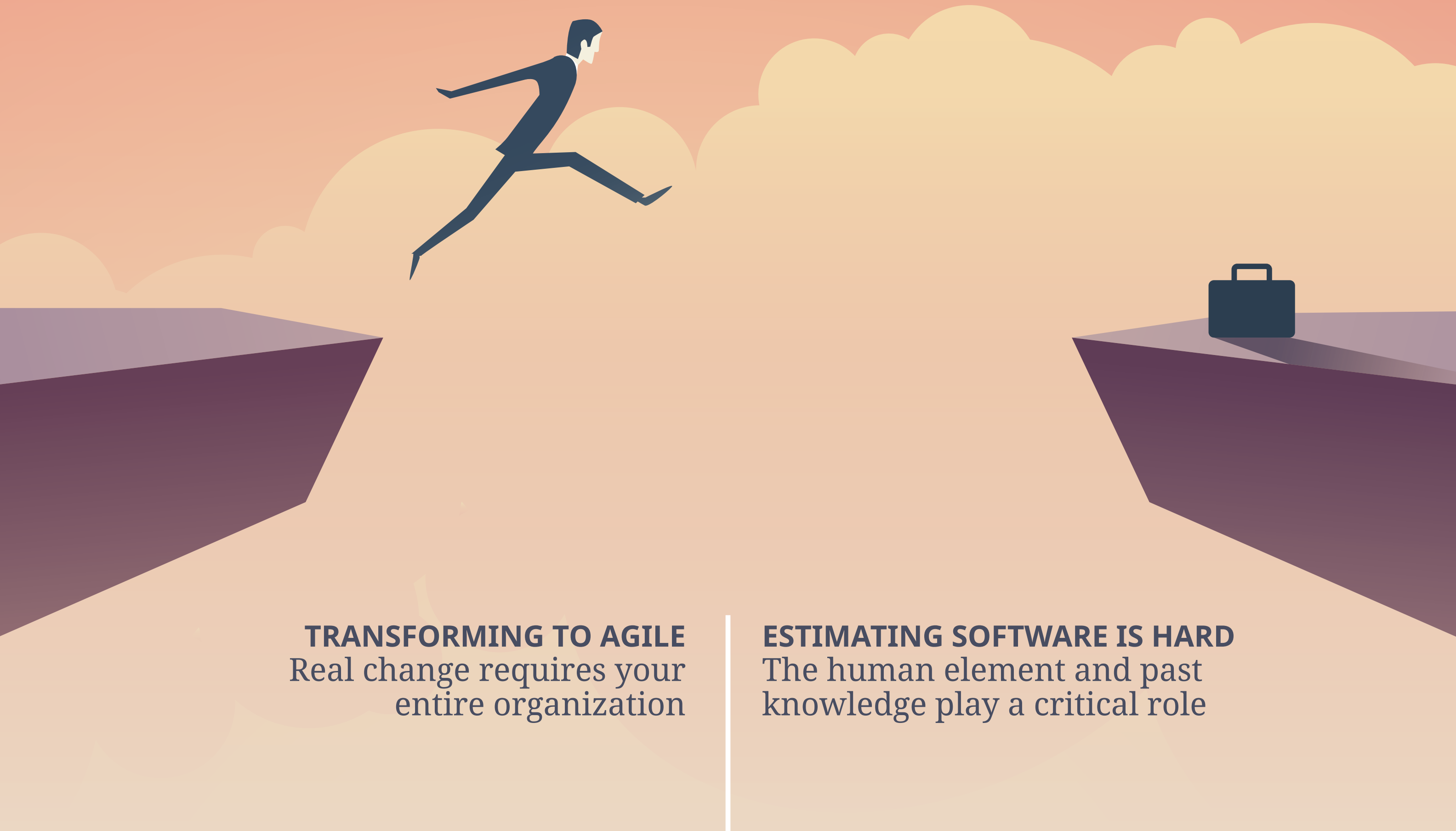
SPRING 2017

[BETTER SOFTWARE]™

A TECHWELL™ PUBLICATION

Do You Really Want to Be a Manager?



TRANSFORMING TO AGILE
Real change requires your entire organization

ESTIMATING SOFTWARE IS HARD
The human element and past knowledge play a critical role

What are Past Attendees Saying?

about the TOPICS

"I very much enjoyed being able to cross-attend the varying topics to gain a large content of ideas."

Tim Robert, Systems Analyst,
State Farm

about the KEYNOTES

"The keynotes were inspiring! There were several practical talks. Gave me time to think and network to develop actionable takeaways."

Pete Lichtenwalner, Sr. Engineer Manager, Verint

about the SPEAKERS

"Great speakers that show they are passionate about what they do. Plus they are open to share ideas and experiences."

Verita Sorsby, QA Manager, Tio Networks

about the TUTORIALS

"Excellent conference. The tutorials were invaluable to me and my group."

Jennifer Winkelmann, Business Analyst, TD Ameritrade

Agile Dev topic areas:

- Agile Development
- Agile Testing
- Agile Implementation
- Agile Techniques
- Scaling Agile
- Agile Metrics
- and more

Better Software topic areas:

- Leadership
- Process & Metrics
- Test & QA
- BA Requirements
- Project Management
- Design & Code
- and more

DevOps topic areas:

- Architecture & Design
- Configuration Management
- Adopting DevOps
- Continuous Delivery
- Continuous Integration
- Regulation & Risk
- and more

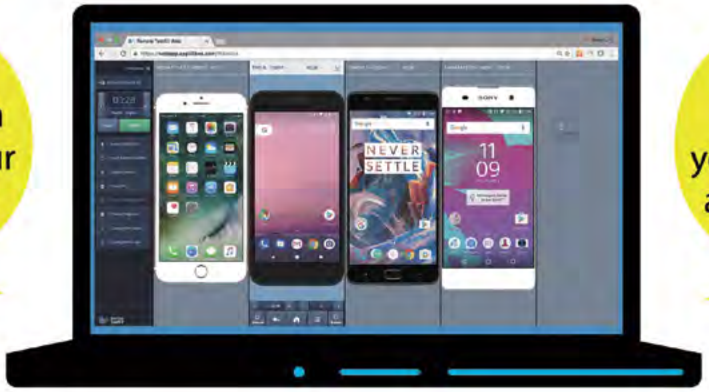
Agile Dev
Better Software
DevOps WEST
A TECHWELL EVENT

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

Register by
May 5, 2017
with code BSMCW
to save up to
$400 off your
conference*

TO REGISTER CALL 888.268.8770 | BSCWEST.TECHWELL.COM

*Discount valid on packages over $400



INSIDE

Volume 19, Issue 2
SPRING 2017

On the Cover

Features

Columns

Departments

SQE TRAINING
A TECHWELL COMPANY

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

Software Testing Training Week

https://www.sqetraining.com/trainingweek

June 12–16, 2017
Chicago, IL

Software Tester Certification—Foundation Level

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

April 24-28, 2017
Virtual Classroom

May 7–9, 2017
Orlando, FL

June 4–6, 2017
Las Vegas, NV

June 19–23, 2017
Virtual Classroom

Fundamentals of Agile Certification—ICAgile

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

April 18–20, 2017
Virtual Classroom

May 7–8, 2017
Orlando, FL

June 4–5, 2017
Las Vegas, NV

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

Agile Dev
Better Software
DevOps WEST
A TECHWELL EVENT

June 4–9, 2017
Las Vegas, NV

LEARN MORE

STAR CANADA
A TECHWELL EVENT

October 15–20, 2017
Toronto, Canada

LEARN MORE

STAR EAST
A TECHWELL EVENT

May 7–12, 2017
Orlando, FL

LEARN MORE

STAR WEST
A TECHWELL EVENT

October 1–6, 2017
Anaheim, CA

LEARN MORE

Agile Dev
Better Software
DevOps EAST
A TECHWELL EVENT

November 5–10, 2017
Orlando, FL

LEARN MORE

EDITOR'S NOTE

Spring-Clean Your Processes

The goal of *Better Software* magazine is to give you innovative ideas so you and your team can create *better* software. Our spring issue has a wide range of articles to motivate you to improve how you do the business of software development. I am proud that this issue has the largest page count by far. With several new advertisers and close to 100,000 worldwide subscribers, *Better Software* is definitely fulfilling a real need in the software development community.

The success of any organizational transformation requires the support and brilliance of management. Many of you are questioning whether you should enter management. Our featured cover article, "Do You Really Want to Be a Manager?" by Ron Lichty and Mickey Mantle, summarizes the basics you should consider before stepping out of your technical contributor role.

Jason Little offers a unique view for taking a nontraditional, multiphase approach toward your project team and your organization becoming agile in "Reshaping Our View of Agile Transformation." Speaking of agile, Jeremy Jarrell offers a simplified approach to risk management in "Managing Risk in an Agile World." And this is without the need for complex analysis tools.

Christian Mackeprang, in his thought-provoking "The Impossibility of Estimating Software" article, questions how some techniques we use to confidently predict schedules are doomed to fail. Not all is lost, according to Christian, as long as you employ more intuitive techniques for specific types of projects.

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. 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

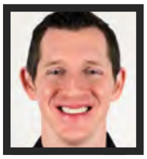
EDITORS:
editors@bettersoftware.com

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

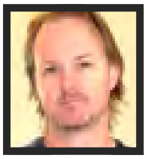
TECHWELL™

CONTRIBUTORS

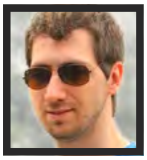


Jeremy Jarrell is an agile coach and author who helps teams improve on what they love. He is heavily involved in the technology community, both as a highly rated speaker and syndicated author whose articles and videos have appeared in numerous publications. When he's not running, Jeremy loves to discuss all topics related to agile methodologies. Reach Jeremy on Twitter @jeremyjarrell or at www.jeremyjarrell.com.

Ron Lichty has been managing and consulting in software development and product organizations for more than twenty-five years. His consulting practice primarily focuses on assisting software development teams to deliver projects on time. Ron and Mickey Mantle coauthored *Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams* and created the *Managing Software People and Teams* video training series. You can reach Ron at ron@ronlichty.com.

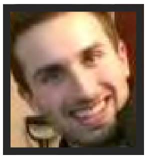


Jason Little began his career as a web developer and moved into management, agile coaching, and consulting. Bumps and bruises along the way brought him to the realization that helping organizations adopt agile practices is not so much about the practices but is all about managing change. An international speaker, Jason is the author of *Lean Change Management* and the *Agile Transformation: A Guide to Organizational Change* video series. Contact Jason at jason@leanintuit.com.



Christian Mackeprang is a software consultant with more than two decades of experience in software development, web architecture, and programming languages. After leading teams through complex projects in highly dynamic environments, Christian spends time contributing to open source projects and writing about software craftsmanship. He likes to focus on how to organize and establish effective software teams. Contact Christian at chris@chrismm.com.

Mickey Mantle held executive engineering management positions at Pixar, Brøderbund, and Gracenote before partnering with Ron Lichty as the coauthor of *Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams* and cocreator of *Managing Software People and Teams LiveLessons* video training series. Today, Mickey consults and helps start-ups become major companies and is also the founder and CEO of Wanderful interactive storybooks. Contact Mickey at mmantle@wanderfulstorybooks.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 wrote for popular video game journalism websites like GameSpot, IGN, and Paste Magazine, and now acts as an editor for an indie project 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.

Eric Robertson is responsible for the DevOps product line at CollabNet. Previously, he served as director of services and portfolio management for enterprise solutions at Unisys and Cisco, where he led product management for Cisco's cloud automation and SAP ALM extension offerings. Eric has held product development, services, and management roles with enterprises and start-ups. He has extensive experience providing consulting services to Fortune 500 companies. Reach Eric at erobertson@collab.net.

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

The Reality of Test Artifacts

CONSIDERING THE EFFORT SPENT ON DOCUMENTING TEST PLANS, THERE HAS TO BE A BETTER WAY TO IMPROVE QUALITY WHILE RESPONDING TO CHANGE.

by Justin Rohrman | *rohrmanj@gmail.com*

The phrase "can create and execute test cases" appears in many testing job descriptions and help-wanted postings. The software development community accepts the idea that testers *plan the work* and *work the plan* by creating detailed documentation of what they intend to test and following it to the letter. At many companies I work with, testers accept test cases and documentation as a fact of their work lives.

I am no longer a fan. I think test cases are worthless and, in some cases, produce negative value.

The Fantasy

Sit down next to a tester and watch her work. You will see her typing, clicking buttons, and pausing to think. Then she will make a few mouse clicks and think for a few more minutes. Eventually, she will get a surprised or confused look on her face, start taking notes, and talk with programmers. But is any testing taking place?

The first myth is that test cases are tools testers must have to guide their work. They are something managers use to make sense of a job that lives nearly entirely in a person's brain. Unlike bricklaying or programming, the results of testing are often invisible. I could spend an hour testing and not have any code or report to show what I did. I regularly hear from friends that they write test cases as part of their role on a team. A tester is usually consumed by converting part of a specification or user story into test cases at the beginning of a sprint.

Test cases give exact instructions on how something should be tested: "Navigate to this page, enter -2 into the number field, click submit, and verify that you get the expected result." One script is typically created for each data value and outcome that the tester thinks is important to validate.

That isn't test *planning*. Rather, this is simply *translating*.

It isn't difficult, either—just laborious. I believe that most professional testers can be trained to do that work in real time as they test. The idea that a tester can't test software and discover information about product quality without detailed instructions doesn't make much sense. The customer certainly doesn't need them.

The second and more dangerous myth is that test cases give insight into what testers are doing.

In my experience, bugs found during regression testing are discovered most often when testers go "off the map." A tester might start with a test case, notice something interesting, and jump off the plan to investigate. At the end of a test cycle, managers only see a list of passed tests.

Managers miss out on the skilled exploration that happened, as well as the bugs discovered when testers left test cases behind.

> "The idea that a tester can't test software and discover information about product quality without detailed instructions doesn't make much sense. The customer certainly doesn't need them."

The Reality

Think about the last time you did regression testing—or any sort of testing. What was the flow like? Let's say you started with a test case covering an analytics report. You might have begun by entering the values as instructed to create some data, saved the page, and then gone to check the report. And what if nothing happened? The software seemed to work, as the test showed the expected result. But, you might have arrived at a few ideas to investigate further. Rather than moving on to the next test case in the stack, you produced an export of a customer database to be used with your build. You tried navigating to the report once more and encountered problems. There was too much data in the report to make any sense, and the new report was not designed to handle the type or amount of data that real customers use.

It is impossible to capture a test idea exactly on paper. I might write, "Navigate to amazon.com, search for the book *Constructing the Subject*, select it, and verify that book details are displayed."

But in practice what will happen is that I'll navigate to amazon.com, filter by books, search for a book, notice something I wasn't looking for and check that out, then maybe, after a long delay, end up back with my original book.

We rarely do things the same way every time we test software—and that is a good thing. Even if we could capture every conceivable user interaction in testing documentation, would that be a good idea? No matter how detailed, testing documentation only captures vague ideas a user might want to interact with a product.

The Big Problem

The biggest sin in test documentation is forgetting that test cases are based in abstraction and standardization. The process of documenting test ideas is also a process of washing away the in-the-moment magic of testing. The results can be disastrous, with a misunderstanding of coverage, metrics consisting of test case counts, and a lot of wasted time.

Looking at test artifacts in this way is easy. It is seductive. And it can lull you into a false sense of security that testing is simply an activity with a definite start and end. The impression is that testing is something that can be written down in steps and performed by anyone capable of typing and moving a mouse.

Look at the work you do. Is all that test documentation helping you and others? Does it describe the work you do? If you stopped doing it, would anyone notice? [BSM]

"If we allow ourselves the ability to fail and not see it as a totally destructive thing … then it really makes it a planned part of your strategy. I know that sounds weird to say, but part of our strategy is to fail."

"If we have test designs that are not flexible, they're going to be hard to automate. They're going to be hard to maintain. They have to be designed with change in mind."

"Unfortunately, when most people learn test design, they don't learn it in a way to make it adaptable to change. They also don't learn it in a way that can be scalable."

> **A lot of people don't understand the difference between the strategy of something and the tactics of something—of how to do it.**

Randy Rice

Years in Industry:	**39**
Email:	**webrequest@riceconsulting.com**
Interviewed by:	**Josiah Renaudin**
Email:	**jrenaudin@techwell.com**

"Yeah, automation is good. It's a fundamentally different way of thinking about and doing testing, so we have to figure out what can be automated. What should be automated? What shouldn't be automated? And how do we carry that forth?"

"One of my great fears is that testing will be relegated to this idea of regression testing. That's just a small subset of all the testing we do."

"Experiment. Pilot things. Then always be willing to come back and do a restart if you need to. Give yourself that permission to fail until you finally have it right."

"No matter what you're doing, you're going to have to be paying attention to make these test designs, these concepts, to make them an actual reality and make them workable."

CLICK HERE FOR THE FULL INTERVIEW ››

Do You Really Want to Be a Manager?

Ron Lichty

AND

Mickey Mantle

Managing programmers is hard! As we point out in our book, *Managing the Unmanageable*, [1] programmers are an interesting management challenge to begin with. They tend to be free spirits, playful, curious, and very independent.

The transition from programmer to manager is made additionally treacherous by the dramatic difference between what made us successful as programmers and what it takes to successfully manage others. The fact that few programming managers receive management training before they start managing further complicates the transition. Not to mention that the approaches to management—managing people in every role and domain—continue to dramatically evolve. This rapid evolution leaves us bewildered and stranded without an adequate supply of role models to follow.

With the software industry continuing its high growth path and with software development boot camps churning programmers into the workforce, more managers are needed more urgently than ever before.

> *“Programming managers do need to garner respect, and respect usually proceeds from their having been decent, if not stellar, programmers. But managers who code rarely do both well.”*

Most of us who manage programmers were promoted because we were great programmers and showed some people skills. We demonstrated a capacity—and perhaps even an inclination—to direct the activity of other programmers. But in our informal polls to more than a thousand managers, we found fewer than 5 percent had even a day of training before becoming managers.

Here are a few things you can do to ease the transition into management.

The Path to Management

Let’s start with how success drivers change when we become managers. The very skills that make a great programmer or technologist often get in the way of being a good manager.

The ability to focus helps make us great at programming. We have learned to shut out the world and climb into the microprocessor to listen to the gates open and close. We become one with the computer, one with our code, and one with the machine executing it.

Focus comes, in part, from personality. Many of you have probably taken the Myers–Briggs personality assessment to determine cognitive style. Did you know that the majority of programmers share the same “I” introverted personality type? Introversion serves all of us well as programmers.

Unlike programming, managing requires a focus on others. We not only need to place a welcome mat at our office door but also must proactively invite and welcome interruptions from our staff. Each programmer’s concerns and challenges become ours. Counseling and coaching them through their issues are paramount to their success, the team’s success, and our own success.

The transition from follower to leader is not as profound as the transition from technical contributor to manager, but it is certainly a change. We’ve probably shown leadership skills already. It’s why we’re selected to take ownership of our teams. But managing is a different kind of leading. Now we’re setting the expectations for our teams, and we’re responsible for establishing the environment in which they can be happy, fulfilled, and successful.

Because of who we are as programmers, we tend to gravitate to detail and to solving problems ourselves. We often hold our own problem-solving skills in higher regard than those of our staff. However, managing is about delegating and growing our people to do what we once did.

Our hope must be to grow each programmer in our organization to surpass our own abilities.

But I Still Want to Code

There are many managers who want to continue to program. That’s admirable, but in our workshops we frequently ask programmers and managers to think about the best manager they ever had and to list their qualities. Not once has “the ability to write code” been on the list of responses.

Programming managers do need to garner respect, and respect usually proceeds from their having been decent, if not stellar, programmers. But when managers also code, they rarely do both well. Because of the continual pressure to deliver code, it is usually the managing and leading that go by the wayside. When managers take on even the tiniest slice of their teams’ critical work, they risk failing their staff.

The Importance of Training

New managers typically neglect to seek out training because we’re busy taking on the new responsibilities. But we also tend to enter management with supreme confidence that we can conquer management as easily as we did coding. Think about how many languages, how many algorithms, and how many pesky bugs we conquered.

Unfortunately, we don’t know what we don’t know.

Treat people-management training as a project. Prioritize it. It’s a growth opportunity. Identify books and classes that might make a difference. Network with peers to learn what they have found valuable. You should have a list long enough that you have to order it like a Scrum backlog. Then, for the books at the top of the backlog, order them. For the training classes at the top of the backlog, sign up for them in advance, blocking time on your calendar so you can't use the excuse of conflicts to bow out.

Your Role as an Enabler

It's really important to find the intrinsic motivations in being a manager. When we were programmers, solving challenging programming problems resulted in ecstatic joy. There were times when I literally jumped out of my chair and yelled "I did it!"

It isn't easy to find a single moment in management to match coding's moments of overpowering accomplishment. But we *have* experienced pride in creating environments and cultures that enabled entire teams of programmers to experience the joy of solving hard problems. Managing is fundamentally about enabling and supporting your people.

Take the Opportunity to Ask for Help

Talking to other managers can ease the transition. Who better to learn from than those who have gone before us? Ask your boss, peers, and mentors to share the most important lessons they learned as they became managers. What mistakes did they stumble into, and how did they successfully exit them? What surprised them—both good and bad? What gave them strength? Where did they find support?

Then ask your new reports what they want in a manager. This is a good time to practice reflective listening. Play back, frequently, what you're hearing; make every attempt to paraphrase in your own words what you just heard; ask for confirmation that what you've heard is what they meant. This is the opportunity to ask the people you'll manage what, to them, makes a good manager good. You might also ask, "If you had a magic wand, what would you change about the team? How would you have it be managed differently? How would you have your teammates interact differently with each other and with those outside the team? What would increase productivity? What would make your job more fun?"

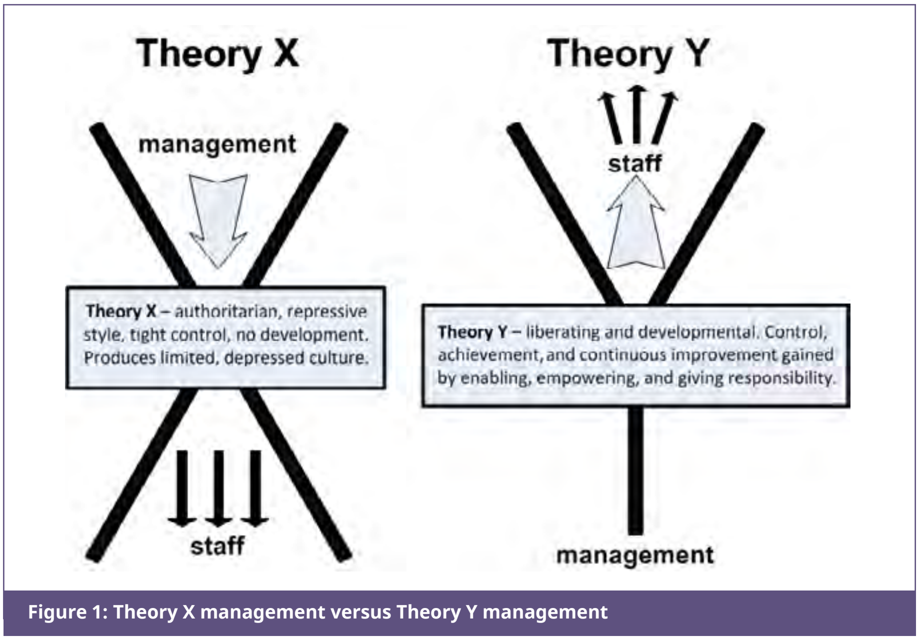
Let your team know that, like becoming a programmer, you don't expect to become a great manager on day one. But with their help, support, suggestions, and feedback, you intend to support them and to keep learning how to do that better.

Becoming a Servant Leader

In the 1950s, Douglas McGregor developed two theories for motivation and people management, which he called Theory X and Theory Y. [2] Some managers tend toward Theory X (authoritarian) and generally get poor results. Enlightened managers, McGregor said, use Theory Y (empowering), which produces better performance and results, and allows people to grow and develop (see figure 1).

The term *servant leadership*, coined a quarter century later, was described in terms almost identical to Theory Y management. But neither was widely practiced in software circles until agile embraced the foundations underlying both.

One way to help embrace servant leadership is to change your concept of what being a manager means. Figure 2 on the next page shows how most people view management as a top-down, authoritative position on top of the pyramid.



Theory X – authoritarian, repressive style, tight control, no development. Produces limited, depressed culture.

Theory Y – liberating and developmental. Control, achievement, and continuous improvement gained by enabling, empowering, and giving responsibility.

Figure 1: Theory X management versus Theory Y management

Consider, instead, yourself to be at the bottom of a pyramid with your staff on top. This view, showing staff doing the actual work on top, is what we call the inverted pyramid. The managers on the bottom are delegated problems to resolve that have stymied the staff on the top. Use this simple image to visualize your role as a manager, and share it with your programmers if you think it helps convey the way you want to work and be perceived.

> *“You can build trust with your team by accepting blame and responsibility when there are problems, and saying thank you and praising performance when there are successes.”*

The Makings of a Great Manager

There are so many things that go into being a great manager that we can’t begin to touch on them all here. But whether you are still coding or shifting your time to focus on managing, two things are vitally important—being fair and being ethical.

One of the hardest lessons we learned on our journey to becoming managers is that you cannot always be fair. A manager must strive to be as fair as possible. This is most important when it comes to compensation. Early in our managing careers we found that salary inequities are inevitable. There are those who, by virtue of experience or happenstance, are paid more than others who perform equally. New hires straight out of school usually earn less than equally competent, but not top-performing, programmers on the same team. It is difficult to double someone’s salary, let alone to halve someone else’s salary, even when it would be the fair thing to do.

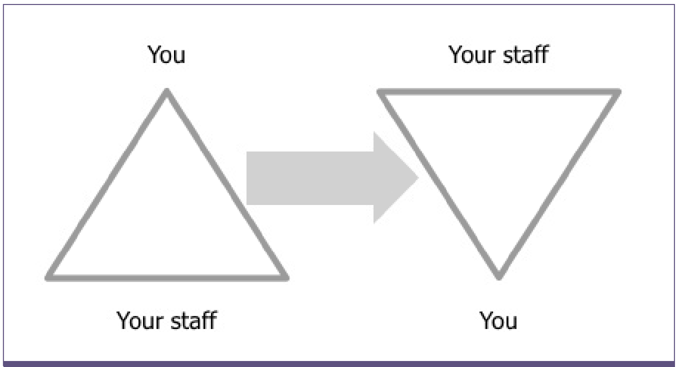


Figure 2: The common view of management and the rise of servant leadership

Still, managers always need to deal with this. Being fair comes down to correcting salary inequities over time by giving greater pay increases to top performers and smaller increases to poorer performers. It’s only fair.

Being ethical means that you are forthright and honest in dealing with your staff and others. You can build trust with your team by accepting blame and responsibility when there are problems, and saying thank you and praising performance when there are successes.

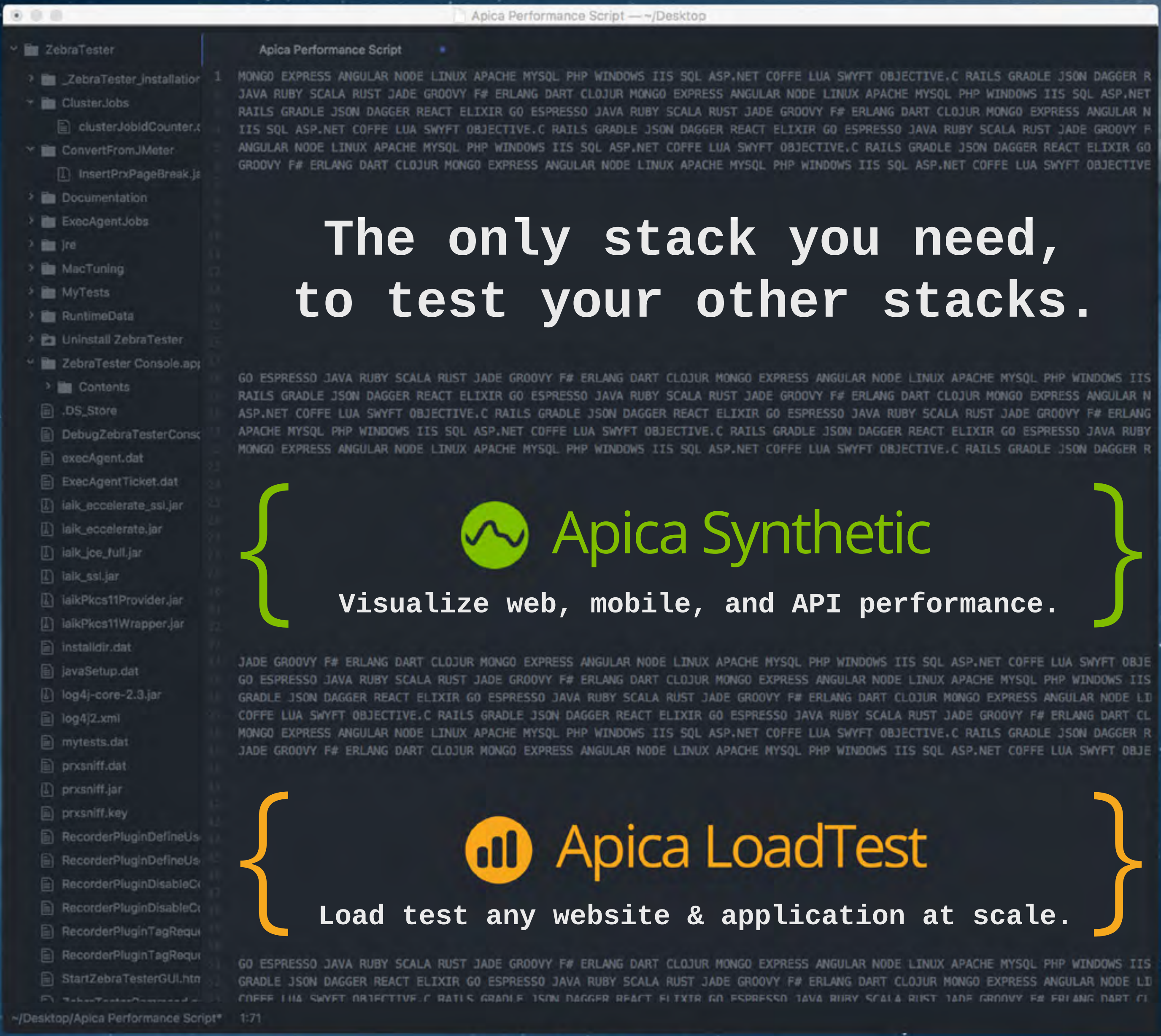
Any manager who does this will be on his way to becoming a great manager—someone his team and others want to work for.

To Be a Manager or Not

Managing programmers is certainly not the piece of cake we thought it would be when we were programmers. In retrospect, avoiding management pitfalls should be obvious, yet anticipating many situations is not possible. You’ll definitely learn by doing—almost like on-the-job training. Becoming a manager isn’t a suitable career path for every programmer, even those who show evidence of people management skills. By treating the transition to management as a learning challenge and committing to the joy of enabling others to succeed, you have a chance to become the manager and the leader you always wanted to work for. [BSM]

ron@ronlichty.com
mmantle@wanderfulstorybooks.com

CLICK FOR THIS STORY’S REFERENCES



```
MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER R
JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET
RAILS GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR N
IIS SQL ASP.NET COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F
ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER REACT ELIXIR GO
GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJECTIVE
```

The only stack you need, to test your other stacks.

```
GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS
RAILS GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR N
ASP.NET COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG
APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY
MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER R
```

{ Apica Synthetic

Visualize web, mobile, and API performance. }

```
JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJE
GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS
GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LI
COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CL
MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER R
JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS SQL ASP.NET COFFE LUA SWYFT OBJE
```

{ Apica LoadTest

Load test any website & application at scale. }

```
GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LINUX APACHE MYSQL PHP WINDOWS IIS
GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CLOJUR MONGO EXPRESS ANGULAR NODE LI
COFFE LUA SWYFT OBJECTIVE.C RAILS GRADLE JSON DAGGER REACT ELIXIR GO ESPRESSO JAVA RUBY SCALA RUST JADE GROOVY F# ERLANG DART CL
```

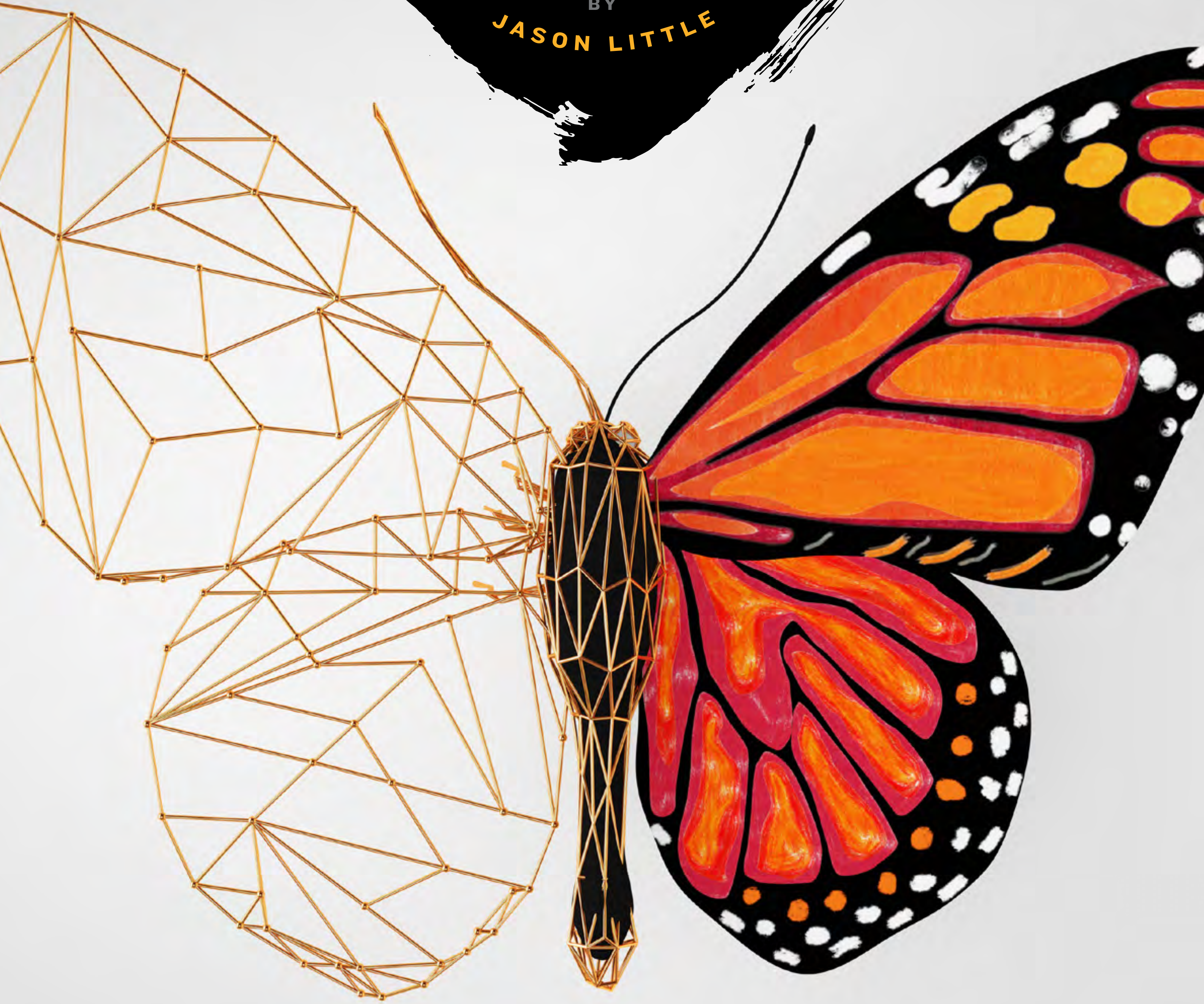
Everyone has performance issues.
Apica can find yours before your customers do.

Request a trial at apicasystem.com/start-trial
Email sales@apicasystem.com or call (310) 776-7540

Apica

RESHAPING OUR VIEW OF AGILE TRANSFORMATION

AN ARTICLE BY JASON LITTLE



Ten years ago, I sent an email to my certified ScrumMaster training instructor and asked if we were supposed to estimate product defects. I learned that Scrum emphasizes that quality isn't negotiable and defects go to the top of the backlog.

But what if there are too many defects found and no features could get done? Scrum says we use a team's velocity as a guide for how much work we can pull into a sprint. But how can a team estimate the number of defects they could finish in a sprint if they don't estimate each one first? A few months later I concluded that perhaps we should just estimate the bugs but not include them in velocity calculations. [1]

I mistakenly thought we should deal with the problem by creating more process. I realized there were more important questions to ask:

Why do we have so many defects?

Would it be better if we estimated correcting defects?

Do customers care what our backlog looks like?

Should this be called a feature or bug? And does the definition really matter?

Over the past years, I found myself becoming uninterested in exploring specific quality and defect identification techniques in my Scrum practice. Rather than continue my focus on helping teams with quality and agile concepts, I decided there was a bigger opportunity: transforming entire organizations to agile.

From Agile Community to Agile Organizational Change

Over the years, I've drifted away from the agile community and moved toward the organizational change community out of necessity. Most of my work early on was in enterprise organizations, where models designed for single agile teams break down. I realized early on in my career that agile was a trigger for organizational change, and many more parts of the organization outside of IT were impacted by it.

An agile transformation is mistakenly viewed as a project: Create a plan, execute the plan, and close the project. This view is typical because someone needs to pay for the change program, someone needs to report project status, and someone needs to justify why money is being spent on agile coaches and not features. And, of course, someone needs to be held accountable for the transformation.

Our brains love the idea of the transition from current state to future state, even though that's a false sense of certainty. There is no current state nowadays, and by the time we think we have reached the future state, that reality has already changed. Deep organizational change doesn't take place according to a project budget or schedule, no matter how much we want it to.

Much like I reshaped my view of handling defects in software, our community would benefit from reshaping our view of agile transformation by looking at how change actually happens.

Change, whether in our organization or society, happens in precisely the same way. Someone is dissatisfied with the status quo and takes action. After that, the movement takes shape, and either change occurs or it dies. As your organization moves toward being more agile, the people who stand to lose something create a counter-movement to put a stop to it. Whoever has the most influence usually wins.

The Phases of Agile Organizational Transformation

Sociologist Herbert Blumer identified four lifecycle stages that social movements typically go through. They describe how social change begins, how it organizes, and how work gets done based on agreements and negotiations. [2] Other academics have since renamed these stages, but the underlying themes have remained constant. Figure 1 shows how an agile transformation can mimic this theory.

Here's a description of the modern phases:

Emergence: Something triggers the change, and we start to see change emerge. At this stage, there is little to no organization—even if the organization has hired an agile coach or created a new vice president of agile role. We have such a strong desire for certainty that starting off with official titles and accountability for agile sounds reasonable and comforting. The challenge is that we can't know how agile will affect our organization until we live with it through our regular organizational ceremonies, such as annual project planning and budgeting, performance reviews, and more.

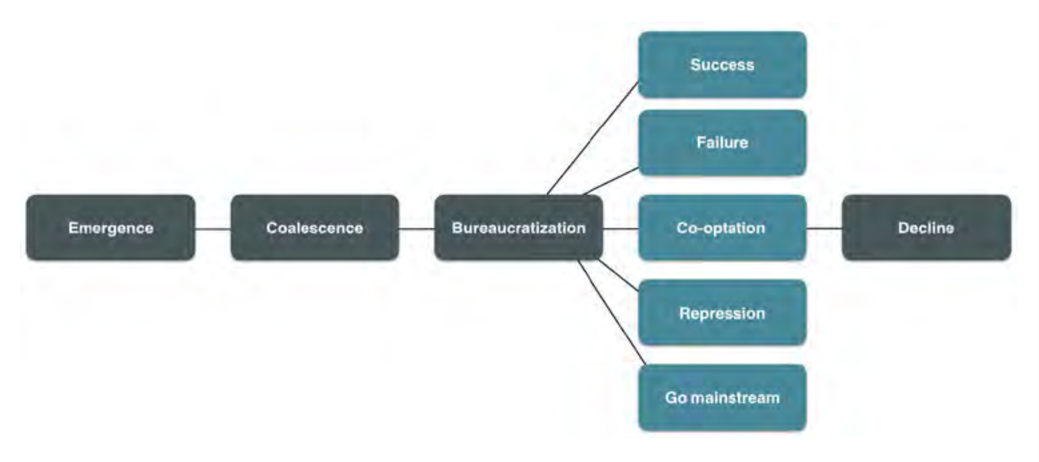


Figure 1: The phases for how social change happens

Many organizations start with pilot projects. As pilot teams experiment with agile, the impact of agile on the organization becomes clearer. There may be a perception that some people are resisting the change, but that's a natural response and much better than apathy. Once agile is brought into an organization, political posturing may start from those who think they stand to lose something, namely control or status.

Consider avoiding defining a solid plan, identifying success metrics and measurements, or assigning solitary accountability. The goal of this early phase is to learn how agile affects your organization. The emergence phase might take well over a year, and it could be painful. Rely on short feedback loops, reuse existing rituals (such as existing department meetings or town halls) for a while to build momentum, and then sunset those rituals for new ones that align with an agile way of working.

Coalescence: Here, things start to become more formal. You're at this phase when you hear questions like, "So, you're the agile guru?"

The first time agile disturbs the organizational norms, the organization will tend to react negatively. Agile exposes risk sooner than traditional approaches, so a project status report may go red earlier in the project. Agile will be blamed by the detractors, and agile will be praised by the promoters. During coalescence, the pressure to define agile processes will increase. People may start running regular lunch and learns or informal meetings, requesting agile training, joining book clubs, or attending clubs and meetups.

> "IF THE ORGANIZATION IS SERIOUS ABOUT AGILE, THEN RED TAPE AND ROADBLOCKS THAT ARE PREVENTING TEAMS FROM DELIVERING MUST BE EXPOSED."

If you're in a leadership position, provide air cover for the agile folks who are running amok. If the organization is serious about agile, then red tape and roadblocks that are preventing teams from delivering must be exposed.

Again, be prepared for the detractors to be vocal that agile doesn't work and causes all these problems. Be on the lookout for breaks appearing in the organization. This can be anything related to project management, release process, technical debt, or teams fighting for space to collaborate. Explore your ecosystem using Jay Galbraith's star model as a diagnostic tool. [3] This model describes how all five points of the star need to be aligned for change to work. Anytime you change a process, the structure, rewards system, HR processes, and budget processes must also be changed.

I worked in an organization where ninety teams went agile within a year. The centralized operations team was consumed with work because so many teams moved to an iterative model, driving up demand for more frequent integrations. The individual project teams had no control over the demand overload yet still relied on this centralized team. This Ops team decided to make it harder for teams to release instead of listening to what the organization needed.

The reaction is a symptom that the centralized operations team should be merged with the delivery team. In effect, the bloated organizational hierarchy needed to go.

Bureaucratization: The pain exposed through emergence and coalescence leads to this stage, where the desire to standardize and formalize agile processes reaches its peak. Even though teams and business folks love it, agile can look like chaos from the outside. Other affected departments—legal, marketing, auditing, customer support, operations, and any other centralized function—often feel the pain of keeping up with the delivery teams.

The Turning Point

Although the term *decline* sounds negative, it is not necessarily bad. It simply means that the period of social movement is over and has reached stabilization. An organization can react in a few ways that align with the five possible paths for the decline of the movement, as shown in figure 1. Entering this phase is the turning point in an agile organization's transformation.

Success: Even though the change worked, we often fail to recognize it. Indicators of a successful agile transformation include the completion of a reorganization, departure of staff, presence of new agile-thinking staff, a move to team-based performance reviews instead of individual-based ones, and a better cadence of solution delivery that customers love.

This may feel like a win for some, but for others it feels like a failure. It could be that some agile proponents have lost interest and are bored now that it's working. Those people may leave for another organization.

Failure: For whatever reason, the change didn't work at all. Move on and don't beat yourself up about it. Sometimes we can't explain why it didn't work, and that may be okay. It is important to emphasize learning over failure. Although everyone knows change is hard, the first time through an agile transformation is always the toughest.

Agile retrospectives, a cornerstone of many agile methods, play a key role. By embracing a large-scale retrospective with teams, managers, and executives, everyone can learn why the transition didn't work out the way you wanted it to. The team wipes the slate clean, adjusts, and moves on.

Co-optation: Some believe agile is a mindset; you've probably heard that you can't *do* agile, you have to *be* agile. Co-optation means that instead of living by the new set of values that agile brings, you install agile into your old existing cultural values like a piece of software. An organization that values hierarchy, structure, and control will introduce agile practices in an overly structured, control-driven way, instead of using an agile values-based approach. Agile becomes a process choice, yet no meaningful change occurs.

"IF WE STICK TO LINEAR, PROJECT-BASED THINKING, AREN'T WE GOING TO KEEP REPEATING THE SAME MISTAKES?"

If this happens, the early adopters will likely leave the organization. I've seen two instances of full agile change teams walking out within eighteen months.

Regression: This means the change was too difficult and everyone gave up. It could also be that whoever was leading the charge for agile gave up and either resigned or was pushed out. Either way, the pain of the change was more than the pain of working the old way, so the organization regressed.

Go mainstream: This phase of decline wasn't documented in Blumer's original work, but it was added later to specifically show that the movement declined because the goals and values of the movement spread throughout the social system and simply weren't needed anymore. To me, that's the same as success, but you can choose to keep that subtle difference.

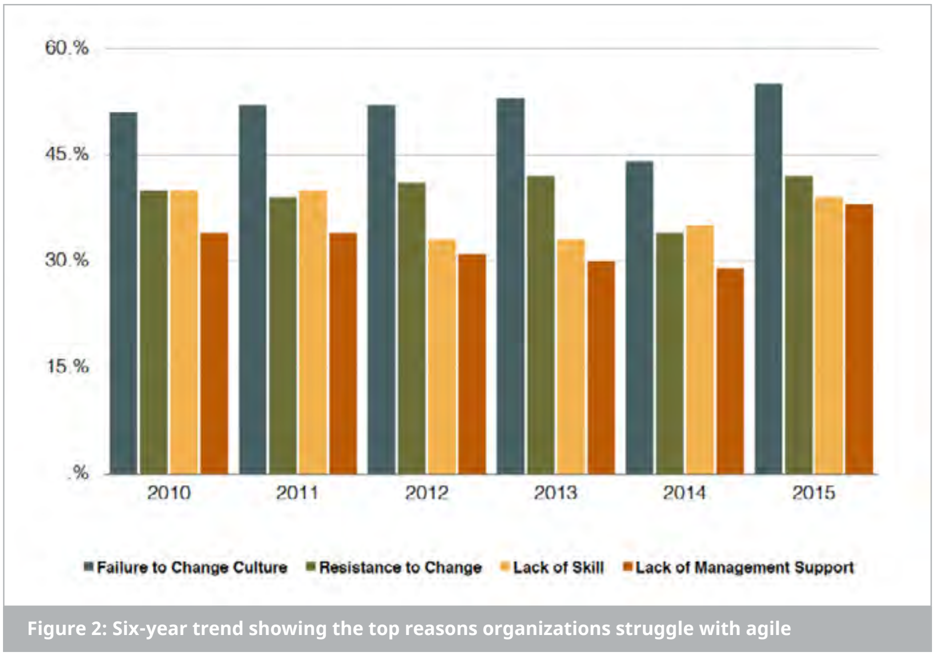


Figure 2: Six-year trend showing the top reasons organizations struggle with agile

Our organizations evolve the same way our societies do. At some point, someone is upset by the status quo and injects a change into the organization. That starts the movement, but where it ends up is based on many other people's actions.

I believe we need a different frame for helping our organizations change. Change management studies have concluded that 70 percent of changes fail. [4] When you examine the data, it shows that change programs work about 30 percent of the time, are challenged 56 percent of the time (meaning they were late, over budget, or delivered less than expected), and outright fail 18 percent of the time. The data also fails to differentiate between changes, so a large transformation change project is lumped into typical IT system changes. [5]

The results arrive at two conclusions: We lack a standard change method, and people are unpredictable.

Where Are Organizations Getting Stuck?

Figure 2, a summary of the last six Version One "State of Agile" reports, shows the reasons survey respondents' software development organizations struggle in transitioning to agile. [6]

There is no doubt that transitioning an organization to agile is difficult. If we stick to linear, project-based thinking, aren't we going to keep repeating the same mistakes over and over again?

Our organizational behavior mimics our social behavior, so it seems it would be a good idea to use a different frame for transforming our organizations.

There Is Hope

Even if your declining movement ended in failure, it may not be worth worrying about why it didn't work. Organizational transformations can fail due to perceived lack of leadership buy-in, poor communication, and staff resistance. Adopting a better change process may not have made a difference.

Knowing that people are social in nature, there are five techniques I recommend for success.

1. ***Honor the past:*** Our brains are like Velcro when it comes to bad things and Teflon when it comes to good things—

that is, we remember the bad and tend to forget the good. Before embarking on your transformation journey, capture the good things by sharing stories about past successes and good things that you don't want to change. Honor the things you already do well.

2. ***Establish new rituals:*** Accept that you can't predict how complex change will work out. Instead, focus on establishing new rituals. For example, you can conduct monthly, executive-sponsored lean coffee sessions [7] that put the focus on dialogue. Leaders should have the courage to walk into any group session and take unfiltered questions from the staff. For example, at a change intervention for a large organization, I used a tool called Slido to harvest anonymous questions from the audience. Attendees voted on the questions management would be asked. In one case, the top-voted question directed at the CTO was, "You keep saying that we need to change. What are you prepared to change?"
3. ***Make space:*** We live in an age when Google knows everything. Just search "how to transform your organization" and you'll find millions of results. *Knowledge* isn't the problem anymore; *time* is. People need time to learn about agile, and the best way you can make space for the agile movement is to stop a percentage of your in-process projects. Time is your only enemy.
4. ***Find your metaphor:*** I worked for an organization that used the Rockefeller habits [8] to align our departments. Our metaphor was moving boulders. We identified four big boulders to move for the year, and every quarter we broke those boulders into rocks that were easier to move. The key was adding celebrations to the mix. We used decorative pebbles as thank-you notes. If someone from any department helped out, we recorded a sixty-second video clip in a room containing thank you bowls for all employees. We'd drop a decorative pebble in the bowl and at the end of the month, everyone watched the videos, and whoever had the most pebbles received vouchers for dinner and a movie.
5. ***Use storytelling:*** Ever since humans learned to draw on cave walls, we have made progress by sharing stories. Technology has pushed us toward tools and processes, but people still remember and are inspired by stories. Make sure your stories are honest and share the good, bad, and ugly.

Change works when enough people embrace it. Start with the heart, and the mind will follow. [BSM]

jason@leanintuit.com

CLICK FOR THIS STORY'S REFERENCES

PARASOFT®

COMPLETE IoT SOFTWARE TESTING SOLUTION

- STATIC & DYNAMIC ANALYSIS
- UNIT TESTING
- API, LOAD, PERFORMANCE, & SECURITY TESTING
- SERVICE VIRTUALIZATION

Learn more at parasoft.com/iot

LEARN MORE

MANAGING RISK IN AN AGILE WORLD

BY JEREMY JARRELL

Software projects are no stranger to risk. There's the risk that the team will build the wrong thing for the customer. There's the risk that the team will build the right thing for the customer but build it wrong, usually resulting in cost and schedule overruns.

Or there's the risk that the team will deliver an over budget, late project that customers reject.

So, how do you manage risk on your project? If your team is following an agile methodology, focusing on small units of work and validating progress frequently with stakeholders, then you may already be closer than you think. Let's first examine how work is defined on agile projects.

On agile projects, small units of work are called user stories, product backlog items, or simply cards. The term will vary with the specific agile methodology your team uses, but for simplicity, we'll refer to them as items. Each item is self-contained, adds some measurable piece of value, and has the potential to carry some amount of risk to the final project. But to address the risk that may be present in each item, you must first identify it.

Risk analysis is a huge subject filled with complicated algorithms, statistical and stochastic modeling, and complicated mathematics. But it doesn't have to be.

Historically, risk analysis has been complex because the results of a failure could be devastating. And if an error is made while building a bridge or performing a medical procedure, then the result can be catastrophic.

These errors are significant because they occur on such a large scale. But, if items are broken down into multiple, smaller chunks, then you reduce the risk associated with each item. This results in a less-severe impact if those risks actually come to pass.

Once you've broken your work down into smaller items, it's simply a matter of evaluating the risk of each item. And often analyzing risk can be very complicated.

Risk Analysis That's Easy to Implement

At its core, risk analysis comes down to two simple questions: "How likely is it to happen?" and "What would happen if it did?" But to get started, you'll first need to brainstorm anything that could possibly go wrong with each of the items your team is working on. Let's look at an example.

Imagine your team is building an app to help users track their personal finances. This app does things like categorize a user's spending, create and track simple budgets, and report changes in net worth over time. To bring this product to market, the team will need to develop many different items. Some items may be new features, others may be defects that need to be corrected, and others are simply tasks that are necessary to keep development moving forward. Each item comes with its own risk potential that will need to be identified and assessed.

To understand how risks can be managed, there are a few approaches the team can use to identify risks that may be present.

Avoid a Single Point of Failure

Considering we'll be creating an app for personal finance, our first feature should be an easy way for a user to set up a budget for monthly expenses. Then our users will need to easily see how they are pacing against that budget throughout the month or how much of their budget they've already spent compared to the remaining days in the month. Figure 1 presents a great way to convey how a user is pacing to their budget. A stacked bar chart showing how much of their budget users have consumed is an effective feature that should set your product apart from your competitors.

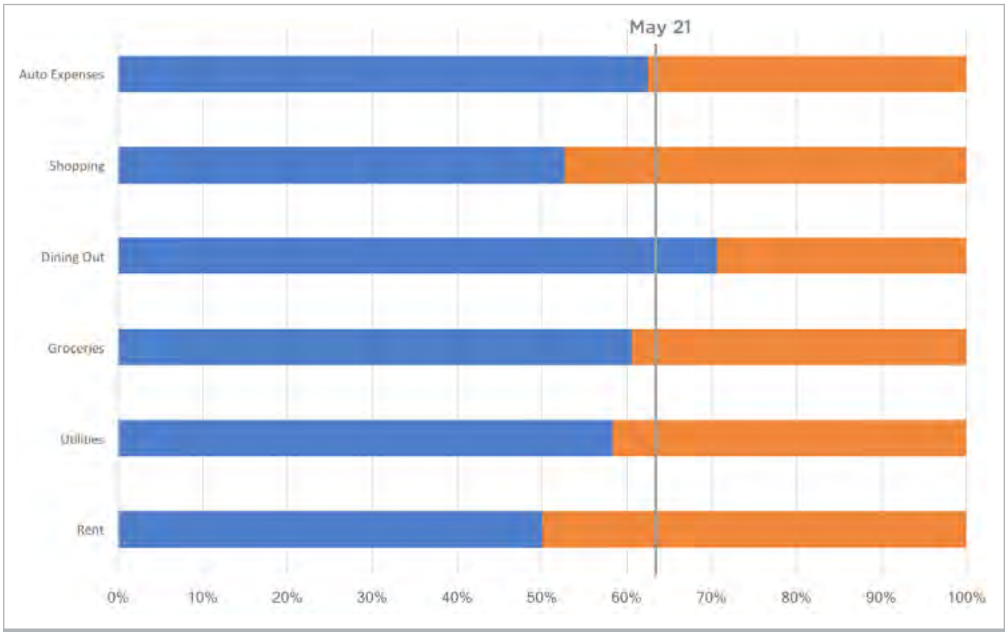


Figure 1: A stacked bar chart is a great way to convey how a user is progressing toward budget goals

But, uh oh, there's only one developer on your team who's capable of creating these types of visualizations. This shouldn't be a problem as long as nothing unexpected happens to that developer during the course of your project. Otherwise, your entire team could be stuck until that developer is available.

You've just discovered your first possible risk.

A developer may find herself unavailable for any number of reasons. Maybe she was pulled off to a more critical product elsewhere in the organization, or maybe she came down with a particularly nasty flu and will be out of commission for an entire week, or maybe she has a two-week vacation coming up right in the project's crunch time that she simply forgot to mention. Whatever the reason, having a critical skill set concentrated in a single team member can easily lead to problems you'll have to solve down the road.

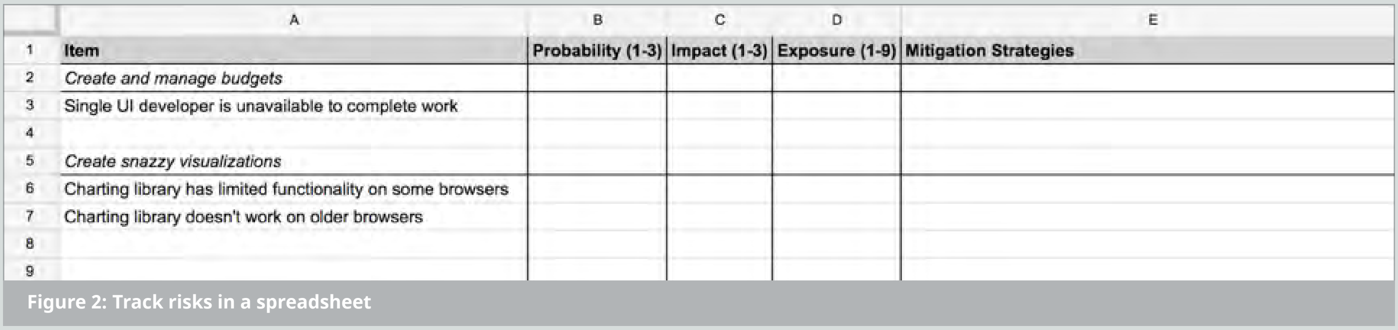
Support for Legacy Platforms

And speaking of fancy visualizations, they can pose their own problems. Maybe your UI developer stumbles upon a snazzy new charting library and uses it to create a show-stopping budget visualization. Perhaps he uses that same library to create other compelling visualizations of the user's financial health throughout the entire product. Everything is looking great—until your test team starts to put the product through its paces on all the different browser platforms that must be supported.

Suddenly you discover that, while the library is fully functional on the latest, most popular browsers on the market, it seems to have only limited functionality on some less popular browsers or doesn't function at all on some the oldest browsers your customers are using. If a third of your users can't see them, they're not going to add the value to your product that you expected.

Score Your Risks

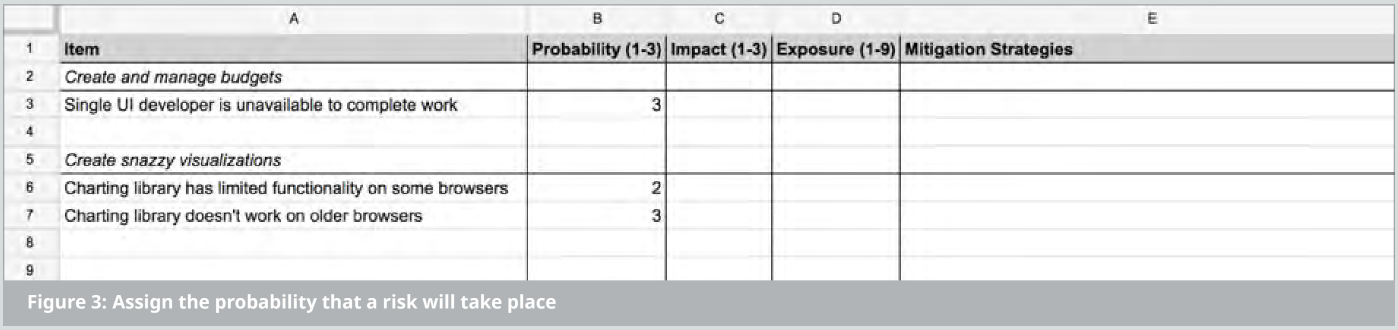
As risks are identified, use a spreadsheet to keep track of them, as shown in figure 2.



	A	B	C	D	E
1	Item	Probability (1-3)	Impact (1-3)	Exposure (1-9)	Mitigation Strategies
2	*Create and manage budgets*				
3	Single UI developer is unavailable to complete work				
4					
5	*Create snazzy visualizations*				
6	Charting library has limited functionality on some browsers				
7	Charting library doesn't work on older browsers				
8					
9					

Figure 2: Track risks in a spreadsheet

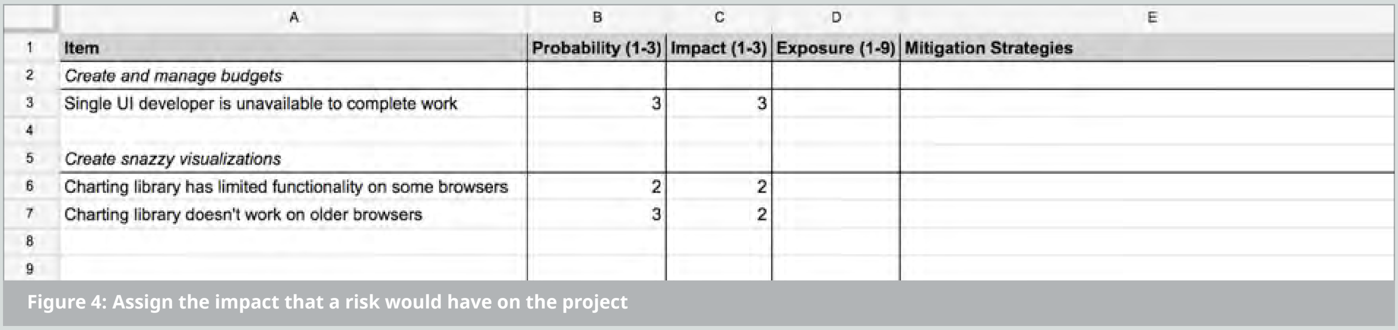
Now that we have a list of the possible things that may go wrong with each of our features, quantify how exposed your team is to each. Work through each risk on the list and assign each a score of one to three based on the probability of its occurring—with one being somewhat unlikely and three being very likely (see figure 3).



	A	B	C	D	E
1	Item	Probability (1-3)	Impact (1-3)	Exposure (1-9)	Mitigation Strategies
2	*Create and manage budgets*				
3	Single UI developer is unavailable to complete work	3			
4					
5	*Create snazzy visualizations*				
6	Charting library has limited functionality on some browsers	2			
7	Charting library doesn't work on older browsers	3			
8					
9					

Figure 3: Assign the probability that a risk will take place

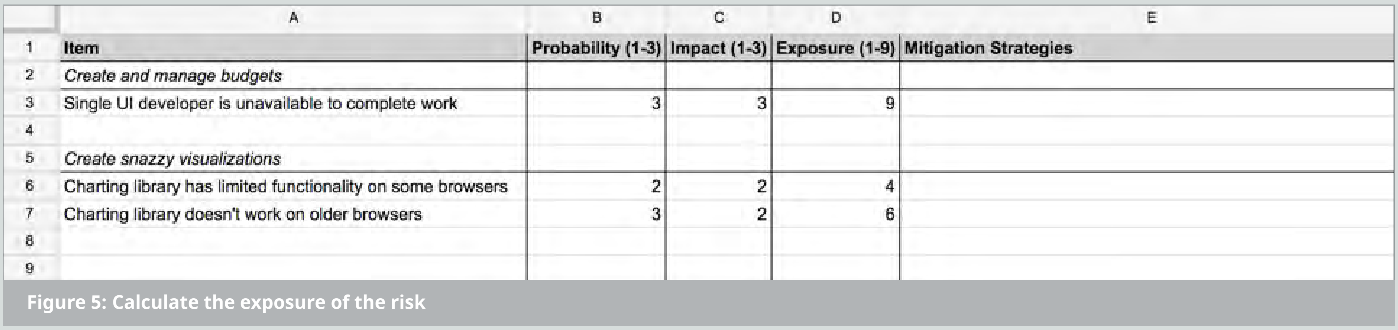
Next, assign a second score of one to three to each risk based on the impact if the risk occurs. This score, shown in figure 4, captures how damaging the risk would be to your project if it comes to pass. An impact score of one correspond to risks that may slow the team down initially but can be easily overcome. An impact score of three is reserved for risks that could stop a project dead in its tracks.



	A	B	C	D	E
1	Item	Probability (1-3)	Impact (1-3)	Exposure (1-9)	Mitigation Strategies
2	*Create and manage budgets*				
3	Single UI developer is unavailable to complete work	3	3		
4					
5	*Create snazzy visualizations*				
6	Charting library has limited functionality on some browsers	2	2		
7	Charting library doesn't work on older browsers	3	2		
8					
9					

Figure 4: Assign the impact that a risk would have on the project

Once probability (P) and impact (I) scores are assigned to each risk, calculate the exposure (E) to that risk by multiplying its probability by its impact using P × I = E, as shown in figure 5. It's as easy as PIE!



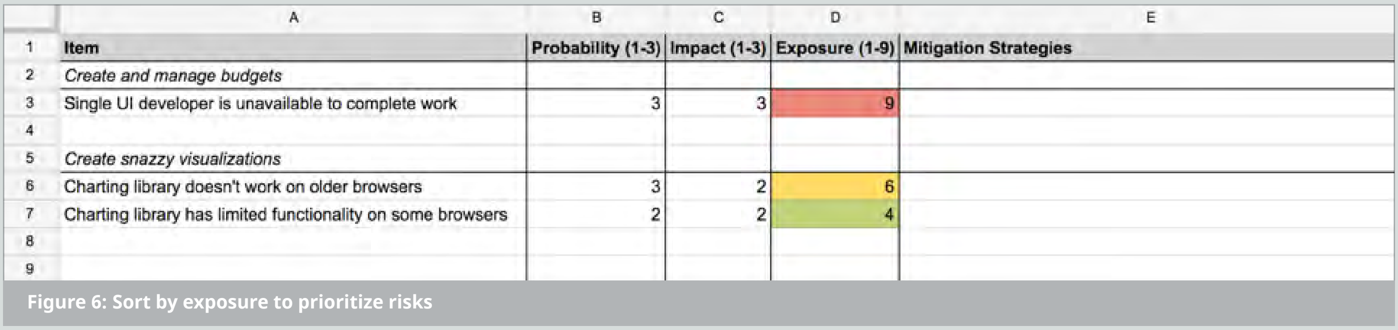
	A	B	C	D	E
1	Item	Probability (1-3)	Impact (1-3)	Exposure (1-9)	Mitigation Strategies
2	*Create and manage budgets*				
3	Single UI developer is unavailable to complete work	3	3	9	
4					
5	*Create snazzy visualizations*				
6	Charting library has limited functionality on some browsers	2	2	4	
7	Charting library doesn't work on older browsers	3	2	6	
8					
9					

Figure 5: Calculate the exposure of the risk

Understand Risk Exposure

Because exposure is a function of probability and impact, exposure scores fall into a range between one and nine. A good rule of thumb is to consider any risk with an exposure score of six or higher as warranting the creation of a mitigation strategy. Any risk with an exposure score of less than six is either not likely to occur or isn't serious enough to warrant the upfront planning.

To focus on the most important risks, sort the rows by their exposure score in descending order. Figure 6 shows this order, along with color-coding, with red being the most critical.



	A	B	C	D	E
1	Item	Probability (1-3)	Impact (1-3)	Exposure (1-9)	Mitigation Strategies
2	*Create and manage budgets*				
3	Single UI developer is unavailable to complete work	3	3	9	
4					
5	*Create snazzy visualizations*				
6	Charting library doesn't work on older browsers	3	2	6	
7	Charting library has limited functionality on some browsers	2	2	4	
8					
9					

Figure 6: Sort by exposure to prioritize risks

"YOU'RE LIKELY TO FIND THAT MANY RISKS YOU ONCE THOUGHT WERE NEAR CERTAINTIES WILL SEEM LESS AND LESS LIKELY AND EVENTUALLY DISAPPEAR AS NEW AND UNEXPECTED RISKS EMERGE TO TAKE THEIR PLACE."

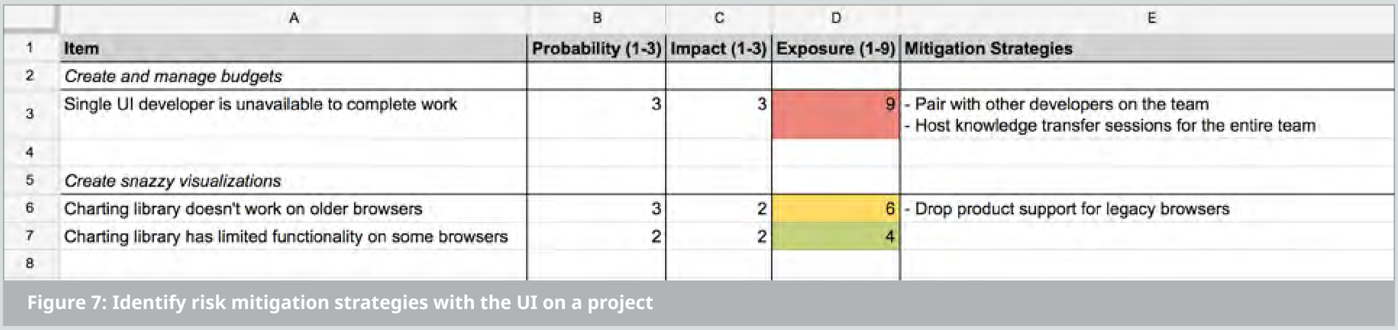
How to Approach Risk Mitigation

Once high-risk items have been identified, the next step is working with your team to brainstorm a few possible mitigation strategies for each one. You don't need to create the perfect solution for each; only a few starting points for discussion are necessary. You won't know exactly what the resulting effects of each risk will be until it occurs, so you won't be able to define the perfect mitigation strategy until the time comes. Consequently, you should treat these strategies simply as options to have on the table if the risk occurs.

Some of these options may be ruled out immediately, while others may serve as the basis for better ideas later. Regardless of the outcome, the goal is to have at least a few options already in mind so you're in a better position to respond quickly and appropriately. Think of this stage of risk analysis as a fire drill. No one knows exactly what will happen if a fire occurs, but we do know that it's much easier to find the exits before the building is filled with smoke.

Figure 7 shows risk mitigation strategies associated with UI work. To mitigate this risk, you can pair your UI developer with other members of the development team for knowledge sharing. Perhaps she can even host a few brown bag sessions on the charting library she is working with.

The fact that the library doesn't support older browsers is a major area of risk. The team needs to decide how to handle support in legacy browsers. One option is to drop support entirely for browsers that are no longer supported.



	A	B	C	D	E
1	**Item**	**Probability (1-3)**	**Impact (1-3)**	**Exposure (1-9)**	**Mitigation Strategies**
2	*Create and manage budgets*				
3	Single UI developer is unavailable to complete work	3	3	9	- Pair with other developers on the team - Host knowledge transfer sessions for the entire team
4					
5	*Create snazzy visualizations*				
6	Charting library doesn't work on older browsers	3	2	6	- Drop product support for legacy browsers
7	Charting library has limited functionality on some browsers	2	2	4	
8					

Figure 7: Identify risk mitigation strategies with the UI on a project

Risk Analysis Is an Ongoing Process

You now have a head start on identifying and assessing the risks that may be present throughout your product, but you're far from done. Your list of risks and the associated PIE score for each is a living document that should be revisited regularly throughout the life of your product.

You're likely to find that many risks you once thought were near certainties will seem less and less likely and eventually disappear as new and unexpected risks emerge to take their place. And, of course, the probability, impact, and exposure scores are likely to change as more is learned about each item. The key is to frequently review this list to ensure it's always fresh and represents the most up-to-date understanding of the risks that can threaten your project.

An underlying tenet of all agile methodologies is that teams forgo extensive upfront planning when beginning their projects in exchange for planning in smaller, more frequent increments throughout the entire lifecycle of the project. This is because teams learn more about their project the longer it runs, and they can use this additional information to make better and more informed decisions along the way.

Agile risk analysis works the same way. Instead of performing an extensive risk analysis at the outset of the project and then trying to manage a risk profile that becomes irrelevant shortly thereafter, agile risk analysis encourages teams to build their risk profile from the risks that are most likely to occur as a result of the activity actually taking place throughout the project. Teams are then encouraged to discard risks that are no longer relevant to their project, making room for new risks that no one could have predicted at the beginning.

Risk is a normal part of software development, and no team will completely eliminate all the risks that lurk in the shadows. But by regularly considering mitigations for risks that could befall your project, you'll significantly improve the odds of success and be ready for whatever your project throws at you. [BSM]

jeremy@jeremyjarrell.com

LIVE VIRTUAL TRAINING

ATTEND LIVE, INSTRUCTOR-LED CLASSES VIA YOUR COMPUTER.

Live Virtual Courses:

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

Convenient, Cost Effective Training by Industry Experts

Live Virtual Package Includes:

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

SQETRAINING.COM/LIVE-VIRTUAL

SQE TRAINING
A TECHWELL COMPANY

Advanced Automation Secrets for Selenium & Open Source

STAR CANADA

A TECHWELL EVENT

Oct. 15–20, 2017
Toronto, ON
Hyatt Regency Toronto

WE'VE RESERVED YOUR SEAT

CONFERENCE TOPICS INCLUDE:

- Agile Testing
- Mobile Testing
- Continuous Integration
- Cloud Testing
- Performance Testing
- DevOps
- Test Design
- Test Automation
- Test Management

RESERVE YOUR SPOT NOW FOR THE BEST PRICING

Better Software subscribers can receive up to an additional $200 off with code **BSM**

Learn More: **https://well.tc/sc17bsm**

THE IMPOSSIBILITY OF ESTIMATING SOFTWARE

BY CHRISTIAN MACKEPRANG

I remember the first time I was asked to estimate a software project. My boss came over to my desk, where I was intensely focused on solving programming problems. He asked me to create a list of work items, each with an estimate of hours for completion. It seemed like a simple enough request. I did not know what I was getting into.

Estimating task durations is a broad problem with many special cases. It reminds me of Terry Bollinger's quote:

> "The creation of genuinely new software has far more in common with developing a new theory of physics than it does with producing cars or watches on an assembly line." [1]

Depending on the situation, schedule estimating can be unrealistic. Articles and books have been written about the art of estimating, and they are filled with inscrutable equations—only to arrive at inaccurate schedules. Not even the most powerful computers can project how long it will take to run an algorithm to arrive at a reasonable schedule.

Estimating, like an algorithm, is performing a sequence of steps. However, this specific algorithm represents a *halting problem*. This occurs when a computer gets stuck on a problem, and there's no way to tell when it will get unstuck—just like we experience on our projects.

Having some perspective on the subject of estimation can be revealing. Let's look at some of the issues you may encounter so we can identify situations when making an estimate is possible—and when all hope might as well be lost.

Dealing with Project Complexity

The first roadblock you'll run into when estimating a new task is unexpected complexity. Your understanding of a problem is seldom clear from the start. You're almost certain to find situations become more complex as you make progress. The waterfall methodology, for example, can be used to solve the problem of estimation, but even if you try to outline every single detail, you'll only discover implementation issues during development.

Complexity can escalate in many ways during development. Good candidates for unexpected complexity are tasks with a creative component, such as building a user interface; businesses that are implicitly hard to understand, such as finance and artificial intelligence; problems that require solving, as in math and physics; and any area in which you are not a subject matter expert.

Experienced developers have a better chance of making accurate estimates simply because they've been exposed to more situations and aren't as easy to surprise. Still, everyone is vulnerable to unexpected complexity.

Software variety and trends also bring complexity with them. It takes time to become proficient with a new software library, and each has its own corner cases. So even if you're an expert with one library, learning a new one will expose you to new tradeoffs and edge cases made during its development.

It may even be that a specific problem you're tackling requires a new type of software architecture or arrangement of components that you are unfamiliar with. This has happened multiple times during the evolution of game development. Drawing pixels initially required a few geometrical calculations, so procedural programming was enough. Then, 3D came along, and with it matrix algebra and more complex games. This led developers to favor object-oriented programming. Eventually, performance had to be squeezed from any possible place, and some people tried data-oriented programming as an alternative.

Game development is also a good example of unexpected escalation of complexity on seemingly simple applications. Seasoned game developers often own quite a few books on math, software architecture, and programming languages.

The Importance of Previous Knowledge

I recognized the importance of experience and familiarity with a task many years ago while playing around with a Rubik's Cube on a long bus trip.

The distance from Rio de Janeiro to Buenos Aires runs a cool 1,200 miles, and I had about forty-two hours to kill. I decided it was a good time to play with an old Rubik's Cube I had lying around. After spending many hours on it, I never finished the last side. It didn't help my mood when I later watched a YouTube video showing a kid solving one in just a couple of minutes.

"NOT EVEN THE MOST POWERFUL COMPUTERS CAN PROJECT HOW LONG IT WILL TAKE TO RUN AN ALGORITHM TO ARRIVE AT A REASONABLE SCHEDULE."

However, I did see the similarity between Rubik's Cube algorithms and software development. This got me thinking about estimates for software projects.

Expert-level cube solvers can complete them very quickly. If you ask one of them how long he would take, you might get an estimate in the three to four minute range. If you asked me how long I would take, it would be hard to come up with a reasonably accurate range.

Familiarity with a task is a main component of accurate estimates, yet many software development methodologies seem unconcerned with it. We routinely see teams meeting to discuss how long a task will take. However, in many cases, there is no discussion about who the expert is on the subject matter at hand and what his estimate might be.

An example is the use of planning poker in estimating agile software development projects. Each team member comes up with a number representing the days he thinks a task will take to complete. Unfortunately, this approach disregards the possibility that only a few people on the team are experts on a specific task and, thus, able to estimate it accurately. The solution is to have the experts try to convince everyone else that their estimate is the right one. But because estimating often requires applying intuition and gut feeling, it's hard to convince the rest of the team that you're right.

Scheduling using a traditional waterfall process isn't much better. The waterfall approach relies on having a few people come up with a detailed specification and a list of tasks. The approach follows these steps: estimate the effort, select a team, and distribute the tasks among the team. The problem here is that the person doing the estimating is rarely directly responsible for the implementation. Consequently, there is a high degree of uncertainty in the estimate for each task and for the overall project.

The Impact of Human and Cognitive Factors

The human element is a critical side of software engineering that affects estimates. We are all familiar with the many interactions that take place on a project. We have to deal with coworkers interrupting us, a manager assigning too many tasks, key staff getting switched to a different project, and poor documentation resulting in time wasted asking for clarification.

Because it's so hard to quantify the cognitive elements, there is no way to accurately measure worker productivity, so we rely on metrics to give us a hint of the true schedule status. Classic examples are the number of tickets solved, the time spent on an issue, and the code commit rate. Unfortunately, the metrics are often inaccurate and, in my experience, scarcely useful, considering they attempt to conflate all cognitive factors into a single number.

Agile processes combat this problem in an interesting way. Instead of basing estimates on an average of how much work the team completed during a given time period, agile teams measure how much things improve or degenerate. With this information, called *velocity* in Scrum, it's possible to apply direct measures to improve the workplace and track how they perform over time.

For example, a team with a high incidence of interruptions, multitasking, and project switching will spend more time on each task. But there are many factors that could adversely impact estimates. We can apply reasonable measures to improve productivity and keep measuring the velocity to see if it improves. This approach is based on an iterative learning process, so it can't be used for upfront project estimates.

Is It Possible to Estimate with Confidence?

Approaches that take statistical or historical data as a ground truth are doomed to fail. The problem is that new software almost invariably brings new situations into play. Even small details have a chance of greatly affecting the complexity of the software you've developed. Experienced developers will be familiar with the situation of introducing a small change that ends up completely blowing apart the scope of a project.

> "YOUR SUCCESS IN ESTIMATION IS A MATTER OF REFLECTING ON WHAT YOUR PROJECT IS ABOUT AND HOW PREDICTABLE YOUR PARTICULAR CASE IS."

A paper by J.P. Lewis on software estimating [2] stresses the importance of valuing intuition, wisdom, and experience while being cautious about using objective processes. Lewis states that it's possible to define a framework for estimates based on a particular measure, such as an agile team's velocity. In that case, the burden should prove that the measure you're using correlates with productivity and development time. However, this approach is relevant only for teams repeatedly implementing similar software.

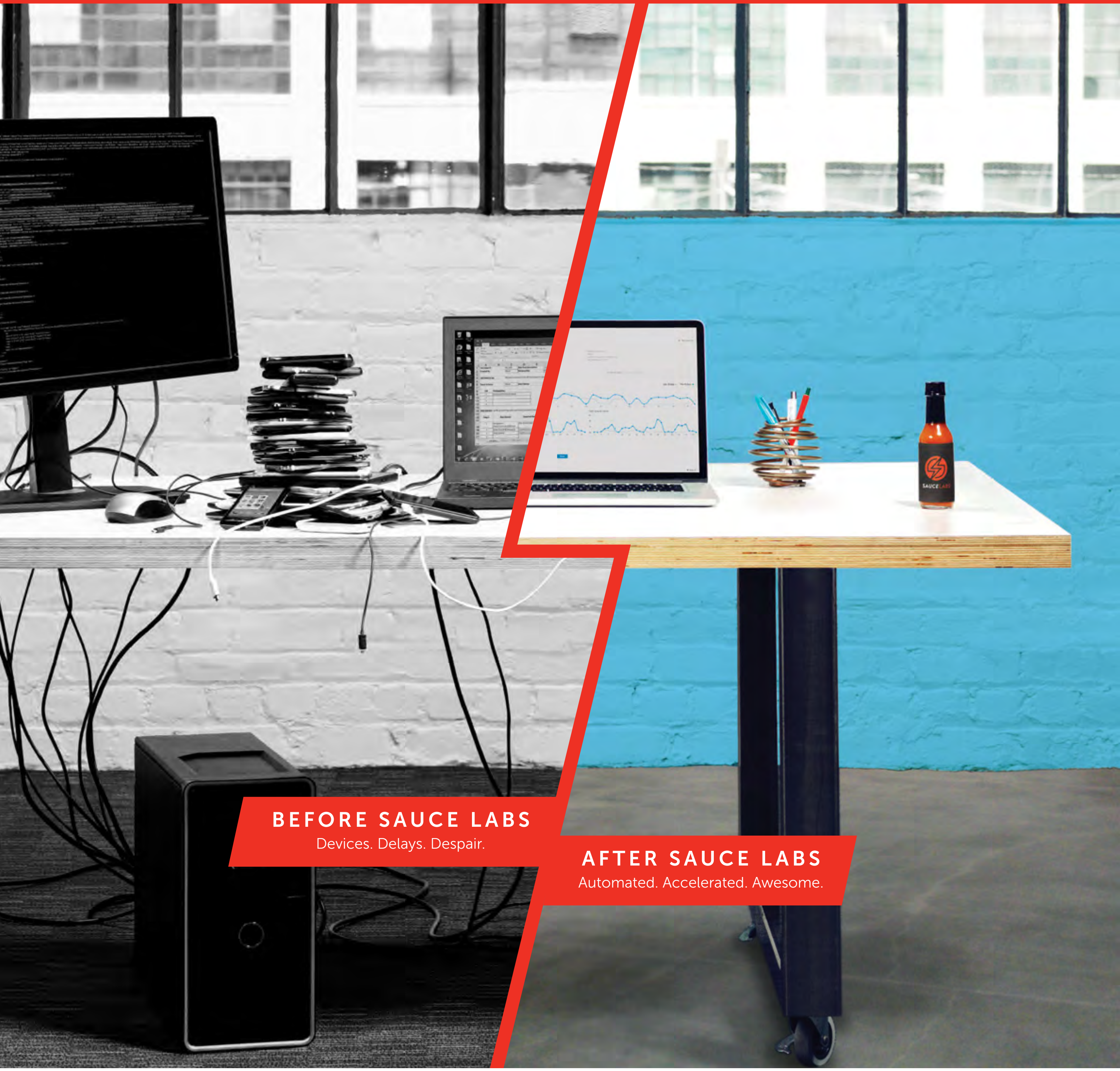
Most of the time, software development is more comparable to problem-solving and developing new theories of science than to a path that can be walked in a set amount of time.

Your success in estimation is a matter of reflecting on what your project is about and how predictable your particular case is. Are you working on simple, sure-fire projects that come with little creative freedom? If so, it might be time to automate. On the other hand, if your project requires creative thinking, you can't expect to provide accurate estimates up front.

Be sure you have the right people on your team, because it's time to trust your developers to make the best estimation effort they can. [BSM]

chris@chrismm.com

CLICK FOR THIS STORY'S REFERENCES



BEFORE SAUCE LABS
Devices. Delays. Despair.
AFTER SAUCE LABS
Automated. Accelerated. Awesome.

TECHWELL INSIGHTS

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!

The Value of Falling into Software Testing

By Justin Rohrman

To become a software tester, there are generally no required degrees or certifications. Consequently, many testers sort of "fall into" the job. But that doesn't mean they won't do outstanding work. Coming from all walks of life and having varied work experiences can help testers find problems no one else can.

Read More

Managing the Risks of Cloud Storage

By Dale Perry

When managing and storing information, the cloud is a reasonable place to do that, but you need to realize that, as with a personal computer or any other device, it needs to have a backup (or more than one, for important things). Luckily, there are several ways to make local backup copies of critical data.

Read More

5 Questions to Ask in a Project Review

By Payson Hall

Project managers often dread doing reviews, but they're necessary to make sure the project is on the right track. Progress can be affected by unclear definitions, risk, schedules, and cost, so it's important to evaluate whether the project manager, sponsors, and team members are all on the same page.

Read More

Testing in Agile and DevOps: Where Are We Going?

By Hans Buwalda

When looking at what the software market is currently talking about, the top item is DevOps and Continuous Integration/Deployment, which seems to be taking over some of the spotlight from agile and is now a widely accepted new normal. Hans Buwalda looks at where the future of software testing is going.

Read More

Testing the Security of the Internet of Things

By Rajini Padmanaban

The Internet of Things (IoT) has made it on many of the trends lists for the year. Given that security issues can make or break market acceptance for IoT solutions, security testing as a quality attribute is expected to gain a lot of prominence in app development again this year.

Read More

Delivering Successful Software Requires You to Fail Faster

By Josiah Renaudin

The concept of failing has changed from something people dread to a necessary part of creating secure, functioning applications. That means that you don't want to have one major failure at the very end of the development lifecycle—you need to continue to fail before release to find real success.

Read More

IBM's Watson Will Help You File Your Taxes at H&R Block

By Beth Romanik

Customers at H&R Block will be able to get tax advice from IBM's famous supercomputer, Watson. Watson has been fed all 74,000 pages of the US tax code and will use its natural language processing to interact with clients in order to answer questions, uncover deductions and credits, and help calculate refunds.

Read More

What Not to Do If You Want Satisfied Customers

By Naomi Karten

You may think that overperforming would ingratiate you to your customers. But customers don't always want you to go above and beyond—often, they just want what they asked for. Don't fall for this common misconception. The trick to customer satisfaction is delivering just what they want—and good communication.

Read More

The Problem with Software Measurement and Metrics

By Lee Copeland

Many software practices rely on setting target numbers for the team to hit. But when a measure becomes a target, it ceases to be a good measure. People start gaming the system by changing their behavior in such a way to favorably adjust the measure in order to achieve the target. Don't get hung up on metrics.

Read More

Mobile and IoT Challenges: What Testers Need to Know to Improve Their Careers

By Jon Hagar

Many of the skills and knowledge areas that testers have in the IT, web, PC, and even mobile world will have application in the IoT. However, there are some knowledge domains that may be new or have some twists, and if testers understand them, they will be able to separate themselves from other job seekers.

Read More

3 Fundamentals of a Successful Testing Team

By Greg Paskal

When it comes to equipping a QA team to reduce risk, test quality, and deliver world-class products, there are more important things than tools. Fundamentals such as a common language, core testing concepts, and a smart automation strategy are essential to setting up testing teams for success.

Read More

DevOps Helps Enterprises Deliver Better, Faster Software for the IoT

By Anders Wallgren

As the world becomes more connected, it's changing the way we do things, especially in relation to software delivery. For starters, software development for IoT applications presents obstacles concerning security, privacy, and unified standards. But we need look no further than DevOps to find the answers.

Read More

Why You Need to Unify Agile Methodologies among Teams

By Sanjay Zalavadia

Agile software development is a complex initiative to undertake, especially when a dispersed team is involved. Organizations must establish a unified agile methodology to ensure that everyone is on the same page and understands what is expected of them in these efforts.

Read More

NASA Code Available for Down-to-Earth Apps

By Pamela Rentz

The 2017–2018 NASA catalog has hundreds of free, downloadable software codes in categories ranging from aeronautics and autonomous systems to environmental science and vehicle management.

Read More

Creating Your Organization's Agile Culture

By Johanna Rothman

Some organizations decide they can just "install" agile by simply telling the technical team members what to do. It never occurred to the managers that much of what makes agile successful is the organizational culture. It's important to recognize that agile is something you work toward—with the whole team.

Read More

Do Your Part—Engage in Cyber Hygiene

By Mukesh Sharma

Cyber crimes can be more alarming and can have a more devastating impact than even some physical crimes. Consulting firms have stated that cyber crimes are the fastest growing economic crimes today, with a 20 percent increase since 2014.

Read More

2017 Is a Pivotal Year for DevOps

AS CUSTOMERS DEMAND REAL-TIME SOFTWARE UPDATES, DEVOPS IS NO LONGER AN AFTERTHOUGHT. NOW IS THE TIME FOR DRIVING INNOVATION.

by Eric Robertson | *erobertson@collab.net*

The complex nature of software development and delivery, especially at an enterprise scale, has resulted in DevOps gaining importance in recent years. How many of us have been frustrated by online banking or some other vendor interaction? How many of us have left a vendor for another that offers a better customer experience? Today's digital world shows us how the speed and quality of software delivery can either help or harm customer satisfaction and affect business outcomes.

Better customer experiences are driven by better software, and Microsoft CEO Satya Nadella saw it coming at the company's annual Convergence conference in 2015 when he stated, "Every business will become a software business, build applications, use advanced analytics, and provide SaaS services." [1]

All organizations are impacted by software, and all businesses are in the software business.

The quality and functionality of a company's software affects everything from competitive differentiation to customer support and, ultimately, employee satisfaction. So why aren't all private and government organizations delivering better offerings and better service at greater speeds?

The Importance of DevOps

Traditional efforts to deliver innovative software solutions are often hampered by the limitations of the disparate tools, methods, and platforms in use today. Teams tend to be spread out geographically, and today's software development requires collaboration between R&D and IT operations.

DevOps will become an even higher priority to the enterprise as IT professionals learn how it helps bring innovative ideas to life by accelerating and improving software development. Companies that expand their DevOps practices will experience the benefits of better teamwork between development and other groups across the enterprise.

Next-generation DevOps tools are starting to deliver comprehensive views of software release cycles. They combine those views with operational data that teams can use to make better-informed decisions. Key performance indicator (KPI) data will come into play, providing a link between an organization's software development lifecycle and its business. Fundamentally, DevOps will change in 2017 to usher in these and other advances that connect software development to the heart of the enterprise.

> "Traditional efforts to deliver innovative software solutions are often hampered by the limitations of the disparate tools, methods, and platforms in use today."

Here are some of the trends I see developing in the next year as the DevOps market evolves.

The Left Shift

Companies are beginning to leverage specific DevOps tools, which has led to a rapid uptick in automated testing and continuous delivery. The adoption of automated testing created a leftward shift in the pipeline, resulting in smaller-scale tests that are completed earlier and faster. Quality continues to be a focus, and micro-services are helping accelerate that drive by enabling the deployment of higher-quality code at a smaller risk to the business.

These shifts in testing and code deployment have pushed feedback further left in the pipeline, so teams are receiving responses earlier in the delivery lifecycle. As more organizations pick up on this shift, software quality will improve, and risks will be reduced

further down the pipeline. This lowers the potential impact to the customer in terms of outages and operational costs by lessening the number of service requests.

Driving DevOps with Analytics

Focusing on analytics provides a more holistic and comprehensive approach to DevOps. Over the next year, we will see more connected tools and processes, as well as KPI data that can enable new levels of decision making by leveraging operational data to provide intelligent correlation and traceability. Through KPI data, for example, organizations can unwrap the hidden issues within a software release that led to a jump in service tickets. With this type of powerful analytic data, detailed and revealing reports can be used to collect metrics from the tools and activities from chained associations.

DevOps Becomes an Integral Part of the Project Lifecycle

While DevOps tools are meeting the needs of organizations involved with the software development lifecycle, many organizations have evolved. They now need to understand their DevOps value stream across the software development and delivery lifecycle—from planning to operations. This enables organizations to deliver end-to-end traceability across every DevOps tool chain component and to leverage objective metrics and KPIs. This ensures that the delivered value is always operational and meets service-level agreements for the business. Essentially, this means continuous monitoring and feedback across DevOps tool chains.

This year, DevOps excellence is expected and has become the catalyst for successful software solutions. We are witnessing the importance of DevOps even at the executive level of enterprise software solutions. Consider bringing DevOps to the very beginning planning stages of your project lifecycle and coupling KPIs with analytics to measure operational success. By prioritizing these considerations, enterprise leaders will better leverage existing investments and set themselves up for future success in an industry that is constantly changing. [BSM]

CLICK FOR THIS STORY'S REFERENCES

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.