

BETTER™ SOFTWARE

A TECHWELL™ PUBLICATION

REDUCE PROJECT RISK
Apply test-driven
development to your
agile project

FROM GOOD TO GREAT
Become a great
ScrumMaster



AGILE
DEVELOPMENT
CONFERENCE
EAST

BETTER
SOFTWARE
CONFERENCE
EAST

DEVOPS
CONFERENCE
EAST



Register By
September 11
**AND SAVE UP
TO \$400**
Groups of 3+ save
even more

**3 Conferences
in 1 Location!**

CHOOSE FROM A FULL WEEK OF LEARNING, NETWORKING, AND MORE
SUNDAY Multi-day Training Classes begin
MONDAY-TUESDAY In-depth half- and full-day Tutorials
WEDNESDAY-THURSDAY Keynotes, Concurrent Sessions,
the Expo, Networking Events, and more
FRIDAY Agile Leadership Summit

ADC-BSC-EAST.TECHWELL.COM | [#BSCADC](https://twitter.com/BSCADC)

 Project
Management
Institute

PMI® members can
earn PDUs at this event

Keynotes

BY TOP-DOGS
IN THE INDUSTRY



The Care and Feeding of Feedback Cycles

Elisabeth Hendrickson, Pivotal



Continuous EVERYTHING: How Agile Is Changing Our World Forever

Jeffery Payne, Coveros, Inc.



Introducing the GROWS™ Method for Software Development

Andy Hunt, Pragmatic Bookshelf



Scaling Agile: A Guide for the Perplexed

Sanjiv Augustine, LiteSpeed



Innovation Thinking: Evolve and Expand Your Capabilities

Jennifer Bonine, tap|QA, Inc.

Principles and Practices of Lean Software Development

Ken Pugh, Net Objectives

Test Estimation in Practice

Rob Sabourin, AmiBug.com

Giving Great Presentations: The Art of Stage Presence

James Whittaker, Microsoft

Measurement and Metrics for Test Managers

Michael Sowers, Software Quality Engineering

Continuous Delivery: Rapid and Reliable Releases with DevOps

Bob Aiello, CM Best Practices Consulting

Coaching and Leading Agility: Tuning Agile Practices

David Hussman, DevJam



NOVEMBER 11-12

The Expo

Visit Top Industry Providers Offering the Latest in Software Development

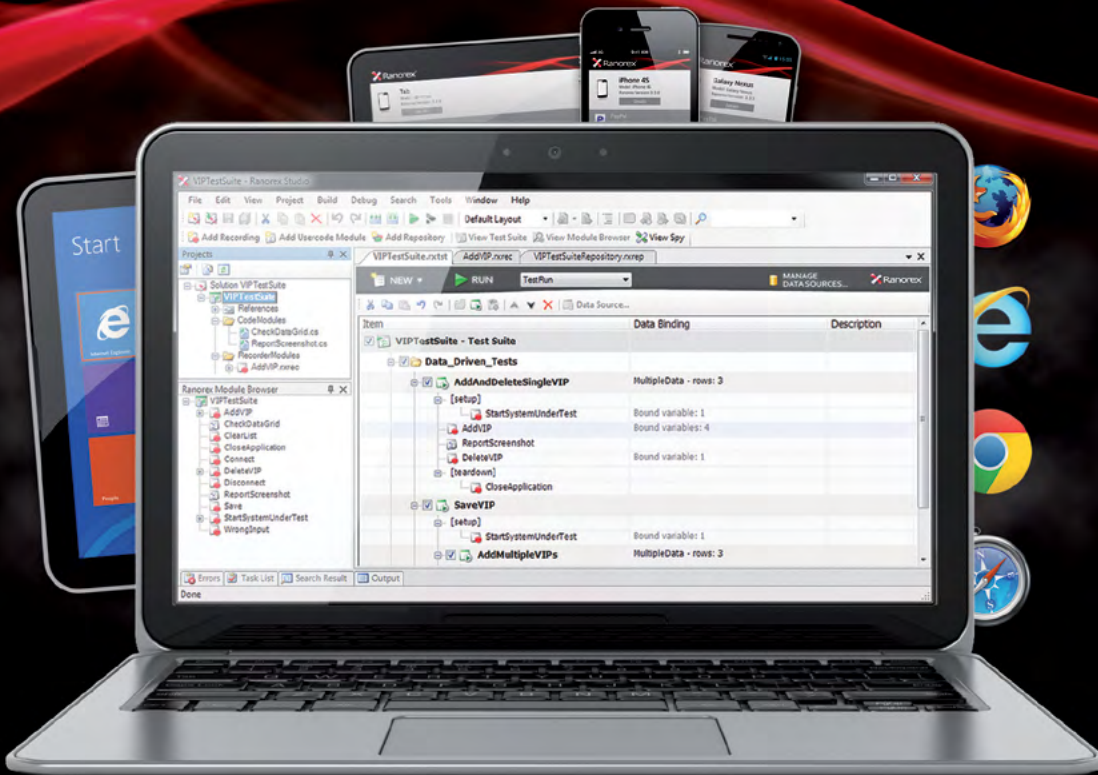
**TOOLS
SERVICES
TECHNIQUES
DEMOS**

Who Should Attend?


Software managers, directors, CTOs, and CIOs, project managers and leads, measurement and process, improvement specialists, requirements and business analysts, software architects, security engineers, test and QA managers, developers and engineers, technical project leaders, testers, process improvement staff, auditors, business managers

TO REGISTER CALL 888.268.8770 | ADC-BSC-EAST.TECHWELL.COM


Automated Testing of Desktop. Web. Mobile.



 Robust Automation

 Broad Acceptance

 Seamless Integration

 Quick ROI

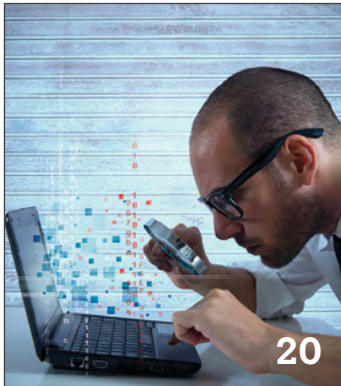


Why Use Ranorex
www.ranorex.com/why





CONTENTS



20



24

in every issue

Mark Your Calendar	4
Editor's Note	5
Contributors	6
Interview with an Expert	10
FAQ	35
Ad Index	37

Better Software magazine brings you the hands-on, knowledge-building information you need to run smarter projects and deliver better products that win in the marketplace and positively affect the bottom line. Subscribe today at BetterSoftware.com or call 904.278.0524.

features

This issue has two cover features presenting opposing views of the new ISO 29119 software quality standard.

12 COVER STORY SOFTWARE TESTERS SHOULD KNOW ABOUT ISO 29119

The ISO/IEC/IEEE 29119 has defined a set of requirements for testing software. As a member of the ISO working group, Jon Hagar wants you to know the basics and why testing teams should consider this recommendation.
by Jon Hagar

16 COVER STORY WHY ISO 29119 IS A FLAWED QUALITY STANDARD

Never afraid to voice his opinion, James Christie doesn't object to the adoption of any recommendations that improve software quality. He does, however, believe that ISO 29119 is fundamentally flawed.
by James Christie

20 APPLYING TEST-DRIVEN DEVELOPMENT TO AGILE

Test-driven development (TDD) is fundamental to agile, but to most of us caught up in projects, there's never enough time to commit to it. Erick Fleming shows how to use TDD to improve product quality and time to delivery.
by Erick Fleming

24 PLANNING TO PERFORMANCE TEST YOUR APP? THINK AGAIN!

To complement functional validation, software teams are expected to validate performance. But, according to Jun Zhuang, you must be prepared to invest time, personnel, and resources to benefit from performance testing.
by Jun Zhuang

30 BECOME A GREAT SCRUMMASTER

Performing all the functions required to facilitate project teams as ScrumMaster can be a task. Zuzi Sochova describes creative ways to become a master of Scrum by adopting a ScrumMaster state of mind.
by Zuzi Sochova

columns

7 TECHNICALLY SPEAKING

HOW TOUCH TIME IMPACTS DELIVERY

What does a developer do after a task is completed and testing takes over? Wait for testing results? Matt Heusser presents innovative techniques to keep everyone's pace going even after handoffs.
by Matt Heusser

36 THE LAST WORD

THE EVOLUTION OF TESTING CENTERS OF EXCELLENCE

According to Rajini Padmanaban, a testing center of excellence (TCoE) must be instituted for the establishment of any enterprise software development organization. Rajini presents the latest TCoE trends used by successful IT organizations.
by Rajini Padmanaban

MARK YOUR CALENDAR

training weeks

Testing Training Week

<http://www.sqetraining.com/trainingweek>

September 21–25, 2015

Washington, DC

October 19–23, 2015

Dallas, TX

November 2–6, 2015

San Francisco, CA

software tester certification

<http://www.sqetraining.com/certification>

September 22–24, 2015

Salt Lake City, UT

October 6–8, 2015

Portland, OR

October 13–15, 2015

New Orleans, LA

October 20–22, 2015

Philadelphia, PA

November 8–10, 2015

Orlando, FL

mobile application testing

<http://www.sqetraining.com/map>

September 27–28, 2015

Anaheim, CA

October 21–22, 2015

Dallas, TX

October 22–23, 2015

Chicago, IL

November 4–5, 2015

San Francisco, CA

conferences

STARWEST

<http://starwest.techwell.com>

September 27–October 2, 2015

Anaheim, CA

Disneyland Hotel

Pre-Conference Training (Anaheim)

<http://starwest.techwell.com/conf-training>

September 27–28, 2015 (2 days)

Agile Tester Certification

Fundamentals of Agile Certification—ICAgile

Mastering HP LoadRunner® for Performance Testing

Mobile Application Testing

September 27–29, 2015 (3 days)

Software Tester Certification—Foundation Level

Requirements-Based Testing Workshop

Real-World Software Testing with Microsoft Visual Studio®

Agile Dev, Better Software & DevOps Conference East

<http://bsceast.techwell.com>

<http://adceast.techwell.com>

<http://devopseast.techwell.com>

November 8–13, 2015

Orlando, FL

Hilton Orlando Lake Buena Vista

Pre-Conference Training (Orlando)

<http://adceast.techwell.com/precon-training>

November 8–9, 2015 (2 days)

Agile Tester Certification

Certified ScrumMaster® Training

Fundamentals of Agile Certification—ICAgile

Integrating Test with a DevOps Approach

Leading SAFe—SAFe Agilist Certification Training

Product Owner Certification

November 8–10, 2015 (3 days)

Software Tester Certification—Foundation Level



WHY QUALITY MATTERS

A sizeable percentage of *Better Software* subscribers are involved with software testing, and there's a software testing standard, ISO/IEC/IEEE 29119, that you all should know about.

Similar to the debates between political parties gearing up for the 2016 US presidential elections, our industry has seen a good deal of both positive and negative commentary on whether this standard even makes practical sense.

To get the debate going, our cover feature consists of two articles—a first for our magazine. Jon Hagar, a committee member involved with the creation of ISO 29119, presents a wonderful introduction to this standard and why the testing community should pay attention to it. James Christie gives a convincing perspective on the dangers of adopting any standard, but especially this one.

To continue our theme of software quality, Jun Zhuang puts together his thoughts on the real effort it takes to embrace performance testing in his article, "Planning to Performance Test Your App? Think Again!" Although I have been an advocate for upfront test development for years with my teams, I have a much better perspective of the *why* behind the benefits of upfront testing after reading Erick Fleming's "Applying Test-Driven Development to Agile."

To further the agile cause, Zuzi Sochova puts together a checklist of what she believes are the most important ingredients for project facilitators to get right with "Become a Great ScrumMaster." In addition to these insightful feature articles, we have several fantastic columns I think you'll enjoy reading.

We truly value your feedback. Let us and our authors know what you think of the articles by leaving your comments. I sincerely hope you enjoy reading this issue as much as I enjoyed working with our authors.

Ken Whitaker
kwhitaker@sqe.com
Twitter: @Software_Maniac

Publisher

TechWell Corporation

President/CEO

Wayne Middleton

Director of Publishing

Heather Shanholtzer

Editorial

Better Software Editor

Ken Whitaker

Online Editors

Josiah Renaudin

Beth Romanik

Production Coordinator

Donna Handforth

Design

Creative Director

Catherine J. Clinger

Advertising

Sales Consultants

Daryll Paiva

Kim Trott

Production Coordinator

Alex Dinney

Marketing

Marketing Manager

Cristy Bird

Marketing Assistant

Tessa Costa

FOLLOW US



CONTACT US

Editors: editors@bettersoftware.com

Subscriber Services:

info@bettersoftware.com

Phone: 904.278.0524, 888.268.8770

Fax: 904.278.4380

Address:

Better Software magazine
Software Quality Engineering, Inc.
340 Corporate Way, Suite 300
Orange Park, FL 32073

Contributors



ARLEN BANKSTON is an established leader in the application and evolution of agile software development processes such as Scrum, kanban, and Extreme Programming. He is a lean Six Sigma master black belt, certified ScrumMaster trainer, and certified Scrum product owner. Arlen has led the integration of interaction design and usability practices into agile methodologies, presenting and training frequently at industry conferences and to Fortune 100 clients. You can contact Arlen at arlen.bankston@lithespeed.com.



JAMES CHRISTIE is a self-employed testing consultant with thirty-two years of IT experience. Before moving into testing, James spent six years as an IT auditor. He has also worked in information security management, project management, business analysis, and development. He is particularly interested in links between testing, auditing, governance, and compliance. James spent fourteen years working for a large UK insurance company, then nine years with a big IT services supplier working with large clients in the UK and Finland. He has been self-employed for the past eight years. Please email James at james@clarotesting.com.



ERICK FLEMING (CSM, CSD, JCD, MCT, MSCD) has more than seventeen years of programming, project management, and training experience. Erick is an instructor for The Braintrust Consulting Group, a provider of complete project management solutions for a wide range of software related projects. Erick has spent most of his career training and consulting with organizations on technologies ranging from .NET and Java to Linux and other open source tools as well as specializing in cloud-based web solutions. Erick can be reached at erick.fleming@braintrustgroup.com.



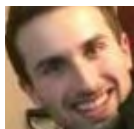
JON HAGAR has more than thirty-five years' experience as a software tester, thinker, and teacher supporting software product integrity, verification, and validation. Jon is the IEEE project editor for ISO/IEEE 29119 and co-chair for OMG UML testing profile. If that's not enough, he works on IEEE1012 V&V plans. Jon is the author of *Software Test Attacks to Break Mobile and Embedded Devices* and regularly consults, publishes, teaches, and mentors on quality assurance topics. Contact Jon at embedded@ecentral.com.



MATT HEUSSER, the managing consultant at Excelon Development, is probably best known for his writing. In addition to currently serving as managing editor of StickyMinds.com, Matt was the lead editor for *How to Reduce the Cost of Software Testing*. He has served both as a board member for the Association for Software Testing and as a part-time instructor in information systems for Calvin College. Matt can be contacted at matt@xndev.com.



As senior director of engagement, **RAJINI PADMANABAN** leads the engagement and relationship management for some of QA InfoTech's largest and most strategic accounts. She has more than twelve years of professional experience, primarily in the software quality assurance space. Rajini actively advocates software quality assurance through evangelistic activities including blogging on test trends, technologies, and best practices and providing insights on software testing to analyst firms such as Gartner and IDC. She can be reached at rajini.padmanaban@qainfotech.com.



A long time freelancer in the tech industry, **JOSIAH RENAUDIN** is now a web content producer and writer for TechWell, StickyMinds.com, and *Better Software* magazine. He also writes for popular video game journalism websites like GameSpot, IGN, and Paste Magazine, where he writes reviews, interviews, and long-form features. Josiah has been immersed in games since he was young, but more than anything, he enjoys covering the tech industry at large. Contact Josiah at jrenaudin@sqe.com.



ZUZANA (ZUZI) SOCHOVA has more than sixteen years of experience in the IT industry. As an independent agile coach and trainer, Zuzana specializes in organizational and team coaching, facilitations, and culture change using agile and Scrum practices at both startups and big corporations. She has experience with agile adoption in telco, finance, health care, automotive, mobile, and high-tech software companies all over the world. Zuzana can be reached at zuzana@sochova.com.



JUN ZHUANG has worked in all aspects of software testing since 2000. He has led and been involved in the design and build-out of various automation frameworks and the performance testing of applications across multiple industries. Jun currently works for Hobsons as a senior performance testing engineer. Please contact Jun at jun.jz.zhuang@gmail.com.

How Touch Time Impacts Delivery

Clever techniques like pair programming, continuous testing, and one-piece flow can maximize everyone's time on any schedule.

by **Matt Heusser** | matt@xndev.com

A small team is developing software, and on Monday, the programmer pulls a feature request. She completes the work in about an hour and kicks off a build. The software is ready by 10 A.M., but the tester can't start because he is busy working on something else. As a result, the just-completed work goes to the bottom of the stack. On Tuesday morning, the tester pulls the feature and begins testing, finding a serious bug within a half-hour. But the programmer can't make the fix because she is working on something else. Wednesday morning, she fixes the bug in about ten minutes.

You know how this story is going to play out. The tester can't work on the new fix until Thursday, but he has a question about the proper behavior, and the story is completed on Friday.

What could have been completed in four hours can drag out over a period of forty business hours. That represents a touch time of 10 percent. The concept of touch time is represented as a percentage: the amount of time a piece of work is actively being worked on divided by the amount of time it is in progress.

If you've never experienced a low touch time environment like this, you are more fortunate than most. The example above may seem a bit extreme, but once you consider the time wasted waiting for analysis or preparing work waiting to be programmed, touch time is often below 10 percent. If you include the associated time spent planning and waiting for handoffs to deployment, touch time can be as low as 1 percent.

Why Do We Do It?

Delays do happen when we develop software, and we want something to work on while we are waiting. So we pick up a second piece of work to complete, even if the first item becomes unblocked. As a result, touch time is reduced even further, and time to market unfortunately becomes longer. In the spirit of resource utilization, it seems horribly inefficient to let a computer professional sit around waiting for a task to become unblocked.

Yet that is exactly what happened in the example. If the two-person team were operating at maximum capacity, they could finish ten stories a week. If they multitasked to fill in time while waiting for work to be completed, how many features would get done? Fewer than ten; likely much fewer.

Single-Tasking and Partner Pairing

Taking a single-tasking approach with a team comprised of a developer and a tester, they can work on the same piece of work at the same time. Pairing resources may be viewed by executives as lazy or redundant. In this case, the two roles are doing different things: The programmer is writing code while the tester is looking for problems with that code. Put simply, the tester is testing and the programmer is programming. There is no redundancy.

Meanwhile, the tester is asking, "What about this?" "What about that?" "Why did you do this?" or "What happens if I put international characters in?" When the two switch to test mode, the programmer benefits by learning how the tester thinks. The programmer will remember the techniques the tester uses and will consider them as she codes, preventing categories of defects from

occurring. As a result, the test/fix/retest cycle will have fewer loops improving quality and time to market. The same phenomenon will naturally occur when the tester asks a question during programming that prevents an entire fix and retest cycle.

The benefits of pairing can really take place by not having to wait for a build while testing on the programmer's computer. This introduces the "it works on my machine" risk, but in my experience, when you track those problems down, they often involve something a pair partner would catch, such as hard coding files on the programmer's computer without putting them in version control. Pairing also limits the risks involved in a handoff when the tester is given work he does not understand or know what to do with. Unnecessary questions that usually cause delays are eliminated.

“In the spirit of resource utilization, it seems horribly inefficient to let a computer professional sit around waiting for a task to become unblocked.”

Achieving One-Piece Flow

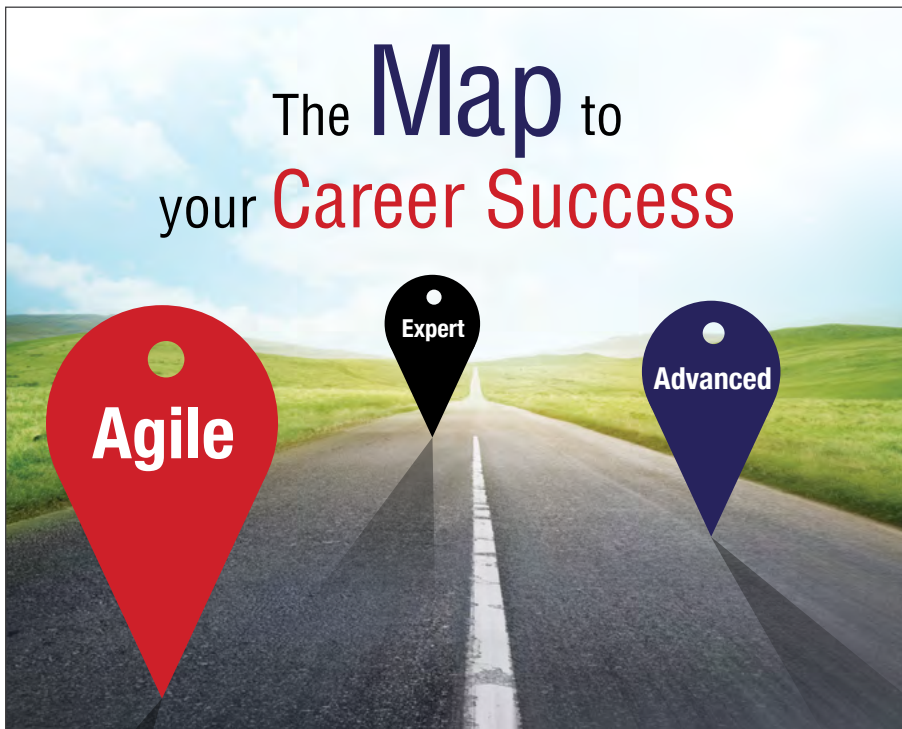
One-piece flow, another term for achieving near 100 percent touch time, occurs when the team improves the speed of delivery for each feature (or story), which results in faster overall project delivery.

Given the many disruptions attacking developers throughout the day (email, meetings, support calls, and short breaks), it is unlikely the organization will achieve one-piece flow. Even if it comes close, there are risks. For example, the testers might not

be comfortable with the programming language used by the developer and might tune out, creating risk. Some programmers won't like workplace disruption when testers ask questions.

The first step in moving toward one-piece flow is simple: Measure your touch time as a percentage of time the software is being actively worked on divided by total elapsed time. If touch time is low, make changes to increase it. A second possible measure is work in progress (WIP), which you can calculate with the techniques documented in the book *Why Limit WIP: We Are Drowning in Work*. [1]

For teams that have a low touch time, reducing wait states, getting rid of the piles of stuff on top of desks, and eliminating multitasking can dramatically improve overall individual and team performance. If your touch time is reasonable and you want to take things to the next level, consider an experiment with developer/tester pairing. **{end}**



Sticky
Notes

[Click here](#) to read more at StickyMinds.com.

■ References

Get to the next level in your software testing career.

ISTQB Software Tester Certification has agile, advanced and experts paths that can show you the way, giving you recognition that sets you apart.

If you are ready to take the next step in your career, choose your software testing career path now at www.astqb.org/map.

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

Genefa Murphy

Years in Industry: **10 in application delivery**

Email: genefa.murphy@hp.com

Interviewed by: **Josiah Renaudin**

Email: jrenaudin@sqe.com

“If you want to be successful in testing the real app's performance, it's not just looking at the core fundamentals, but also looking at network conditions and the context of how that user is going to use your mobile application.”

“We've seen a resurgence in the past few years around the value of user experience and what credibility people put around user experience.”

“Being able to simulate or virtualize those network conditions in the testing side of the house, before you go to production, that's the real key to making sure you're testing the full context of the app.”

“There are so many companies out there ... where their primary revenue stream, their primary interaction point with the customer is via a mobile application.”

“One of the biggest issues with mobile security is that it's still treated, in some respects, as an afterthought.”

“

With mobile, you have to test the three core pillars that we talk about. You have to test the functionality, you have to test the performance, and you have to test the security of the application.

“You don't want to wait until you're in production to use your customers as guinea pigs.”

“Today, we actually see a lot more customers starting to develop hybrid-based applications. The skills that they have in house tend to be more hybrid-based.”

”

For the full interview, visit <https://well.tc/IWAE17-4>

MOVE YOUR TESTING FORWARD


With 16 specialized courses, SQE Training's Testing Training Weeks give software testers and QA professionals the skills they need to boost productivity, improve testing procedures, and build better software. Create a customized week of instruction and maximize your training investment—the more training you take, the greater your savings.



2015 FALL SCHEDULE

TESTING TRAINING WEEKS

- September 21–25, 2015**
Washington, DC
- October 19–23, 2015**
Dallas, TX
- November 2–6, 2015**
San Francisco, CA

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
Software Tester Certification—Foundation Level			Mastering Test Design	
Security Testing for Test Professionals		Integrating Test with a DevOps Approach		DevOps Test Integration Workshop
Fundamentals of Agile Certification—ICAgile		Agile Tester Certification		Agile Test Automation—ICAgile
 Green background Indicates courses pre-approved for Project Management Institute PDUs.		Mobile Application Testing		Mobile Test Automation Workshop
Essential Test Planning and Management		Measurement & Metrics for Test Managers	Leadership for Test Managers	Test Improvement for Agile
Risk-Driven Software Testing		Performance Load and Stress Testing		

A photograph of two hands shaking against a teal background. The text 'Software Testers Should Know' is written in a large, red, stylized font with a blue shadow, following the curve of the hands. Below it, 'ABOUT ISO 29119' is written in a smaller, white, blocky font with a red shadow.

Software Testers Should Know

ABOUT ISO 29119

BY JON HAGAR

In 2013, after many years of development by international working groups, the International Organization for Standardization (ISO) and Institute of Electrical and Electronics Engineers (IEEE) approved and jointly published the first three parts of the ISO/IEC/IEEE 29119 software test standard [1]. The standard is segmented into five parts: test definitions and concepts, test processes, test documentation, test techniques, and keyword-driven testing. Part four (test techniques) should be approved within the next year, and part five (keyword-driven testing) should become available soon thereafter. For those unfamiliar with this standard, I'll present an overview of the new ISO 29119 and explain why I believe it is important for the software testing community.

A Little History

ISO/IEC/IEEE 29119 is part of a series of ISO and IEEE standards dealing with processes, concepts, and artifacts for engineering systems, software, and information technology. Each part of the test standard was developed through consensus by a working group of volunteer experts from more than twenty countries. The standards were refined and repeatedly voted on by national body members of the ISO and IEEE open balloting of volunteer groups.

I have participated in the production of numerous standards over the years—including as the IEEE project editor overseeing each part of the 29119 standard. The opinions and information presented in this article are my own and do not necessarily represent the positions of the ISO, IEEE, IEC, or other members of the working group and balloting groups who developed and approved this standard.

Why a Testing Standard Is Necessary

After thirty-five years in software and testing, I retired from full-time work to focus on numerous fronts. I'm now involved in speaking, writing, and supporting the software testing industry because I want to see it evolve to a higher standard of professionalism. When I started in software testing, there was almost no reference material. *The Art of Software Testing* [2] was just being published, and certainly no worldwide industry testing standards existed. I found then, and continue to find today, that many organizations desire a common baseline reference rather than a subjective, individualistic view of software testing.

My concept of a baseline reference comes from studying civil engineering before working with software. In land surveying, a baseline is a reference starting point. The early land survey baselines were often inaccurate, but they were the best available for activities such as defining a boundary for a person's property. Over time the methods have evolved, baselines have improved, and the definition of property lines changed. I look at ISO 29119 as an analogous baseline. Process standards in software may not be for every organization, but I have encountered companies, organizations, and even countries that seek a reference baseline supported by a standards body.

A standard is typically viewed as an idea used as a measure, norm, or model in comparative evaluations. That is slightly different from the ISO definition of "A standard is a document

that provides requirements, specifications, guidelines or characteristics that can be used consistently to ensure that materials, products, processes and services are fit for their purpose." [3] Both definitions allow the standard to be used as a reference point baseline. But for those organizations needing ISO requirements, where strict conformance is a goal, the standard supports this. Practitioners of ISO 29119 should keep the concepts of baseline, guidelines, context, and comparative evaluation in mind when starting to use it, especially in the relatively immature software test industry.

Today's standard is a starting point. Ideas and open subjects not yet defined or incorrect in the current ISO 29119 standard are subject to correction and improvement over time, based on test industry use and input. This is how the ISO and IEEE maintain and improve their standards.

If any standard does not properly address industry input, then over time it has to be changed in order to provide value to the community it serves. However, before judging the validity of the ISO 29199 standard, we should first read and consider its concepts and practices instead of forming opinions based on articles, conference proceedings, and blog postings.

The Five Parts of ISO 29119

ISO/IEC/IEEE 29119-1 SOFTWARE TESTING: CONCEPTS AND DEFINITIONS

This first part introduces test vocabulary and concepts. It does not contain requirement statements. It includes examples of testing roles within different lifecycle models. This is beneficial to a testing team that does not have definitions of common test concepts or terminology. I have found this useful for organizations that are looking at ways to improve internal testing or work with an outside test contractor.

ISO/IEC/IEEE 29119-2 SOFTWARE TESTING: TEST PROCESSES

Part two of the standard defines a generic software test process model, addressing the testing organization, test management, and test execution levels relying on a risk-based testing approach. Major software test management stages are defined, including test planning, monitoring and controlling, and completion. Test processes document the importance of test design and test implementation, environment, execution, and reporting. This part includes descriptions that can be used with contractual or regulatory documents. In ISO 29119, each test process is further described with an introduction, a purpose, outcomes, activities and tasks, and informational items. An appendix provides a partial example of a test design process.

When there is no existing test organization or test processes, a testing team should be able to use this standard to establish a minimal set of baseline processes. Another team lacking organization processes might use the standard to justify establishing a company software test policy statement. Finally, an established testing organization might use 29119-2 to perform an internal process improvement assessment.

ISO/IEC/IEEE 29119-3 SOFTWARE TESTING: TEST DOCUMENTATION

The third part of the standard aligns test documentation with test processes of part two. Figure 1 shows a summary of the documentation outlined and defined in part three. The standard provides examples of how this documentation can be used for traditional and agile projects. 29119-3 can be used by itself without other parts, and not all defined documents would need to be produced with tailored conformance.

This documentation standard not only works with an internal testing department, but also can be used to define the documents a testing service provider should be required to create.

ISO/IEC/IEEE 29119-4 SOFTWARE TESTING: TEST TECHNIQUES

This part provides definitions of common software test design techniques to aid in the development of test cases. The current list of test techniques was selected because they had historic usage and references; not every test technique in the literature is presented. Part four addresses structural white box and specification black box test techniques, as well as testing of quality characteristics, as shown in figure 2.

The steps necessary to derive test conditions, coverage items, and test cases define each technique. And for each test technique, support measures are defined that can be used to assess test coverage. I expect that this part of the standard will be used so that test techniques will be consistently applied and consistently measured.

In future revisions of the standard, new test techniques that are identified can be easily added.

ISO/IEC/IEEE 29119-5: KEYWORD-DRIVEN TESTING

Part five defines software tests using common structures, called keywords. Users of this part will create keyword-driven test specifications, corresponding frameworks, or test automation based on those keywords. Keyword-driven testing can be used when creating a tool that complies with an ISO/IEEE industry standard, as well as projects looking for a standard to drive keyword test cases.

- **Organizational test documentation**
 - Test policy
 - Test strategy
- **Project test documentation**
 - Project test plan
 - Test project completion report
- **Test level documentation**
 - Test plan
 - Test specification
 - Test results
 - Anomaly reports
 - Level test status report
 - Test environment report
 - Test level completion report

Figure 1: List of potential test documents

Applying ISO 29119 to Your Organization

In my experience, standards and models, such as the CMMI, waterfall lifecycle, or agile approaches, have to be tailored to the local software context. The same should be true for any organization considering using ISO 29119. In organizations not seeking complete conformance to the standard, clause two in each part of the standard defines how to perform conformance tailoring. As with any changes to conformance requirements, tailoring a standard assumes that you'll need stakeholder approval.

Critics to the standard have expressed concern that users will rigidly apply a mandatory standard and inhibit new testing ideas. Any technology or standard can be the subject of misuse, but this does not mean approaches should not be standardized. Ultimately, the marketplace will decide if ISO 29119 is a net positive gain toward improving the consistency and reliability of our industry. Many companies, organizations, and even countries are starting to use parts of this standard and finding it helpful. As with any standard, teams must carefully and thoughtfully craft their application of the standard rather than naively follow the full process set.

The ISO 29119 standard is not intended to be viewed as the state of the art or a best practice. Lack of quality testing standards is a potential issue for parts of our industry, and this standard captures existing processes, concepts, and methods that work for ISO/IEEE-supporting organizations. No standard can be a panacea, but the standard can provide a baseline reference for potential users with the following benefits:

- Companies need a common basis for conducting testing, internally or with other companies.
- Companies engaged in international business with language barriers need a unified standard (the standard has been translated into the languages of ISO members).
- Immature test organizations can treat the standard as a quick-start reference on software testing, including test policy, test management, test planning, and test case design.
- Regulatory organizations and governments need an ISO/IEEE industry-approved baseline.
- Companies and organizations performing process assessment or continuous improvement can use the stan-

Specification-Based Test Design Techniques (black box)	
	Equivalence Partitioning
	Classification Tree Method
	Boundary Value Analysis
	Syntax Testing
	Combinatorial Test Design Techniques
	Decision Table Testing
	Cause-Effect Graphing
	State Transition Testing
	Scenario Testing
	Random Testing
Structure-Based Test Design Techniques (white box)	
	Statement Testing
	Branch Testing
	Decision Testing
	Branch Condition Testing
	Branch Condition Combination Testing
	Modified Condition Decision Coverage (MCDC) Testing
	Data Flow Testing
Experience-Based Test Design Techniques	
	Error Guessing

Figure 2: List of potential test design techniques

dard as an accepted ISO/IEEE baseline.

- Testing tool vendors can use the standard as a baseline for testing software features and functions.
- Training and research organizations (professional and universities) can train a new generation of testers with an established baseline.

The Journey Continues

In a world where software quality must dramatically improve, I view the debate on the effectiveness of the ISO 29119 standard as necessary and as a potential source of improvement. As a contributor to the standard, I will do my best to ensure constructive comments from the test community are considered in future versions and parts under ISO and IEEE procedures for due process and consensus. The approaches and concepts of this standard represent input from members of the ISO and IEEE—organizations that have a history of producing standards. If the standard is found to be prescriptive and overly detailed in process, then this needs to be fed back into future revisions.

None of us recommending the use of standards can stop managers, companies, and countries from abusing standards—especially those who feel their individual expertise or creativity is being devalued. We can, however, encourage testers to build skills, learn about the standard, carefully consider it, and tailor it for their specific projects. The ultimate goal is to use a baseline standard that provides customer value. **{end}**

embedded@ecentral.com

**Sticky
Notes**

[Click here](#) to read more at StickyMinds.com.

■ References

NEWSLETTERS FOR EVERY NEED!

Want the latest and greatest content delivered to your inbox every week? We have a newsletter for you!

- **AgileConnection To Go** covers all things agile.
- **CMCrossroads To Go** is a weekly look at featured configuration management content.
- **DevOps To Go** delivers new and relevant DevOps content from CMCrossroads.
- **StickyMinds To Go** sends you a weekly listing of all the new testing articles added to StickyMinds.
- And, last but not least, **TechWell Insights** features the latest stories from conference speakers, SQE Training partners, and other industry voices.

Visit [StickyMinds.com](#), [AgileConnection.com](#), [CMCrossroads.com](#), or [TechWell.com](#) to sign up for our weekly newsletters.

A photograph of two hands shaking against a teal background. The text is overlaid on the hands. The top part of the text is in a bold, white, sans-serif font with a red drop shadow. The bottom part is in a large, red, cursive font with a white outline and a dark blue drop shadow.

WHY ISO 29119 IS A
Flawed Quality Standard

BY JAMES CHRISTIE

In August 2014, I gave a talk attacking ISO 29119 at the Association for Software Testing's conference in New York. [1] That gave me the reputation for being opposed to standards in general—and testing standards in particular. I do approve of standards, and I believe it's possible that we might have a worthwhile standard for testing. However, it won't be the fundamentally flawed ISO 29119.

Technical standards that make life easier for companies and consumers are a great idea. The benefit of standards is that they offer protection to vulnerable consumers or help practitioners behave well and achieve better outcomes. The trouble is that even if ISO 29119 aspires to do these things, it doesn't.

Principles, Standards, and Rules

The International Organization for Standardization (ISO) defines a standard as “a document that provides requirements, specifications, guidelines or characteristics that can be used consistently to ensure that materials, products, processes and services are fit for their purpose.” [2]

It might be possible to derive a useful software standard that fits this definition, but only if it focuses on guidelines, rather than requirements, specifications, or characteristics. According to ISO's definition, a standard doesn't have to be all those things. A testing standard that is instead framed as high-level guidelines would be consistent with the widespread view among regulatory theorists that standards are conceptually like high-level principles. Rules, in contrast, are detailed and specific. [3] One of ISO 29119's fundamental problems is that it is pitched at a level consistent with rules, which will undoubtedly tempt people to treat them as fixed rules.

Principles focus on outcomes rather than detailed processes or specific rules. This is how many professional bodies have defined standards. They often use the words principles and standards interchangeably. Others favor a more rules-based approach. If you adopt a detailed, rules-based approach, there is a danger of painting yourself into a corner; you have to try to specify exactly what is compliant and noncompliant. This creates huge opportunities for people to game the system, demonstrating creative compliance as they observe the letter of the law while trashing underlying quality principles. [4] Whether one follows a principles-based or a rules-based approach, regulators, lawyers, auditors, and investigators are likely to assume standards define what is acceptable.

As a result, there is a real danger that ISO 29119 could be viewed as the default set of rules for responsible software testing. People without direct experience in development or testing look for some form of reassurance about what constitutes responsible practice. They are likely to take ISO 29119 at face value as a definitive testing standard. The investigation into the HealthCare.gov website problems showed what can happen.

In its March 2015 report on the website's problems, the US Government Accountability Office checked the HealthCare.gov project for compliance with the IEEE 829 test documentation standard. [5] The agency didn't know anything about testing. They just wanted a benchmark. IEEE 829 was last revised in 2008; it said that the content of standards more than five years

old “do not wholly reflect the present state of the art.” [6] Few testers would disagree that IEEE 829 is now hopelessly out of date.

The obsolescence threshold for ISO 29119 has increased from five to ten years, presumably reflecting the lengthy process of creating and updating such cumbersome documents rather than the realities of testing. We surely don't want regulators checking testing for compliance against a detailed, outdated standard they don't understand.

Scary Lessons from the Social Sciences

If we step away from ISO 29119, and from software development, we can learn some thought-provoking lessons from the social sciences.

Prescriptive standards don't recognize how people apply knowledge in demanding jobs like testing. Scientist Michael Polanyi [7] and sociologist Harry Collins [8] have offered valuable insights into tacit knowledge, which is knowledge we possess and use but cannot articulate. Polanyi first introduced the concept, and Collins developed the idea, arguing that much valuable knowledge is cultural and will vary between different contexts and countries. Defining a detailed process as a standard for all testing excludes vital knowledge; people will respond by concentrating on the means, not the ends.

Donald Schön, a noted expert on how professionals learn and work, offered a related argument with reflection in action. [9, 10] He argued that creative professionals, such as software designers or architects, have an iterative approach to developing ideas—much of their knowledge is understood without being expressed. In other words, they can't turn all their knowledge into an explicit, written process. Instead, to gain access to what they know, they have to perform the creative act so that they can learn, reflect on what they've learned, and then apply this new knowledge. Following a detailed, prescriptive process stifles learning and innovation. This applies to all software development—both agile and traditional methods.

In 1914, Thorstein Veblen identified the problem of trained incapacity. [11] People who are trained in specific skills can lack the ability to adapt. Their response worked in the past, so they apply it regardless thereafter. Kenneth Burke built upon Veblen's work, arguing that trained incapacity means one's abilities become blinders. [12] People can focus on the means or the ends, not both. Their specific training makes them focus on the means. They don't even see what they're missing. This is goal displacement, and the dangers for software testing are obvious.

The problem of goal displacement was recognized before software development was even in its infancy. When humans specialize in organizations, they have a predictable tendency to see their particular skill as a hammer and every problem as a nail. Worse, they see their role as hitting the nail rather than building a product. Give test managers a detailed standard, and they'll start to see the job as following the standard, not testing.

In the 1990s, British academic David Wastell [13] studied software development shops that used structured methods, the dominant development technique at the time. Wastell found that developers used these highly detailed and prescriptive

methods in exactly the same way that infants use teddy bears and security blankets: to give them a sense of comfort and help them deal with stress. In other words, a developer's mindset betrayed that the method wasn't a way to build better software but rather a defense mechanism to alleviate stress and anxiety.

Wastell could find no empirical evidence, either from his own research at these companies or from a survey of the findings of other experts, that structured methods worked. In fact, the resulting systems were no better than the old ones, and they took much more time and money to develop. Managers became hooked on the technique (the standard) while losing sight of the true goal. Wastell concluded the following:

“Methodology becomes a fetish, a procedure used with pathological rigidity for its own sake, not as a means to an end. Used in this way, methodology provides a relief against anxiety; it insulates the practitioner from the risks and uncertainties of real engagement with people and problems.” [14]

Developers were delivering poorer results but defining that as the professional standard. Techniques that help managers cope with stress and anxiety but give an illusory, reassuring sense of control harm the end product. Developers and testers cope by focusing on technique, mastery of tools, or compliance with standards. In doing so they can feel that they are doing a good job, so long as they don't think about whether they are really working toward the true ends of the organization or the needs of the customer.

Standards Must Be Fit for Their Purpose

Is all this relevant to ISO 29119? We're still trying to do a difficult, stressful job, and in my experience, people will cling to prescriptive processes and standards that give the illusion of being in control. Standards have credibility and huge influence simply from their status as standards. If we must have standards, they should be relevant, credible, and framed in a way that is helpful to practitioners. Crucially, they must not mislead stakeholders and regulators who don't understand testing but who wield great influence and power.

The level of detail in ISO 29119 is a real concern. Any testing standard should be in the style favored by organizations like the Institute of Internal Auditors (IIA), whose principles-based professional standards cover the entire range of internal auditing but are only one-tenth as long as the three completed parts of ISO 29119. [15] The IIA's standards are light on detail but far more demanding in the outcomes required.

Standards must be clear about the purpose they serve if we are to ensure testing is fit for its purpose, to hark back to ISO's definition of a standard. In my opinion, this is where ISO 29119 falls down. The standard does not clarify the purpose of testing, only the mechanism—and that mechanism focuses on documentation, not true testing. It is this lack of purpose, the why, that leads to teams concentrating on standards compliance rather than delivering valuable information to stakeholders. This is a costly mistake. Standards should be clear about the outcomes and leave the means to the judgment of practitioners.

A good example of this problem is ISO 29119's test com-

pletion report, which is defined simply as a summary of the testing that was performed. The standard offers examples for traditional and agile projects. Both focus on the format, not the substance of the report. The examples give some metrics without context or explanation and provide no information or insight that would help stakeholders understand the product and the risk and make better decisions. Testers could comply with the standard without doing anything useful. In contrast, the IIA's standards say audit reports must be “accurate, objective, clear, concise, constructive, complete, and timely.” Each of these criteria is defined briefly in a way that makes the standard far more demanding and useful than ISO 29119, in far less space.

It's no good saying that ISO 29119 can be used sensibly and doesn't have to be abused. People are fallible and will misuse the standard. If we deny that fallibility, we deny the experience of software development, testing, and, indeed, human nature. As Jerry Weinberg said, “No matter how it looks at first, it's always a people problem.” [16]

Any prescriptive standard that focuses on compliance with highly detailed processes is doomed. Maybe you can buck the system, but you can't buck human nature. **{end}**

james@clarotesting.com

Sticky
Notes

[Click here](#) to read more at StickyMinds.com.

■ References

WANTED! A FEW GREAT WRITERS!

I am looking for authors interested in getting their thoughts published in *Better Software*, a leading online magazine focused in the software development/IT industry. If you are interested in writing articles on one of the following topics, please contact me directly:

- Testing
- Agile methodology
- Project and people management
- DevOps
- Configuration management

I'm looking forward to hearing from you!

Ken Whitaker

Editor, *Better Software* magazine

kwhitaker@sqa.com

The Best Place for Software Testing Solutions

Save up to \$400 when you register by March 4, 2016.

"The most effective training a QA manager can have."

Mike Chernick, QA Manager

"Wonderful first time experience."

Ken Malin, QA Test Analyst

"Great opportunities to learn new methodologies and better approaches."

Don Bloemer, Software Engineering Manager

<https://well.tc/GOSTAREAST>

STAR EAST

A TECHWELL EVENT

Orlando, FL
May 1-6, 2016

Renaissance Orlando
at Sea World®

APPLYING TEST-DRIVEN DEVELOPMENT TO AGILE

BY ERICK FLEMING



THINKSTOCKPHOTOS.COM

In the agile software development community, we value the delivery of working software through an incremental and iterative approach. Teams are encouraged to maintain close collaboration with project stakeholders in order to elicit feedback on their progress. This feedback is then incorporated into future iterations and, ideally, will increase the likelihood of producing the most valuable features.

However, as projects evolve, the code typically becomes increasingly more complicated and requires additional amounts of cognitive reasoning. This mental overhead can affect the team's ability to adapt to new requirements and can ultimately hinder the agile process. Complexity is an agile killer and should be mitigated before too much technical debt is accumulated.

As a point of fact, great results can be obtained simply through the use of core engineering practices and test-driven development (TDD).

Combating the Effects of Complexity

To combat the spiraling effects of an ever-expanding set of requested features, teams can adopt various engineering practices that help maintain a reasonable level of complexity while ensuring the necessary flexibility with regards to changing business needs. These practices include the use of modularization, design patterns, pair programming, continuous integration, and test-driven development.

Continuous integration can help strengthen the stability and confidence of a system that relies heavily on the presence of automated tests. Essentially, these tests provide constant verification of new and existing behavior that enables the team to experiment with alternative functional or technical implementations. Unfortunately, most teams struggle with adding a competent level of tests that voids many of the benefits gained from continuous integration.

In test-driven development, incremental and iterative programming concepts are married with the discipline of writing automated tests. If you look at any definition for TDD, you will find references to Kent Beck's *Test-Driven Development*. [1] In this influential work, he describes the practice of driving development with automated tests—with the ultimate goal of producing clean code that works—with two simple rules: "Write new code only if an automated test has failed" and "Eliminate duplication."

Starting the TDD Flow

In the TDD process, start with a particular story chosen from a prioritized list of backlog items that has been carefully crafted with appropriate information based on business value. With value in mind, the story should consist of a vertical slice of functionality, meaning that all components of the system are incorporated into the implementation. For a web application, this will typically include user interface elements and the associated server side functionality.

Let's look at an example story about password strength. A security expert stakeholder needs a feature that encourages users to choose an appropriately complex password in order to

help mitigate security breaches in the system (figure 1).

With additional conversations with stakeholders, the team

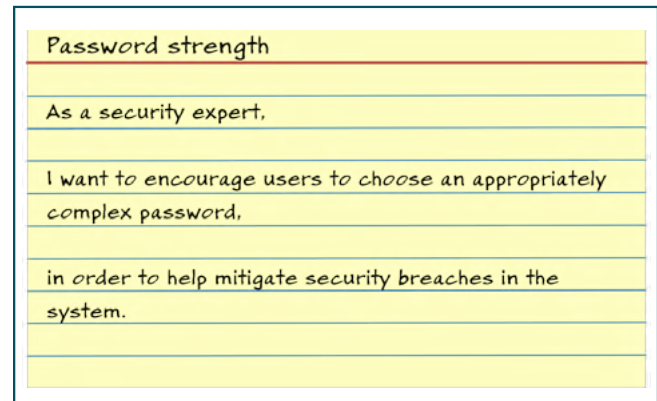


Figure 1: Initial password strength story card

has established a set of criteria to help complete the story forming the basis of the automated tests (figure 2).

In this example story's acceptance criteria, the use of imprecise

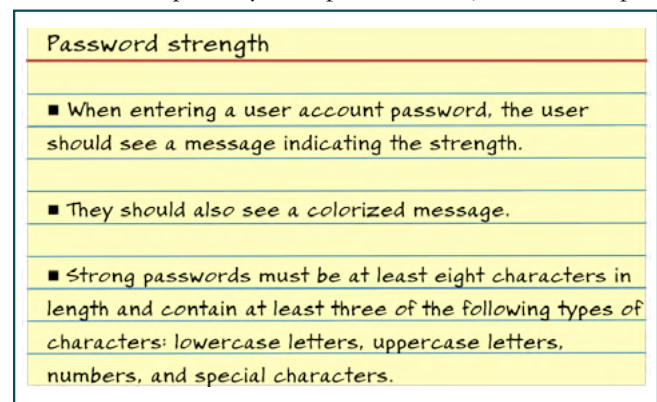


Figure 2: Completed password strength acceptance criteria

language is an attempt to vaguely document what is expected to validate. This is intentional and will become evident as an important concept when applying TDD to the agile process.

Test-Driven Functional Tests

The first bullet point on the story card in figure 2 describes an expected behavior from a user interface perspective and represents a good starting point for our overall story. This will be used to begin the test-driven approach.

In true test-driven fashion, a JavaScript test is written before implementing any production code, and because the scenario implies user interactions, an automation tool will be used to control a browser and to assert the expected outcome from the app's user interface (figure 3).

This test specifically opens a browser window, navigates to the appropriate section, enters a value for the password, and verifies that a particular message is displayed on the screen. Because it incorporates multiple parts of the overall story (a vertical slice), it is a functional (or end-to-end) test that is the primary driving force for the rest of our test code. Some people may recognize this approach as reminiscent of acceptance test-

driven development.

The test will initially fail due to the fact that none of the functionality in question has been implemented. The next step is to identify the minimal amount of code necessary to make the failing test pass to refactor or eliminate any code duplication. At this point, we could simply implement a shell of the overall functionality in order to get the test to pass. TDD encourages the simplest implementation, in contrast to adding code that may not be needed.

The second item of acceptance criteria in figure 2 is very similar to the first test case, but with a different expected outcome, verifying that the message output will have different colored font.

Test-Driven Unit Tests

To finalize the test case development for this sample story, the third criterion needs to be treated differently. While the test case in figure 3 was primarily focused on high-level interactions of the UI, isolated unit testing could be used to validate the third acceptance criterion.

```
describe( "password strength messages", function() {  
  
  it( "display a warning, when entering a weak password", function( done ) {  
  
    browser.url( "http://localhost:8080/new-user-registration.html" )  
      .setValue( "#password", "weak-password" )  
      .getText( "#strength-indicator-message", function( err, message ) {  
        assert.equal( "Your password is too weak!", message )  
      } )  
      .call( done );  
  } );  
});
```

Figure 3: User interface test written up front

For example, figure 4 shows a test case that uses an algorithm that verifies password strength, starting with the simplest test that describes an expected outcome and then drives the implementation from there.

```
describe( "password strength", function() {  
  
  it( "less than 8 characters are not strong", function( done ) {  
    var result = Passwords.isStrong( "short" )  
    assert.equal( result, false )  
  } );  
});
```

Figure 4: Test case that verifies password strength

This test uncovers the details of our initial implementation, such as establishing an appropriate naming convention for the function along with its inputs and expected outputs. It may seem trivial, but it's an intricate part of the test-driven mindset and critical for the overall flow of TDD.

Once the minimal code necessary has been implemented to make this test pass, move on to another scenario. Figure 5 shows testing for a negative condition and should be easy to add to the original test case.

Adding test coverage to an established baseline to reject weak passwords has the benefit of evolving the test code in

```
describe( "password strength", function() {  
  
  describe( "passwords with 8 characters", function() {  
  
    it( "letters, numbers and no specials are NOT strong", function( done ) {  
      var result = Passwords.isStrong( "1234567a" )  
      assert.equal( result, false )  
    } );  
  
  } );  
});
```

Figure 5: Test case enhanced to validate negative conditions

an incremental fashion. This approach can be pivotal in a developer's adoption of TDD, as it takes time to learn what sequence of tests is needed to drive an implementation. Figure 6 verifies the positive outcome of providing a password that is approved by the strength algorithm and should help drive the bulk of the implementation logic. When implementing the minimal amount of test code, existing test cases should be run to provide feedback on the effects of any new code.

```
describe( "password strength", function() {  
  
  describe( "passwords with 8 characters", function() {  
  
    it( "letters, numbers and specials are strong", function( done ) {  
      var result = Passwords.isStrong( "123456a!" )  
      assert.equal( result, true )  
    } );  
  
  } );  
});
```

Figure 6: Enhancing test case

With practice, an engineer can easily move between the test and implementation code with a natural cadence.

Conclusion

TDD is a disciplined technique for validating implementations, but it is also useful to incrementally discover the best implementations of a proposed system. This has amazing benefits:

- Encourages radical simplification of the code
- Allows the design to progressively evolve as business requirements change
- Encourages developers to remain focused on implementing what is needed to make a test pass
- Provides a significant level of test coverage to help maintain a healthy confidence in the system

Incorporating TDD into your software development process is a mechanism for increasing developer confidence around code changes, and, therefore, it can encourage experimentation even in the wake of a growing code base. **{end}**

erick.fleming@braintrustgroup.com



Click here to read more at StickyMinds.com.

■ References



ATTEND LIVE, INSTRUCTOR-LED CLASSES VIA YOUR COMPUTER.

Live Virtual Courses:

- » Agile Tester Certification
- » Fundamentals of Agile Certification—ICAgile
- » Testing Under Pressure
- » Performance, Load, and Stress Testing
- » Get Requirements Right the First Time
- » Essential Test Management and Planning
- » Finding Ambiguities in Requirements
- » Mastering Test Automation
- » Agile Test Automation—ICAgile
- » Generating Great Testing Ideas
- » Configuration Management Best Practices
- » Mobile Application Testing
- » and More



Convenient, Cost Effective Training by Industry Experts

Live Virtual Package Includes:

- **Easy course access:** You attend training right from your computer, and communication is handled by a phone conference bridge utilizing Cisco's WebEx technology. That means you can access your training course quickly and easily and participate freely.
- **Live, expert instruction:** See and hear your instructor presenting the course materials and answering your questions in real-time.
- **Valuable course materials:** Our live virtual training uses the same valuable course materials as our classroom training. Students will have direct access to the course materials.
- **Hands-on exercises:** An essential component to any learning experience is applying what you have learned. Using the latest technology, your instructor can provide students with hands-on exercises, group activities, and breakout sessions.
- **Real-time communication:** Communicate real-time directly with the instructor. Ask questions, provide comments, and participate in the class discussions.
- **Peer interaction:** Networking with peers has always been a valuable part of any classroom training. Live virtual training gives you the opportunity to interact with and learn from the other attendees during breakout sessions, course lecture, and Q&A.
- **Convenient schedule:** Course instruction is divided into modules no longer than three hours per day. This schedule makes it easy for you to get the training you need without taking days out of the office and setting aside projects.
- **Small class size:** Live virtual courses are limited to the same small class sizes as our instructor-led training. This provides you with the opportunity for personal interaction with the instructor.



***Planning to Performance
Test Your App?
Think Again!***

by Jun Zhuang



Severe performance issues in the production environment could have a devastating impact on a company's revenue, as well as on its image. According to a study by Strangeloop Networks, a site that loads in three seconds experiences 22 percent fewer page views, a 50 percent higher bounce rate, and 22 percent fewer conversions than a site that loads in less than one second. [1]

However, there is more to performance testing than just hiring someone who claims to be a performance test expert. If you expect quick return on your investment, you may be in for a rude awakening. There are many aspects you might want to consider before committing to performance testing:

- You have to make so many decisions that it will make your head spin
- Good performance test engineers are in short supply
- Performance testing is expensive—very expensive
- Adopting performance testing may cause tension among your employees
- The return on investment is hard to measure

Decisions, Decisions, Decisions

Figure 1 shows a high-level decision-making processes that should be taken into account when planning for performance testing.

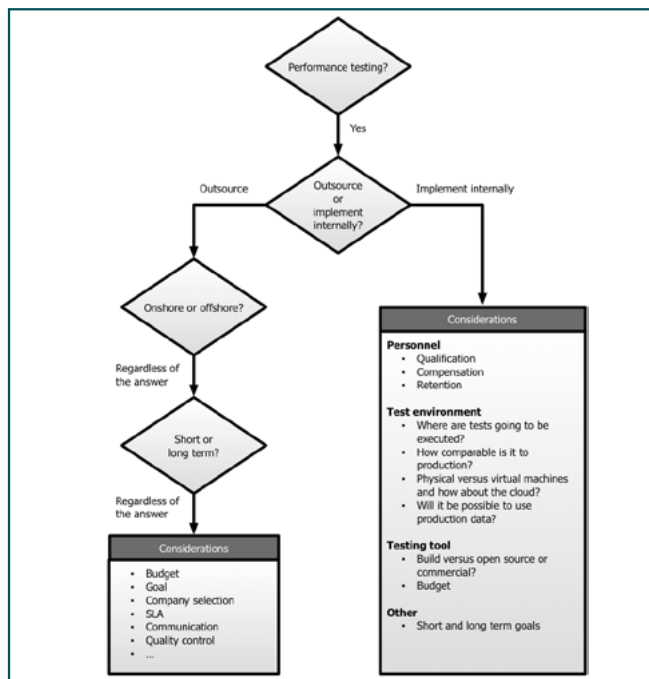


Figure 1: Performance testing decision process

Your first decision is whether to outsource performance testing. If your organization doesn't have the capacity or expertise, outsourcing may be the best choice. However, outsourcing can carry significant risks. For example, the actual delivery quality is not always as high as promised.

Outsourcing can be onshore or offshore, with contracts ranging from short term to long term. Regardless, outsourcing

is never completely hands off. Be prepared to manage an investment of your and your team's time. In the long run, there are many benefits to implementing performance testing with internal resources, some of which include better integration with other teams involved in the SDLC, better quality control, and potentially more cost-effective.

Details regarding outsourcing considerations are out of the scope of this article. Instead, I will focus more on the internal implementation.

Finding a Good Performance Test Engineer Isn't Easy

Taking advice from the saying "Well begun is half done," you need an experienced, skilled performance engineer to jump-start a performance testing initiative. You should expect him to lay a solid foundation and establish the necessary process for the practice to be successful from the very beginning.

A good performance engineer should be experienced with all facets of performance work, including requirements gathering, test planning, test design, scripting, test execution, problem-solving, and performance analysis. This person must have a firm grasp on the overall process and broad knowledge of tools, protocols, infrastructure, and programming and scripting languages.

The bad news is that good performance test engineers are in short supply.

One of the companies I worked for had an opening for a senior performance engineer. We interviewed candidate after candidate but could not find one who was up to our standards. Some candidates claimed they had done performance testing before, but what they really did was run scripts created by others. Some might have worked under the supervision of someone who was more experienced and knew part of the process but not everything. Keep in mind that a résumé does not necessarily reflect one's true capability.

Now, let's forget this challenge for a moment and think about the time and money you have to spend on listing the position, paying a headhunter, searching, screening, interviewing, and bringing the new person onboard. Do you still want to do it?

Performance Testing Can Be Expensive

There are several costs associated with performance testing.

PERSONNEL COMPENSATION AND RETENTION

Congratulations for finding someone you think can do the job. Now, how much are you willing to compensate her? Fair market value of an experienced performance engineer should be no less than a good software developer.

If this person turns out to be a huge benefit to your organization, you must do everything you can to keep her; otherwise, you will have to go back to the market and find a replacement. It's going to be more costly this time because when an employee exits, she takes with her the knowledge she gained about the system, process, and infrastructure during her tenure with your company. To make matters worse, critical project delivery could be jeopardized.

One of my friends started the performance testing practice for a company, then left to pursue a better opportunity. More than a year later, I happened to learn that the company was still looking for his replacement. When this takes place, a company may decide to drop the performance testing position and won't reconsider hiring a performance engineer again until performance issues erupt in production, which could be very costly.

TESTING ENVIRONMENT

To avoid disrupting the normal production operation, there should be a dedicated environment that is identical to production to run performance tests against. Identical to production generally means the test environment uses the same hardware, configuration, and software.

This ideal environment is often too costly or impractical to build. As an alternative, a possible compromise is to create a test environment that is as similar as possible to production. For example, you could use a system with half or a third or a quarter of the production's capacity, but except for capacity, you must make sure this system is identical in all other aspects. The more the testing environment deviates from production, the more difficult it is to use performance testing results to forecast actual production performance or to replicate production issues.

You also need machines to drive the desired load and run other software to gather the performance data when running tests. The good news is that these should be available at a much lower cost.

Don't forget about ongoing costs for continuous operation. The performance test environment needs to be updated and maintained with any production changes, including hardware updates, configuration changes, and software version upgrades and updates.

TESTING AND MONITORING TOOLS

Now that you have a performance test engineer on board and a performance test environment, you'll need to acquire the necessary tools to test and gather performance measurements. Most performance testing tools have basic monitoring capability built in, but sometimes you might need another one in order to gain more in-depth information regarding what is taking place within your system. Generally speaking, you have three options: build a tool yourself, pick an open source tool, or purchase a commercial tool.

Unless you want to enter the software performance testing market or have an enormous testing budget, the build-it-yourself option could be the most expensive option, and success cannot be guaranteed.

The second option is to use open source tools, and there are some good ones readily available. But there is a catch: If you run into issues, there is usually no one to call for support. However, you can get answers or advice from community support most of the time.

The third option is to purchase a commercial tool. This usually includes technical assistance, though it does not necessarily mean you will receive quality and timely customer support. My word of caution is to not acquire a tool based on a salesperson's

pitch or from viewing a demo. The tool may be able to accomplish what you want to achieve, but there might be a less expensive one, too. Over the years, I have known companies that have bought performance tools only to let them sit on shelves collecting dust due to the fact that no one knew how to use them properly.

Tool selection is typically the performance engineer's choice, i.e., he recommends a tool he is comfortable with using as long as it's suitable for the job. Of course, your company may have other considerations in tool selection, such as cost or standard tools already in use. It is always a good idea to do some comprehensive evaluations to decide on a tool that is the cheapest but still able to satisfy your short-term and long-term testing needs.

Performance Testing Can Cause Tension

Now that you have the primary resources in place, can testing start? It depends. I guess a better question to ask is: Is your organization ready? Your company's readiness can be assessed from the following aspects:

- Do you have performance requirements?
- Do developers have something in mind (like a database query that can use some performance tuning)?
- What's your short-term and long-term plan regarding performance testing?
- Have you communicated to developers that they need to carve out time to help the performance test engineer when needed?

Software performance is the responsibility of the entire company, and sometimes, it requires a companywide culture change. This can cause tension among employees if not handled properly. If your developers are not accustomed to following coding standards, unit testing their own code, or performing code reviews, they may be in for a shock. I have worked with developers who want to do nothing but write code; their answer to solving performance issues in their code is to purchase better hardware.

Performance testing and analysis require a team commitment, and your entire team must be willing to make adjustments based on performance testing results.

Return on Investment Can Be Difficult to Measure

The return on performance testing investment can be difficult to measure. Every now and then performance testing reveals a severe problem, but most of the time, it's going to be eventless—especially if your developers proactively unit test their own code with the intent to find performance bottlenecks up front.

If you do not experience severe performance issues in your production environment anymore, my suggestion is to assume that your performance test engineer has done what he is expected to do, and you should continue to give him your support.

In Conclusion

In this age of computing, nobody has the patience to wait for a page to load. In addition to functional product defects,

bad performance can easily contribute to losing customers to a competitor. Humiliation and lost revenue aside, your company's survival could be on the line.

Just because a product functionally works doesn't mean you don't have serious performance problems lurking. Performance testing should still be considered. Truly embracing performance testing takes planning, buy-in, and commitment, but it is worth the investment. **{end}**

jun.jz.zhuang@gmail.com

Sticky
Notes

[Click here](#) to read more at StickyMinds.com.

■ References



CURIOUS ABOUT DEVOPS? START HERE!

We can trace the beginnings of the DevOps movement to a Belgian named Patrick Dubois. In 2007, he lamented that the two worlds of development and operations seemed miles apart. He observed that development teams and operations teams tended to fall into different parts of a company's organizational structure (usually with different managers and competing corporate politics) and often worked at different geographic locations, causing conflicts and miscommunication.

Suspecting there were others who shared his frustration, Dubois took to Twitter in 2009 to discuss bridging the gap between development and operations. People responded, and DevOps was born as a grassroots initiative for brainstorming solutions. Since then, the DevOps movement has gained global momentum and become a lightning rod for people who have something to say about how IT is—or *should be*—running.

The approach found further traction online through social media and discussion boards. DevOps is clearly touching a nerve within the industry—likely because it was created for practitioners, by practitioners; it's not a product, specification, or job title. DevOps is an experience-based movement about cooperation and collaboration.

In response to the call for more information and resources about DevOps, TechWell is introducing the inaugural [DevOps Conference East](#) from November 12-13 at Hilton Orlando Lake Buena Vista in Orlando, Florida. [DevOps Conference East](#) will accompany the fifth annual collocated Agile Development & Better Software East conferences, the premier events for software professionals. This year's program is even more robust, bringing all aspects of the software development lifecycle to the forefront.

[DevOps Conference East](#) features industry practitioners passionate about the DevOps movement and focuses on topics like:

Continuous Delivery: Rapid and Reliable Releases with DevOps—Bob Aiello explains how to implement DevOps using industry standards and frameworks in both agile and nonagile environments, focusing on automated deployment frameworks that quickly deliver value to the business.

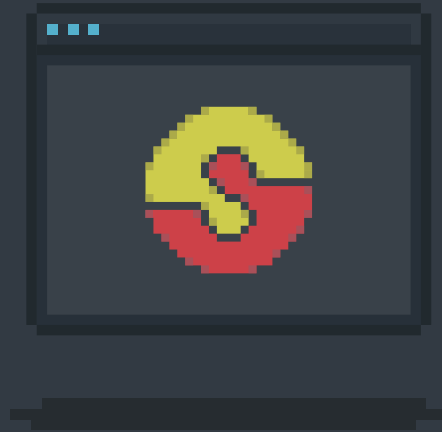
Agile and DevOps Transformation in Large Organizations—Siraj Berhan explores common challenges—people, processes, technology, and operations—in the agile journey of large-scale organizations. Siraj explores a project suitability assessment tool for evaluating as well as mitigating risks specific to agile delivery, incorporating a time-and-material funding model, and maintaining a cross-functional self-managing team with a generalist-specialist attitude.

Rethinking Test Automation in a DevOps Era—Kalyana Konda shares insights about how test teams can contribute to delivering high-quality applications by becoming an upstream quality co-creator rather than a downstream quality validator, thus implementing a continuous feedback loop to respond to customer needs and concerns. Kalyana discusses how to create a DevOps-friendly test automation strategy to ensure you overcome bottlenecks and reap the benefits of a successful DevOps implementation.

We hope your DevOps journey brings you straight to [DevOps Conference East](#). See you in November!



MARTIAL ARTS
HAS BRUCE LEE.



AUTOMATED TESTING
HAS SAUCE LABS.

Maybe you can't do a one-fingered push-up, but you can master speed and scale with Sauce Labs. Optimized for the continuous integration and delivery workflows of today and tomorrow, our reliable, secure cloud enables you to run your builds in parallel, so you can get to market faster without sacrificing coverage.

Try it for free at saucelabs.com and see why these companies trust Sauce Labs.



YAHOO!

PayPal

mozilla

VISA



 SAUCE LABS

Be On the Lookout!

2 Conferences
1 Location

Back in 2016!

Mobile Dev + Test

A TECHWELL EVENT

The Mobile Dev + Test Conference is jam-packed with keynotes from top industry thought-leaders, full- and half-day tutorials, concurrent sessions on hot mobile topics, an Expo that is full of industry solution providers in the mobile space, and a full-day Mobile Innovation and Leadership Summit. Join us to find out where the future of smart and mobile software is headed.

- Mobile Development for iOS and Android
- Cross Platform Mobile Development
- Mobile App Testing
- Mobile Performance Testing
- User Experience (UX) Design
- Mobile App Security

NEW for 2016!

IoT Dev + Test

A TECHWELL EVENT

The challenges of developing and testing in the new digital frontier of Internet of Things are sometimes unique. The newly-added IoT Dev + Test conference focuses specifically on developing and testing for IoT and embedded systems. You will have the opportunity to learn from experts in the field where the future of IoT is headed and how to position yourself to be a part of this revolution.

- Smart Home and Office Apps
- Embedded Systems Development
- Wearable Technology
- Embedded System Security
- IoT and DevOps
- IoT Testig and Performance

April 17-22, 2016

San Diego, CA | Westin San Diego

Mobile-IoT-DevTest.TechWell.com





The role of a ScrumMaster is one of the most difficult to explain and understand, yet it is critical since it is my belief that a team is only as good as its ScrumMaster. Let's explore what it takes to become a great ScrumMaster.

Basic ScrumMaster Responsibilities

The most basic thing every ScrumMaster must understand is that his job is to remove impediments that could negatively impact the successful completion of a project. This focus makes the life of team members easier and more relaxed so they can spend their time more efficiently on creating a product.

The ScrumMaster should motivate and help team members develop professionally. This can result in increased employee satisfaction, especially if the team members' participation in the process and decision-making helps them relate to why their project is important. Keeping the team focused on one goal can empower employees to become responsible and take commitments seriously.

The ScrumMaster is responsible for leading the Scrum process by facilitating team meetings, which enable efficient communication flow.

But, there is a downside to the ScrumMaster assuming all these responsibilities. The easiest way to remove obstacles is to work them off oneself, making life easier for the team. Instead, where it makes sense, a great ScrumMaster will occasionally delegate some of the responsibilities to the team.

Not only will the team have a greater respect for what it takes to be a ScrumMaster, they will also grow by broadening their own roles.

Never delegating responsibilities doesn't build a good self-organized or confident team. Instead, the team continues to rely on the ScrumMaster to take care of everything, and they won't develop any further.

Be One Step Ahead of the Team

The ScrumMaster's approach must adjust depending on the state of the team. Using educational psychologist Bruce Tuckman's team development model, [1] if a team is in the initial *forming* phase, it is operating as a group of individuals, not as a team. The trust among team members is limited, and they don't openly discuss issues to identify solutions, usually due to a desire to avoid conflict.

Forming is often the first phase in agile transformation—when team members who were organized in separate departments find themselves working in cross-functional Scrum teams. During this phase, the ScrumMaster's role should be providing guidance and direction so the team can move forward to the next phase of team development. The ScrumMaster cannot allow the team to remain long in forming. Recognizing the situation, the ScrumMaster must be one step ahead of the team and push the team to change and move forward.

The next phase, *storming*, is more collaborative, but usually a number of conflicts appear among the team members. Team meetings can appear disruptive and require discussions about the working style, team values, and agreements. The ScrumMaster's role is still explanation and guidance, but focuses more on facilitation of team discussions.

In the *norming* phase of the team development model, everyone feels good about team collaboration and productivity. Team members trust each other and offer assistance to their colleagues. In the norming phase, the ScrumMaster role is accepted as an impediment remover and someone who takes care of the team. For example, team members understand and appreciate the Scrum rules but rely on the ScrumMaster to tell them if there is something going wrong.

In the *performing* stage, the final phase of the team development model, a team easily self-organizes, eagerly takes on responsibility, and is unified to achieve one goal. Team members

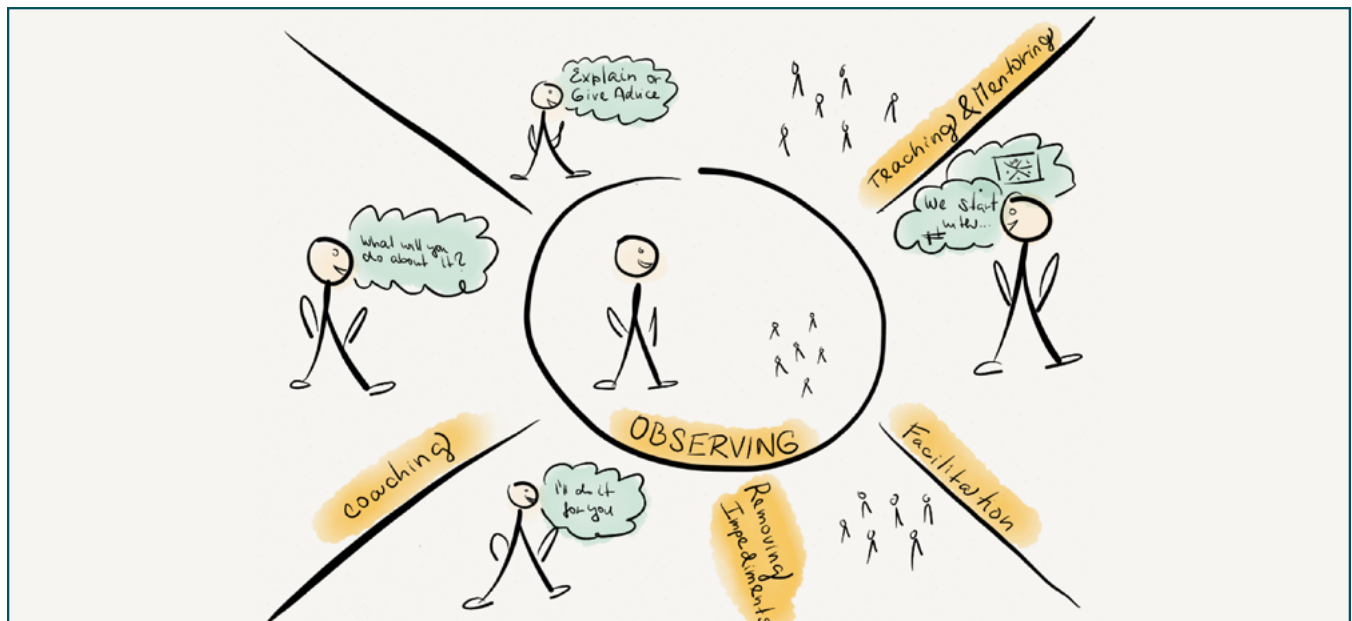


Figure 1: The ScrumMaster state of mind

identify and mitigate difficulties by themselves. The ScrumMaster's role at this stage is mostly coaching and facilitation.

From my experiences, ScrumMasters often get stuck at a certain phase when they already should have been adjusting their behavior to the next phase. It's natural to assume that if a technique worked on a prior project, it should work with any team. Why change something that works? But in order to help the team through all these phases, a ScrumMaster must be one step ahead of the team, sometimes pushing team members into an uncomfortable zone. On the other hand, make sure that you are not too far ahead, as the team may not understand your approach and might not accept you as a team member.

Become a Master of Scrum

Even a Certified ScrumMaster may not be a master of Scrum. To become a master of something, you need to have years of experience, preferably with a wide range of environments and situations. Good ScrumMasters are gardening the Scrum process. Helping the team understand all the reasoning behind the Scrum process, especially in the early phases, can require constant repetition throughout a development lifecycle. Every situation of change is different, and it takes time for a team to become accustomed to a new, agile way of working. The Scrum artifacts, processes, and celebrations may not make sense at first, and it is the ScrumMaster's responsibility to make them work.

In addition, great ScrumMasters spend their time learning more about Scrum and trying different approaches and recommendations. Don't get stuck with what you learned at your first training or what you read in your first Scrum book. Scrum's weak point is that it looks so simple, but in reality, each implementation is different and will naturally change over time. A great ScrumMaster will take on a teaching role to bring new ideas, methods, and approaches of a Scrum framework to teams.

ScrumMaster State of Mind Model

As a ScrumMaster, you should adjust your approach based on the state of the team and a company's agile adoption. There is a useful model that can help you change your approach intentionally. It's called the ScrumMaster state of mind, and it includes four core segments, as shown in figure 1.

Teaching and mentoring: This role assumes that you explain Scrum basics, the reasons for Scrum adoption, how the individual practices are supposed to work, and why an agile team does them. In addition, you should suggest new practices and methods based on your experiences.

Facilitation: This is where you make sure team meetings run smoothly and communication flows efficiently.

Removing impediments: A great ScrumMaster should start each day with the philosophy "What can I do to make it easier for my team to perform their work?" Your contribution to the team in removing obstacles can dramatically improve morale and project progress.

Coaching: Probably one of the most important roles a great ScrumMaster needs to take on is caring about the development of individual team members and building a self-organized team. Coaching the team should result in team members' becoming

more responsible and self-confident. Eventually, they must be able to solve impediments by themselves.

Based on the maturity of your team and the fact that every team is different and needs different things, you could be spending time in multiple segments. However, all of them should be used at every team development stage. The segment to focus on should be the one that helps you reach your goal. While the responsibilities presented so far are important, your immediate project goal can help you decide where you need to be with respect to this model.

The ScrumMaster's ultimate goal is building a good self-organized team, which mistakenly implies that you'll end up doing nothing. This is untrue, and you should never stop being able to provide value to the team. Your role as an obstacle remover will always be needed to get rid of roadblocks as you enable the team to self-organize. In addition, your role for explanation can become critical as you take on pure coaching activities with the team. Any action you take that strengthens the team's responsibility and commitment is the right action to take.

The Missing Piece of the Puzzle

While the four segments of the ScrumMaster state of mind are important to becoming a great ScrumMaster, there is still one very important item missing: observing, which is shown in the center of figure 1. Take the opportunity to be quiet and let the team take over during team activity. There is no rush to take control. You can easily observe the team another minute before you explain, facilitate, coach, or try to fix the problem yourself. If you resist the urge to solve every issue as fast as possible so the team can get back to work again, you will be much closer to the goal of having a self-organized team.

And from any other state of mind role, always step back to the observing circle. There is truth to the adage that listening is one of the most important aspects of communication and decision-making. Replay past projects and think about how listening could have improved the outcome while you were teaching, facilitating, coaching, and removing impediments. I believe you will find some situations where you would have decided differently if you had practiced this model before.

Coaching with Intention

As a ScrumMaster, you should adapt your approach to always be one step ahead of the team, pushing the team members out of their habits and customs. The ScrumMaster state of mind model is particularly important during the transformation phase, but even for experienced ScrumMasters with teams already used to Scrum, this model is extremely powerful.

It helps the ScrumMaster decide which approach to use and makes it more an intention than a guess—and this is a sign of a great ScrumMaster. {end}

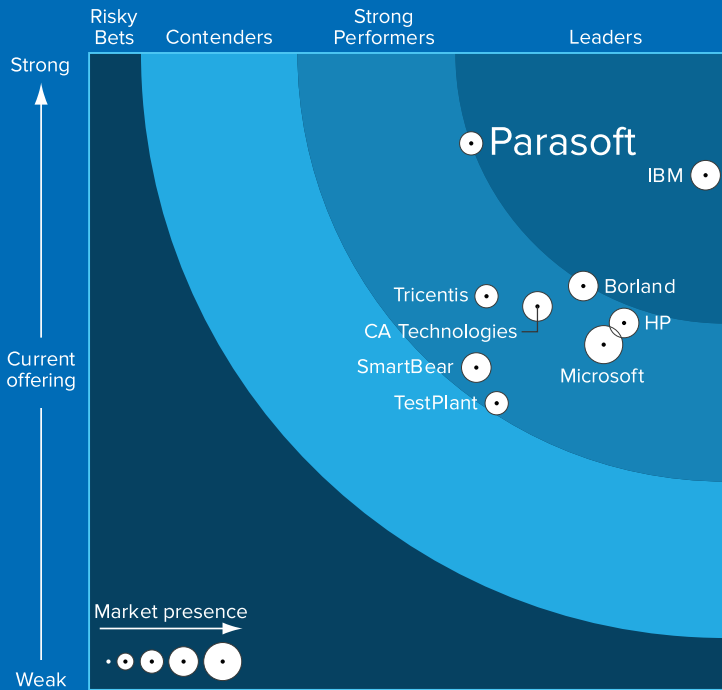
zuzana@sochova.com

Sticky
Notes

[Click here](#) to read more at StickyMinds.com.

■ References

Parasoft Service Virtualization



Source: Forrester Research, Inc. Unauthorized reproduction or distribution prohibited.

Forrester recognized Parasoft for the strongest current offering for Functional Test Automation

The top award-winning Service Virtualization technology



Download the Forrester Research and Service Virtualization Guide:
www.parasoft.com/bettersoftware_SV

Never compromise on quality.



MARKET PRESSURE to reduce time-to-market, increased product complexity, lower budgets and shorter test cycles do not have to mean less test coverage and increased risk.

Beyondsoft's automated testing solutions eliminate the inefficiencies of manual testing, dramatically reducing cycle times while increasing testing efficacy and product quality – so you don't have to compromise.

With tens of thousands of hours of test development and automation experience over more than 18 years and 200 customers around the globe, Beyondsoft's no-compromise, benchmark-driven testing consistently delivers high quality test results and peace of mind.

Download our white paper on no-compromise testing at offers.beyondsoft.com/nocompromise

Contact Beyondsoft at 1-877-896-5859 or visit www.Beyondsoft.com to learn more.



IT Management and Services • Product Engineering • Testing • Mobility • Ecommerce • Cloud Computing

© 2015 Beyondsoft Corporation All Rights Reserved

FAQ

expert answers to
frequently asked
questions

Balancing Waterfall Predictability with Agile Flexibility

As popular as agile methods have become, most companies have not adopted them completely, instead straddling between traditional waterfall ways and the new horizons of agility. One of the core reasons for this is the simultaneous desire for flexibility and predictability, which are at some level fundamentally opposite.

However, the situation isn't hopeless. The aim of agile methods is to eventually speed up project delivery so that long-term projections are largely unnecessary. There are techniques that can help bridge this gap in the meantime. This allows organizations to experiment, innovate, and adjust without the substantial baggage of large plans and commitments.

Determining potential project costs up front: To estimate both a short-term and long-term budgeting forecast, use historical project data from your project team that will be performing the work. And the more recent, the better: Whatever you spent last quarter should be similar to what you'll spend now, as long as you're not substantially changing team composition or making capital expenditures. Use very coarse-grained estimation units, such as T-shirt sizes: small, medium, large, and extra-large.

You may want to consider using incremental funding strategies. The expectation that business planning for software projects should be an annual activity is an unfortunate artifact of waterfall project management, as evidenced by the time organizations spend redoing budgets following the annual effort. A better approach is to keep track of how much you need to spend in total, then allocate as your project progresses based on current needs. The benefit of this approach is that it gives the business more flexibility in managing its budgets and the development efforts being funded.

Knowing when the project is going to be done: Agile planning largely relies on a rearview mirror approach, and in longer-term budgeting activities, this is usually the most reliable method. By using a project's average velocity, generally calculated based on the team's accepted output over a series of sprints, teams get a good idea of their average capacity over time. Teams who ignore their previous velocities often overcommit.

Improving estimates and projections: The best advice I have found is to keep teams as stable as possible. This makes everything more accurate and consistent—notably, velocity—and should improve team productivity and collaboration. Bring work to established teams rather than establishing teams around projects. Constantly adjusting resource allocations makes an organization's true development capacity extremely fuzzy to measure, not to mention the frustration it brings to the team.

Once teams get underway, start gathering data. This will improve their familiarity with what they're building and improve planning and forecasting activities.

Promote regular progress updates to stakeholders instead of long-term projections and promises; the latter won't be accurate, but they'll still hold you to it. Assure stakeholders that you won't allow any nasty surprises to sneak up on them and will provide frequent updates and more accurate forecasts over time. Don't make promises many months out, especially if you won't be able to keep them.

Consistently meeting sprint commitments: Focus on effective product backlog grooming and story splitting. Stories ready for sprints ideally should require just a few days for a team to deliver—likely one to two points, or six to eight stories per sprint.

In summary, a few themes emerge in balancing waterfall predictability with flexible, agile methods: stable teams, consistent communication over long-term promises, incremental budgeting and funding practices, and data-driven projections. Following these guidelines, classically waterfall organizations can grow more adaptable and agile. **{end}**

by Arlen Bankston
arlen.bankston@lithespeed.com

The Evolution of Testing Centers of Excellence

Ever wished that your organization's commitment to quality could be enhanced? Consider the TCoE as a better way to embrace total quality.

by **Rajini Padmanaban** | rajini.padmanaban@qainfotech.com

For years, testing centers of excellence (TCoE) have helped testing organizations deliver a focused and comprehensive effort to improve quality. A TCoE brings together testing people, processes, tools, and frameworks to promote better usage of resources while reducing overall cost of quality and operations. A TCoE balances testing resources, such as tools and testers, and promotes knowledge sharing across testers to better execute their tasks.

Traditionally, TCoEs leverage all the resources within testing groups working on various projects. If, for example, there are three teams working on a set of software products, testing resources could be shared among them. For small companies, sharing of resources brings in greater economies of scale and knowledge. For large organizations, TCoEs can be set up for specific product groups. In this structure, test training, bug bashes, open discussions, and forums can be moderated by the TCoEs.

In addition to core test teams, a virtual group of specialized teams can leverage TCoEs to help with niche testing requirements, such as performance testing, security testing, and localization validation. Specialized group of testers bring specific skills available that are sometimes difficult to find.

While TCoEs usually enhance test team efficiency and productivity, the added communication and collaboration can be a challenge as agile teams scramble to keep up with short release cycles. As a result, there is a risk of introducing chaos when shared resources are brought in as part of a TCoE. This can occur even within a core product team.

Successfully implementing a TCoE is an art and a science—from its initial setup to its evolution as the organization's needs change. To many, TCoE organizations can appear to be added bureaucracy and needless overhead. I recommend the following proven practices to ensure that a TCoE provides significant value and remains relevant.

Bring in other disciplines: Collaboration between disciplines in a product development team is important, so de-

velopers and designers should help testers improve product quality. For instance, non-testing disciplines can participate in bug bashes to help find defects. Where kanban is being used, work is balanced among multiple disciplines, and, as a result, the TCoE model opens doors to external departments. This lets everyone benefit without overwhelming or distracting them from their core areas of focus.

Testdrive ALM: Application lifecycle management (ALM) tools help product teams deliver quality results in a more productive and consistent manner. Each discipline has its own workflows within these ALMs. For example, the Visual Studio product development team has a Visual Studio test team

helping testers run through their full testing lifecycle.

A TCoE deals not just with people and processes, but also with tools and tool migrations. There is now a trend to consolidate test tools and test frameworks to bring in more rigor and consistency to the overall operation.

On several recent projects, teams are increasingly interested in integrating test case management with defect management. Instead of using two separate tools, such as Bugzilla and Testopia, some project teams are now using Zephyr as a plugin with JIRA. This approach unifies reporting and usability with a single user interface.

Commit specialized testing resources: When TCoEs were originally conceived, a goal was to promote the sharing of specialized testers and test resources across departments in areas of performance, security, localization, usability, and accessibility. While this may be true even today, given the change in development methodologies, it is becoming increasingly difficult to share specialized resources across projects. This is due to two factors. First, specialized testers have tasks that require an ongoing commitment rather than becoming consultants on demand. Second, project timelines are becoming much shorter, requiring resident experts to be available during the entire project lifecycle.

“Successfully implementing a TCoE into any organization is an art and a science, starting with its initial setup and its evolution as the organization’s needs change.”

This is a big change from the original intent of establishing a TCoE where specialized testers should be available as shared resources when needed. While specialized testers may not be able to work across projects, it is a good idea to embrace the concept of a TCoE, as it encourages sharing knowledge across various test areas—including specialized test areas. You could view a TCoE as a group of testing subject matter experts. The value of such knowledge sharing should not be discounted. The same goes for sharing resources. Resources, such as mobile devices or data servers, may not be available given a project's timeline, but at the least, consolidating resources under a TCoE can benefit an entire testing group.

The value a TCoE brings to the overall software product development team is immense. As with any model, evolutions require change, and a TCoE is no exception. This is especially true with agile projects where testing is expected throughout the entire project lifecycle. The benefits that a core TCoE model offers will only help the test team validate and verify product quality. **{end}**

index to advertisers

Display Advertising
advertisingsales@sqe.com

All Other Inquiries
info@bettersoftware.com

Better Software (ISSN: 1553-1929) is published four times per year: January, April, June, and September. Print copies can be purchased from MagCloud (<http://www.magcloud.com/user/bettersoftware>). Entire contents © 2015 by Software Quality Engineering (340 Corporate Way, Suite 300, Orange Park, FL 32073), unless otherwise noted on specific articles. The opinions expressed within the articles and contents herein do not necessarily express those of the publisher (Software Quality Engineering). All rights reserved. No material in this publication may be reproduced in any form without permission. Reprints of individual articles available. Call 904.278.0524 for details.

Agile Dev, Better Software & DevOps Conference East	http://adc-bsc-east.techwell.com	Inside front cover
ASTQB	http://www.astqb.org/map	8
BeyondSoft	http://www.beyondsoft.com	34
Mobile Dev + Test & IoT Dev + Test 2016	http://mobile-iot-devtest.techwell.com	29
Parasoft	http://www.parasoft.com/bettersoftware_sv	33
Ranorex	http://www.ranorex.com	2
Sauce Labs	http://saucelabs.com	28
SOASTA	http://soasta.io/cloudtestnow	Back cover
SQE Training	http://sqetraining.com/trainingweek	11
SQE Training: Live Virtual	http://sqetraining.com/virtualtraining	23
STAREAST2016	http://stareast.techwell.com	19
TCS	http://www.tcs.com/offerings/assurance_services/Pages/default.aspx	9

In today's online world, milliseconds mean \$millions.

Are Your Apps Performing at Their Peak?

- **Monitor** real user behavior & business metrics in real-time
- **Test** continuously across web & mobile applications to ensure availability
- **Optimize** via powerful predictive analytics & data visualization for fastest resolution



“SOASTA, with its unique capabilities to correlate information across a range of metrics and excellent real-time dashboard technology, enables us to spot issues, make smarter decisions and align our efforts to achieve the best possible performance and business outcomes.”

