March/April 2014 www.TechWell.com

BETTER SOFTWARE™

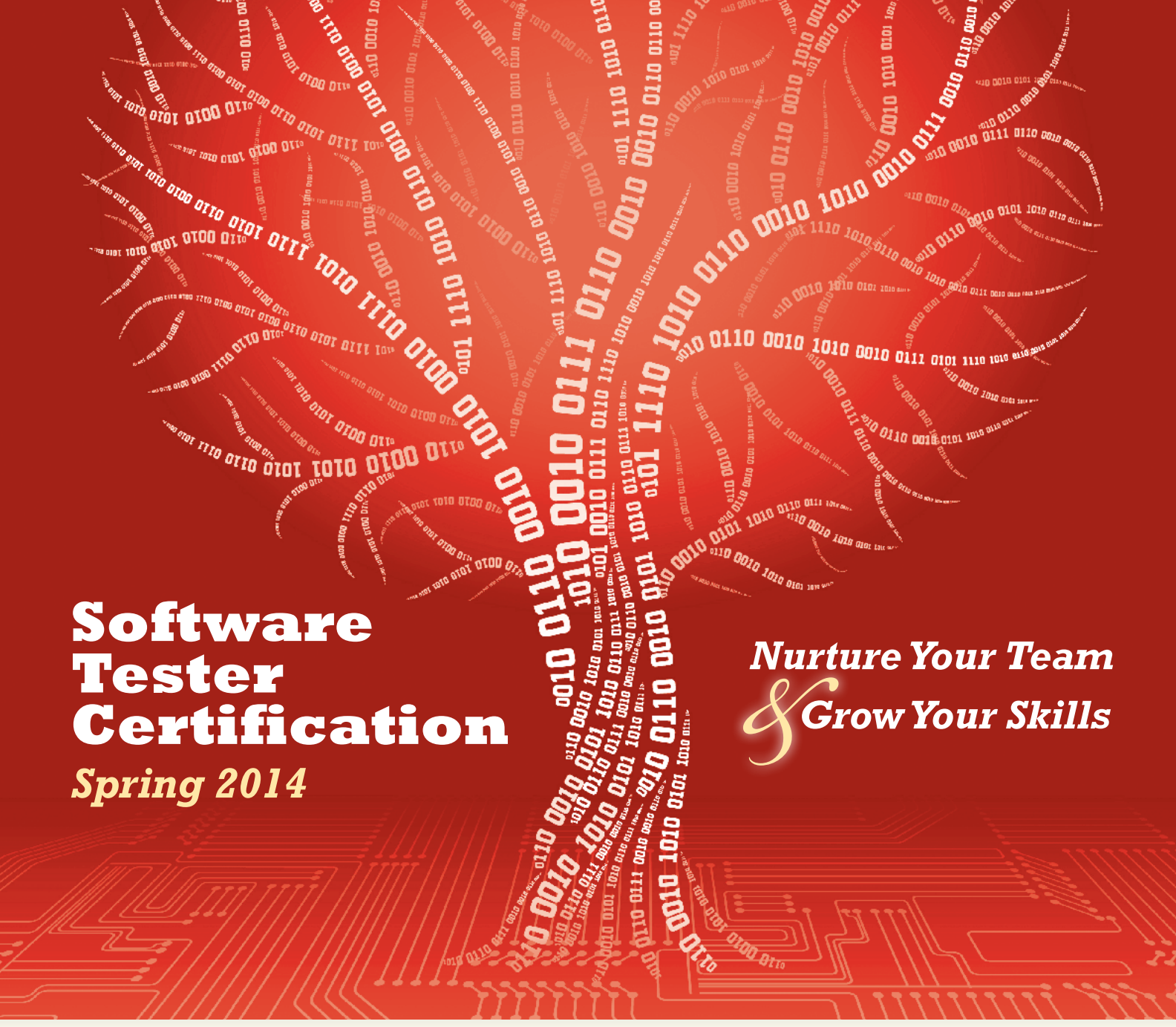
A TECHWELL PUBLICATION

AUTOMATE YOUR UNIT TESTS
To be truly agile, unit tests need to run unattended

WHY DO DEFECTS ESCAPE?
Explore the true cost of bugs found too late

DO MOBILE AND EMBEDDED SOFTWARE REALLY NEED COMPREHENSIVE TESTING?


GREAT SOFTWARE REVOLVES AROUND GREAT TESTING
STAR
CANADA
APRIL 5–9, 2014 | TORONTO, ONTARIO | HILTON TORONTO
REGISTER USING CODE BSMSCI4 AND SAVE UP TO $150 OFF YOUR REGISTRATION! (RESTRICTIONS APPLY)
STARCANADA.TECHWELL.COM



Software Tester Certification

Spring 2014

Nurture Your Team & Grow Your Skills

Make 2014 the Year You Become an ISTQB® Certified Tester

San Francisco, CA (Advanced Tester Certification)	March 3–6
Philadelphia, PA	March 4–6
Cincinnati, OH	March 11–13
Denver, CO	March 11–13
Raleigh-Durham, NC	March 18–20
Boston, MA	March 24–26
LA Area, CA	March 25–27
Seattle, WA	April 1–3
Toronto, ON	April 5–7
Houston, TX	April 8–10
Nashville, TN	April 8–10
Atlanta, GA (Advanced Tester Certification)	April 28–May 1
Orlando, FL	May 4–6
Vienna, VA	May 6–8
Boise, ID	May 6–8
San Diego, CA	May 12–14
St. Louis, MO	May 13–15
Dallas, TX	May 20–22
Las Vegas, NV	June 1–3
Chicago, IL	June 9–11
Tampa, FL	June 17–19

ISTQB Certified Tester accredited training provider

PMI Project Management Institute R.E.P.

Earn 22.5 PDUs

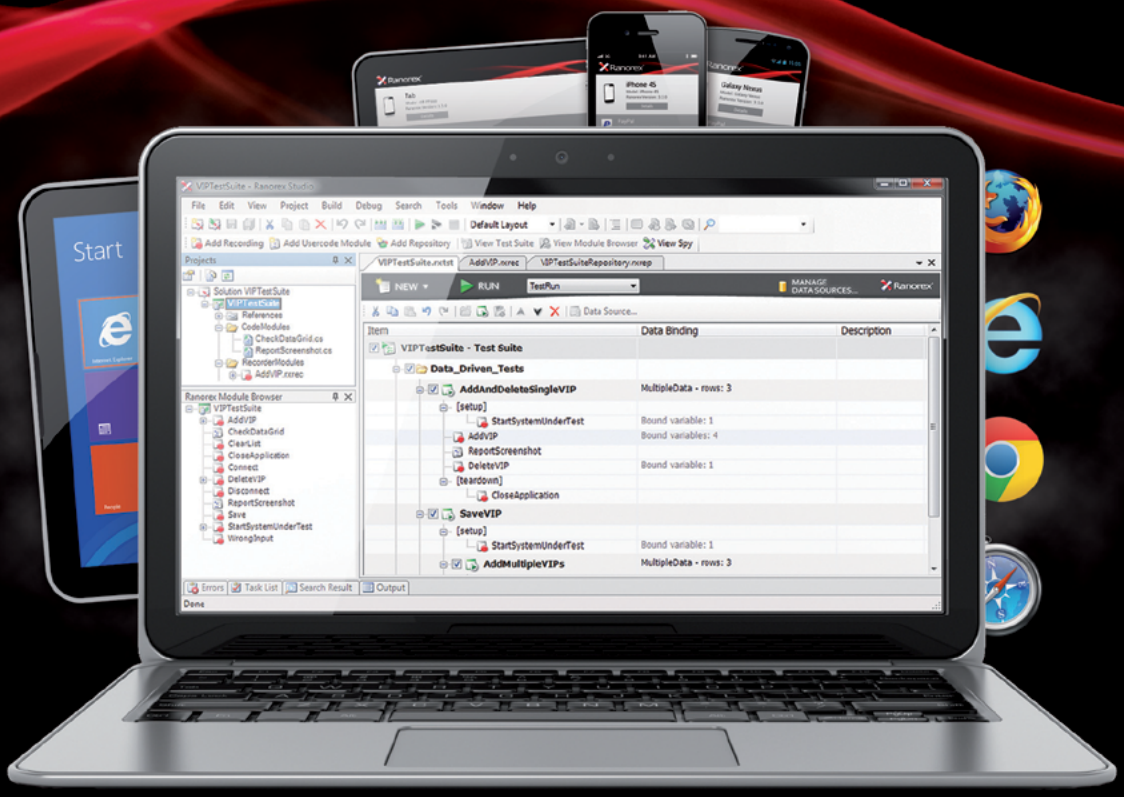
- **Basics of testing**—goals and limits, risk analysis, prioritizing, and completion criteria
- **Testing in software development**—unit, integration, system, acceptance, and regression testing
- **Test management**—strategies and planning, roles and responsibilities, defect tracking, and test deliverables

WWW.SQETRAINING.COM/CERTIFICATION

SQE TRAINING

Ranorex®

Automated Testing of Desktop. Web. Mobile.



- ✓ Use connectors for data-driven tests
- ✓ Build robust test automation frameworks
- ✓ Generate EXEs for pure flexibility
- ✓ Write custom code in C# or VB.NET

Why Use Ranorex

www.ranorex.com/why

Award-winning test automation tools provide seamless testing of a wide range of desktop, web and mobile applications.
HTML5 | IE | FF | Chrome | Safari | WPF | Flash/Flex | Silverlight | Qt | SAP | .NET | MFC | Delphi | 3rd Party Controls | Java

BETTER SOFTWARE
A TECHWELL PUBLICATION
Volume 16, Issue 2 • MARCH/APRIL 2014

CONTENTS

in every issue

features

columns

MARK YOUR CALENDAR

SQE TRAINING

software tester certification

http://sqetraining.com/certification

Foundation Level Certification
March 11–13, 2014
Denver, CO

March 24–28, 2014
Boston, MA

March 25–27, 2014
Los Angeles, CA

April 1–3, 2014
Seattle, WA

April 5–7, 2014
Toronto, ON

April 8–10, 2014
Houston, TX
Nashville, TN

April 29–May 1, 2014
Atlanta, GA

Advanced Tester Certification
April 28–May 2, 2014
Atlanta, GA

training weeks

http://sqetraining.com/trainingweek

Testing Training Week
March 24–28, 2014
Boston, MA

May 12–16, 2014
San Diego, CA

June 9–13, 2014
Chicago, IL

Agile Software Development Training
June 1–3, 2014
Las Vegas, NV

conferences

STAR*CANADA*
http://starcanada.techwell.com
April 5–9, 2014
Toronto, ON
Hilton Toronto

STAR*EAST*
http://stareast.techwell.com
May 4–9, 2014
Orlando, FL
Rosen Centre Hotel

Agile Development Conference West
http://adcwest.techwell.com
June 1–6, 2014
Las Vegas, NV
Caesars Palace

Better Software Conference West
http://bscwest.techwell.com
June 1–6, 2014
Las Vegas, NV
Caesars Palace

STAR*WEST*
http://starwest.techwell.com
October 12–17, 2014
Anaheim, CA
Disneyland Hotel

Agile Development Conference East
http://adceast.techwell.com
November 9–14, 2014
Orlando, FL
Walt Disney World Dolphin

Better Software Conference East
http://bsceast.techwell.com
November 9–14, 2014
Orlando, FL
Walt Disney World Dolphin

BETTER SOFTWARE™
A TECHWELL PUBLICATION

Publisher
Software Quality Engineering Inc.

President/CEO
Wayne Middleton

Director of Publishing
Heather Shanholtzer

Editorial

Better Software Editor
Ken Whitaker

Online Editors
Cameron Philipp-Edmonds
Beth Romanik
Jonathan Vanian

Production Coordinator
Donna Handforth

Design

Creative Director
Catherine J. Clinger

Advertising

Sales Consultants
Daryll Paiva
Kim Trott

Sales Coordinator
Minda Crosby

Marketing

Marketing Manager
Jonathan Greene

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

Why Quality Is Such a Big Deal

With the upcoming STAR*EAST* testing conference taking place in Orlando this May, our focus with this issue of *Better Software* magazine is on quality and testing.

Three of the four feature articles focus on quality. With so much emphasis these days on the adoption of advanced quality assurance techniques and sophisticated testing tools, we decided to get down to testing basics.

I think you're going to enjoy Jon Hagar's discussion of the special testing demands with mobile and embedded software development. As software solutions become more connected with the latest Internet of Things craze, his approach for testing small systems is certainly a must-read.

Gil Zilberfeld promotes the importance of adopting automated unit testing and reinforces the fact that quality testing is everyone's responsibility, not just the job of the QA department.

Mayank Sharma explains how defects escape into deployed production environments and offers suggestions about how to find defects before your customer does.

Natalie Warnert's entertaining article is about the life and times of a first-time ScrumMaster. For those of you starting out as a ScrumMaster or wanting to become a ScrumMaster, you better read her article on the tough lessons she learned the first time she managed Scrum projects.

Don't forget to spread the word to your coworkers about *Better Software* magazine. Just go to http://www.stickyminds.com/BetterSoftware to register for a complimentary subscription.

See you at STAR*EAST*.

Ken Whitaker
kwhitaker@sqe.com, Twitter: @Software_Maniac

Contributors

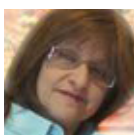
Bob Aiello is a consultant, technical editor of CMCrossroads, and the author of *Configuration Management Best Practices: Practical Methods that Work in the Real World*. Bob has served as the vice chair of the IEEE 828 Standards working group (CM Planning) and is a member of the IEEE Software and Systems Engineering Standards Committee (S2ESC) management board. Connect with Bob on LinkedIn or at bob.aiello@ieee.org.

Kaushal Amin is chief technology officer for KMS Technology (http://www.kms-technology.com), an offshore product development and IT services firm based in Atlanta, GA, and Ho Chi Minh City, Vietnam. He was previously VP of technology at LexisNexis and a software engineer at Intel and IBM. You may reach him at kaushalamin@kms-technology.com.

Jon Hagar has been a software tester, thinker, and teacher supporting software product integrity, verification, and validation for over thirty-five years. Jon works on various standards including ISO, IEEE, and OMG. Jon consults, publishes, teaches, and mentors regularly and has a published book, *Software Test Attacks to Break Mobile and Embedded Devices*. Jon can be reached at embedded@ecentral.com.

Linda Hayes is a frequent industry speaker and award-winning author on software quality. She has been named as one of *Fortune* magazine's People to Watch. She is a regular columnist and contributor to StickyMinds, *Automated Testing Institute* and *Better Software* magazines, and author of the *Automated Testing Handbook* and coeditor of *Dare to be Excellent* with Alka Jarvis. Her article "Quality is Everyone's Business" won a Most Significant Contribution award from the Quality Assurance Institute and was published as part of the Auerbach Systems Development Handbook. Contact Linda at lhayes@worksoft.com.

When not working on his theory of time travel, Cameron T. Philipp-Edmonds is writing for TechWell, StickyMinds, and AgileConnection. With a background in advertising and marketing, Cameron is partial to the ways that technology can enhance a company's brand equity. In his personal life, Cameron enjoys long walks on the beach, romantic dinners by candlelight, and playing practical jokes on his coworkers. He can be reached at cedmonds@sqe.com.



Leslie Sachs is a New York state-certified school psychologist and the COO of Yellow Spider Inc. (http://yellowspiderinc.com). She is the coauthor of *Configuration Management Best Practices: Practical Methods that Work in the Real World* (http://cmbestpractices.com). A firm believer in the uniqueness of every individual, she has recently done advanced training with Mel Levine's All Kinds of Minds institute. She can be reached at LeslieASachs@gmail.com or link with her http://www.linkedin.com/in/lesliesachs.

Mayank Sharma is a principal technical expert at the global development center of Landis+Gyr in India. With more than eighteen years of experience, Mayank has worked as a test professional, process improvement leader, and QA manager in the software industry. He has authored a number of articles and white papers on software testing, agile practices, and the smart grid technology. You can reach Mayank at mayank.sharma@landisgyr.com or http://www.linkedin.com/pub/mayank-sharma/1/626/abb.

Natalie Warnert is an agile enthusiast, practitioner, and coach focused on learning through experience and helping others do the same. In the past couple of years, she has worked on a number of agile and agile-like projects. Throughout her career, Natalie has played many roles, including developer, business analyst, project manager, and ScrumMaster. She is a Certified ScrumMaster (CSM), Professional ScrumMaster I (PSM I), and has her Six Sigma Yellow Belt. She enjoys blogging and speaking about Scrum, agile, and UX. Natalie can be reached on http://www.nataliewarnert.com.

Gil Zilberfeld is the product manager at Typemock, working as part of an agile team in an agile company, creating tools for agile developers. He promotes unit testing and other design practices, down-to-earth agile methods, and cool tools. Gil speaks at local and international venues about unit testing, TDD, and agile practices and communication. In his spare time he shoots zombies for fun. Gil blogs at http://www.gilzilberfeld.com on different agile topics, including processes, communication, and unit testing.

How DevOps Drives the Agile ALM

With some of the recent enterprise software rollout disasters, it is time to get back to basics with utilizing DevOps with your agile ALM.

by Bob Aiello and Leslie Sachs | bob.aiello@ieee.org and LeslieASachs@gmail.com

DevOps is getting a lot of attention these days as the benefits of improving communication and collaboration between development and operations is becoming readily apparent. The need for DevOps is especially obvious in the wake of software and systems glitches that have impacted major financial services, including large banks, trading firms, and the trading exchanges themselves. The healthcare.gov website is the latest high-profile software system that failed to meet its goals, due in part to problems related to software and systems reliability. These problems lead us to wonder if the technology industry is really capable of creating reliable enterprise software.

Software developers, by and large, are very smart and highly skilled technology professionals. Equally skilled are the operations experts who establish the IT controls necessary to ensure that large-scale systems are available continuously, scaling them to meet the capacity demands required during peak usage. Both development and operations teams bring much expertise to the table, but they have fundamentally different perspectives. Developers are expected to write code that implements new features, while operations is charged with seeing that systems maintain a high degree of reliability even under heavy system load. In addition, there are actually many other key stakeholders, without whom we could never create robust enterprise-wide software systems.

Technology organizations consist of a wide array of professionals, from business analysts to QA and testing professionals, each of whom is essential to the successful development of complex software systems. Coordinating their work is no easy task, and most organizations employ a number of full-time project managers who track and report on the work being accomplished by each member of the team. But managing the development of large-scale software systems involves a lot more than just creating Gantt charts and resource reports.

Developing large-scale software systems requires the coordination of hundreds (or even thousands) of tasks, each with its own set of dependencies that are rarely completely understood up front. Helping the entire team understand what needs to be done is what application lifecycle management (ALM) is all about. With many organizations embracing the enhanced productivity and quality that comes from employing agile principles, the agile ALM is becoming an essential software methodology. It turns out that agile ALM benefits greatly from the very same principles behind the DevOps revolution.

> "One of the most effective approaches to DevOps involves moving the automation of the application build, package, and deployment upstream to the beginning stages of the software development lifecycle."

Enhancing collaboration between development and operations works because each group brings a set of complementary skills to the table that, when integrated, enhances both productivity and quality. Developers know the technology they have been creating better than anyone else. They should; most developers have months to get up to speed and focus on changing technologies and software development frameworks. Developers often get to choose which technologies to use in creating systems and then focus on building their expertise on a daily basis. Operations professionals need this information in order to be successful. The operations team understands what happens when a critical system is unavailable for any period time. One of the most effective approaches to DevOps involves moving the automation of the application build, package, and deployment upstream to the beginning stages of the software development lifecycle—an industry best practice long before DevOps became as popular as it is today. [1]

True DevOps groups involve development and operations teams working collaboratively to automate the complete application build, package, and deployment process, creating what is becoming known as "the deployment pipeline." This practice enables the team to best support iterative development by emphasizing the synergy between development and operations. The rise of DevOps demonstrates the powerful synergy that can be achieved with close collaboration between development and operations.

Recognizing that the other stakeholders also possess expertise, it becomes clear why improved collaboration and communication throughout the complete lifecycle can help the entire team achieve a high degree of personal productivity and effectiveness. The fact is that encouraging other stakeholders to apply these principles often yields significant benefits as well. Each team member can benefit significantly by better communication and collaboration.

Information security (InfoSec) is often in the position of trying to ensure the integrity of complex systems they do not completely understand. Similarly, testing and QA professionals are expected to ensure that complex systems are defect-free. Each stakeholder on the team brings expertise, and successful companies are realizing that DevOps really applies to the entire ALM. This is particularly apparent in an agile development methodology embracing iterative development.

The agile ALM helps brings structure to the demanding and constantly changing application lifecycle that is part of any agile development effort. While the level of ceremony and software process maturity may vary a great deal from one project to another, there is always a need for just enough structure so that each stakeholder understands what he needs to do on a day-to-day basis. The Scrum methodology provides an excellent basis for communicating and sharing knowledge and is used successfully by many highly effective self-organizing teams. In an agile ALM, information security professionals have the ability to start looking at working milestones of the system much earlier in the process. This enables InfoSec to understand the interfaces and core requirements for ensuring systems security. Similarly, QA and testing professionals who get involved early in the process build quality in from the beginning [2] and are better equipped to develop and automate robust testing frameworks. The DevOps environment involves moving work upstream and creating milestone releases that have fully automated deployment pipelines and robust testing frameworks, including information security.

Within the agile ALM, the customer or his representative is also a key stakeholder. Applying DevOps to the agile ALM ensures that systems meet their specifications and also satisfy their business purpose. Many technology teams struggle to fully understand the business requirements up front, and iterative development provides an excellent means to allow business experts early access to release milestones to ensure that the system meets its intended requirements and, more importantly, that the requirements are indeed correct. Competitive pressures, including new and ever-changing regulatory requirements, mean that understanding a given system's requirements can be very complicated and involve aiming for a moving target. As a result, DevOps applies to a lot more than just development and operations. The agile ALM needs to ensure that each stakeholder embraces the collaborative synergy of sharing knowledge and better communication as the core lesson that DevOps brings to the table! **{end}**

Sticky Notes

Click here to read more at StickyMinds.com.

- References

DELIVER *Brilliant* SOFTWARE

AGILE DEVELOPMENT CONFERENCE WEST

BETTER SOFTWARE CONFERENCE WEST

JUNE 1–6, 2014 | LAS VEGAS, NV | CAESARS PALACE

Super Early Bird Savings!

REGISTER BY APRIL 4 AND SAVE UP TO $400

GROUPS OF 3+ SAVE EVEN MORE!

Two Conferences in One Location

PMI Project Management Institute R.E.P.

PMI® members can earn PDUs at this event

Explore the Full Program at

ADC-BSC-WEST.TECHWELL.COM

ALL ATTENDEES WILL HAVE THE OPPORTUNITY TO BUILD A CAR WITH TEAM WIKISPEED—HOW AWESOME IS THAT!

Joe Justice

Years in Industry: **10**
Email: **Justice@ScrumInc.com**

Interviewed by: **Cameron Philipp-Edmonds**
Email: **cphilippedmonds@sqe.com**

"A visible impediment list—anything that the delivery team thinks is preventing them from accelerating—is critical. If that's not easily visible and anyone can't easily add items to it, it's going to really slow everybody down."

"Not that Scrum is the only answer—agility as a whole is—but Scrum is the rightest left way to do it with teams. It has a highly successful track record, which helps these companies. We are seeing it transform at a global political level right now."

"Personal kanban is what lets me keep that sustainable and help me keep my velocity high, because I always know what's happening next, and when I have a distraction or an interrupt, I always know what's next again."

"We are able to propagate skill throughout the team, which reduces our "bus factor"—how many people in the team would it take to get hit by a bus before the project would stop? For most teams, it's one person."

"Cultural inertia is the biggest block I've hit when I'm working with companies. They say, "Well, this is the way we've always done it, why would we change?" Yet they have a mandate saying we have to produce the next tractor or the next software package in half the time and half the price if we are going to remain competitive."

"Some teams hit 10x velocity, the amount they are able to get done in a given increment of time with quality. If they do that, what are they going to do with all that extra time they have? What we propose is that they should do some social-good work, and that's what Team WIKISPEED is for."

> If we want teams to rock as hard as they can, then they've got to be performing like top-level sports teams. That's absolutely a mindset, a way of thinking.

"The biggest win we could possibly have is for people to understand what a high-performing team feels like. That's difficult to describe in a book, and it's difficult to create that experience for people to know what it feels like."

Unprecedented velocity and uncompromised quality? You can have it all

Test Continuously with HP Solutions for ALM, Quality and Performance

Test early, test often, test continuously. Under the pressure to release faster, quality can be a casualty with costly consequences. So how do you inject quality in an ever-shrinking development window? The answer is in relentless automation, lifecycle integration and service virtualization. Combine them together and watch your Agile teams fly with quality that rocks customer satisfaction.

See it in action at STAR*CANADA*, STAR*EAST* and STAR*WEST*, visit HP in the solution showcase and learn more at **www.hp.com/go/ALM** today!

Interested in writing an article for *Better Software* magazine?

Contact Ken Whitaker at kwhitaker@sqe.com. We're looking for article proposals for agile, testing, project and people management, configuration management, ALM, development, and any other topic you think is relevant to today's software professionals.

NEWSLETTERS FOR EVERY NEED!

Want the latest and greatest content delivered straight to your inbox every week? Have we got a newsletter (or four) for you! AgileConnection To Go covers all things agile. CMCrossroads To Go is a weekly look at featured configuration management content. StickyMinds To Go sends you a weekly listing of all the new testing articles added to StickyMinds.com. And, last but not least, TechWell To Go features updates on the curated software development stories that appear each weekday at TechWell.com. Visit StickyMinds.com, AgileConnection.com, CMCrossroads.com or TechWell.com to sign up for our weekly newsletters.

Do Mobile and Embedded Software Really Need Comprehensive Testing?

BY JON HAGAR

THINKSTOCKPHOTOS.COM

"Because we are testing a simple mobile app, our app doesn't need extensive testing to submit to the app store."

"Because we are testing embedded software, the software is already unit tested by our developers and we have the best software development processes for our small amount of code. As a result, we won't have bugs in the field."

I focus my energy working with mobile and embedded product development teams, and these quotes are all too familiar to me. On the other hand, we all have read a similar headline in the press: "Large auto manufacturer loses lawsuit because of defective software in an electronic engine controller."

In app store sites, I see hundreds of product reviews with no stars and comments, like "This app is buggy and I can't figure out how to use it, so I deleted it after twenty minutes of frustration."

Both of these stories are situations the software producers would like to have avoided, so perhaps the attitude that "we don't need much testing" is wrong. Worse, this lack of attention to quality costs companies millions of dollars every day.

What Are Mobile and Embedded Systems, Anyway?

Mobile "smart" systems are small, handheld devices, usually connected to communication networks and powered by batteries. They share many common features with embedded devices and traditional computers, yet they have limited resources.

Examples of these devices include cellphones and smartphones, tablets, medical devices (such as pacemakers and defibrillators), automobiles and other forms of transportation (like cars, buses, trains, trams, and trolleys), and factory and industrial systems (PLCs, robots, and so on).

Embedded software systems consist of unique hardware or systems with dedicated software that solve specialized problems, often in real time. Embedded systems have the following unique characteristics.

Unique hardware: Software interacts with special hardware, providing interface and control support.

Constrained resources: The systems have limited resources, such as RAM, ROM, stack, power, speed, or time.

Limited user interface: Embedded systems typically have a restricted or no user interface.

Examples of embedded software systems include software-controlled robotics, avionics systems, control devices, and smart electronics. In fact, users may not even be aware a device has software.

Testing Considerations

What testing approaches should be considered for mobile and embedded systems? The simple answer is to use the same techniques and testing approaches used for developing IT software, PCs, and the web. You'd think that with heavily constrained systems, testing would be dramatically less expensive. On the contrary, testing mobile and embedded systems may cost more and require more effort than expected. Instead, mobile and embedded validation might include the kinds of attack testing listed in table 1, which focuses on the kinds of errors commonly seen in mobile and embedded systems. [1, 2]

The attacks outlined in table 1 may appear to be common to other software environments, but specific patterns of attacks in the mobile and embedded space will be dramatically different. The attack patterns of table 1 are unique to mobile and embedded environments and would also need to be customized for specific types of software, such as medical, transportation, industrial, space, gaming, mobile information, sales, and so on.

Attack Type	Finds	Notes on the Attack
Developer level attacks	Code and data structure problems	Almost a quarter of errors in mobile and embedded can be found by structural testing.
Control system attacks	Hardware and software control system errors	Many critical bugs in mobile and embedded are centered in the control logic.
Hardware-software attacks	Communication and interface integration issues	The software works with unique hardware that must be assessed.
Communication attacks	Digital communications problems	Mobile and embedded systems communicate with hardware, networks, and software with complex interfaces.
Time attacks	Time, performance, sequence, and scenario bugs	Embedded and some mobile apps have critical timing and performance factors.
User interface attacks	Problems between man and machine	The usability of devices and software is critical to success.
Smartphone attacks	Issues specific to smart device configurations, including gaming and cloud bugs	Gaming and cloud computing comprise a majority of the apps being deployed.
Security attacks	Bugs that can expose devices to security threats	Security of devices is increasing in importance.
Generic functional attacks	Requirements and interoperability bugs	These are the basic checks testers should conduct on mobile and embedded devices.

Table 1: Types of attacks on mobile and embedded systems

The concept of test attacks to break software is a popular basis for testing and is detailed in a variety of books and standards [1, 3] that go beyond basic requirements verification checking. Test attacks are patterns of testing to find errors in the software based on common failures. Test attack patterns will be customized for the local context, considering factors such as the software under test, who does the test, where the attack is done, available resources, and the goals of the test.

This menu of attack types and test approaches will leave most test teams concerned that to implement a full range of tests will take too much time and money. In order to prioritize the best test approach, I'd recommend considering the following approaches for mobile and embedded testing.

Focus testing on what is important: Conduct ongoing, risk-based test planning and prioritization. [3]

Coordinate test scope: Define your test scope in a test strategy or plan to gain agreement with your stakeholders. [4]

Employ early testing: During development of the code, use exploratory, nonautomated tests to provide useful information to the team. [5]

Stay agile: Evolve the test plans and strategies to provide more information to your stakeholders and to determine if more or less attack testing is needed.

Using Simple Risk-Based Testing

Although there is no best or single way to test apps, always consider the heuristics behind test approaches, planning, and techniques.

Risk-based testing uses product risks identified by a team's stakeholders to determine the areas most critical to product success. Risks are then used to focus test activities on critical high-risk concerns. If functional and nonfunctional qualities are quickly attack tested with cost-effective exploratory testing (not highly scripted testing), data about product trustworthiness can rapidly provide information to the team to aid development and release decisions using an informed agreement approach. Using risk-based testing may result in some functional areas being undertested or not tested at all, but the full team will get to decide the amount and approach of acceptable testing for the product.

Defining Your Scoping Strategy with Early Testing

Closely related to risk-based testing is the determination of test strategies as part of higher-level test planning. For example, on one mobile app software project the team decided to use agile developer-based testing combined with a risk-based attack during development. As the software became mature, they employed a crowdsourced third-party group of testers to follow a new detailed plan of validation checks, which focused on potential bugs and risks that might likely impact the user of the app. This was performed with a small team of developers, testers, and a crowd team. Once the product received positive customer reviews, the team expanded the product and attack testing. This early and fast feedback testing approach during initial app development not only found critical errors but also was cost- and time-effective. Once the deployed product gained user acceptance, more testing was performed to find latent bugs before users found them.

Pulling It All Together

There is no best set of test attacks, number of tests, or combination of approaches that can be generically used for all mobile and embedded systems. A good approach should be to focus on attack-based testing early, frequently consider the risks to refine scope, and adjust test plans. Such a strategy can be done within a project's cost and schedule constraints because mobile and embedded software is often developed under tightly constrained budgets and very aggressive schedules.

For many projects, budget and schedule constraints can be changed as testers provide information on the quality status of the software under test. There are times when schedule changes and increased budgets need to be considered by demonstrating with test data that functional bugs or other software quality problems exist in the product. You may want to avoid releasing buggy software when the team knows about the quality issues. There are times, however, when the team will have to make an informed release decision by accepting the risks.

A major consideration in risk-based and attack-based testing is that the tester must go beyond just checking functionality. If testers only verify requirements, usually with simple "happy path" test cases, many errors may remain in the software. Missed errors can result in unhappy users, resulting in negative feedback. System testing that tries to show the software does not work should be a common practice of mobile and embedded testers within project constraints. Such "break-it" system testing, with a focus on risks and using attacks or complementary test techniques, can provide valuable information to the team and decision makers. The balancing of constraints and test approaches requires highly skilled and practiced testers. This is true in general for most software, but mobile and embedded systems can have significant risk involving safety, hazards, finances, legal issues, or other factors, which should elevate the need for comprehensive testing.

In Summary

Risks of bugs, bad customer feedback, and other undesirable results should be considered by the team when selecting an optimal testing approach to take. Skilled mobile and embedded testers who balance testing to constraints will most likely release products that are good enough and meet expected quality goals. Unfortunately, as we have all witnessed with quality disasters in PC and web app environments, many of the newer mobile and embedded teams who don't learn the testing lessons of the past may not be around in the future. **{end}**

embedded@ecentral.com

Click here to read more at StickyMinds.com.

- References

Featuring fresh news and insightful stories about topics that are important to you, TechWell.com is the place to go for what is happening in the software industry today. TechWell's passionate industry professionals curate new stories every weekday to keep you up to date on the latest in development, testing, business analysis, project management, agile, DevOps, and more. Here is a sample of some of the great content you'll find. Visit TechWell.com for the full stories and more!

Ants in Space? That's a Good Thing

by Pamela Rentz

The orbiting International Space Station (ISS) just got 600 or so new visitors who arrived via cargo spacecraft.

They'll be taking part in a new research project to help determine how they adjust to microgravity conditions, and mapping their behavior could lead to more refined algorithms for solving complex problems—such as how robots could better search for survivors in a burning building or at a disaster scene.

Continue reading at https://well.tc/qan

The Benefits of Making Deliberate Mistakes

by Naomi Karten

In his book *Antifragile: Things That Gain from Disorder*, Nassim Nicholas Taleb describes a loser as someone who "after making a mistake, doesn't introspect, doesn't explain it, feels embarrassed and defensive rather than enriched with a new piece of information, and tries to explain why he made the mistake rather than moving on." These types, he points out, often view themselves as victims of a large plot, a bad boss, or bad weather.

That's quite a characterization, given how tempting it is when we make a mistake to run from it, rationalize it, conceal it, or blame others. But the reality is that sometimes a mistake is exactly what's needed to make progress. So, sometimes you require not just a mistake—but a deliberate mistake.

Continue reading at https://well.tc/c2A

Is It Time for Cloud Providers to Control Malware Distribution?

by Rajini Padmanaban

Is malware only about unwanted software gaining access to secure information or the process of infecting websites? There is clearly more to this, including complex transfer distribution systems that make even trusted sites and applications play a role in this malicious activity.

There are several categories of people who may be involved in malware creation and several categories of malware itself—making this a very specialized segment of security engineering in the software development world. Malware distribution by and large is an illegal activity and government bodies continue to take condemning actions against malware creators and distributors.

Continue reading at https://well.tc/cLV

Wearable Apps Not Wearing Out Welcome

by Cameron Philipp-Edmonds

Wearable tech and accessories that interact and communicate with your tablet or smartphone aren't overwhelmingly popular—yet. But research from Gartner shows that many analysts expect a significant uptick in the number of wearable devices and connective mobile applications. In fact, Gartner's prediction goes so far as to suggest that as many as half of all app interactions will come from wearable devices.

Brian Blau, an analyst for Gartner, explained that because most wearable devices can share the interface with a mobile device, the manufacturers are able to keep the devices small, efficient, inexpensive, and easy to maintain.

Continue reading at https://well.tc/cnw

Is a Framework Needed to Scale Agile?

by Kent J. McDonald

As larger organizations adopt agile, there is an increased focus on figuring out how to apply agile across a large part of the organization and how to deal with obstacles and dysfunctions that are more prevalent in larger organizations compared to smaller ones. These obstacles often relate to the size of projects that are attempted, the number of people who have to be involved with them, and the organizational structures that are involved in delivering new software assets or changes to existing assets.

This focus, usually labeled as "scaling agile" or "enterprise agile," is viewed by some in the agile community as the next step in the agile evolution. There are varying reactions to the need to scale agile. Some in the community have identified new methods along with training, consulting, and certification to help with adoption. Two of these methods are the Scaled Agile Framework (SAFe) and Disciplined Agile Delivery (DAD).

Continue reading at https://well.tc/cux

Nervous about Your Big Presentation? Don't Try to Relax—Get Excited

by Beth Romanik

"Keep calm and carry on"? When it comes to preperformance jitters, it turns out a better mantra might be "Get excited and try to fight it."

For years, people who experienced sweating, a racing heartbeat, and nervous thoughts before being in the spotlight were advised to just take deep breaths and try to keep themselves calm. However, new research from the American Psychological Association suggests that getting excited before a presentation is more effective for decreasing anxiety than trying to relax.

Continue reading at https://well.tc/cua

YOU CAN'T BE AGILE WITHOUT AUTOMATED UNIT TESTING

BY GIL ZILBERFELD

THINKSTOCKPHOTOS.COM

Agile methodologies are approaches to managing software development based on short-term, iterative, and incremental deliveries, enabling continuous feedback and flexible response to change.

Stemming from a rapidly evolving business environment that demands faster product improvements and modifications, the agile methodology promotes the organizational qualities of speed, responsiveness, and adaptability throughout the entire application management process, from defining product requirements to coding, testing, and, finally, release management.

This article explains why agile development cannot be implemented effectively without unit testing—and especially automated unit testing.

The Importance of Code Quality

Developers have known for decades that the further into a project timeline a bug gets discovered from its insertion point, the more costly it is to fix. When a developer finds a bug, it can sometimes take minutes to fix. If it slips through testing and finds its way to the customer, figure 1 shows that mitigation can be exponentially more expensive to fix. [1]

Correcting quality issues can take months of rework, cost millions of dollars, or, if done too late, may even cost lives. Take, for example, the first launch of the Ariane 5 rocket in 1996. Its flight abruptly terminated just thirty-seven seconds after liftoff, taking with it hundreds of millions of dollars in invested effort. Also think of Toyota recalling four hundred thousand vehicles because of a bug in the brake control system, costing an estimated three billion dollars.

While we can't eliminate all bugs, we can fight them by baking quality into the code. There are many ways to define code quality depending on the perspective of the customer or the developer.

The customer expects working software. Customers do not care how the code is written—they just need the software to work. When developers talk about code quality, they talk about code that is easy to maintain, easy to read, and risk-adverse to change. Each perspective takes the cost of bugs into consideration. The customer knows that for each bug, he'll lose precious business hours or days. The developer knows that each returning bug means considerable time spent fixing it instead of working on new features.

Agile methodologies take working software and combine it with early feedback. For example, early releases can get user feedback about how well the software operates. To give the developers confidence that their code works, unit testing gives the fastest available quality feedback.

The earlier defects are found, the cheaper they are to fix. As agile methodologies encourage high code quality, the team should run lots of unit tests. Similarly, automated tests give the developer early feedback on the quality of the software in a repeatable fashion prior to release.

What Is Unit Testing?

Unit testing is a methodology where individual units of software, associated data, and usage procedures are tested to determine whether they operate correctly. The unit is usually a small piece of code—for example, a single function. The unit test is a short function that tests the behavior of the unit that produces a pass/fail result. This is achieved by performing the tested function on a known value with a single correct result. Unit tests often use mock objects to simulate the behavior of dependencies in a predictable way.

The main purpose of unit testing is to allow developers to identify as many problems as possible at the development stage and to do it in an automated, repeatable fashion that can be applied for every code change.

This makes developers directly responsible for producing working code, even before it reaches the quality assurance team.

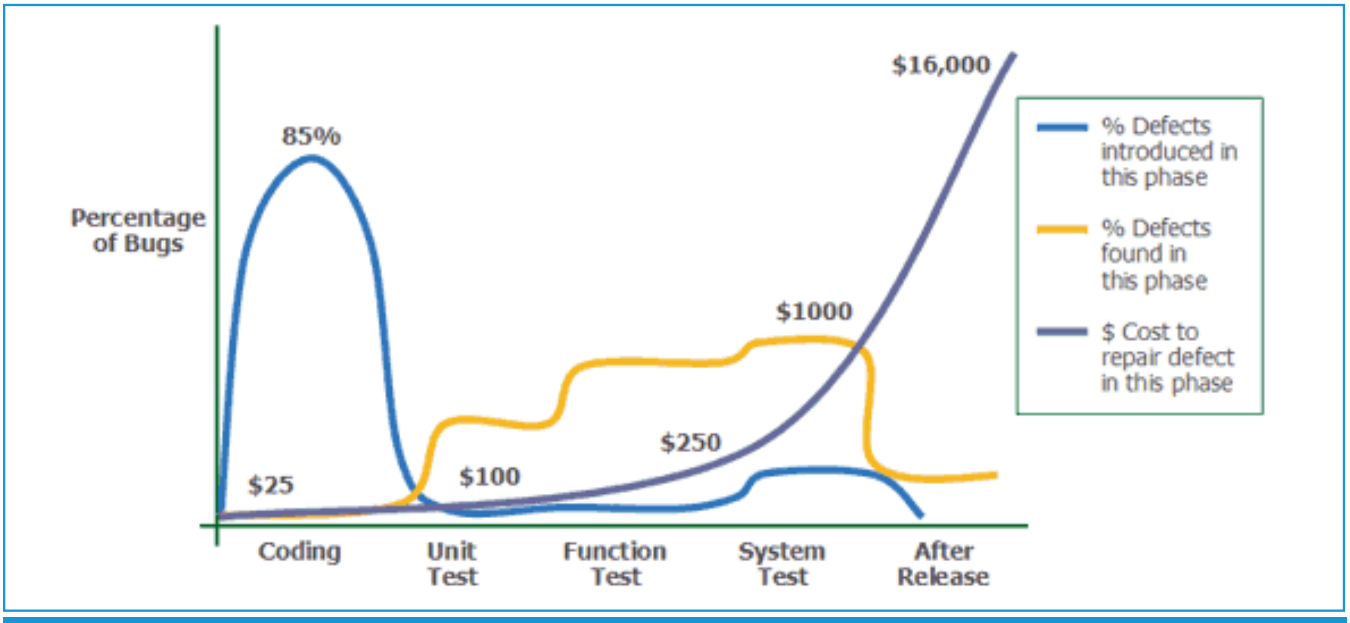


Figure 1: Example of the cost of defect correction during a software project's development lifecycle

What Does Unit Testing Have to Do with Agile Development?

I think the two are closely linked. In fact, I believe you can't be truly agile without implementing automated unit testing as an integral part of the development process. Automated unit testing has several benefits that align closely with agile development principles.

The central benefit of unit testing is that it produces working code faster and with fewer bugs. The ability to automate tests and catch bugs at the development stage reduces a huge amount of overhead that is otherwise spent on releases that are immediately rejected by QA due to basic functionalities being broken. Unit testing increases the chances of a new feature working correctly upon first delivery, as it becomes the developer's responsibility to verify that he is delivering working code.

Another reason that unit tests cut down on development time is that their fine resolution allows them to pinpoint precisely the location of a problem. A failed unit test can direct the developer to the exact location of the problem in the code, allowing him to quickly resolve it. This minimizes or even eliminates the time that would otherwise be spent locating the problem.

Unit testing may not be able to catch all bugs, but it is highly effective in catching regression bugs that are defects that break existing functionality. These bugs hamper progress and waste valuable development and QA resources as code is sent back and forth between the two departments, delaying new versions of existing products and new product releases. Without automated testing, it is virtually impossible to detect bugs during the development phase. This causes sprints to become bogged down as developers need to spend more and more time fixing regression bugs in order to keep producing working software. It becomes impossible to maintain a steady and predictable software delivery schedule while also maintaining quality. When a release date draws near and the product is not working, panic sets in, software is released without enough time to test it, and more bugs are introduced, creating a vicious cycle.

Code that is not properly maintained very quickly becomes legacy code that developers either refuse to change or insist on rewriting themselves. To keep code alive, you need to be able to change it and be confident that your changes won't break anything. Unit testing promotes this confidence. Without it, you end up either refusing to change older code or investing large amounts of time rewriting it every so often. In order to respond quickly to change, you need to be able to modify all parts of your code quickly and confidently. Some tools even allow you to develop unit tests for older code without having to change the code itself.

Agility through Automation

The platform for unit testing is implicit, and we usually omit the word automated before it. In reality, unit testing is a collection of processes, skills, and tools that support agility. For example, writing the tests is an actual skill. I look at tests I wrote five years ago and think, "How would anyone let me write this?" (I'm sure I'll feel the same in five more years about what I'm writing now.)

In addition, using isolation and mock objects correctly is a capability that improves over time. Refactoring of the tested code or changing code design can fill up a three-day workshop, and much like design, it can be improved and lead to maintainable test design.

When we improve our skills, we can move more quickly and change directions as we go with agility. But without automation, we won't be able to use our skills effectively in a repeatable fashion.

Automation is the foundation that gives the power to get quick feedback from running tests. It gives us the ability to cover more code and know we didn't break anything. And it gives us the independence to change our design when we need to without risk and to mold the software the way we want it.

In the end, the Agile Manifesto favors working software. Automated unit tests bring us close to that point quicker than other processes.

Conclusion

The benefits of unit testing are closely aligned with the principles of agile software development. Unit testing allows you to make code changes while remaining confident that they will not break existing functionality and that the major part of new functionality will work on first delivery. This enables frequent, timely delivery of working software, which in turn enables swift response to changes in requirements. Automated unit testing also promotes a transparent view into the code's health by producing reports that allow anyone to see which problems occur and their precise locations in the code. Further, automated unit testing reduces the number of regression bugs, preventing development sprints from becoming bogged down and enabling developers to maintain a constant, sustainable work pace.

Together with the agile methodology, an integrated, automated unit testing tool that works well within your programming environment is a crucial necessity for managing modern software development. **{end}**

gilz@typemock.com

Sticky Notes

Click here to read more at StickyMinds.com.

- References

www.virtusa.com

Virtusa QA Revolution

The future of quality is here

Social and millennial QA, 'connected quality' and managed testing services driving innovative quality outcomes

Gold sponsors at STAREAST May 04 – 09, 2014

Virtusa is proud to partner with Per Scholas to build US based QA resources leveraging their Software Testing Education Program (STEP) in underserved communities

virtusa®
Accelerating Business Outcomes

Why Do Defects Escape?

Can Continuous Analysis of Production Defects Improve Quality and Customer Experience?

By Mayank Sharma

THINKSTOCKPHOTOS.COM

An important quality measure of a software solution is the rate at which customers find defects after a release is deployed in a production environment. A high rate of production defects not only poses a risk to how well the solution works as expected, but also impacts the customer experience. The situation is more intimidating with contractually obligated ventures and mission-critical systems. Moreover, it is a well-known fact that the cost of quality exponentially increases when more defects than expected are discovered either late in the project lifecycle or after the release is deployed into production. An effective defect analysis practice plays a vital role in an organization's ability to prevent defects from escaping into production.

A famous quotation reminds us that "Those who cannot remember the past are condemned to repeat it." Measure the statistical trend of production defects and establish a profound defect analysis methodology in order to take the initial strides toward the realization of quality from the customers' perspective. The key to improvement of quality is to tackle the prevalent causes affecting the quality over time.

In order to begin our journey of quality improvement, my company envisioned a defect analysis methodology with the intent to minimize the number of production defects by targeting the defects that need engineering resolution. Gradually, a repeatable practice should evolve as we seek answers to some fundamental questions after consistently measuring and analyzing production defects every month.

Identifying Weak Process Phases

In the first phase of our investigation, we categorized the key process areas of the overall release lifecycle. Using information collected within my company, figure 1 shows the distribution of production defects by overall time spent in project phases.

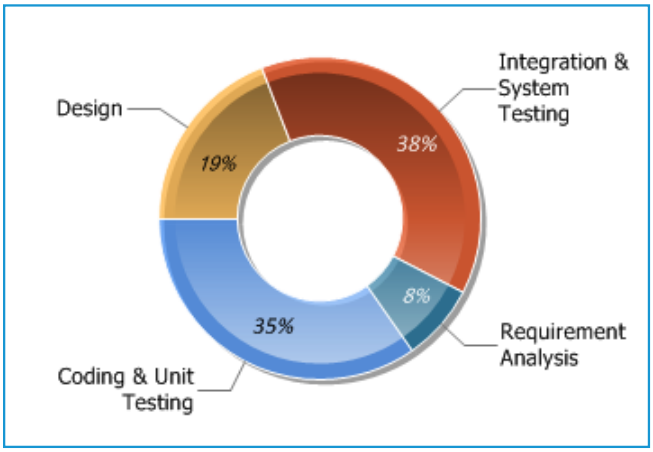


Figure 1: Distribution of production defects by overall time spent in project phases

The distribution of production defects in the graph depicts that the majority of production defects escaped from the integration and system testing phase and the coding and unit testing phase. As a result, these two process phases need an in-depth analysis in order to understand and address the root causes of detected problems. To ensure a comprehensive analysis, further investigation of other factors, such as the type of defects and their areas of origin, can help pinpoint the root causes.

What Are the Most Overriding Behaviors of Escaped Defects?

In the next step of our analysis, we identified the broad categories of defects considering the most commonly occurring patterns in production. Ideally, it is a good idea to classify the nature of defects based on various components and the overall architecture of the system. Such a classification allows isolating the most repetitive natures of defects based on the predefined defect types.

The distribution of defects in figure 2 illustrates that the majority of defects are of the timing or serialization type and exception handling type, followed by the functional behavior type. The next logical step is to dig into the origins of these defect types in order to determine preventive actions.

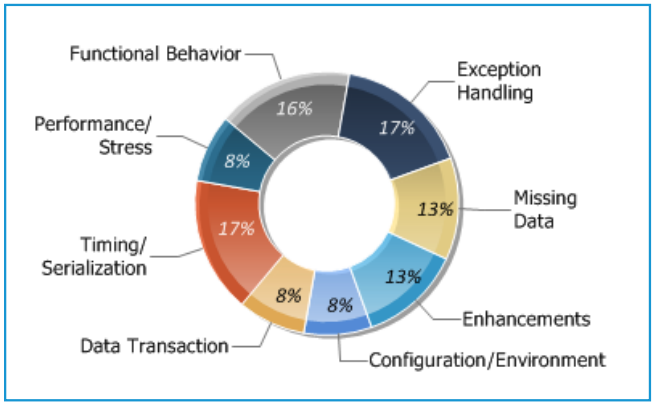


Figure 2: Distribution of defect types

At 13 percent, missing data defects can pose business risk in terms of system reliability in reporting accurate data to other applications such as the billing system of the utilities. As a result, it is important to reduce the number of defect types with missing data in order to make the system reliable to comply with customer contractual obligations. In spite of a relatively lesser percentage, such defects taking place in the field would need immediate attention due to their high impact on the business.

Where Is the Root Cause of a Defect?

In order to prevent defects from escaping into production, it should be a priority to understand the source of when the defects are being introduced. A table consisting of the commonly repeating defect types with their origins helps in isolating the root cause of defects. Figure 3 shows major categories of defect origin areas classified by defect types in order to segregate root causes of production defects.

In a later stage of our analysis, we define the underlying areas from which the majority of defects possibly originate. These areas are identified based on the most common origins found during the root cause analysis of the defects when they are resolved and the associated corrections in code. It may not

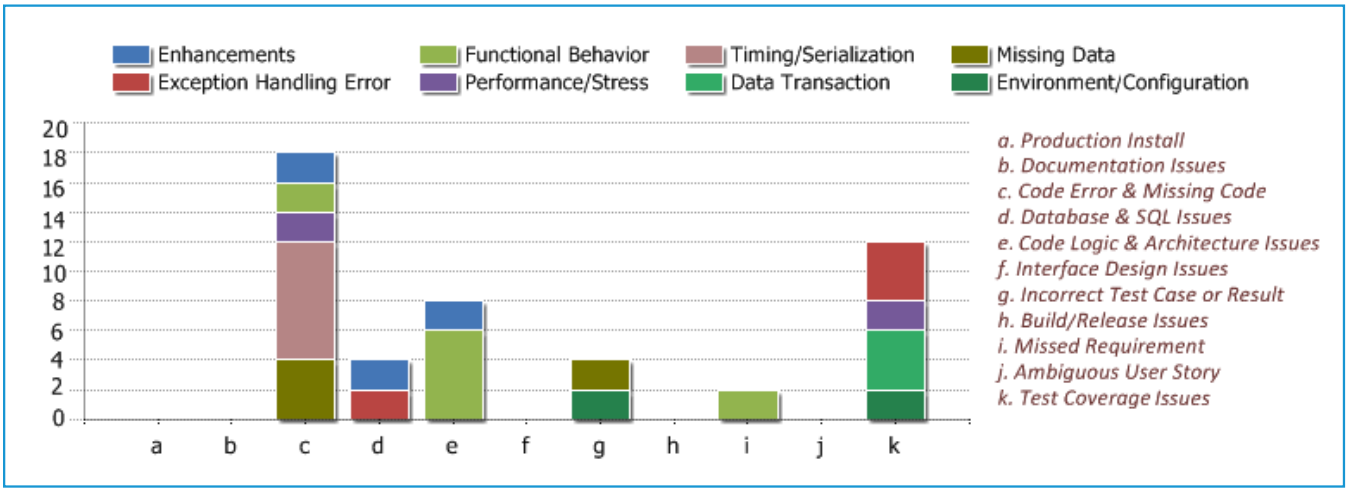


Figure 3: Defect origin areas classified by defect types

be viable to ascertain the exact origin in some cases if the root cause analysis is not conclusive. In that case, I recommend digging into the underlying cause once the defect is resolved. It is always a great idea to have a detailed discussion with the developers and testers to help everyone understand how your findings can reduce escaped defects.

Figure 3 depicts code error and missing code issues as some of the most prominent causes for escaped defects. The portion of code error and missing code defects is approximately 37 percent of total defects. The distribution of defects by overall time spent in project phases in figure 1 also supports the fact that 35 percent of escaped defects take place during the coding and unit testing phases in a project lifecycle. Timing or serialization defects and missing data defects are the two most prevalent types within the code error and missing code category. As a result, these areas demand greater attention to the effectiveness of code reviews and coding guidelines compliance in order to prevent such defects from occurring in the future.

Another problem area within the same category is functional defects originating due to code errors. One of the solutions to this issue may be to review the existing unit testing and code coverage analysis practices with the aim of preventing such defects from occurring in code logic. In addition, a large number of functional defects are in the code logic and architecture issues category. This implies the need for a review of the robustness of the high-level design and architecture before the implementation is performed because the cost of nonconformance can be significantly higher if the design defects are discovered after the release into production.

Testing everything before going into production is neither desirable nor realistic. So, how much do you test? The lack of test coverage is another important cause of defects escaping into production, as evidenced by the data in figure 1 showing that 38 percent of defects occurred during the integration and system testing phases in a project lifecycle.

The combination of the data transaction area and the exception handling area in figure 2 stands out after taking a deeper look at the defects happening within the test coverage issues category. To mitigate these defects, test design should focus on better test coverage for data transactions and negative test cases for exceptions in handling error conditions.

Last but not least, the performance or stress type and the configuration or environment type from figure 2 are areas of concern for defect escapes. A thorough review of your test framework (also known as a test harness), improved test design techniques, and reprioritizing your test strategy can help in overcoming these test gaps. The identification of critical areas of defect origins helps to eradicate the root causes of defects and, subsequently, to prevent their reoccurrence.

Miles to Go

Defect analysis findings need consensus from all stakeholders in order to move forward with a defect escape mitigation plan. The corrective actions to overcome defect escape gaps should be tracked until closure is made in order to complete a defect analysis cycle. Preventing defect escapes is an ongoing journey toward improving the overall quality from the customers' perspective. It is a good idea to periodically repeat the defect analysis cycle because it helps to continuously reinforce defect prevention.

The adoption of an effective defect analysis practice does not mean there won't be any more defects reported from the field. But reducing the total number of defects (and especially in critical defects) can be a realistic expectation. This will take time, and a sound defect escape mitigation practice seeded today will certainly ensure a positive outcome in the future. **{end}**

mayank.sharma@landisgyr.com

Performance Driven.
Quality Assured.

For most organizations, IT systems and software applications provide the foundation for the organization's core offering, reinforcing its brand and reputation. However, inadequate software testing can increase risk, potentially leading to loss of revenue and customer trust.

At Capgemini, we've developed innovative, business-driven quality assurance and testing services, combining industry recognized leading testing methodologies TMap® and TPI® (Test Process Improvement). We offer a comprehensive portfolio of Testing Services (including Managed Testing) that take a proven approach to industrialized and structured software testing. Our testing services produce clearly identifiable results, notably the reduction in cost of software development and maintenance by up to 30%; enhanced quality levels, improved time-to-market, and importantly mitigation of technical and business risks.

Over 500 businesses trust us to deliver consistently. We know where testing should be and how to take organizations there.

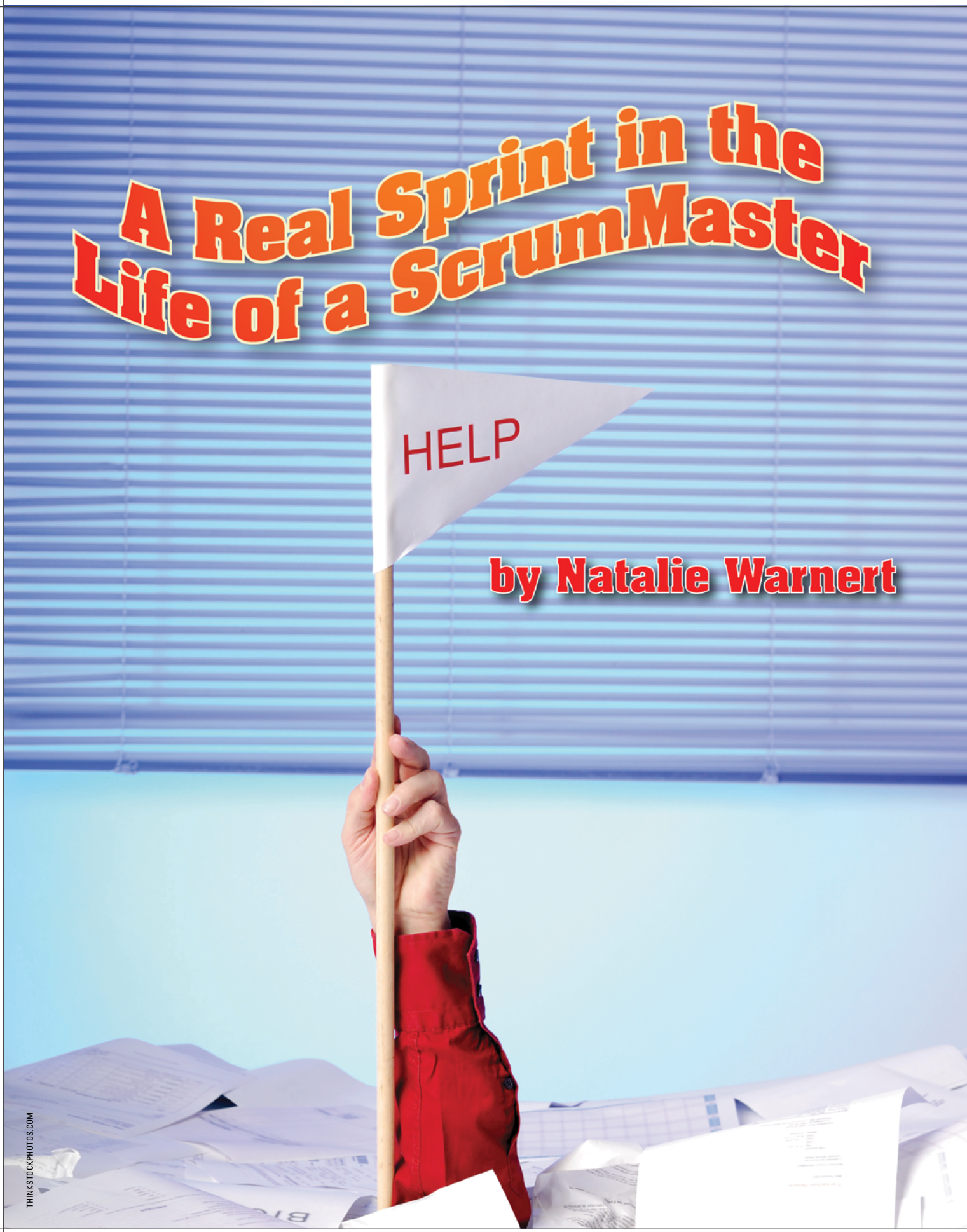
More information is available at:
http:// www.capgemini.com/testing

SOGETI

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

A Real Sprint in the Life of a ScrumMaster

by Natalie Warnert



THINKSTOCKPHOTOS.COM

You will face many challenges that ScrumMaster training does not prepare you for, from management and corporate roadblocks to disgruntled team members and self-organization follies, and pure excitement alone does not compensate. I compiled some lessons I learned the hard way in my first few years of being a ScrumMaster. As many ScrumMasters know, a real sprint is very different from the controlled, safe environment of ScrumMaster training. There are ten lessons learned that I'd like to share.

1. Change is a journey, not a destination

I find it's not prudent to try to change everything at once, especially if the organization is new to Scrum implementations. Take it in steps, take your time, modify where necessary, and expect there will never be a steady state. I realized this as I was doing a Scrum implementation. When I first started, the idea of Scrum was a foreign concept to the product development team, and I wanted to completely retrain them and the way they worked. I was met with tremendous opposition to a new way of working. There was uneasiness to commit to fast-paced sprint iterations, and the team members feared they would be in trouble if they didn't complete what they committed to. This led to the team undercommitting by padding estimates.

On my next project, I decided to take the "Scrumness" out of team conversation entirely. I started by asking about work items and writing them down without calling them stories. This slowly evolved into planning meetings and continually documenting work items with the product owners. To avoid tool complication, all of this was done by hand and put on a Scrum board for visualization. I kept setting small goals with the team to get to the next implementation milestone and then evolving those goals as things were met. Was it perfect? No. But it was constant evolution and iteration to meet our next goal in being a great Scrum team and less pressure than implementing all pieces of Scrum at once. Change takes time.

2. Self-organization takes longer than you think

There is no single prescriptive way to know if your team has self-organized. One size does not fit all. Sometimes the team has to take two steps back to advance one step forward.

What does self-organization really look like? Scrum projects rely on teams' self-organizing, yet does the team know they're supposed to be doing that? I felt that something wasn't going well when I wasn't getting invited to some team meetings and gatherings. I was nervous and panicked, and I was sometimes unaware that meetings were taking place unless I saw the team gather. Then I realized this was the team members' form of self-organization, and they didn't always need me to be present. It was extremely rewarding to see them solve their own problems in the way they saw fit.

3. Be authentic and vulnerable

It's OK to not know everything or to ask for help. Giving up power makes us vulnerable. Admitting mistakes makes us authentic. If you do, the team will learn that being wrong is part of team growth. They'll see you as human and will be more likely to take the risks that Scrum touts as the way to be efficient and agile.

One example that comes to mind was working with a difficult leadership team on a new Scrum implementation. The essence of Scrum was being compromised with testing and requirements performed in different sprints. The team was working in mini-waterfalls, all supported and promoted by leadership. I struggled to influence project decisions for months, with little to no success. I consulted with a leader—with more authority and experience—outside the project and department.

I admitted that I needed help to influence the project leadership team. I received great advice on "managing up" that I wouldn't have known if I hadn't sought help.This leader also took an active interest in the project and eventually helped to influence the leadership direction toward true Scrum. It was a very humbling experience, and I wish I hadn't waited so long to admit I needed help.

4. Find your ScrumMaster style

Is your style similar to being a Scrum purist? Are you more or less flexible than other ScrumMasters? What approach works best for you? What works best for the team? How much are you willing to adjust while still staying true to yourself and the Scrum guidelines?

When I was given the reins as ScrumMaster on my first large Scrum project, my style was that of a Scrum purist. I was not in favor of detailed release planning and I was totally focused on story points over ideal days. I even created a deck weighing out the pros and cons of both to make my point clear to the team and leadership.

Not only did this brand me as a "Scrum crazy person," but it also demonstrated to the team members my inflexibility about their needs. I wasted a lot of energy fighting "Scrum-But" (meaning being Scrum-like, but not really) when I could have been improving other things that would have helped the team more.

Now I've adapted my style to be more flexible, and I only have a few key Scrum practices I make sure the teams stick to.

5. What you do is as important as what you don't do

Although you can't and shouldn't solve every problem, find out how to make things easier for the team without doing everything yourself. Your team needs to remove some of its own obstacles or self-organize to solve the problems and can't always be dependent on you.

One of my teams did not like to update its tasks in the agile lifecycle management software. This agile software was difficult to use and the web interface didn't work well in Safari, so I didn't blame them. For some period of time, I updated the information for them. In fact, I was handling all of the project planning, and when I was not available to the team, sprint planning would not take place. Could they have done it by themselves? Sure, but I had made it too easy, and they got lazy.

By taking a different approach, I laid out cards to make a Scrum board instead of using software tools. I had everyone write stories during planning and put them on the board to avoid any excuses for not being able to access the ALM system.

Did they complain? Yes. But did they learn and self-organize? Yes.

6. Let the team fail

A key tenet of agile is the need to trust your team members to do what is right—for themselves, the project, and, ultimately, the customer. Sometimes team members will go down the wrong path. And sometimes, even though you know that a wrong path is being taken, you have to let them. By jumping in all the time as the problem solver, you undermine them. That can result in a huge step backward because team members may feel that you distrust them, or you may be hindering an idea that in fact works great for the team.

I worked with a team who didn't like the daily stand-up. The members didn't see value in it, and while I see it as an important part of Scrum, why should I make them do it if it's not valuable to them? They suggested changing it to twice a week, and while I disagreed with the idea, I agreed to try it. After awhile, the benefit of frequent team collaboration and communication eroded rapidly. They eventually realized that they needed to meet more frequently, and the team agreed to meet daily. It was vital that I let them try it to show I trusted their decision-making.

The other important piece of information here is that I didn't challenge them to prove me wrong or make a big deal about their modifying Scrum. This likely would have added more resistance and made it a "team versus ScrumMaster" mentality, something I've seen before from a ScrumMaster who was too purist and wanted Scrum, as a methodology, to prove all things right. It is of great benefit to use Scrum as a powerful project framework, but it is the team's interactions and everyday decisions that truly result in project success or failure.

7. Being a ScrumMaster is not always glamorous

You're removing obstacles, protecting the team, and acting as a servant leader. That also means you're doing a lot of busy work required to keep day-to-day projects going. Embrace it! No one person on the team is more or less important than anyone else, and you're not above any kind of work that is going to help in the end goal to produce software project delivery.

Regarding busy work, this includes a whole lot of things. I set up meetings, I send status reports, I make project plans, and I order food. I had to order food multiple times in a city I didn't even live in for a few release planning sessions. At the time, I thought this was menial work and I felt like I was being undervalued. Surprisingly, I was the most important person of the meeting, according to everyone else on the team. I was asked multiple times every day what awesome food I had picked for the day and when it was coming. It was like a herd of elephants was coming at me when I brought in the afternoon coffee. Did I like ordering food? No, and I still don't. But by doing it, I freed up the team to do the work they committed to, and really, that is what my commitment is. As the ScrumMaster, you must embody the agile values, too, and walk the talk to be a part of the team, no matter how unglamorous it is.

8. Perform a personal retrospective

A self-review and team member review should take place at determined intervals so everyone can see how far they have come, not only as a team, but also as individual technical contributors.

I did a personal retrospective at the end of the year. I had made some large changes in the past year, both personally and professionally. When I looked back, I realized that I had been witnessing a lot of things as failures instead of learning opportunities, and it made me think of the impact on my team. If I were feeling this way, were they? Instead, I wanted to see what I had really accomplished and learned. I turned this around and did an individual exercise with the team to see their accomplishments and learning. Failure was not a term used during the discussion, and from every issue we identified, at least one lesson was learned. These retrospectives were extremely powerful, and we made a great poster of our successes in the past year to hang up by our working agreements.

9. Don't forget to celebrate

Celebrate small wins. Celebrate at the end of a feature completion, sprint, or any other important project event. It doesn't need to be a huge party every time, and it doesn't have to be expensive. But make it fun with small gestures of gratitude.

Sometimes I use retrospectives as a form of celebration by hanging up accomplishments around the team room. You can do this by printing out a particularly complex piece of code, a hard bug that was corrected, some important screen shots of user interface improvements, and so on. Another thing I like to do is have others recognize their teammates by writing colleagues' accomplishments on sticky notes and sharing them with the group. On the more social side of things, there are potlucks, happy hours, and game ideas. There are endless ways to recognize small wins.

10. Learn something new every day and encourage your team to do the same

Whether this is inspecting and adapting, reading a blog, or talking to a team member one on one, it's important to always strive to improve and learn.

One way I like to do this is by using Agile in a Flash cards from The Pragmatic Programmers. They give a new tip or anecdote to talk about every day after the stand-up or during a lull. Though the team groans sometimes, I know they like them. I also often post links on the team wiki with good articles I come across on Twitter or share good tweets. Finally, I put my agile books out on the table, and whenever I get some spare time, I pick one up. When the team saw this, they started doing it too, and it almost sent me through the roof celebrating that win. Talk about authentic enthusiasm.

There are many things traditional ScrumMaster training does not prepare you for, and much of the time, lessons need to be learned through practice. I hope some of these examples of how I learned things the hard way will help you become a better ScrumMaster to your team and organization. And maybe you will avoid some of the frustration I went through as I learned from my mistakes. **{end}**

http://www.nataliewarnert.com

Coverity Inc., Announces Coverity Development Testing Platform 7.0

Coverity, Inc., a development testing company, announced the availability of the Coverity Development Testing Platform 7.0, the next-generation of its software testing platform that enables development organizations to create and deliver better software, faster. The rapid growth of cloud, mobile, and web-based application development in enterprise IT organizations has elevated development testing to a business-critical process in the software development lifecycle (SDLC), arming developers with a way to quickly and efficiently test their code and address critical quality and security issues as it is written.

The new version of the Coverity Development Testing Platform is the industry's first enterprise-scale solution that combines code analysis, change-aware unit test analysis, and policy management across C/C++, Java, and C#. With this release, Coverity has built on its market leadership and multiple patents for scalable and accurate defect detection with new innovations to C# and Java code analysis.

http://www.coverity.com

Linode Launches Linode CLI

Linode, an established leader in cloud hosting, launched Linode CLI, a new tool that provides a convenient way to provision and manage Linode cloud services from the command line. The Linode CLI enables users to easily automate common tasks, such as creating, rebooting, or resizing servers, while also making it straightforward to manage DNS records and distribute workloads across backend Linodes. While some users will be content using the Linode Manager or Linode Mobile to administer their services, those who prefer the convenience of the command line will find Linode CLI a time saving alternative. Not only can Linode CLI be used to provision Linodes and manage NodeBalancers, it also makes it easy to modify DNS zones and records and output to JSON format for scripting of repetitive tasks.

http://www.linode.com

East Coast Datacom, Inc., Releases Stateful Traffic Generator

East Coast Datacom, Inc., a communications specialist, released the Stateful Traffic Generator, STG-10G based on the generation engine, D-ITG. The STG-10G is composed of a graphical user interface (GUI) that wraps the D-ITG engine, INTEL DPDK Fast Packet Technology, and other test tools.

The STG-10G produces IPv4 and IPv6 traffic by accurately replicating the workload of current user applications. The platform supports eight-ports 10/100/1000 and four-ports of 10GbE traffic generation managed via the easy to use GUI. This allows users to perform load tests on hardware prior to deployment and to simulate wired or wireless network traffic behavior.

The STG-10G supports UDP, TCP, ICMP, DCCP, SCTP protocols and soon to be released support for IGMP. The STG-10G also supports replay of Pcap files with an easy to use Pcap player.

http://www.ecdata.com

AvePoint Unveils DocAve Online Service Pack (SP) 3

AvePoint, a provider of enterprise-class big data management, governance, and compliance software solutions for next-generation social collaboration platforms, unveiled DocAve Online Service Pack (SP) 3. This update features data protection, governance, and reporting enhancements that allow organizations to maintain the same level of protection and control over their cloud-based assets as they have with on-premises solutions.

Updates in DocAve Online SP 3, AvePoint's software-as-a-service platform for Microsoft Office 365 management, include policy enforcement, granular content protection, and configuration reports.

http://www.avepoint.com

Compuware Corporation Announces New Features to Compuware Workbench

Compuware Corporation, a technology performance company, announced new features to Compuware Workbench, a modern, intuitive, Eclipse-based mainframe development environment. First released in 2010, the Workbench has evolved into a solution that's made application development, testing, and tuning faster and more efficient for both new and experienced developers across the globe. Updated offerings include significantly more robust data search and editing capabilities, additional debugging support, and a code coverage reporting feature. The Workbench has also been more tightly integrated with Compuware APM for Mainframe and the company's other developer productivity solutions, improving the way teams collaborate when resolving application exceptions and performance problems.

http://www.compuware.com/en_us.html

Blueprint Releases Blueprint v5.4

Blueprint, a leading provider of enterprise requirements definition and management (RDM) software, released Blueprint v5.4, the latest version of its best-in-class RDM solution. Blueprint v5.4 features new application lifecycle management (ALM) integration capabilities that align cross-functional teams; and includes Blueprint Analytics, a solution that gives enterprises clear visibility into the early stages of large, complex IT project portfolios.

Blueprint v5.4 features new and enhanced integration capabilities that enable deep bi-directional data synchronization with leading agile ALM solutions, including: IBM Rational Team Concert, JIRA, Microsoft Team Foundation Server, Rally, and VersionOne. This integration allows these systems to "talk" to one another, enabling better communication between business analysts, developers, testers, and all other team members who use these products. Customers can unify their ALM infrastructure with Blueprint v5.4 to deliver complete traceability, improved visibility, and efficient collaboration between cross-functional teams throughout the application development lifecycle.

http://www.blueprintsys.com

Red Hat Makes Available Red Hat Enterprise Linux OpenStack Platform 4.0

Red Hat announced the availability of Red Hat Enterprise Linux OpenStack Platform 4.0. Red Hat Enterprise Linux OpenStack Platform offers IT infrastructure teams, cloud application developers, and experienced cloud builders a clear path to the open hybrid cloud without compromising on availability, security, or performance.

Red Hat Enterprise Linux OpenStack Platform 4.0 also continues Red Hat's integration of its infrastructure product portfolio with OpenStack. Red Hat CloudForms 3.0, sold separately, can now manage and deploy workloads to Red Hat Enterprise Linux OpenStack Platform and provides management across OpenStack, Red Hat Enterprise Virtualization, VMware vSphere, and Amazon Web Services (AWS). Red Hat Storage Server, also sold separately, offers Red Hat Enterprise Linux OpenStack Platform an open, software-defined, and distributed storage foundation that provides a massively scalable and highly available storage platform and native compatibility with OpenStack storage interfaces.

http://www.redhat.com

Typemock Launches Isolator V7.4.3 for .NET

Typemock, a provider of easy unit testing solutions, launched Typemock Isolator V7.4.3 for .NET. The new Typemock Isolator V7.4.3 takes developer productivity even further, speeding up programming processes and eliminating development down-time due to redundant debugging and QA. Uninterrupted development flow is at the heart of this version release. With FastFix, the relevant test is "pinned" for debugging, while the status of all tests is visible as well. This eradicates time-consuming navigation, expediting bug-fixing.

The new version has improved productivity for faster development flow with enhanced usability features, including: new shortcuts, updated menus, highlighting capabilities, and easier navigation.

V7.4.3 also supports the newly released Visual Studio 2013 as well as all other leading development tools.

http://www.typemock.com

Centrify Corporation Announces Centrify Developer Site

Centrify Corporation, the provider of unified identity services across data center, cloud, and mobile, announced the Centrify Developer Site for easy and direct access to Centrify's software development kits (SDKs), giving enterprise application developers and ISVs the resources and support needed for integration of Centrify's identity management into their cloud, mobile, and datacenter applications and systems. The Centrify Developer Site also serves as a hub for newly introduced and updated SDKs, technical resources on integration, code samples, interaction with Centrify developers and the developer community, and more.

http://www.centrify.com/developers

WANdisco Unveils New Version of Git MultiSite

WANdisco, a provider of high-availability software for global enterprises to meet the challenges of big data and distributed software development, announced the next release of Git MultiSite, the company's performance, scalability, and continuous availability solution that provides LAN-speed Git access and collaboration to developers everywhere, even across a WAN.

The new features of Git MultiSite 1.2 include centralized management and replicated configuration settings for simplified administration and enhanced security across multiple sites. In addition, Git MultiSite 1.2 integrates seamlessly with common ALM toolsets with enhanced support for distributed notification mechanisms. These features alleviate administrative burdens and boost security for global enterprises looking to streamline their source control management systems.

http://www.wandisco.com/git/multisite

Compuware Corporation Announces New Release of Data Center Real User Monitoring Solution

Compuware Corporation, a technology performance company, announced a new release of its Data Center Real User Monitoring solution (DC RUM). Enhanced analytics and new network packet capture and analysis capabilities simplify identification and triage of performance issues across applications, infrastructure, and network. Now application operators can monitor and understand the network impact on application performance down to the transaction and user level at packet-level depth. The new availability analytics span all layers of a business transaction, from the network TCP session all the way up to the application logic.

http://www.compuware.com/en_us/application-performance-management/products/application-aware-network-monitoring.html

Seapine Software Unveils TestTrack 2014

Seapine Software unveiled TestTrack 2014, the latest version of its product development management solution. TestTrack 2014's new interface integrates the functions of Seapine's three previous web clients, further enhancing the user experience with a modern interface, improved usability, and additional features.

TestTrack 2014 replaces its three previous web clients—TestTrack Pro Web, TestTrack RM Reviewer, and TestTrack TCM Test Runner—with a cleaner interface that is optimized for usability. It features new search filtering capabilities and enhanced options for removing interface elements to increase usable screen real estate. Deployment is easier with TestTrack 2014 because it eliminates the need to install desktop clients on multiple computers.

TestTrack 2014 also includes new, integrated dashboard capabilities for the ALM Reporting Platform. Teams can now easily launch ALM RP dashboards to monitor project progress across an organization.

http://www.seapine.com/almnewfeatures.html

FAQ

expert answers to frequently asked questions

Should QA Perform Unit Testing?

by Linda Hayes
lhayes@worksoft.com

The short answer is no, absolutely not.

The long answer goes like this: You know that old saying that indicates the later you find a defect, the more it costs to fix? Well, it's true; yet few make decisions as if it is. Rigorous testing in design and development lifecycle phases is rare, as is root cause analysis to determine where the defect was introduced (requirements, design, and build) and where it was detected (engineering, QA, and production). But without this analysis, you can't really tell where to adjust your processes for improvement.

You can't fix quality in QA

Unit testing is the responsibility of engineering in most system development lifecycles. The developer who creates or modifies a unit of code should be accountable for testing whether the code meets its technical requirements before it is delivered to QA. While some developers are diligent about unit testing, others may not be, and still others simply lack education. Even worse, schedules rarely allocate time for testing, and developers are usually only rewarded for delivering code, not necessarily for delivering quality.

If you skimp on or skip early-stage testing activities, the impact inevitably flows into QA. Low quality code complicates or blocks functional, regression, and performance testing. This forces QA to perform unit testing in self-defense. This comes at the sacrifice of QA's own responsibilities, given the reality of limited schedules and resources. The more unit testing is performed by QA testers, the less other types of testing gets done, and, as a result, the more defects escape into production.

The ultimate irony of this predicament is that in most cases, QA is held accountable for escaped defects instead of engineering. Show me an engineering organization that is lax about unit testing and I will show you a testing organization that is compromised. We have all witnessed companies throw more and more resources into QA to counter low quality yet completely ignore upstream processes like review of requirements definitions and design specifications.

If you are part of a QA organization that finds itself being sucked into unit testing because low-level capabilities like field edits, menu navigation, and error handling are missing or unreliable, here is some advice:

1. Review your company's systems development lifecycle (SDLC) documentation and review the section about unit testing. I will lay odds that it belongs to engineering and not to QA.
2. Analyze the types of errors you are encountering and identify those that should have been uncovered with appropriate unit testing.
3. Schedule a review with management and present your data as objectively, but firmly, as possible. Point out that your responsibilities are being compromised by noncompliance with upstream SDLC activities. Augment your data with industry data that shows how much more costly defects are to fix the later they are detected.
4. Present alternatives, which should start with educating engineering in unit testing techniques and formalizing the process to assure compliance. Make it clear to management that unit testing is outside your scope and that it will cost time and resources to include it.
5. Finally, no matter what the outcome, institute root cause analysis for every defect, not only where it was introduced and discovered, but also where in the process it should have been detected. Then, make sure you include this information in every project status review.

I can't guarantee you will get everything you ask for right away, but if you continually arm yourself with data and educate management with ways to improve product quality, simple economics and the positive impact to customer satisfaction should eventually prevail. The earlier you uncover a defect, the cheaper it is to fix. *Really*. **{end}**

The Rules for Writing Maintainable Code

There are three fundamental rules that every software developer should take into account when creating code to support long-term maintainability.

by Kaushal Amin | kaushalamin@kms-technology.com

When a developer writes code, he imagines that he will be the only one working on it in the future. But the reality is that someone else will have to work on it. This may be due to a number of reasons: New functionality may be required, changes will be needed for existing features, and fixes for defects will need attention. The latter is a certainty. All of this work is often performed long after the original code was written and by a developer who did not write it. The challenge is to make changes without breaking the existing code. This situation can be complicated by the fact that there may be little technical documentation summarizing what the code actually does, and any future work will typically have tight schedule demands.

If we accept Robert L. Glass's assertion in his post "Frequently Forgotten Fundamental Facts about Software Engineering" for the IEEE Computer Society [1] that software maintenance accounts for 40 to 80 percent of total software development costs, then we can understand the importance of writing maintainable code from the start. Focusing on rushing the product out the door and failing to make code easily understandable for those who will work with it in the future dramatically increases the cost down the line.

> "It's a simple truth that the more maintainable your code is from the start, the longer its lifecycle will be."

Starting over from scratch because you're afraid that everything will break if you make too many changes is hugely disruptive and costly. It's a simple truth that the more maintainable your code is from the start, the longer its lifecycle will be.

The question is, how do you write maintainable code? These three simple rules will keep you firmly on the right track.

1. Write Code that Is Easy to Understand and Debug

If the next developer to work on your code can't understand what you've done or why you've done it a *specific* way, then they'll usually throw that code away and start over. It takes longer to understand poorly written code than to write new code from scratch. Write structured code with a clear format, follow conventions, and, if it isn't self-explanatory, make sure the code is fully commented. Here are a few best practices.

Choose a clear coding style: Keep your function and data naming consistent.

Optimize for the reader, not the writer: Saving time while you write code can cause serious frustration and confusion for anyone reading that code later.

Include concise comments: If it isn't obvious what's happening when you look at the code or you've implemented something a little unusual, make sure you include good comments to explain it. Don't write your comments for yourself—imagine someone else trying to understand your code cold.

Always do the smallest, simplest thing to add value: Always focus on the task at hand and write the best code you can to achieve your current aim. Don't do anything unusual. Writing code with one eye on future requirements is a recipe for disaster. Designing your code in a modular fashion with separate, discrete parts is much easier to understand.

KISS (Keep It Simple, Stupid): Don't assume the next person working on your code is going to be at the same level of understanding or experience. Write code that a novice can understand and leave out the experimentation and excessive optimization out.

Ensure good logging of code execution: Effective debugging requires good code logging. You need evidence of what was going on when the code was written. Log actions, entry points, exit points, and parameters, and make the code configurable. When you log from the beginning, it will be easier to pinpoint specific errors and the origins of those errors down the line.

2. Write Code that Is Easy to Modify and Enhance

To write code that is truly maintainable, it must be easy to add new functionality and features. Extensibility is vital. If a single change is liable to break the code in ten different places, then you're in serious trouble. It is possible to make your code easier to change down the line. Here's how.

DRY (Don't Repeat Yourself): Many developers have a nasty habit of writing code for one purpose and then copying and pasting elsewhere to do something else. If it gets used in multiple places and there's something wrong with it, then you've just multiplied the defect. If you're tempted to copy and paste code, consider extracting the common functionality to be available throughout your code base.

Separate concerns: You should modularize code based on distinct features that overlap as little as possible in terms of functionality. If the code needs to do fifteen things, then split it up into fifteen modules that each do one thing. Don't try to do all fifteen things in one module because that will make it tougher to make changes without breaking everything else.

Separate code and data: You should always externalize text into separate files. For example, it takes additional effort to isolate menu options and error messages into an external file, but if you put text in the code, it will be more difficult to change it later. Make sure you use a consistent nickname in language text file names. This approach enables text to be updated by nondevelopers without letting them near the actual code.

Avoid long statements and deep nesting: Don't write all your code in one big function because it's really tough to understand if it's performing too many tasks. In my experience, a single function that is more than a couple of printed pages long is way too long and should be subdivided.

3. Write Code that Is Easy to Test

Saving the best for last, a good suite of tests can serve as documentation, indicating how the code is supposed to behave while making sure that the code actually supports the expected behavior. Even better, great tests can give you confidence that your code still works after you've made your changes.

Automated unit testing should be implemented from day one so that when you make changes, the automated testing program will run and you can see what needs to be fixed immediately. In agile, even though it takes more time at the outset to write test programs and code concurrently, comprehensive tests should save major time and resources in the long run. **{end}**

Click here to read more at StickyMinds.com.
- References

index to advertisers

Display Advertising
advertisingsales@sqe.com

All Other Inquiries
info@bettersoftware.com

Better Software (ISSN: 1553-1929) is published six times per year: January/February, March/April, May/June, July/August, September/October, and November/December. Back issues may be purchased for $15 per issue plus shipping (subject to availability).


Cognizant
fastest
Rapid
Response.
Assured.
Experience the speed, finesse and precision in Quality Assurance.
fastest Cognizant's catalog-based testing ecosystem of choice.