staff

Hopefully the right people were hired at the start of the project. As part of the initial investment, training was done on the tool(s) and technologies the automation team was going to work with. We need to keep any new members of the automation team trained in the framework and the libraries. Also, additional in-depth training in the tool(s) is needed. Any additional training in the technologies being tested, along with programming and development methods can be beneficial to the automation team.

Other Testers and BA/SME's will need to be trained in how to use the scripts and data files to create usable tests. Make sure they are trained on how to use the SEARCH method effectively. Consistency in how tests are built will make maintaining it all easier later on. If everyone builds tests in different ways you will soon have a maintenance nightmare and again incur a technical debt that will kill the automation effort.

Also other groups, like Management, will need to be educated on what automation can and cannot do, and why. Set expectations properly and be sure people understand. Otherwise you will be setting yourself up for a world of hurt later on.

Conclusion

In conclusion; we've discussed, the Test Execution Plan and why it is needed. We discussed why grouping tests for better execution efficiency is a smart thing to do. We learned what the SEARCH method is and how it improves the usability of automation. In addition we discussed the benefits of Test Labs and using Virtualization to add efficiency, and why test data and databases for automation are important. We now understand why Utility Scripts are important for keeping automation running properly, and how vital maintenance is for the entire automation system. Last but not least we talked about the need for training staff, before during and after each project. The benefits of doing all these things is simple, it protects the investment in automation and helps to ensure the usefulness, maintainability and longevity of it. After all it's Automation, not Automagic.

REFERENCES

"How We Test Software At Microsoft," pg. 187, Alan Page et al., Microsoft Press, 2009.

"How to Automate Testing: The Big Picture," Keith Stobie & Mark Bergman, 1992.

"Virtualization: The Path to Multiple Efficiencies," Alan Page, http://www.hwtsam.com/star/Virtualization.pdf, STARWest 2009.

"Software Test Automation," Mark Fewster & Dorothy Graham, Addison-Wesley, 1999.

"Automated Software Testing," Elfriede Dustin et al., Addison-Wesley, 1999.

"Implementing Automated Software Testing," Elfriede Dustin et al., Addison-Wesley, 2009.

"Seven Steps to Test Automation Success," Bret Pettichord, http://www.io.com/~wazmo/papers/seven_steps.html, 2001.

"Success with Test Automation," Bret Pettichord, http://www.io.com/~wazmo/succpap.htm, 2001.

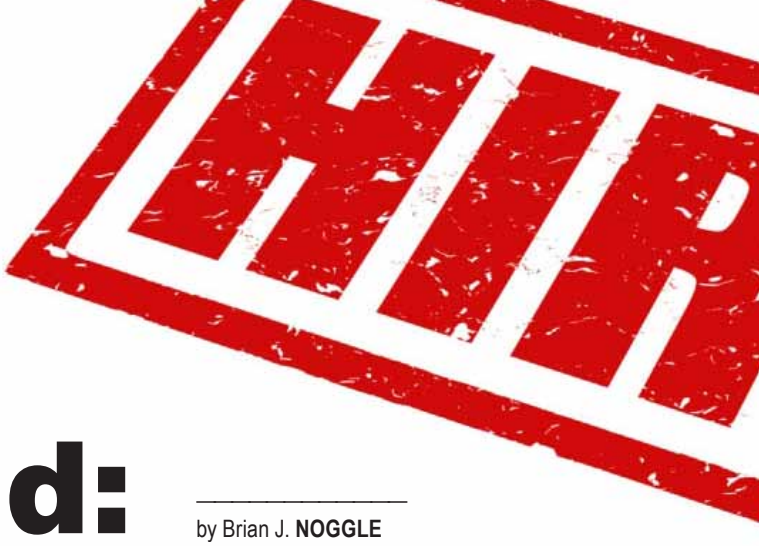
Test Studio

Easily record automated tests for your modern HTML5 apps

Test the reliability of your rich, interactive JavaScript apps with just a few clicks. Benefit from built-in translators for the new HTML5 controls, cross-browser support, JavaScript event handling, and codeless test automation of multimedia elements.

www.telerik.com/html5-testing

telerik



Cast Your Eyes Afield:

by Brian J. **NOGGLE**

Hiring Testers From Outside The IT Industry

When it comes time for you to write a job ad to fill a lower rung position in your quality assurance department, you might be tempted to merely copy the same old, same old boilerplate from other job postings and put this out there:

Job Requirements:

- Bachelor's Degree or minimum 5 years' experience testing software applications
- Experience with particular tools or graphical software, testing or otherwise.
- Excellent written and verbal communication skills
- Self-directed
- Solid, demonstrated knowledge of quality assurance and testing methodologies
- Experience writing, executing and maintaining test plans and test cases
- Experience with defect tracking tools and processes
- Experience working in an Agile Development environment
- Ability to write and maintain automated test scripts a plus
- Experience with *some testing package* would be a big plus

The human resources hive mind proffers some similar combination of job requirements for each junior level job, but confining your candidate search to existing members of the IT field might yield you junior tester candidates who are looking at this position as a stepping stone to their real goals—development or design positions. Perhaps you'll poach a couple of people making lateral moves because they're unhappy with their current equivalent positions.

Instead of recycling dissatisfied testers, you might consider hiring people from outside the IT world into your starting-level QA positions. In the 21st century, many people have a basic understanding of computer behavior compared to what you would have found twenty years ago. Now, almost everyone knows how to close windows and to launch programs. A lot of people know the rudiments of website pages and their behaviors as well, even if they don't understand the technologies behind them. Their experience as *software users* can readily serve the purpose of *relevant experience* in job requirements.

1. Advantages of Non-IT Testers

Although it might not seem obvious at first, hiring testers from outside the professional information technology world offers some distinct advantages.

First and foremost, people hired from outside of IT view websites and applications like real users do. They have not spent several years' worth of weekdays (and some weekends) plunking at keyboards running the same test cases over and over. Instead, they approach applications with fresh eyes, ready to identify things that might be obvious oversights or problems but that the organization has overlooked or ignored because it was looking at macro-level considerations. An absence of a forestry degree allows one to appreciate the individual trees.

Unseasoned candidates also lack experience in the bad habits of information technology workplaces. This can include some complacencies in the way things are done in your organization and in the industry as a whole. For example, someone not steeped in IT might not understand why a set of actions that crash the application is not a real problem. He or she might not know that nobody would do what he has just done. Or the new guy might not understand that it is customary to overlook mere misspellings.

When you hire some experienced testers, those testers can bring a wealth of experience about how they did things at the old place. In many cases, you can learn something from how things are done elsewhere, and this information can help improve your organization's process. However, sometimes an experienced tester can waste time trying to make your organization do things "the right way"—that is, the way the tester is already comfortable doing things. Sometimes people resist altering their habits, and the effort to retrain someone

ingrained in one way of doing things can equal the effort to train someone in the first place.

Inexperienced candidates are not Dryden's noble savages, but hiring someone from outside the IT industry to a junior position on your team does offer some possible advantages. The key, though, lies in the candidate. Although I cannot tell you the competencies and abilities of individual people, you might find some resume items could indicate a hidden aptitude for software testing.

2. Finding Outsiders

Not all non-IT candidates are the same. You might find that the resumes include experience that is not directly related to software testing but that can highlight a candidate amenable to the software testing way of thinking, concentration, a good attention span, tenacity, and attention to fine detail.

People Working in Other Precise Professions

You might not think that people who work in manufacturing and other hands-on, non-office jobs could have any skills your junior software tester needs. Think again. Many professional trades require a keen eye for detail and a quick appraisal of defective product. Printers, for example, can pull a sheet of paper from a conveyor belt and instantly identify the misalignment of elements by fractions of an inch, improper colors, and other flaws that they need to correct before the mistake is replicated—expensively—thousands of times. For that matter, your local copier operator from FedExKinkos should have the same eye for detail. Machinists and machine operators might not only know how to adhere to processes and procedures to improve quality, but if you talk about certain quality methodologies, such as Six Sigma and LEAN principles, they can relate.

People with Precise Hobbies

A large number of crafting hobbies require a precise eye and patience. Anyone who knits, paints lead figurines, weaves elaborate tapestries with beads, or builds china cabinets in the workshop already demonstrates a commitment to concentration and, quite possibly both patience to make something right and impatience to things that are imperfect. Crafters vary in skill of course, but most people who have practiced a hobby for any length of time have evolved some skill in it. Ask your inexperienced interviewees what sorts of hobbies they pursue, and delve into precision hobbies.

Former Military Servicemen and Servicewomen

Popular depictions of servicemen and women in film tend to center on two archetypes: psychos and Sergeant Bilko types. In reality, former members of the military are well-adjusted and, if they've served any length of time, they're not bumbling clowns like Bill Murray or Phil Silvers. They're professionals accustomed to process and procedure and to making quarters bounce off of bunks. They understand teamwork and cohesion. They've learned to adapt to conditions and situations outside of control or that deviate from the original plan and to survive those situations. While QA is not life-or-death, that spirit and mindset comes in handy. You might want that sort of five-by-five signal on your team. And if the developers fear that they'll make Tst. 2nd Class Reever lose it if they don't fix issue #1308, so much the better.

3. Conclusion

With our heads down and our attention focused squarely on IT problems and challenges, when it comes time to hire junior level positions for software testers, our immediate bias is toward the IT industry and candidates within it. In QA, though, we look at software applications with an eye to doing something different and unexpected to beneficial effect. When it comes time to look at job applications, a little of the unconventional might prove advantageous to the QA team and to the organization.

About The Author

Brian J. Noggle *has worked in quality assurance and technical writing for over a decade, working with a variety of software types and industries. He currently works as a freelance software testing consultant through his own company, Jeracor Group LLC.*