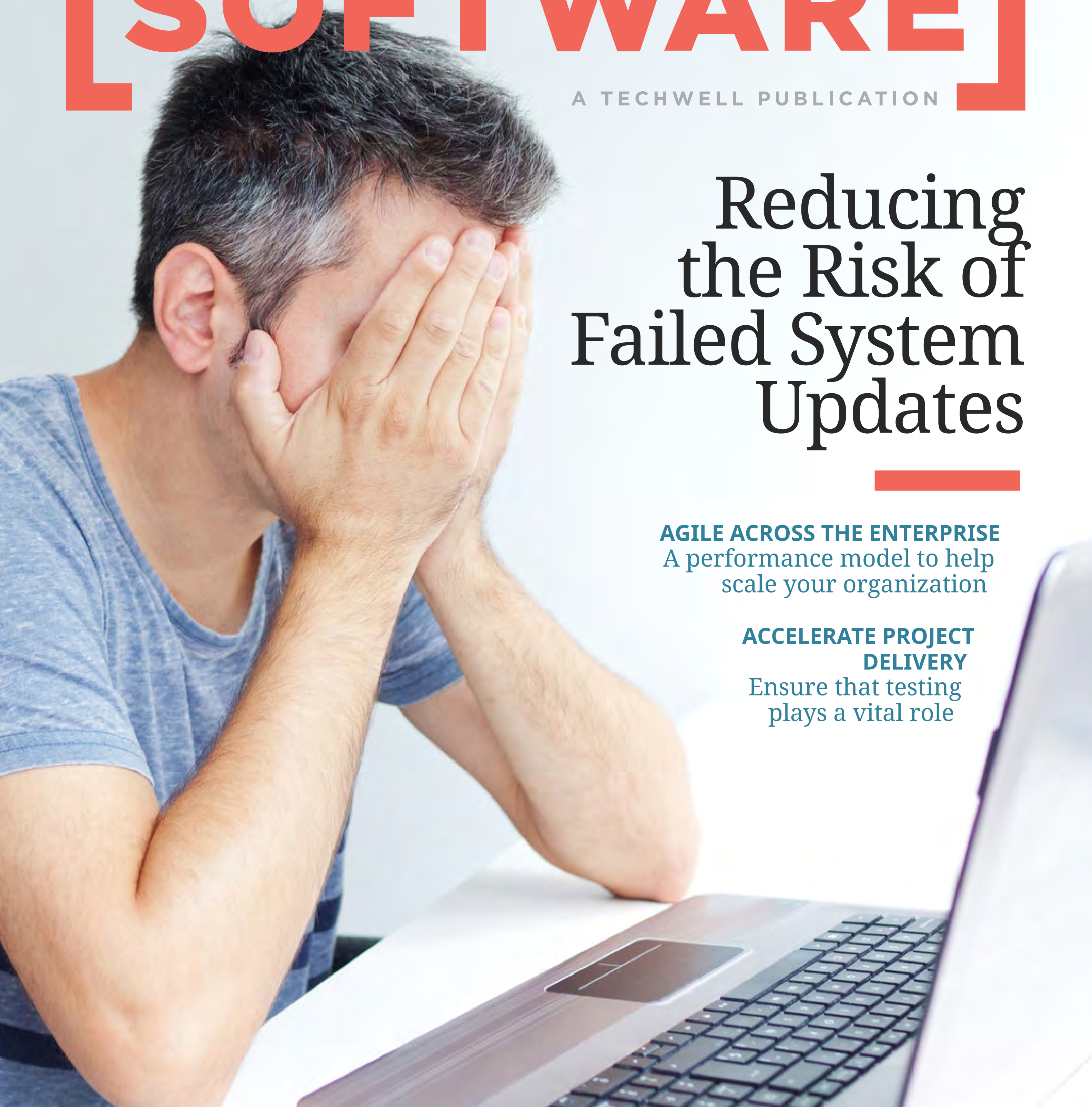
WINTER 2017

BETTER SOFTWARE™

A TECHWELL PUBLICATION



Reducing the Risk of Failed System Updates

AGILE ACROSS THE ENTERPRISE
A performance model to help scale your organization

ACCELERATE PROJECT DELIVERY
Ensure that testing plays a vital role

The Year's Most Memorable Software Testing Event

STAR EAST

A TECHWELL EVENT

Orlando, FL | May 2–7, 2017

Rosen Centre Hotel

TO REGISTER CALL 888.268.8770

https://well.tc/STAREAST2017

Conference Schedule

Choose from a full week of learning, networking, and more

Sunday Multi-day Training Classes Begin

Monday–Tuesday In-Depth Half- and Full-Day Tutorials

Wednesday–Thursday Keynotes, Concurrent Sessions, the Expo, Networking Events, and More

Friday Testing & Quality Leadership Summit, Women Who Test, & the Workshop on Regulated Software Testing (WREST)

EXPLORE *the* PROGRAM

Beautiful Downtown Orlando, Florida

STAR EAST
A TECHWELL EVENT

Dive deeper into topics at the tutorials

Learn from industry leaders at the Keynotes

Meet face-to-face with top solution providers at the Expo

Network with fellow testers

EXPLORE *the* PROGRAM

Gain new skills by breaking software in the Test Lab

TO REGISTER CALL 888.268.8770
https://well.tc/STAREAST2017

Meet Customer Expectations. Deliver Performance Anywhere, on Any Device.

To performance test web and mobile apps today, real user workloads must be accurately simulated.

Micro Focus can help you do this with sophisticated tools that replicate wide-ranging user access patterns, to make performance testing simpler and more effective.

www.microfocus.com/performance

MICRO FOCUS

INSIDE

Volume 19, Issue 1
WINTER 2017

MARK YOUR CALENDAR

SQE TRAINING
A TECHWELL COMPANY

Helping organizations worldwide improve their skills, practices, and knowledge in software development and testing.

Software Testing Training Weeks

https://www.sqetraining.com/trainingweek

February 27–March 3, 2017
Atlanta, GA

March 6–10, 2017
San Diego, CA

April 3–7, 2017
Boston, MA

June 12–16, 2017
Chicago, IL

Software Tester Certification—Foundation Level

https://www.sqetraining.com/training/course/software-tester-certification-foundation-level

January 31–February 2, 2017
Philadelphia, PA

February 7–9, 2017
San Francisco, CA

February 21–23, 2017
Houston, TX

February 21–23, 2017
Denver, CO

February 28–March 2, 2017
Tampa, FL

March 7–9, 2017
Toronto, ON

Fundamentals of Agile Certification—ICAgile

https://www.sqetraining.com/training/course/fundamentals-agile-certification-icagile

January 17–19, 2017
Your Desktop

March 7–8, 2017
San Diego, CA

April 4–5, 2017
Boston, MA

May 7–8, 2017
Orlando, FL

TECHWELL™ events

Conferences

Cutting-edge concepts, practical solutions, and today's most relevant topics. TechWell brings you face to face with the best speakers, networking, and ideas.

Mobile Dev + Test
IoT Dev + Test
A TECHWELL EVENT
April 24–28, 2017
San Diego, CA
LEARN MORE

STAR EAST
A TECHWELL EVENT
May 7–12, 2017
Orlando, FL
LEARN MORE

Agile Dev
Better Software
DevOps WEST
A TECHWELL EVENT
June 4–9, 2017
Las Vegas, NV
LEARN MORE

STAR WEST
A TECHWELL EVENT
October 1–6, 2017
Anaheim, CA
LEARN MORE

STAR CANADA
A TECHWELL EVENT
October 15–20, 2017
Toronto, Canada
LEARN MORE

Agile Dev
Better Software
DevOps EAST
A TECHWELL EVENT
November 5–10, 2017
Orlando, FL
LEARN MORE

EDITOR'S NOTE

Another Great Year for Creating Better Software

Another year and a new issue of *Better Software* magazine! Our sole goal is to give you innovative ideas so that you and your team can create *better* software.

A year ago, *Better Software's* parent company was officially rebranded from Software Quality Engineering to TechWell Corporation. The new name reflects our continued focus on the entire software development lifecycle. The TechWell website was redesigned, and now it is the magazine's turn. We've overhauled the look of *Better Software* with a simplified layout and a more modern look.

We have some great content, starting with Thom Denholm's article, "Reducing the Risk of Failed System Updates," which shows ways to build software that avoids the nightmare caused by failed software updates. Most of us have experienced "bricked" systems after an operating system update fails, but who would have imagined that the heart of reliability starts with the underlying file system?

There's no doubt that software development teams have embraced agility in their approach to projects. Kirk Botula, the CEO of CMMI, explains how to augment your organization's transition to agile using performance improvement techniques in "Helping Organizations Scale Agile across the Enterprise."

Sherri Sobanski's article "Alternate Testing Models: A Tale of Veggies and Precious Gems" searches for the magic formula to achieve product quality on fast-paced software projects. I promise you'll find her unconventional view of her QA journey both entertaining and enlightening. (Her article mentions Cucumber, a tool that was explained in "What Is Cucumber and Why Should I Care?" in our Fall 2016 issue of *Better Software*.) Justin Rohrman's "Testing as a Development Catalyst: Accelerate Project Delivery" shows how early integration and strong collaboration with the testing team can help deliver projects earlier.

We truly value your feedback. Let us and our authors know what you think of the articles by leaving your comments. I sincerely hope you enjoy reading this issue as much as we enjoy working with these wonderful authors. And please let me know what you think of our new layout.

Don't forget to spread the word to let people know about TechWell and *Better Software* magazine.

Ken Whitaker
kwhitaker@techwell.com
Twitter: @Software_Maniac

BETTER SOFTWARE™ A TECHWELL PUBLICATION

FOLLOW US

PUBLISHER
TechWell Corporation

PRESIDENT/CEO
Wayne Middleton

DIRECTOR OF PUBLISHING
Heather Shanholtzer

Editorial

***BETTER SOFTWARE* EDITOR**
Ken Whitaker

ONLINE EDITORS
Josiah Renaudin
Beth Romanik

PRODUCTION COORDINATOR
Donna Handforth

Design

CREATIVE DIRECTOR
Jamie Borders
jborders.com

Advertising

SALES CONSULTANTS
Daryll Paiva
Kim Trott

PRODUCTION COORDINATOR
Alex Dinney

Marketing

MARKETING MANAGER
Cristy Bird

MARKETING ASSISTANT
Allison Scholz

CONTACT US

EDITORS:
editors@bettersoftware.com

SUBSCRIBER SERVICES:
info@bettersoftware.com
Phone: 904.278.0524,
888.268.8770
Fax: 904.278.4380

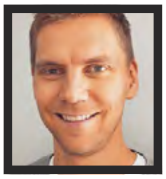
ADDRESS:
Better Software magazine
TechWell Corporation
350 Corporate Way, Ste. 400
Orange Park, FL 32073

TECHWELL™

CONTRIBUTORS

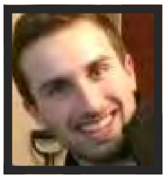
Kirk Botula is the CEO of the CMMI Institute, the home of CMMI, the globally adopted capability improvement framework that guides organizations in high-performance operations. Prior to the CMMI Institute, Kirk served as president of Confluence, a global financial technology firm. He is a global growth company executive whose career has been focused on advancing the common good through the commercialization of technology. For more information, please contact Kirk at info@cmmiinstitute.com.

Thom Denholm is an embedded software engineer with more than twenty years of experience in operating system internals, file systems, and specific knowledge of modern flash devices. His love for solving difficult technical problems has served him well in his sixteen years with Datalight. After hours, Thom works as a professional baseball umpire and an Internet librarian. Although he lives in Seattle, he has never had a cup of coffee. You can reach Thom at thom.denholm@datalight.com.

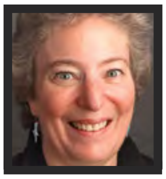


Joshua Dawson works for Three Rivers Technologies as a QA analyst in La Crosse, WI, with interests in agile methodology, training others in test management implementation solutions, mobile, and team integration strategies. Outside work you'll find Joshua with his family, creating and playing custom cornhole games, golfing, snowboarding, and caring for his pug. Please contact Joshua at jdawson@3riverstech.com.

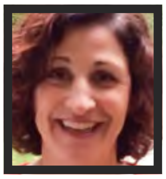
Justin Rohrman has been a professional software tester since 2005. In addition to being editor of StickyMinds.com, Justin is a consulting software tester and writer working with Excelon Development. He also serves on the Association for Software Testing board of directors. As president, Justin helps facilitate and develop programs like BBST, WHOSE, and the CAST conference. Contact Justin at rohrmanj@gmail.com.



Josiah Renaudin is a longtime freelancer in the tech industry and is now a web content producer and writer for TechWell, StickyMinds.com, and *Better Software* magazine. He also wrote for popular video game journalism websites like GameSpot, IGN, and Paste Magazine, and now acts as an editor for an indie project being published by Sony Santa Monica. Josiah has been immersed in games since he was young, but more than anything, he enjoys covering the tech industry at large. Contact Josiah at jrenaudin@techwell.com.



Johanna Rothman, known as the "Pragmatic Manager," provides frank advice for your tough problems. She helps leaders see problems, manage risks, and ease the way for teams to work better. Johanna just published *Agile and Lean Program Management: Scaling Collaboration Across the Organization*, her tenth book. Johanna writes columns for AgileConnection, TechWell Insights, and http://projectmanagement.com, writes two blogs on her http://jrothman.com website, and blogs on http://createadaptablelife.com. Contact Johanna at jr@jrothman.com.



Sherri Sobanski has enjoyed more than twenty years of experience in information technology, as it's the only profession that can keep her thoroughly entertained. She is currently the test manager supporting the global e-commerce website for LEGO Systems. Certified in a bunch of different IT and leadership things, Sherri aspires to leverage her learned knowledge to do wicked cool things for the betterment of IT. Sherri can be reached at sherri.sobanski@lego.com.

Saying No to More Work

THERE'S ALWAYS MORE WORK TO DO AND NEVER ENOUGH TIME. WITH A FULL PLATE ALREADY, HOW SHOULD YOU RESPOND WHEN ASKED TO DO MORE?

by Johanna Rothman | *jr@jrothman.com*

If you're like most people I know, it doesn't matter what approach you take to your projects—your manager has too much work for you to do. Instead of a potential career-limiting conversation, frame the conversation so you can show your manager you're considering his or her perspective.

Here are some options for how to say no and still stay on good terms with your boss.

Ask: "Can We Add More People?"

Does your boss expect you to solve problems alone without working with a team? That is a classic case of optimizing for *resource efficiency*, which is the least efficient way to manage (and deliver) knowledge work.

If you have expertise in a specific area of code, testing, or a certain function, such as database administration, your boss might think you are most efficient when you work as an expert. Your boss might even think that you, alone, can do this work. And that might be true—for *your* part of the work.

The problem occurs when we think of software product development as manufacturing. Manufacturing is a repetitive process. We don't learn as we manufacture.

All software is some form of product development. We learn as we develop the software. We need to build in time to learn together as a team to deliver a working product.

Think about how you perform your work. In all the software projects I've seen, people don't spend the bulk of their time writing code or tests. Rather, people spend their time thinking about the problem: including wrestling with requirements; redesigning the interactions inside the code; and understanding what the tests tell us about the code.

That means—regardless of your approach—you need to work as a team to finish work. Your boss might not realize this yet. In that case, explain *flow efficiency*.

Flow efficiency optimizes for finishing a chunk of work as it flows through the team. In contrast, resource efficiency optimizes for each person's strengths, so it creates queues of partly finished work waiting for the handoff to the next person down the chain.

When you suggest to your boss that you might need more people, explain how a team of people focused on finishing work can deliver finished features faster than experts working separately.

According to Brooks' Law, adding more people to a late project makes it later. I have often found Brooks' Law to be true in practice, especially if you work with handoffs, as in resource efficiency. [1] However, I have not seen the effects of this law when people work together to optimize at the team level to finish features.

If you can't add more people or organize in flow efficiency to help complete the work, try to limit the work.

Ask: "What Should I Stop Doing?"

It is entirely possible that your boss does not know all the work you are trying to accomplish. In that case, consider asking what you should stop working on. This works at the team level and at the personal level.

"It is entirely possible that your boss does not know all the work you are trying to accomplish. In that case, consider asking what you should stop working on. This works at the team level and at the personal level."

When I have too much work, I rank it and put some onto my personal parking lot. I don't forget the work; I postpone it, so I'm not thinking about it for now. When I'm ready to address work in my parking lot, I can reassess my decision.

You can do this for yourself, your team, and your organization. If there is significant customer demand for your products, you will have more work than you can easily complete. Separate the work you need to deliver now from the work you can do later.

Explain: "These Are the Risks I See ..."

Every so often I realize I have taken on a project with too much risk. I don't know how to start, or worse, I don't know what done means. And, for me, one of the biggest risks is that I don't see how to deliver this work in someone else's promised time.

If you are ever in that position, you can say to your manager, "Here are the risks I see if you want me to start on that work." Then, you can explain how you don't see how to do that work with all the other work you are supposed to complete.

Provide your boss facts without whining. Facts will help your case. Whining will not.

You might provide facts about other people's availability, the time it's taken so far to finish whatever you have done or not done, and the general riskiness of the project.

Sometimes, it's not about the project risks. You have too much work and your manager might not realize that. It's time to show your manager, not just tell him.

Visualize Your Work for Yourself and Your Manager

I use pictures to help my manager see what is on my list and when I might finish the work. I like two views of the data: a kanban board and a calendar.

You can start a kanban board with four columns showing the state of the work: To Do, In Progress, Stuck, and Done. If you need more states for your work, or you want to show the team's work, add the states you need. I often see these additional states: Waiting to Discuss, Waiting to Analyze, and Waiting for Testing.

In addition to the kanban board, a calendar view can show when you're supposed to finish the work. You might have two-week intervals on the calendar view. Sometimes, one-month intervals work. I do not like intervals longer than one month. I find that we interrupt ourselves with too much work if the intervals are longer than that.

Now, label the left column "In Progress" with the intervals across. At the bottom of the chart, draw a big, black horizontal line. Under that line, in the left column, write "Unstaffed."

Place a sticky note in each column of the projects you intend to work on during that interval. If you don't plan to work on that project in that interval, put a sticky in the "Unstaffed" row for the relevant column. Once you have a picture of the work, you can have a conversation with your boss when he asks you to perform more work.

Saying No Works

You do not have to say yes. You can say no in at least these four ways and help your boss become more aware of what you are working on and what is practical for you to accomplish. You might even make it a career-enhancing conversation! [BSM]

CLICK FOR THIS STORY'S REFERENCES

Agile Dev
Better Software
DevOps **WEST**

A TECHWELL EVENT

JUNE 4–9, 2017
LAS VEGAS, NV
CAESARS PALACE

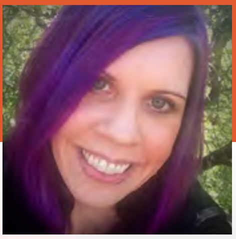
KEYNOTES ANNOUNCED!

Stamp Out Agile and DevOps Bottlenecks
Tanya Kravtsov
Audible

Modern Evolutionary Software Architectures
Neal Ford
ThoughtWorks

Lightning Strikes the Keynotes
Lee Copeland
TechWell Corp.

Big Data: The Magic to Attain New Heights
Ken Johnston
Microsoft


Identify Development Pains and Resolve Them with Idea Flow
Janelle Klein
Open Mastery

WHO SHOULD ATTEND?

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

CHOOSE FROM A FULL WEEK OF LEARNING, NETWORKING, AND MORE

SUNDAY Multi-day Training Classes begin

MONDAY–TUESDAY In-depth half- and full-day Tutorials

WEDNESDAY–THURSDAY Keynotes, Concurrent Sessions, the Expo, Networking Events, and more

FRIDAY Agile Leadership Summit

Special Offer for *Better Software* Subscribers: Register using promo code BSMCW17 by April 7, 2017 to save up to an additional $400 off your conference*

*Discount valid on packages over $400

TO REGISTER CALL 888.268.8770 | BSCWEST.TECHWELL.COM

"Software development and delivery has always been a race against time. But over the last several years, that race has entered an even more challenging phase."

"Software success is increasingly indistinguishable from business success. All business innovation requires new software, changes to software, or both. And business innovations can't wait for long software cycles to finish."

"DevOps is not 'something you do,' but rather a state you continuously move towards by implementing a culture of continuous improvement and by doing many different things."

"As far as the 'speed vs. quality' trade-off, with modern software practices, you should not have to make a choice."

Mark Levy

Years in Industry:	25
Email:	MicroFocusEmailList@vocecomm.com
Interviewed by:	Josiah Renaudin
Email:	jrenaudin@techwell.com

"Accelerating application delivery is the number one reason companies implement agile development methodologies but agile, by itself, is often not sufficient."

"With the explosion of mobile apps and low-switching costs, the business needs to deliver quickly to prove out business ideas and innovations."

"DevOps has proven that speed and quality are not mutually exclusive."

"As development teams transition to faster and more iterative development methodologies, the delivery cycle times will continue to shrink and continuously improve."

CLICK HERE FOR THE FULL INTERVIEW

DELPHIX

The market leader in data virtualization

Delphix data virtualization completes the modern DevOps stack by making it easy for organizations to enable automated, self-service data delivery that accelerates development.

Companies of all sizes use Delphix to:

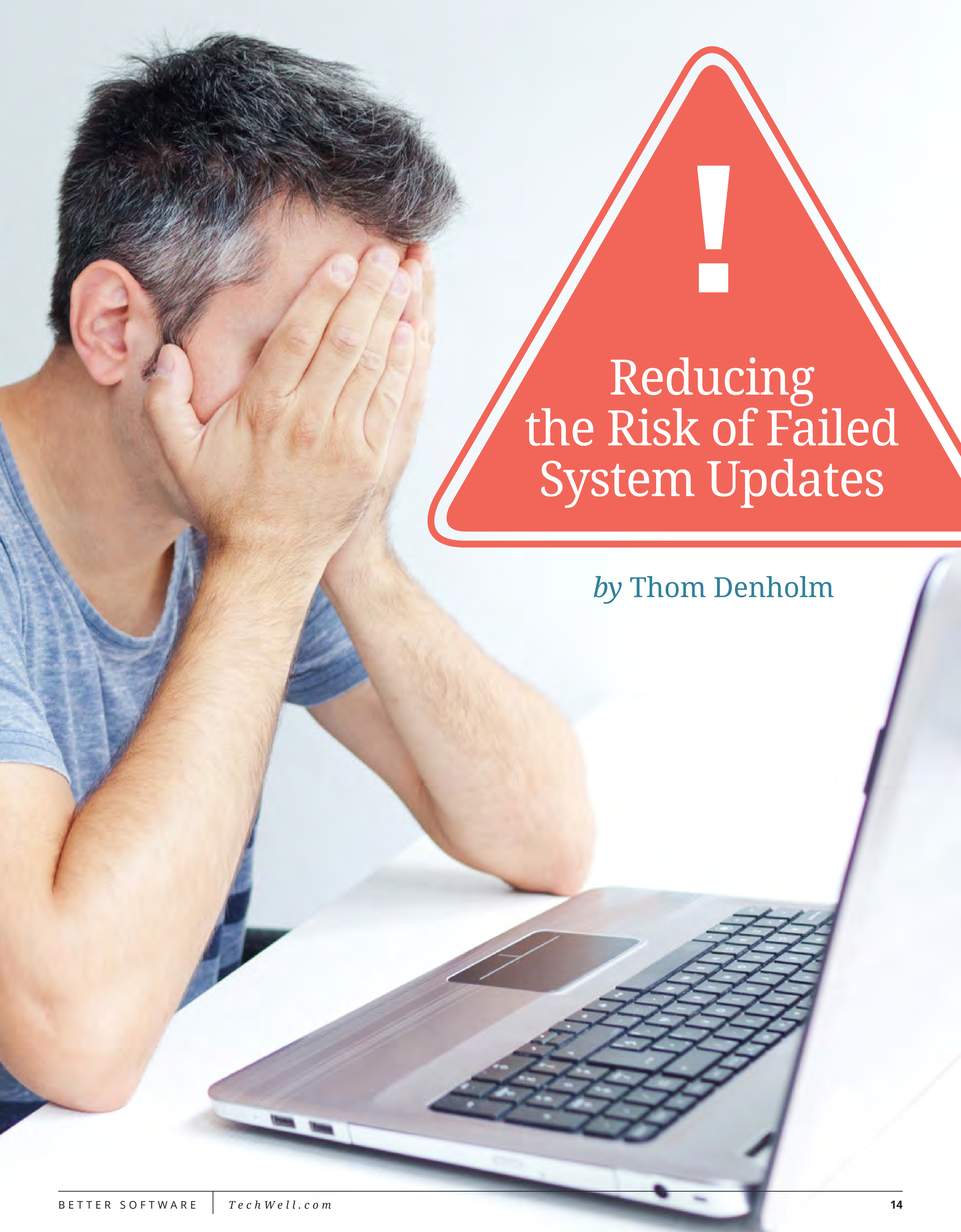
Branch and version data

Integrate with existing DevOps tools

Restore data in minutes

Learn how data virtualization can help your organization

Visit us at delphix.com



Reducing the Risk of Failed System Updates

by Thom Denholm

In our modern world, frequent updates of embedded devices (laptops, smartphones, wearables, and so on) are a reality. These run the gamut from minor bug fixes and security updates to feature releases and full operating system updates. Most of the time system updates take place seamlessly with minimal user knowledge of them happening. While this might seem like a nice convenience, lack of user notification can make it more problematic when a failure occurs. When a simple application update fails, it can be downloaded again. However, if something goes wrong during a software system update, a device can be rendered unusable or, in the worst case, completely dead—no better than a brick.

Some of these update problems are related to the installed update itself, some are problems with the update software or procedure, and others are related to the environment or update conditions. Most software companies do not release full details explaining why or how an update failed, although tech-savvy customers can come up with some guesses.

Challenges with Updates

Update failures happen more frequently than most users are aware of. When Apple released an iOS 10 update in early September, many iPhone and iPad users ran into installation problems.

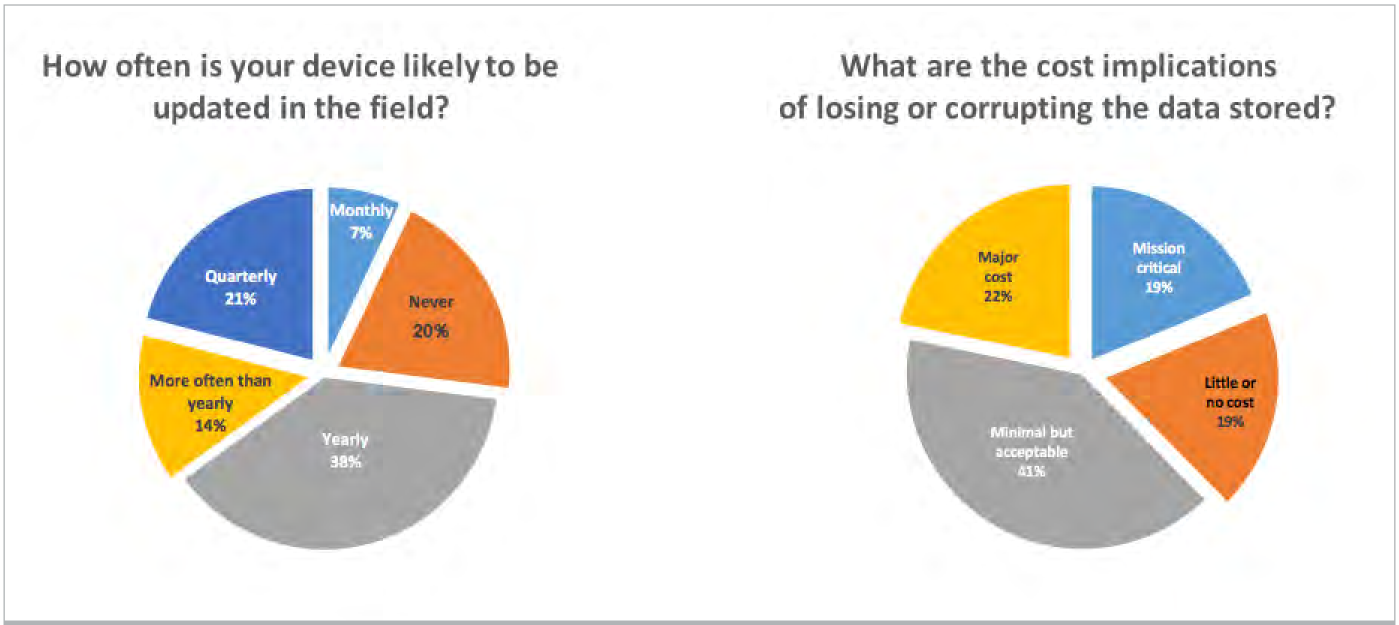


Figure 1: Survey results of the frequency desired for system updates in the embedded industry

For some, the update process failed, with a subsequent error message asking users to plug the device into a PC or Mac for a complete restore of the operating system. By mid-September, a web search resulted in 14 million hits for the keywords “iOS 10” and “brick.”

Fortunately, the problem was fixed later that day. These devices were all recoverable, so they didn’t technically fit the definition of a brick, but the initial bug proved to be a major headache for people who installed the update in the middle of their workday.

Not to be outdone, Microsoft has inadvertently caused repeated problems for its users with its Windows 10 anniversary update. During this update, many users would see the progress indicator progress all the way up to 100 percent before rolling back to the previous system settings in a process that took more than an hour to perform. Microsoft technicians have been working to solve this problem, but the recommendation from industry pundits in August was for users to block this update for the time being.

And if major industry leaders like Apple and Microsoft—with nearly unlimited resources—experience these difficulties, then no company is immune to update failures with their own software products.

The Real Cost of Failure

The company I work for, Datalight, partnered with Embedded Market Forecasters in its 2015 annual survey of embedded engineers to find out customers’ preference for frequency of software updates. Figure 1 compares how frequently developers expected updates to be performed (left) with their perception of the cost of losing stored data when an update occurs (right). According to the survey results, 80 percent of respondents expected updates for their devices at least yearly, 27 percent desired quarterly updates, and 20 percent specified their devices should not be updated at all.

When customers were asked about the importance of the data on these devices, a combined 41 percent considered system update failures to be of major importance *or* mission-critical. For them, a failed update comes with significant cost.

The Nest thermostat offers a recent example of a high-profile failed update in an embedded device. In November 2015, Nest rolled out a firmware release with devices updating to this version over a two-month period. But the update did not go quite as planned.

As Matt Rogers, cofounder and vice president for engineering at Nest, put it, "We had a bug that was introduced in the software update that didn't show up for about two weeks." In January 2016, for those devices that started to go offline, "that's when things started to heat up." [1]

Homes with a problematic Nest just got colder—a real problem in the northern United States and Canada in January. The bug caused the onboard battery to drain to a point where it was no longer able to control the furnace. Once discovered, Nest engineers created a second update, which could fail due to low battery on the device. Eventually, a nine-step procedure was produced to reset affected Nests, and the company even offered to send an electrician to homes of users who continued to experience problems. The cost to Nest was significant—not only in repairs but also in terms of their reputation.

In another recent publicized failure, Fitbit, the maker of the popular wearable activity trackers, ran into update problems. On Reddit, users reported having to attempt an update twenty or more times before it succeeded, or being forced to update through the PC because the Bluetooth update continued to fail.

The point here isn't to pick on any particular company. Technology analyst Alan Zeichick put it best when he said that our aim is to "point out the hazards of pushing out software updates, especially to real-world embedded systems. Sure, it's a pain if our phone app crashes due to an update, or if we have trouble starting a desktop app. However, in our increasingly connected world and the trend toward the Internet of Things, we developers simply should not place our customers in this situation." [2]

How to Avoid Becoming a Statistic or a Headline

Database vendors know how important it is to perform an operation perfectly only once. For example, the mortgage check funds need to be withdrawn from your account and deposited with the loan company in one atomic operation, which is reflected in the acronym ACID. A procedure should be atomic, consistent, isolated, and durable. Those same requirements apply to successful system updates.

Atomic is the first requirement utilized by database vendors. For an atomic update, all of the data is written (or modified) at once, with no partial operations. This is the case whether the system update involves a single or multiple files.

Consistency is also applied to the individual files, mainly by requiring that they be valid. In other words, no download errors or media bit flips are allowed. For that matter, changes should be validated after they are applied, perhaps by performing a cyclical redundancy check of the resulting files, reverting to the previous state upon *any* failure.

The *isolated* part can definitely apply here also. Performing an update while the embedded system is still writing other data to the media increases the likelihood of failure.

The final requirement of the ACID acronym is *durable*. Any update process should have the ability to recover from interruption. Most of the time errors in updates are due to a power failure, but it could also be a media error, failed cyclic redundancy check, or other exception.

Meeting those requirements for an update happens at the file system level, unless the device doesn't have a file system. A design with a minimal or no operating system is either executing code from RAM or a combination of RAM and system flash memory (EEPROM, NOR, or NAND). In the former case, new system data could be written to the storage while the device continues to execute the application in memory. A reload or system restart would subsequently begin executing the new updated system code.

With enough storage available, the device could download and check the system update before it is applied. If additional storage is insufficient, an in-place update could be attempted, but this presents a considerable risk. This same method would apply in a design with just one main system file. Increasing the number of files to be updated also increases the risk.

Using a separate update routine applied at system boot time can mitigate both kinds of risk. This is the method used by Android phones and some desktop computers today. Providing the user with an animated graphic of Andy the Android isn't necessary, but it is always a good idea to have a visible status update with more information than the old system BIOS update message: "Do not reboot or power down until the update is completed."

Using a boot manager and sufficient empty storage space means that the update can be thoroughly validated before it is applied. Many file systems will overwrite files in place, destroying

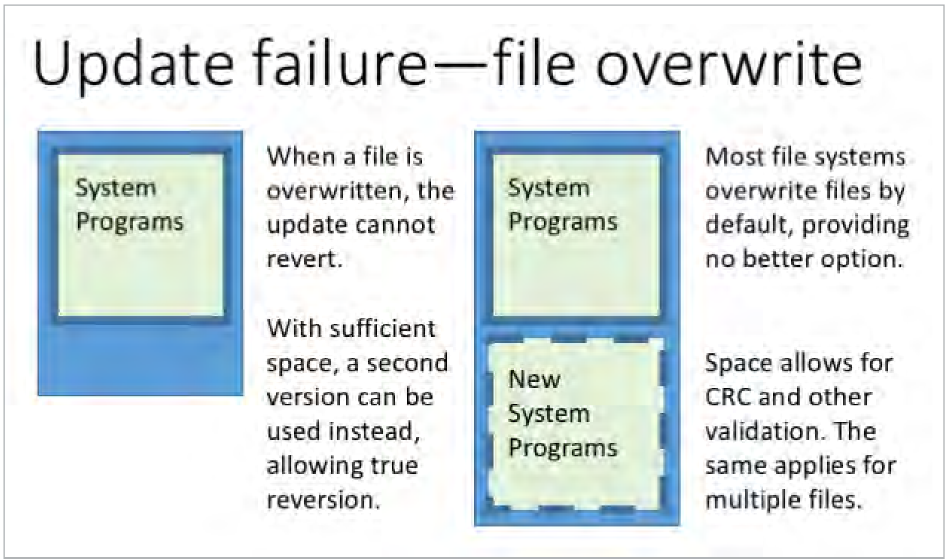

Update failure—file overwrite
System Programs
When a file is overwritten, the update cannot revert.
With sufficient space, a second version can be used instead, allowing true reversion.
System Programs
New System Programs
Most file systems overwrite files by default, providing no better option.
Space allows for CRC and other validation. The same applies for multiple files.

Figure 2: Update system failure due to file overwrites

the original data and making it impossible to recover from an interruption.

A system update manager can provide some protection against power loss by instead writing new files and then replacing the old files after validating the applied update. This would give some degree of protection against power loss during the update process.

For embedded devices, this solution can create a period of downtime, which is an option for some use cases but not for others. Sensitive to this problem, Google will use seamless updates for embedded devices partitioned for Android Nougat. For those designs, there are two system partitions: one active and one dormant. When an over-the-air update becomes available, the active partition downloads and validates it, and it is then applied to the dormant system partition. On the next reboot, the dormant partition becomes active and is fully updated. [3]

Linux BTRFS file system and Microsoft Windows use another related option, referred to as a checkpoint, where a system state is committed before updates start and then again after updates are complete. In some cases, this can allow a user to roll back to a previous state.

These desktop file systems point to an excellent alternative with a file system designed specifically to protect the data on a device. A transactional file system that follows the ACID design methodologies can be used in embedded devices. This provides the developer with complete control over how often data is committed. Using a transactional file system can simplify a reliable update process, saving the need to develop a separate update manager application or reserving storage space for a complete replacement operating system image.

A transactional file system has two metaroots, as shown in figure 3. Each points to the used and free blocks, but one contains the known good state of the media, while the other contains the working state. Unchanged blocks will be pointed to by both metaroots; pointers to blocks that have been changed are only located in one of the metaroots.

"Success is often fleeting. Succeed, and often no one notices. Fail, and everyone notices." !

A transaction point is used to identify the working state as committed, and after this it can become the new known good state. When a power interruption occurs, the system reverts to the known good state on the media, where blocks changed during the working state are still part of the free block pool.

In the context of system updates, one or several files can be

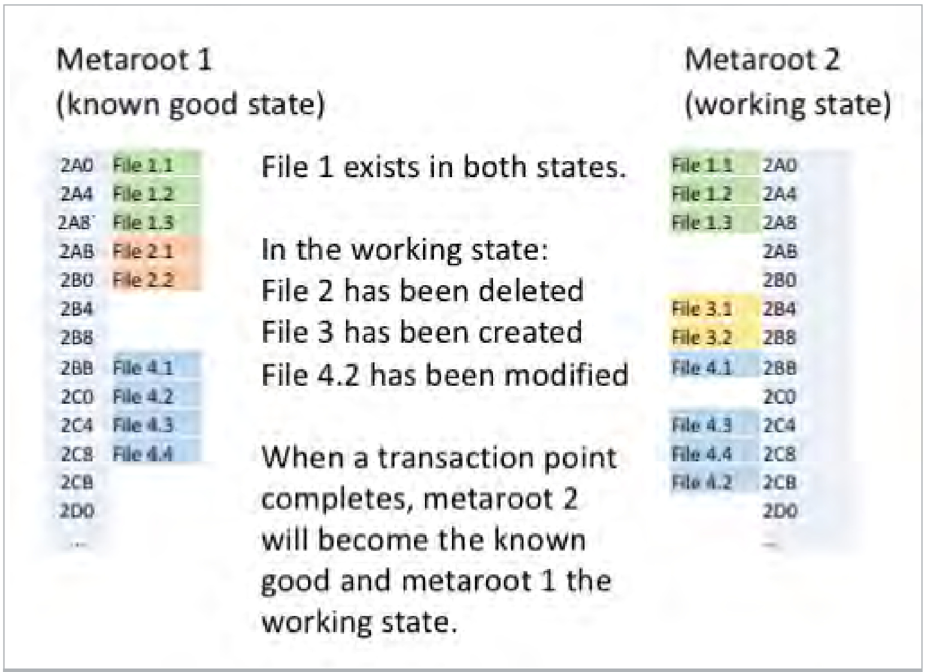


Figure 3: How metaroots improve reliability in a transactional file system

updated within a single atomic transaction point. When all of the updates are performed and validated, a single transaction point can commit this as the new known good state, allowing the system to continue from that point. In the case of any interruption, the system can revert to the previous state where no changes have occurred.

The New Normal

With the increasing frequency of updates and criticality of data, planning for a failsafe update process is now a requirement. This involves more testing of the updates themselves, including the impact of power failure during the update, and a better method for installing software changes. Systems that use a power failsafe transactional file system can simplify this process while using storage resources more efficiently.

Success is often fleeting. Succeed, and often no one notices. Fail, and everyone notices. Reputation is a company's currency, and it should be every software company's goal to avoid costly public failures that could lead to the tarnishing of their reputation. [BSM]

thom.denholm@datalight.com

CLICK FOR THIS STORY'S REFERENCES

TECHWELL™

Two In-Depth Conferences, 1 Great Location

Mobile Dev + Test
IoT Dev + Test

A TECHWELL EVENT

April 24–28, 2017

Westin San Diego
San Diego, CA

#MobileIoTCon

Register by Feb. 24, 2017
AND SAVE UP TO AN ADDITIONAL $400 OFF

MobileDevTest.TechWell.com

IoTDevTest.TechWell.com

Keynotes

Mobile Dev + Test
IoT Dev + Test
A TECHWELL EVENT

A Screenless Future Is Closer Than You Think
Dona Sarkar, Microsoft

A Coded Mind: From Ideas and Assumptions to Applications
TJ Usiayn, The Iron Yard

Enterprise IoT: Solving the Challenges of the Smart City
Bee Hayes-Thakore, ARM

Rooting Your Devices to Test Outside the Box
Alan Crouch, Coveros

JUST A FEW OF OUR IN-DEPTH HALF- AND FULL-DAY TUTORIALS

Swift Programming: From the Ground Up
James Dempsey, Tapas Software

Building Cross-Platform Mobile Applications with C# and Xamarin
Alan Crouch, Coveros

Super Rad Brainstorming
Jaimee Newberry, MartianCraft

Internet of Things: From Prototype to Production
Brian Huey, Sprint
Michael Finegan, MultiTech

Develop Your Mobile App Test and Quality Strategy
Jason Arbon, Appdiff, Inc.

Testing Mobile Apps in the Cloud with Selenium
Max Saperstone, Coveros

36 Concurrent Sessions—Choose Which Topics Matter Most to You

Mobile Testing

Mobile Testing: What To Automate and What Not to Automate
David Dang, Zenergy Technologies

Leverage Node.js for Mobile Test Automation
Stacy Kirk, QualityWorks Consulting Group, LLC

Plus 7 More

Mobile Development

Fun with Enterprise iOS
Joe Keeley, MartianCraft

A Taste of ES6 JavaScript: The Language and the Tools
Rob Richardson, Richardson and Sons

Plus 7 More

IoT Testing

IoT—Let's Code Like It's 1999!
Theresa Lanowitz, voke, inc.

Internet of Fun: Winning Ways for an IoT Hackathon
Alexander Andelkovic, King/Midasplayer AB

Plus 7 More

IoT Development

Harnessing IoT and Mobile Apps to Improve Driver Experience at BMW
Jorge Coca, BMW

Wireless IoT Network Communications: Now and into the Future
Michael Finegan, MultiTech

Plus 7 More

TO REGISTER CALL 888.268.8770 | MOBILEDEVTEST.TECHWELL.COM

HELPING ORGANIZATIONS SCALE AGILE ACROSS THE ENTERPRISE

by Kirk Botula

Whether you are seeking to adopt agile or are in the midst of a complete agile transformation, organizations like yours are increasingly seeking guidance from the Capability Maturity Model Integration (CMMI) to both strengthen and scale their implementation of agile approaches.

As an enterprise-wide performance improvement model that is the de facto standard model for improving quality and performance in the software industry, CMMI helps organizations reap the benefits of agile and scale its adoption across teams, divisions, and the global enterprise.

How CMMI Strengthens Agile

It's no secret that agile is well suited for software projects by encouraging improved team collaboration, transparency, and the rapid delivery of working software. However, its performance advantage often degrades as organizations attempt to scale agile beyond the team level and across the enterprise.

Most agile approaches are derived from a set of common values with high-level guidance for ceremonies, roles, timing, and artifacts. In contrast, CMMI provides a robust set of critical practices that can be used to strengthen agile methods and address behaviors not specifically defined by agile approaches.

CMMI defines the most important practices that organizations must demonstrate to build great products and services, and provides a comprehensive model that organizations can use to assess their capabilities against these practices.

CMMI does not define how to apply specific business processes. CMMI specifies what organizations must do to be successful, but not how. It provides flexibility for organizations to consider their own business environment and organizational context in determining appropriate ways to implement each practice. Most importantly, CMMI provides guidance for numerous behaviors, including those that enhance code quality, peer reviews, product integration, performance measurement, code reviews, large-scale estimation, quality management, and software version and revision control.

The CMMI model reflects best practices in organizational performance. CMMI is framework- and methodology-agnostic and is equally valuable for Scrum, Extreme Programming, V-model, Kanban, and waterfall environments. Organizations can implement business processes using their choice of methodologies or frameworks, including any agile approach. For high-performing organizations, CMMI also provides a set of practices for adopting statistical, data-driven analysis, along with the use of process performance baselines and models, which accelerates performance and dramatically improves the quality of software products.

In short, embracing CMMI will bring the resiliency and predictability required to reliably deploy agile across the enterprise.

The Benefits of Scaling the Agile Enterprise

A recent McKinsey study shows that incorporating agile above and across individual projects can be a formidable task. [1] Organizations that attempt to scale agile need significant structural changes and support. Defining measurable processes that apply to a broad range of behaviors helps technology leadership to scale agile across the enterprise. Agile organizations also adopt ways to increase discipline and improve consistency across projects, which help decrease go-to-market time, improve product quality, and better meet customer requirements.

Located in Bangalore, India, Minacs IT Services is comprised of 450 employees who provide leading banks with technology solutions and support. Internally, the group also supports over 20,000 employees in the business solutions and marketing optimization divisions. To meet current and future business requirements, the division was determined to reduce rework and ensure a faster time to market in their support of services.

Minacs IT Services realized that it needed a solution to help establish processes based on industry best practices for software development and service delivery—and integrate support functions, such as HR, training, and internal IT. The solution also had to integrate with an existing Scrum approach being used for product development projects. By applying CMMI with agile, Minacs IT Services was able to establish clear, defined, lightweight processes for service delivery and product development, enabling service-level agreement improvements and higher customer satisfaction. Specific measured achievements included a 7 percent quarter-over-quarter gross margin increase, 30 percent to 40 percent increase of sprint goal improvement, 30 percent increase in the number of features delivered in a sprint, and a 40 percent increase in on-time delivery.

In addition to these measureable benefits, the effort has transformed Minacs's internal culture from an organization of silos to an organization aligned with a single common vision. Minacs has since scaled its CMMI and agile integration across other Minacs offices, including the North America division and global internal IT functions.

Transferring Knowledge across the Enterprise

Poor knowledge transfer and lack of shared learning are major sources of organizational inefficiency and impediments to continuous improvement. Organizations must create and sustain an information-sharing and problem-solving culture across the enterprise by empowering teams to work through software, product, and management issues in a structured and methodical way. CMMI helps agile teams identify and share impediments early in the development cycle by making lessons learned and solutions available from previous team members within a collective knowledge base shared by the entire enterprise.

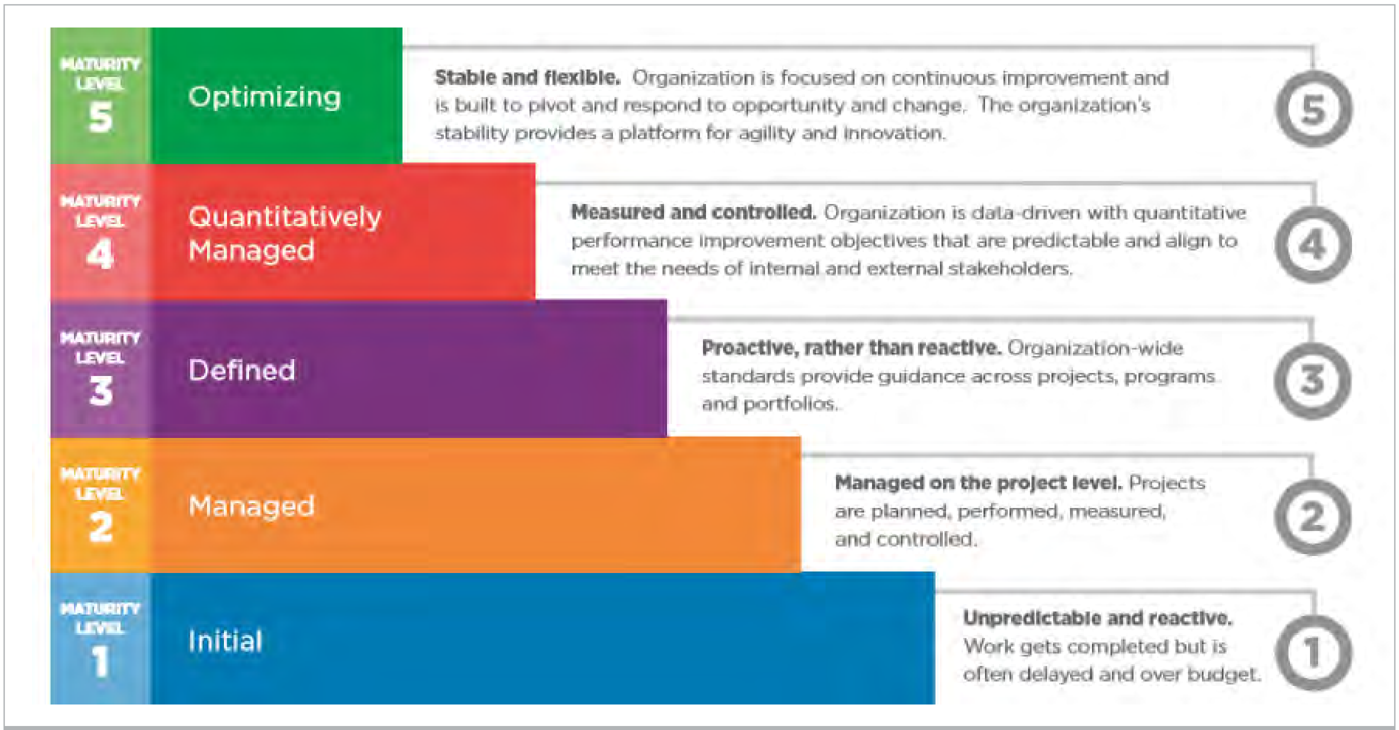
Honeywell Technical Solutions (HTS) is the development and engineering arm of Honeywell Inc. HTS delivers machine-critical products and offerings, so strong processes are required to deliver consistent, high-performance products.

To meet their ongoing business challenges, HTS sought to improve its problem-solving capability among its 7,000 engineers across multiple lines of business. Accomplishing this goal required standard methods of communicating, implementing, and transferring knowledge.

HTS identified several layered outcomes for this initiative. HTS wanted to define a simple, easily adopted problem-solving process. They needed to improve problem-solving accuracy and address root causes effectively and efficiently. Building on this, they wanted to improve systematic and layered problem-solving capabilities for each engineer, commensurate with their roles and responsibilities, training and competency development plans, and mentoring plans. They also had the overall goal of reducing the time engineers spent solving problems.

To achieve these goals, HTS deployed CMMI for Development's Causal Analysis and Resolution (CAR) practices to help the organization bring continuous improvement and best practices to its processes.

The consistency, repeatability, and transparency of the CMMI model helped HTS standardize practices, improve problem-solving abilities, share knowledge, and improve performance of engineering teams across the organization. The result was a 12 percent to 15 percent decrease in the occurrence of functional defects and 15 percent improvement in implementation of kaizen strategy over the past three to four years. This companywide program has resulted in a corporate culture that encourages collaborative problem solving and innovation, increasing knowledge sharing at each tier and shortening the learning curve for employees.



MATURITY LEVEL 5
Optimizing
Stable and flexible. Organization is focused on continuous improvement and is built to pivot and respond to opportunity and change. The organization's stability provides a platform for agility and innovation.
5
MATURITY LEVEL 4
Quantitatively Managed
Measured and controlled. Organization is data-driven with quantitative performance improvement objectives that are predictable and align to meet the needs of internal and external stakeholders.
4
MATURITY LEVEL 3
Defined
Proactive, rather than reactive. Organization-wide standards provide guidance across projects, programs and portfolios.
3
MATURITY LEVEL 2
Managed
Managed on the project level. Projects are planned, performed, measured, and controlled.
2
MATURITY LEVEL 1
Initial
Unpredictable and reactive. Work gets completed but is often delayed and over budget.
1

Figure 1: CMMI defines five maturity levels for organizations

CMMI Adoption in Agile Organizations Is Increasing Worldwide

Organizations that adopt a capability improvement model like CMMI improve agile deployment by scaling agile adoption across the enterprise, strengthening agile performance, and improving their capabilities through organization-wide knowledge sharing.

Adoption of CMMI in organizations implementing agile is steadily increasing. In 2009, 30 percent of CMMI-adopting organizations reported using one or more agile approaches. In 2015, more than 70 percent of appraised organizations reported the same. Multinational companies with technology centers in China, India, and Latin America are using CMMI to scale agile and export that capability into more geographically-distributed operations.

Where Does Your Organization Stand?

According to CMMI, there are five maturity levels that correspond to an organization's overall capabilities compared to CMMI-defined best practices. As shown in figure 1, a CMMI maturity level 1 organization is unpredictable and reactive. Projects are often delayed and over budget. A CMMI maturity level 5 organization is stable and flexible. The organization is focused on measurable and continuous improvement and is built to quickly pivot and respond to opportunity and change. The organization's stability provides a platform for agility and innovation.

The series of levels 1 through 5 encourages organizations to continually self-assess and improve their operations as they achieve higher maturity levels. Agile approaches alone are only sufficient for CMMI maturity level 2, and they begin to fall short as organizations assess practices across the enterprise in pursuit of CMMI maturity level 3.

Every agile software development organization should operate at a CMMI maturity level 3 or above to win in this highly competitive industry. To learn more about how CMMI drives agile performance, browse to http://cmmiinstitute.com/cmmi-and-agile. [BSM]

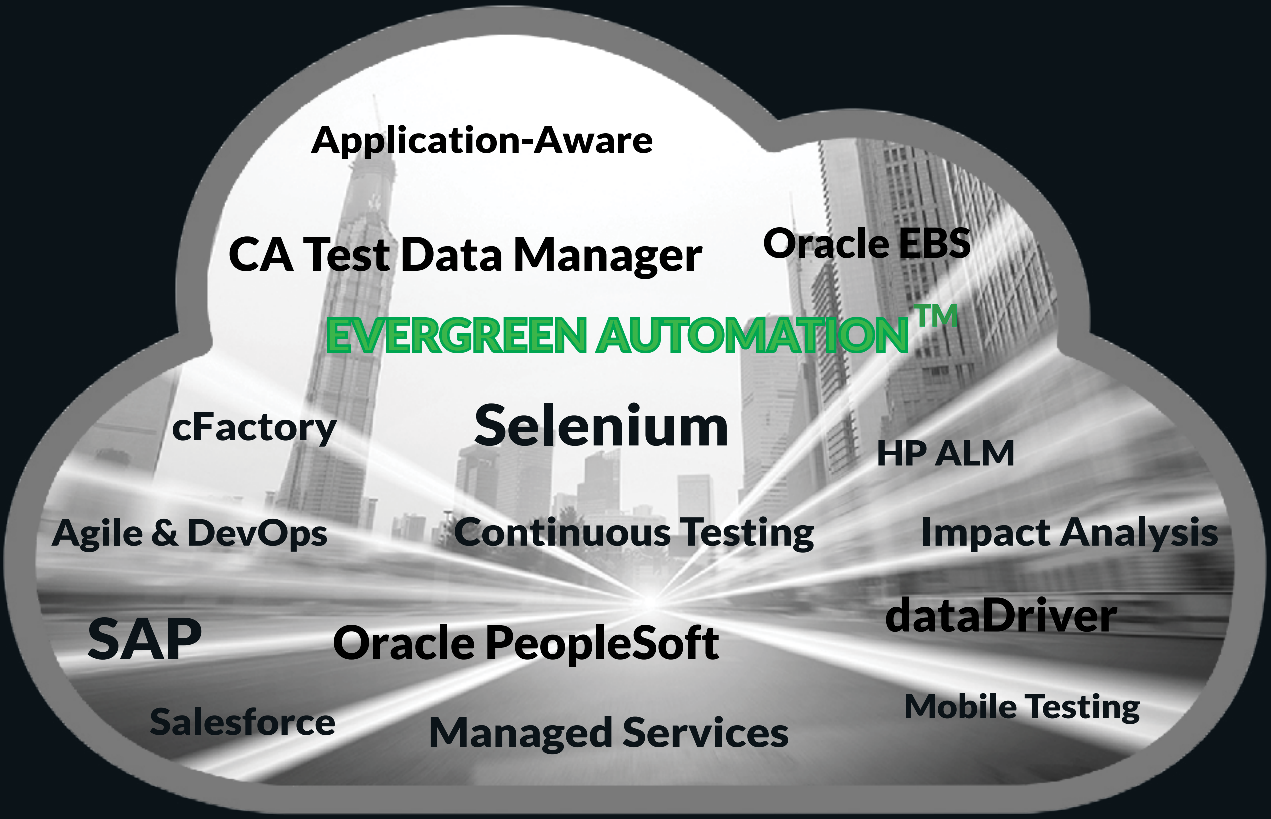
info@cmmiinstitute.com

CLICK FOR THIS STORY'S REFERENCES

turnkey

SMART TESTING. SMART DATA. BUILT FOR SPEED.

Continuous delivery with automated functional testing.
Because that's what technology is designed for.



Get a FREE, HONEST assessment of how automated testing can improve your outcomes.

turnkeysolutions.com/free-assessment

Slash Maintenance Costs by 80% | Increase Coverage by 90% | Cut Test Cycles from 3 Months to 3 Days

Digitally transforming your business?
To get it 'first time right',
there is a certain way.

In a fast-evolving marketplace which demands leadership that brings results, there exists a way of certainty: Tata Consultancy Services (TCS). With TCS as your strategic advisor and partner, the ever-changing new landscapes of business become new vistas of opportunity for transforming your organization.

Within this dynamic environment, does your QA and Testing function ensure you achieve your digital transformation goals the first time, every time? With TCS' Assurance Services Unit (ASU) delivering proven world-class enterprise testing experience and expertise, you can be certain of the quality and speed to market of your transformation initiatives.

Visit **tcs.com/assurance** and you're certain to learn more.
Or write to us at: **global.assurance@tcs.com.**

TATA

IT Services
Business Solutions
Consulting

TATA CONSULTANCY SERVICES
Experience certainty.

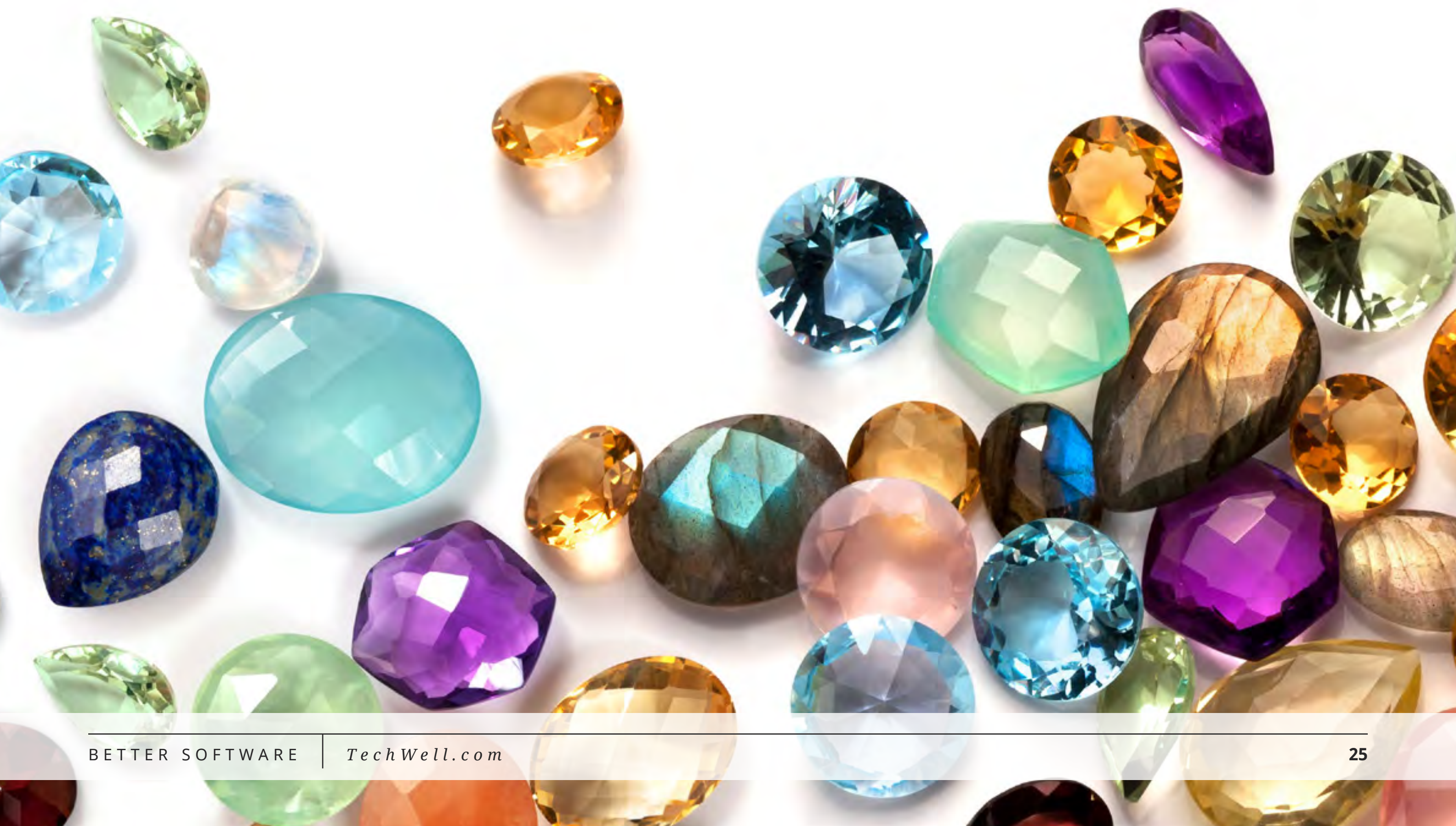
#ThinkAssurance Today!

ALTERNATE TESTING MODELS:

A TALE OF VEGGIES AND PRECIOUS GEMS

by

SHERRI SOBANSKI



I love software testing. Making sure that software rocks is something that makes me absolutely giddy, even after all these years of working in quality assurance. Every day I come in to work all jazzed up to break things, and I like to think I've more than earned my testing mojo. I can plot out edge cases in my sleep and find bugs most people only dream of.

Recently, I was bewitched by an alternate testing model. I'm going to take you on a journey that started nine months ago—a journey that promises to redefine the quality assurance world.

A Shock to the System

Allow me to set the stage. Our department had a relatively young test practice for our e-commerce website, and UI automation was steadily improving over time. The team was doing a great job within the boundaries they were given. Everything was checked off: agile methodology, Java-based automation, test cases, test scripts, and test plans.

At this point, our Scrum team had just finished a difficult upgrade to our e-commerce platform. It was so challenging that we had to put our dreams for a more responsive web design on the shelf. Smack dab in the middle of mourning responsive web design, our team was asked to travel to London to attend a project meeting.

We weren't sure what to expect, and we quickly discovered we were being schooled. A vendor was acquired to take charge and help us move forward with a completely new site design and responsive web design implementation. The perception was that we weren't cutting-edge enough, and the vendor was hired to execute the design and teach us their techniques in the process. It stung at the time, but it was the most efficient way to remove our boundaries and catapult us into a brighter, geekier future.

The core of the changes we were asked to take on were rearchitect, redesign, and become responsive immediately. Redesign and become responsive, no problem. We were already headed in that direction, and it was a comfortable user experience space. But rearchitect? That's a completely different story, one that ultimately sets the stage for our rapid evolution.

Decoupled Architecture Is a Game Changer

The goal to rearchitect primarily consisted of implementing services. So if your world is UI-related, how the heck does a service fit into that world? It doesn't. So how do you test it? Roll up your sleeves and learn to code.

At this point in my career I had mastered managing technology and enjoyed breaking software. However, I wasn't a developer by any stretch of the imagination. As a test manager, I would never push my team in a direction I wasn't willing to journey myself. So I volunteered to learn how to code and implement this new model.

We had been an agile shop since I joined the company, so the fact that we were going to be using that methodology was comforting. The new twist was that we were going to use behavior-driven development, writing our feature files in Cucumber using the Gherkin syntax and code in the Ruby programming language. We would implement continuous integration, leveraging Jenkins with a code repository in HG Mercurial being controlled by RhodeCode. In essence, this combination of methods and tools represented an entirely new way of working.

"AS QA TESTERS, WE AREN'T TYPICALLY DEVELOPERS, BUT WE AREN'T SCREEN JOCKEYS, EITHER. WE ARE SOMEWHERE IN THE GRAY OF THE GEEK PROFESSION."

We Covet the Gray

Why did we select this combination of tools? If you have never coded before, it can be a bit overwhelming. As QA testers, we aren't typically developers, but we aren't screen jockeys, either. We are somewhere in the gray of the geek profession. If we must code, it makes sense that we use tools that are based on simplicity and readability. We are business-oriented, consumer-focused technology warriors. In order to test, we must supply simple, highly readable artifacts that are easy to understand and maintain. Underneath that simplicity holds the more complex code structure that houses our geeky gray goodness.

Now, I know some of you will object that Java reigns supreme as a coding standard. Perhaps it does, but does that really make sense for quality assurance? Our test automation framework was developed in Java prior to this endeavor, but only a handful of people who were skilled in automation could maintain it. This pigeonholed team members into certain roles. With the new approach, there are no more specialists, just highly skilled engineers who happily live in the gray.

Adopting a Behavior-Driven Development Approach

Let's talk test structure and behavior-driven development. Everything in automation is about behavior. Four words are all you need for Cucumber written in the Gherkin syntax: *given, when, and,* and *then*. Here is an example:

Given that I am shopping at my local dollar store
When it's raining on a Saturday
And I have a purple store discount card
Then I purchase twelve bananas

The purpose of this syntax is to start with a readable frame and then turn the text into an automated test. Each line is then broken down into steps and methods that can be used to create the necessary automation. Before coding takes place, this general feature keeps you grounded and gives you lifelines back to the intended purpose. It describes tasks bit by bit rather than creating an overwhelming landslide of functionality.

I've found that an important benefit of this approach is reusability. Let's take the line "And I have a purple discount card." That one line can be used in any test where a discount card is relevant. And if you architect it correctly, this also applies to the code behind it and to the validation of results. I architected full tests that could be completely reused without writing a line of code. By leveraging

existing puzzle pieces and architecting the relevant test for my own purposes, we have reused our test suite more than a hundred times. That is just so beautiful it brings tears to my eyes.

So, where's the code? That purple discount card step will link you to your programming steps and, subsequently, to your Ruby method. Ruby is the backbone for this entire model, but Java or others could be used if desired. Cucumber is a Ruby *gem*, which, in laymen's terms, is a plug-in that allows Ruby to tap into that enhanced functionality. Ruby is known for its simplicity and ease of use, which, applied to QA, is just magnificent because we aren't *building* applications, we are *testing* them.

Testing with a Blindfold On

If you're a UI tester, testing services is very similar to testing with a blindfold on. There is no UI, so you are going to have to communicate differently with the system, which causes mild panic until you accept that your shiny front-end application is no more. With that enlightenment comes the realization that all the UI rules to which you were once bound no longer apply. That means I can break things piece by piece and in all kinds of crazy ways, sort of like a hacker would. Big fun!

The front-end point-and-click is now replaced by automation code. A test harness is helpful, but if you want repeatability, you roll up your sleeves and start coding. It takes some getting used to, but Ruby is a forgiving language and adapting to it doesn't require weeks or months in a classroom. Just dive in, mess things up, and learn hands-on.

Avoiding the Big, Bad Test Set

When we first started this journey, we wanted to test everything. That's a lofty goal, but we didn't lose sight of where we were headed. In a couple of months we had more than five thousand tests running. If we didn't structure our folders and test hierarchy carefully, it would have been unmanageable. Refactoring, changes in approach, and even deleting things at times were a perfectly natural progression.

We reserved ourselves to the organic nature of our test set and allowed it to keep evolving. Easily understood folder structures and naming conventions went a long way toward keeping our sanity when the test sets bulked up. We tried very hard to keep it clean and simple so chaos didn't reign.

Long Live Continuous Integration

With all this automation goodness going on, it would be a downright shame to not automate the execution of the full test lifecycle. In our case we are using RhodeCode as the code manager and Mercurial as the code repository, but these can easily be substituted with other tools if desired. Jenkins comes into play as the orchestrator of the code builds and test execution. Currently, our full automation kicks off every time our QA environment is built with new code. We control the environment builds as well, so we basically have free reign with the automated execution. This allows us to run the test suite at will or in a scheduled fashion. With Jenkins, all I have to do is click four times and the environment builds and automation executes. It couldn't be simpler.

Fast Forward to the Present

Looking back on the past year, I am amazed at what we were able to achieve in such a short period of time. Our previous testing environment was mostly a manual test shop with an automation suite that only specialists could manage. Our new way of working embraces a new test framework, decoupled architecture, and employs full automation coverage with continuous integration—all with a small team of QA engineers. Our headless tests number around nine thousand and run to completion in approximately two hours. It's magical and exciting, and I sleep better at night knowing that my coverage is just shy of 100 percent.

So open yourself up to the experience of alternate testing models. You may find yourself bewitched and experiencing a whole new level of QA enlightenment. [BSM]

sherri.sobanski@lego.com

BUNDLE AND SAVE

ON SOFTWARE TESTING TRAINING WEEKS

When you attend a Software Testing Training Week, you can choose from up to 10 specialized courses developed by some of the industry's most respected and knowledgeable testing/QA professionals.

New Course Bundles

We've bundled some of our most popular courses to give you special pricing on the training you need. Choose one of our course bundle packages and save 10% off the cost of single courses. Or, create your own custom schedule and save 10% when you register for two or more courses.

Software Testing Foundations Bundle
Software Tester Certification—Foundation Level + Mastering Test Design

ISTQB Certification Bundle
Software Tester Certification—Foundation Level + Agile Tester Certification

ICAgile Agile and DevOps Certification Bundle
Fundamentals of Agile Certification—ICAgile + Fundamentals of DevOps Certification—ICAgile

Agile Testing Foundations Bundle
Fundamentals of Agile Certification—ICAgile + Agile Tester Certification

Agile Testing and Automation Bundle
Agile Test Automation—ICAgile + Agile Tester Certification

SEE ALL 10 COURSES *and* LEARN MORE AT SQETRAINING.COM/TRAININGWEEK

2017 Software Testing Training Weeks

Atlanta, GA	February 27-March 3, 2017
San Diego, CA	March 6-10, 2017
Boston, MA	April 3–7, 2017
Chicago, IL	June 12–16, 2017

New Super Early Bird Pricing

Save up to $250 on our standard pricing when you register 8 weeks before any Software Testing Training Week with our Super Early Bird pricing.

SQETRAINING.COM/TRAININGWEEK

SQE TRAINING
A TECHWELL COMPANY

Testing as a Development Catalyst: Accelerate Project Delivery

By Justin Rohrman

Historically, the programmer owned the only technical role on a software team. These people often took the role of a one-person band: programmer, tester, and product person.

As the software development industry matured, software testing became a specialty role, performed independent of programming. This tide change led to people studying testing as a completely separate activity and role from programming. This eventually resulted in the creation of independent test groups. Today, I see these independent programming and testing roles slowly shifting back together, much like tectonic plates. Regardless, the goal of providing better testing service hasn't changed.

"COLLABORATION HAS ALWAYS BEEN A CORNERSTONE OF SUCCESSFUL SOFTWARE DEVELOPMENT."

A catalyst, someone that can help developers make better software faster, can come from any number of unlikely sources. Making testing a development catalyst can be simple, and most people know how to do that, at least intuitively. Learn to work together, and become technical enough so we can participate in every part of the development process.

Let's examine why teams that are optimized for radical collaboration and technical work are able to build great software at a sustainable pace.

The Importance of Collaborative Work

Collaboration has always been a cornerstone of successful software development. But achieving a collaborative culture is more complicated than we like to imagine. The agile literature claims that putting groups of technical people together at a table works like some sort of magic pill. The truth is—if either of these things worked as well and as often as we like to think, software would be delivered with high quality every day.

Collaboration on projects fails in at least one of three ways.

Objections by management: Several jobs ago, I was working for a company making a software product designed to help companies price commodity items. Our project was running late, with entirely too much work left to complete. I wanted cursory testing to be performed before code was committed to our code repository. Because builds took a couple of hours, this approach should save time. A development manager saw what we were doing, shooed me off, and had a not-so-pleasant conversation with the test manager. "How can we get these bugs fixed if testers are always interrupting our programmers?" As a result, I was banished to the testing cubicles. This was followed by a "do not disturb" policy email sent to our programmers. Ugh.

That manager was rejecting two teammates who wanted to work together in the name of productivity. He was completely ignoring the time sucked up by builds (that ultimately will fail) and a more effective find-fix-retest cycle.

Personality differences: Another company had a manager who acted mostly in a facilitation role. He was happy to let the programmers work and interact however they wanted as long as the product was getting delivered on time and the customer was happy. Developers occupied space in the back corner of the office. The lights were low, the blinds were closed, and it was very quiet. People were working separately on their own computers in their own cubicles.

This office arrangement tended to suggest a "leave me alone" and "I don't have time to help you" culture. For highly reactive [1] or introverted people, being in a noisy room around other people is often uncomfortable and a drain on productivity. It was rare to find programmers working together on code, even if a programmer were stumped for several days.

Questionable effectiveness: A year ago, a client proclaimed that their teams regularly do *mobbing*, a development strategy where a team of people work on the same programming problem one at a time. I had read about mobbing but had never actually witnessed it. I was super excited to see it in practice. Instead, what I witnessed was a group of people sitting around one monitor. One person was working on code, one person was watching over her shoulder, a few people off to the right were having a separate conversation, and one uninterested person in the back was playing on his phone.

My observations could have been related to the Hawthorne effect, where people change their behavior if they are being watched. [2] Regardless, the mobbing I observed seemed suboptimal. Was the resultant code a byproduct of the mob, or was it due to the most extroverted person winning arguments in a group setting?

The stories I hear about successful pairing and radical collaboration—where two or more people work on a software problem continuously until it is delivered to production—come from places where that is built into the culture. For example, interviewing candidates for a job can be effectively performed in pairs. This allows the hiring company not only to assess a skill and culture match but also to show a candidate's ability to facilitate work and exchange power back and forth. Whenever code changes are made, it is also done in pairs. There is no intermittent collaboration, getting together randomly to solve a specific problem. That is just how the work is done, and to do otherwise is unthinkable.

Despite difficulties with team collaboration, groups built on radical collaboration, like constant pairing, can be wildly successful. Something special can happen. Typical low-hanging fruit for testers, such as boundary conditions and layout issues, disappears. These types of bugs can be discovered in code while a test expert and a programmer work closely together. These groups inevitably need fewer, but more skilled, testers in that dedicated role.

The Role of the Technical Testing Expert

The role of an independent tester is like a net below a circus trapeze. Ideally, anything big that falls should automatically bounce back up. A technical tester working closely with developers is more like the harness that keeps the trapeze artist from hitting the net. This role has four key characteristics, all of which offer great benefits to the software development process.

Working alongside developers: Technical testing is a spectrum. It starts with basic skills, such as opening developer tools in a web browser, and ends with the ability to read and write code. My best experiences doing technical testing work involved building test automation in parallel with a developer writing a new feature. We didn't have any product management at that company, so discovering requirements was an exploratory process by the team. The developer and I sat next to each other. As he wrote code initially creating APIs, I asked questions and stubbed out tests using a JavaScript framework called Frisby. Throughout this collaborative process, we discovered missing data, incorrect data types, authentication problems, and unusable workflows. A nontechnical tester might have to wait until the feature was nearly complete and a UI was in place to access those parts of the code.

Code was changed, tested, and packaged days before a front end was even available.

Rather than working in a function mostly independent from the developers, technical testers act as technical contributors with developers. Whenever a new code change is being made, the tester is either making the change or acting as an adviser, questioning how the product will be used.

Coaching: Most of the code quality tools available (unit testing, behavior-driven development, test-driven development, and so on) help developers confirm their own assumptions by performing very narrow and simple checks. The technical tester-as-coach role can help drive overall product quality rather than a focus spent on typical testing tasks.

The coaching role could teach testing skills to developers. For example, this person might visit a development team and run through a game with testing themes. At the end of the game, the tester would debrief and point out the testing skills that were used, such as note-taking, experiment design, and observation. These lessons at least seed the idea that there is more to testing and writing code than confirming what you already know.

Everyone on a technical team tests software to some degree; this testing coach can help developers become aware of that activity and focus on developing that skill.

Continuous integration, continuous delivery, and monitoring: This is outside the boundaries of what most people think of as the role of a tester. Continuous integration, continuous delivery, and monitoring are tools for reducing project risk and lowering defect exposure.

A trend I see these days is when a company wants automation yet stumbles on making it useful.

Several years ago, I was working on a UI automation framework built using Watir. We had a small piece of framework, just big enough to provide the DSL to build a couple of tests. Developers responsible for the build process had other higher priority work than getting these new tests running after each new build. As the framework grew, the number and complexity of tests grew along with it. Eventually we had an unwieldy beast, and no one wanted to spend the time getting it in the build system or dealing with the pain of discovering everything we would have to change to make it useful and relevant.

When I do this sort of work now, I build enough framework to get one test running. Then I work with developers to get the appropriate packages running in Jenkins so that tests can run with every build.

Automation that isn't hooked up to the build is often forgotten or not acknowledged by the development team as something useful. Ideally, your build system is encouraging programmers to make smaller changes that can be released more often.

Mastering one skill: Any time spent focused on learning to write code is time not dedicated to testing. Most technical testing experts I work with spend significant time becoming skilled in one area, like programming, and later take on a second skill. This is like growing up in one country, and then moving to another and learning a new language. Maybe you never master the new language, but you become proficient, and that should be enough.

The One–Two Punch: Collaboration and Technical Skills

There is a circular relationship between collaboration and technical skill for software testers. A tester who successfully collaborates with developers will probably learn the technical skills needed to enable deeper testing and faster product releases. A tester who has technical skills to read code, inspect a DOM, search through a database, and write code will probably have an easier time contributing to the product development process. Technically proficient testers usually have an active role in releasing software more frequently.

For testers, collaborative work and technical skill aren't islands. One must be there for the other to thrive.

Testers who once were on independent teams are now commonly embedded in a development team. Strategies that worked then are now a liability. Blending collaborative and technical skills is the key for testers to become a development catalyst in these new teams. [BSM]

rohrmanj@gmail.com

CLICK FOR THIS STORY'S REFERENCES

Featuring fresh news and insightful stories about topics 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. The following is a sample of some of the great content you'll find. Visit TechWell.com for the full stories and more!

Why Your Software Team Shouldn't Aim for a Five-Star App Rating

By Jason Arbon

I hope your app doesn't have five stars.

I'm not being rude. It's just that when your app has five stars, there is nowhere to go but down. Star ratings are important, but many teams are so focused on that metric that it is killing their apps and hurting their business. Having a five-star rating stops innovation, puts teams under pressure, and can even get you fired.

It is an open secret that the best apps from the best companies don't have five stars. Facebook is doing just fine these days, spending millions of dollars each year on testing, and their app hovers around four stars on the Google Play Store. Google Search, Google Maps, and Snapchat also have four stars.

https://well.tc/w422

The Art of People Facilitation: Servant Leadership and Team Dynamics

By Robert Woods

Servant leadership has become a popular term in modern business. Many managers claim to understand what that means, preaching open-door policies and the "I'm just here to help you out" philosophy. But when the rubber meets the road, they simply lack the skill set to carry through on the concept.

Perhaps it's generational. I was describing my job to my seventy-two-year-old father, who had been an ERP implementation manager for twenty years, and we talked about the concept of servant leadership. His gut response was a grunting, "I don't like that term, *servant*."

https://well.tc/w42s

Test Automation and When Enough Is Enough

By Josiah Renaudin

There are seemingly endless reasons why your test team might need to make use of automation. Whether you're working with a web or mobile app that's expected to bring in thousands of users simultaneously, your application contains code that's changing frequently, or you just have piles of tests that need to be run, automation can be invaluable to your success.

However, even if we're at a point where most people agree you need some sort of test automation to compete in today's rapid, agile landscape, using automation indiscriminately will lead to way more headaches than benefits.

https://well.tc/w42e

Changing the Narrative: Using Storytelling in Software Testing

By Isabel Evans

Stories are so powerful. They change our worlds, they change how we think, and they change how we perceive our surroundings. Even testers can use stories in several ways.

There are monsters that snarl at meetings and threaten terrible consequences. There are magic tokens that will solve all our problems. There are impossible tasks to achieve, with fairy-gold rewards that follow. We tell ourselves stories, then watch our narratives come true.

https://well.tc/w42m

Reviving the Master Test Plan in the Age of Agile

By Michael Sowers

Retro is back: Vinyl is in, and clothing styles are a throwback to decades-old flares, fringe, and suede. Of course, Jimmy Buffett seems to have spanned generations, but I digress!

One of the more retro—or, in this case, I prefer "traditional"—software development and testing approaches is the master test plan. The idea is that there is an overarching plan describing all the testing activities for a project or product, with subplans focused on specific testing activities such as integration, system, and acceptance testing, called detailed test plans.

https://well.tc/w42n

Pushing System Performance with Stress Testing

By Dale Perry

In performance testing, you hear people talking about "stress testing" the system to make sure it performs correctly. The challenge comes when you ask them what exactly they mean by "stressing" the system. In performance testing, the term "stress" can have several possible meanings and can represent several different types of stress.

In a typical performance test using an operational profile to place load on the system, the focus is on identifying resources that begin to reach their maximum usable capacity within the system at varying load levels (volume). In load testing, if you were running an average load test and memory utilization began to approach 75 percent to 80 percent, memory is now under "stress," as it is becoming a bottleneck on the system. Increasing load on the system will push this resource beyond acceptable stress limits.

https://well.tc/w42h

Why Process Standardization Is a Terrible Idea

By Johanna Rothman

One of my colleagues wants to standardize all his agile teams on one process. He happens to like iterations, so he wants everyone to use two-week iterations. He wants them to use Scrum rituals and ceremonies.

I understand what he wants to accomplish: gaining the ability to look across the projects and see the same metrics, when teams are stuck, and progress across the organization. That makes sense.

But the teams do not work on similar projects. Some projects have interrupting work in addition to their project work. Some teams aren't cross-functional, so they can't deliver features on their own. Some teams are geographically distributed, so the Scrum ceremonies and rituals don't work because the team members are in too many time zones.

https://well.tc/w427

The Myths behind Brainstorming, Open Office Plans, and Collaboration

By Linda Hayes

More and more companies are moving to some version of open offices or pod configurations in the hopes of inspiring collaboration and improving productivity. But does it work?

If you ask the vendors who sell these new seats, of course they say yes. There are also anecdotal stories from companies that extol the cost savings and the odd inspiration or connections that arise from mixing employee seating up.

But if you ask the scientists who actually conduct objective, rigorous studies, the answer tends to be no. In fact, reducing privacy and increasing proximity is shown to decrease productivity, and it can even harm employee health.

https://well.tc/w42b

Make It Easy for Your Customers to Provide Feedback

By Naomi Karten

Soliciting customer satisfaction feedback can be a heart-pounding experience. Once you know your customers' feedback, you have to either act on it or deliberately ignore it. So maybe it's not surprising that the way some organizations request feedback ensures they don't get much of it. This seemed to be the case at some of the hotels I stayed at during a recent trip.

Two weeks after staying at the first hotel, I received an emailed request for feedback and a link to the web-based feedback form. But by this time, I'd been to several other hotels and this one was at best a blur. Timing matters in soliciting feedback. If you really want that feedback, ask for it soon enough after the delivery of the service that customers can still remember it.

https://well.tc/w426

What Is Continuous Delivery Doing to Software Testing?

By Justin Rohrman

Most of the conversations I've heard about testing over the past few years are related to its death. (Or supposed death.) Some people go as far as saying that having a tester role is an antipattern for software development—something that points to organizational dysfunction.

Software teams using continuous delivery focus on building software in small pieces so that new code can be pushed to production multiple times a day instead of on a sprint cadence. There is also an explicit focus on code quality before production and on monitoring afterward.

So, what is this doing to testing?

https://well.tc/w428

When It's OK to Ignore Company Policy

By Lee Copeland

A few days ago, my wife got a new debit card from Wells Fargo, the kind with the chip in it for better security. When she used it at the grocery store, it was declined. Embarrassed, she came home and told me the story. I told her I would call the bank and find out what had happened.

When I called customer service, I pointed out that there was plenty of money in the account, my wife had activated the card properly, the account had been active for thirty-eight years, and I was joint owner of the account. But when I asked what the problem was, the customer service representative would not discuss it with me because although it was my account, it was not my card. She cited "policy" as the reason. I asked to speak with her supervisor, who parroted the same answer. It was "policy." They could only speak with the cardholder.

https://well.tc/w42u

The Importance of a Dynamic and Open Culture in the Workplace

By Rajini Padmanaban

A culture that is carefully built and nurtured in an organization is key to its long term success. While every organization has its own culture and there is no right or wrong answer to which culture should be fostered, a culture that is open and dynamic goes a long way in connecting with employees and fostering a rapport with them.

The product that the organization works on and the service it provides take a backseat when an organization's culture is under consideration, as a strong culture can do wonders in terms of reviving a sick unit, a strategy that is going south, or a weak positioning against the competition. A positive culture can bring in clear thinking, a long term view, innovation, and an empathy for each other—all of which can push the organization forward. An open culture is one that even large organizations strive to achieve in order to grow. Google touts an open culture and attributes the company's success to its culture.

https://well.tc/w42a

Achieving Success through Servant Leadership

THERE ARE MANY DIFFERENT STYLES OF TEAM LEADERSHIP THAT CAN MAKE OR BREAK THE SUCCESS OF A PROJECT. FOR AGILE TO WORK, CONSIDER ADOPTING SERVANT LEADERSHIP.

by Joshua Dawson | *jdawson@3riverstech.com*

Think of the most difficult team you have ever worked with—a team that was hands down the absolute worst. Think of what made them the worst team ever … and remember that you were part of that team.

Now think of the best team you have ever worked with—the A team. They were the team that consistently got things done and never failed to deliver, no matter how difficult things got. This was the team that could spin gold miraculously out of nothing. Think of what made them the best team ever … and remember that you were part of that team, too.

What are the differences between these two teams? There are the usual suspects: communication, team processes, use of proper tools to perform the work, weak links or superstars, and so on.

It all comes down to this: What makes a good team *good* and a bad team *bad?*

When you get a taste of working with the best team, a few things happen. First, you recognize how good things are, and you do your best to make it last as long as you possibly can. Second, you get as many people as you can to be committed and involved in the work. Last, you look for ways to recreate with other teams what you experienced with this team. We should strive to take success with us everywhere we go.

"Servant leadership has its origins in many different manuscripts and texts in the ancient world. To be a servant leader means to lead others by example of serving them and caring for their needs before your own."

So, what makes a good team good? In my experience, it is *servant leadership*.

Servant leadership has its origins in many different manuscripts and texts in the ancient world. To be a servant leader means to lead others by example of serving them and caring for their needs before your own. This general philosophy integrates well into any software development or testing team—and integrates especially well with teams using agile practices.

When you witness servant leadership being executed by an entire team of people, it might appear to be too good to be true. You and your team will wonder how the team is getting so much done, so accurately to what the customer wants, and so quickly. Achieving team flow through servant leadership can be addicting.

The best team I have ever worked with was agile at its best. This software development and testing team was charged to create for their client a unique piece of software to develop specific healthy living plans tailored to each of their customers. The team consisted of a project manager, a business analyst, developers, and testers. The developers were collocated, and everyone else was geographically distributed.

We started the project with only abstract ideas, and it felt like the Wild West. We had an idea of the tools we wanted to use and processes we wanted to implement, but we adjusted along the way and went to production with code at the end of each two-week sprint. A daily standup with all members provided a forum to bring up issues or bring in new requirements, which would inevitably redirect our sprint focus. At times when high-priority items came to our attention, the team was able to drop something off the sprint and deliver the priority items to production within the sprint.

This is how I learned about the power of servant leadership. Every single team member was focused on the health of the overall team. Everyone was always looking for ways to help other team members. Based on projects I worked on, I have come up with four examples of servant leadership.

Example 1: We're Going to Need a Bigger Boat

Testing tasks are split 50/50 between two testing resources for a sprint. One of the testers figures out that he can't complete his work. His boat is too small. He brings this to the team's attention at the daily standup. Several team members indicate that they have excess capacity to assist with the particular testing needs. In agile, full transparency is expected, and it pays to be honest without fear of being judged.

This gives the tester the idea of using another team member to help execute the testing. When the idea is presented to the team and another resource can assist with testing, the tester's boat gets bigger. The original scope agreed upon for the sprint can be tested, and the expected functionality can be delivered to the client.

Example 2: Dealing with the Busywork Paradox

Every member on our team acted as a servant leader by being honest about estimating his work and actual workload. When a teammate completed work, he asked the team where help was needed. In the rare case that help was not needed in the current sprint, a team member could focus on preparing work for the next sprint. Because each member was serving the team as a whole, his honesty and transparency resulted in an extremely efficient team. Alternatively, open hostility or individual members suffering from busywork can quickly put the kibosh on servant leadership.

Example 3: I See Smoke

Smoke coming out of the hood of a car is invariably a bad sign, signaling that something is wrong. In software development, it is usually the practice of testing a small subset of functionality delivered to an environment to ensure system stability at implementation. This is called smoke testing. As a team, we spent anywhere from fifteen minutes to an hour smoke testing production each time we went live with code. Even if only one or two people did the testing for the sprint, everyone on this team pitched in to smoke test, all while simultaneously coordinating efforts on a web meeting. Smoke testing significantly reduced quality issues, and all it took was each team member's willingness to spend a couple of minutes testing verification points. In addition, focused smoke tests reduced the need for large, biweekly testing tasks.

Example 4: Stop, Drop, and Roll

How many times have you worked with a team that is completely fine with dropping everything and working on something completely different that the client believes is more important? Disruptions like this can be jarring and a little disorienting. In one case, the client came to us with an immediate need based on something that was causing the client's customers to be incorrectly scheduled in their current production system due to a previously unknown requirement.

As a development team, we were able to deliver fully tested code that fixed this in less than a week. Although the team had to change course, they maintained an extremely positive and supportive team demeanor. This represents servant leadership in action.

Servant leadership may seem old and out of style, but there is a reason it is still around. Servant leadership brings teams together in a helpful environment where members will not feel judged. It seeks to build up everyone's confidence. It seeks to help everyone, including those falling behind.

By placing client and coworker needs above your own, you will want to bring servant leadership to every team you join. [BSM]

INDEX TO ADVERTISERS

Agile Dev, Better Software & DevOps West	https://bscwest.techwell.com	11
Delphix	https://www.delphix.com	13
Microfocus	https://www.microfocus.com/performance	4
Mobile Dev + Test and IoT Dev + Test	https://mobile-iot-devtest.techwell.com	18
SQE Training—On-Site Training	https://sqetraining.com/on-site	36
SQE Training—2017 Software Testing Training Weeks	https://sqetraining.com/trainingweek	28
STAR*EAST*	https://stareast.techwell.com/program/keynotes	2
Turnkey	http://turnkeysolutions.com/free-assessment	23
TCS	https://goo.gl/GmUBcZ	24

DISPLAY ADVERTISING
advertisingsales@techwell.com

ALL OTHER INQUIRIES
info@bettersoftware.com

Better Software (ISSN: 1553-1929) is published four times per year: January, April, July, and October. Entire contents © 2017 by TechWell Corporation 350 Corporate Way, Suite 400, unless otherwise noted on specific articles. The opinions expressed within the articles and contents herein do not necessarily express those of the publisher (TechWell Corporation). All rights reserved. No material in this publication may be reproduced in any form without permission. Reprints of individual articles available. Call 904.278.0524 for details.

TRAIN YOUR TEAM ON YOUR TURF

ON-SITE ADVANTAGE

BRING THE TRAINING TO YOU

Software Tester Certification—Foundation Level
Mastering Test Design
Agile Tester Certification
Agile Test Automation—ICAgile
Integrating Test with a DevOps Approach
Mobile Application Testing
And More!

60+ ON-SITE COURSES

40 TESTING COURSES

7 MANAGEMENT COURSES

9 REQUIREMENTS COURSES

4 DEVELOPMENT AND TESTING TOOLS COURSES

17 AGILE COURSES

2 SECURITY COURSES

For more than twenty-five years, TechWell has helped thousands of organizations reach their goal of producing high-value and high-quality software. As part of TechWell's top-ranked lineup of expert resources for software professionals, SQE Training's On-Site training offers your team the kind of change that can only come from working one-on-one with a seasoned expert. We are the industry's best resource to help organizations meet their software testing, development, management, and requirements training needs.

With On-Site training, we handle it all—bringing the instructor and the course to you. Delivering one of our 60+ courses at your location allows you to tailor the experience to the specific needs of your organization and expand the number of people that can be trained. You and your team can focus on the most relevant material, discuss proprietary issues with complete confidentiality, and ensure everyone is on the same page when implementing new practices and processes.

IF YOU HAVE 6 OR MORE TO TRAIN, CONSIDER ON-SITE TRAINING

SQETRAINING.COM/ON-SITE

SQE TRAINING
A TECHWELL COMPANY