THIRTY-THIRD ANNUAL
PACIFIC NORTHWEST

SOFTWARE QUALITY CONFERENCE

P N S Q C ™

October 12-14, 2015

World Trade Center Portland
Portland, Oregon

Permission to copy without fee all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

Welcome

This year's theme, "Brewing Software Quality", has been exciting for us as we have planned the conference. It has inspired many ideas that have allowed us to think outside of our software world about how quality is brewed across other industries. As you go through the conference events you will see how this fits with how others produce high-integrity products. The reality is that we can't just expect a quality outcome unless we plan for it from the beginning. The software industry, like any other, requires us to use superlative ideas and processes. I hope that you will be able to find some of these great ingredients to take back to your organization as you brew quality software.

This year we are pleased to have Casey Rosenthal kick-off our technical conference. Casey is the Traffic and Chaos Engineering Manager at Netflix where his team has a mission to fortify availability in anticipation of domain failures - quickly responding to deviating outages while preserving the quality of service for their customers. Join us as Casey leverages his experience with distributed systems, artificial intelligence, translating novel algorithms and academia into working models as we learn about Chaos Engineering. This is a philosophy of testing that tackles systematic uncertainty head-on.

On Tuesday we welcome David Hussman from DevJam Studios. David guides a collection of mentors who blend technology, people, and processes to create better products in competitive cycles. David will discuss with us *How to Build the Wrong Thing Faster and Learn From It!* David teaches and coaches product discovery through interactive delivery. He has spent the last 10 years coaching agility, Lean practices and producing products.

If you have attended our conference in the past, you will be familiar with our multiple-track format, which gives you plenty of choices throughout the two-day conference. We have also maintained the 45-minute presentations with 10 minutes between sessions to ensure presenters and attendees have plenty of time to get settled and ready for the next topic.

Monday night will include a reception with poster papers on the mezzanine level; hors d'oeuvres and beverages will be served. We have also partnered with Technology Association of Oregon (TAO) to lead an "Open Mic Night" with the theme *Tales of Testing Terrors*. On Tuesday night we will have the pleasure of collaborating with Rose City Software Process Improvement Network (SPIN) to present *Exploring the Landscape of Software Quality – An Agile Park Bench* with David Hussman.

This year we have broken away from our traditional birds of a feather. For Monday's lunch we will be doing a grab and go lunch where you have a choice between quality brewing demonstrations or thought provoking games. During Tuesday's lunch we will have a plated lunch program were you can meet the speakers and participate in a tabletop challenge.

Additionally, we will be having a closing keynote where Scott Poole from Columbia United Providers and Live Wire! Radio will be presenting, *Poetry and Programming: Not as different as you think. One just pays a whole lot better.* Scott is most well known as the "House Poet" on the weekly Live Wire! Radio variety show, taped in Portland and broadcast nationally by Public Radio International. Although he has been writing poetry for 25 years he has also been writing code for 9 years and is currently a Senior Web Developer for Columbia United Providers.

We believe that this year's conference is full of practical, useful, and valuable information. With everyone's help, software quality continues to move forward. We all play an active part! I am glad you are here and hope to associate and network with each of you as we strive to accomplish our mission which is to "Enable knowledge exchange to produce higher quality software".

Doug Reynolds
President, PNSQC 2015

TABLE OF CONTENTS

Continuous Integration/Continuous Delivery

Lifecycles, Processes & Methodologies Track

People & Management Track

Performance Track

Security Track

Test Automation Track

Testing & Quality Assurance Track

PNSQC 2015 – Conference At A Glance

Monday Oct 12, 2015 • Day One • Technical Program				
7:00-8:00 AM	**Registration and Exhibits Open** Level 2 Mezzanine			
8:00-8:15 AM	**Welcome and Opening Remarks** Level 3 Auditorium			
8:15-9:45 AM	**KEYNOTE: Engineering Chaos - About Trust in Unpredictable Systems** Casey Rosenthal, Netflix, Level 3 Auditorium			
9:45-10:15 AM	**Break in Exhibit Hall** Level 2 Mezzanine			
	Level 3 Auditorium	**Level 1 Plaza Room**	**Level 2 Mezzanine Room**	**Level 3 Skybridge Room**
	INVITED SPEAKER	**TEST AUTOMATION**	**PROCESS**	**CI/CD**
10:15-11:00 AM	**Testing in the Age of Distraction - Flow, Focus, and Defocus in Testing** Zeger Van Hese, Z-sharp	**Preventing Skunky Test Automation Shelfware: A Selenium-WebDriver Case Study** Alan Ark, Eid Passport	**Pre-Mortems** Julie Green, Con-way Enterprises	**TestOps in a DevOps World** Simon Howlett, Workiva
11:10-11:55 AM		**Variability vs. Repeatability -- An Experience Report** Jonathan Li On Wing, Groupon	**Improving Quality Team Coding Skills with Code Clubs** Dwayne Thomas, CrowdCompass	**DevOps: Are You Pushing Bugs to Your Clients Faster?** Wayne Ariola, Parasoft
12:00-1:25 PM	**Lunch Program – Grab & Go lunches with quality brewing demonstrations or thought-provoking games.**			
	INVITED SPEAKER	**TEST AUTOMATION**	**PROCESS**	**CI/CD**
1:30-2:15 PM	**Learning to be More User Centered** Eileen Forrester, Forrester Leadership Group	**The Journey of Testing with Stubs and Proxies in AWS** Lucy Chang, Intuit	**How to Develop High Quality Software** John J. Paliotta, Vector Software	**Version Control Your Jenkins Jobs with Jenkins Job Builder** Wayne Warren, Puppet Labs
2:25-3:10 PM		**Write Once, Run Everywhere: Beaker & Puppet Labs** Alice Nodelman, Puppet Labs	**House of Quality for Complex Software Development Decisions in Mobile, Wearables and IoT** Manini Sharma, Intel	**Our Road to Continuous Delivery @ Tango** Amit Mathur, Tango
3:15-3:45 PM	**Break in Exhibit Hall** Level 2 Mezzanine			
	INVITED SPEAKER	**TEST AUTOMATION**	**PROCESS**	**TESTING & SQA**
3:45-4:30 PM	**Acceptance Test-Driven Development: Better Software Through Collaboration** Ken Pugh, Net Objectives	**RESTful Wrappers** Tony Vu, Puppet Labs	**Perils of Legacy Design Description in an Increasing Agile World** Ray Miller, NTT DATA Federal	**Sustaining in an Agile World** Don Hanson II, Intel Security
4:40-5:20 PM		**Web Based Automation Framework for Beginners** Scott Rodgers, Simplifile	**Improving Forecasts Using Defect Signals** Paul Below, QSM, Inc.	**The "Toolbox" Testing Approach** James Gibbard, Intel Security
5:30-6:30 PM	**Conference Social Kickoff – Exhibits & Poster Papers** Level 2 Mezzanine Complimentary hors d'oeuvres & beverages; open to the public			
6:30-7:30 PM	**TAO presents *Open Mic Tales of Testing Terrors***			

PNSQC 2015 – Conference At A Glance

Tuesday Oct 13, 2015 • Day Two • Technical Program				
7:30-8:00 AM	**Registration Open** Level 2 Mezzanine			
8:00-9:30 AM	**KEYNOTE: How to Build the Wrong Thing Faster and Learn From It** David Hussman, Devjam Studios, Level 3 Auditorium			
9:30-10:15 AM	**Break with Poster Papers** Level 2 Mezzanine			
	Level 3 Auditorium	**Level 1 Plaza Room**	**Level 2 Mezzanine Room**	**Level 3 Skybridge Room**
	INVITED SPEAKER	**TESTING & SQA**	**PEOPLE**	**TEST AUTOMATION**
10:15-11:00 AM	**The Improvement Kata: Annual Improvement Planning Meets Agile** Kathy Iberle, Iberle Consulting Group, Inc.	**So You Think You Can Write a Test Case** Srilu Pinjala, PNSQC	**Systems Thinking Tester** Russell J. Smith, Janrain, Inc.	**Obtaining True UI Automation Speed** Sam Woods, Puppet Labs
11:10-11:55 AM		**API Testing: Picking the Right Strategy** Asha KR, Intel Security	**Is the Role Of Test Manager At Crossroads?** Sreeram Gopalakrishnan, Cognizant Technology Solutions	**Using Machine Learning to Predict Bug Outcomes from Automation** Wayne Roseberry, Microsoft
12:00-1:25 PM	**Lunch Program – Meet the Speakers, Meet the Challenge – a plated lunch with a tabletop challenge. Eat and meet, solutions will be your just desserts.**			
	INVITED SPEAKER	**TESTING & SQA**	**SECURITY**	**PERFORMANCE**
1:30-2:15 PM	**Brewer's Yeast: The Product Owner's Influence** Ronald Thompson, Eiscon Group	**Brewing Next Generation Identity** Kalman C. Toth, NextGenID	**Moving up the Product Security Maturity Model** Joshua C. Rebelo, Intel Security (Mcafee India Software Pvt Ltd)	**Brewing Analytics Quality for the Cloud Performance** L Chen, P Jain, K Chow, E Guirguis T Wu, Intel
2:25-3:10 PM		**Misusing the Type System for Fun and Profit** Ian Dees, Tektronix	**Integration Testing Among Enterprise Security Products** Jeyasekar Marimuthu, Intel Security	**Performance Test Modeling with "ANALYTICS"** Jeevakarthik Kandhasamy, Capgemini Financial Services USA
3:15-3:45 PM	**Break with Poster Papers** Level 2 Mezzanine			
	PROCESS	**TESTING & SQA**	**SECURITY**	**TESTING & SQA**
3:45-4:30 PM	**Don't Let Documentation be a Buzzkill** Victoria Palmiotto, MINDBODY, Inc.	**Brewing Quality in Android Robots** Kingsum Chow, Intel	**Web Application Security – What You Need to Know** Bhushan B. Gupta, Gupta Consulting, LLC	**Challenges in Testing an Intelligent Software** Satish Yogachar, Intel
4:40-5:20 PM	**CLOSING KEYNOTE: Poetry and Programming: Not as Different as You Think. One Just Pays a Whole Lot Better.** Scott Poole, Columbia United Providers and Live Wire! Radio, Level 3 Auditorium			
5:30-6:30 PM	**A Networking Town Hall co-sponsored with Rose City SPIN** Complimentary hors d'oeuvres & beverages; open to the public.			
6:30-7:30 PM	**SPIN Presents An Agile Park Bench, hosted by David Hussman, DevJam Studios**			

PNSQC™ BOARD MEMBERS, OFFICERS, and COMMITTEE CHAIRS

Doug Reynolds – Board Member & President
Tektronix, Inc.

Ian Dees – Board Member & Vice President
Tektronix, Inc.

Bill Baker – Board Member, Treasurer & Operations Infrastructure Chair

Brian Gaudreau – Board Member, Audit & Marketing Chair
Avante

Lloyd Bell – Board Member
Crowd Compass

Tim Farley – Board Member, Program Chair
Cambia Health

Phil Lew – Board Member, Social Media Chair
XBOSoft

Srilu Pinjala – Review Coordinator

Jeremiah Burley – Volunteer Chair

Shauna Gonzales – Conference Format Chair
Nike, Inc.

PNSQC™ VOLUNTEERS

Rick Anderson

Ove Armbrust

Sue Bartlett

Surech Chandra Bose

Carol Brands

Sarah Brundage

Jeremiah Burley

Randy Carpenter

Rita Casaverde

Theodore Chan

Moss Drake

Bhushan Gupta

Lory Haach

Aaron Hockley

Launi Mead

Bill Opsal

Shivanand Parappa

Dave Patterson

Randy Pelligrini

Srilu Pinjala

Amith Pulla

Emily Ren

Yumi Rinta

Rajesh Kumar Sachdeva

Jeanette Schadler

Anurag Sharma

Russell Smith

Keith Stobie

Vadiraj Thayur

Dwayne Thomas

Patt Thomasson

Heather Wilcox

Satish Yogachar

PNSQC Call for Volunteers

PNSQC is a non-profit organization managed by volunteers passionate about software quality. We need your help to meet our mission of enabling knowledge exchange to produce higher quality software. Please step up and volunteer at PNSQC.

Benefits of Volunteering: Professional Development, Contribution & Recognition

Opportunities to Get Involved:

- **Program Committee** — Issues the annual Call for Technical Paper and Poster Paper Abstracts. Receives and manages the paper selection and review process and coordinates program layout.
- **Invited Speakers Committee** — Collaborates with the Program Committee to identify and invite leaders in the global quality community to provide conference speakers.
- **Marketing Communications Committee** — Ensures the software community is aware of PNSQC events via electronic and print media; coordinates and collaborates with co-sponsors and other organizations to get the word out. Identifies potential exhibitors and solicits their participation.
- **Operations & Infrastructure Committee** — Develops techniques to enhance communications with PNSQC board and committee members as well as the PNSQC community at large. Responsible for the PNSQC website, SharePoint, and speaker recording.
- **Community & Networking Committee** — Implements networking opportunities for the software community. Manages the social networking channels of communication. Recruits and works with incoming volunteers to place them in committees. Provides programming for the networking opportunities at the conference. This includes the lunch time format and evening sessions. Responsibilities are recruitment of lead participants, compelling topics and titles and structure of the program.

Contact Us: by submitting your name in the conference survey which is a link sent to all PNSQC attendees or complete the contact form at www.pnsqc.org>About>Contact and address it to the PNSQC Volunteer Coordinator.

PNSQC 2016 Call for Presentations

Inspired? Got an idea you want to tell us about? Consider submitting your ideas for PNSQC 2016. All it takes is a paragraph or two and selected authors receive waived fees.

Get more information at www.pnsqc.org, link to the Call for Abstracts page for more details. Become part of the program by submitting an abstract.

Engineering Chaos – About Trust in Unpredictable Systems

Casey Rosenthal, Netflix

We are ushering in an era of pervasive machine learning, artificial intelligence, and loosely coupled service architectures. The future of software is unpredictable in a systemic sense, increasingly opaque even to the engineers building it. We can invest in transparency and systemic understanding, but that expenditure comes at the cost of velocity.

Increasing consumer expectations and competitive pressure will overwhelm that equation, and as an industry we need to come to terms with deploying and operating software systems that no human can reason about. If we can't reason about it, how can we have confidence in it?

Chaos Engineering is a philosophy of testing that tackles systemic uncertainty head-on. Case studies from Netflix, which constitutes about a third of the Internet's packets at peak, illustrate the need for Chaos, numerous ways it can be implemented, and exhibit the trust that can be built into an inherently unpredictable system. Chaos as a discipline is changing the way the industry builds complex distributed systems.

Casey Rosenthal *is the Traffic and Chaos Engineering Manager at Netflix. The Traffic and Chaos Team has a mission to fortify availability in anticipation of domain failures, responding to devastating outages in stride while preserving the quality of service for our customers. As an Executive Manager, Senior Architect, and Software Engineer, Casey has managed teams to tackle Big Data, architect solutions to difficult problems, and train others to do the same. He leverages experience with distributed systems, artificial intelligence, translating novel algorithms and academia into working models, and selling a vision of the possible to clients and colleagues alike. For fun, he models human behavior using personality profiles in Ruby, Erlang, Prolog, and Scala.*

How to Build the Wrong Thing Faster

And Learn From It

David Hussman, DevJam Studios,

Can ‘agile software development’ be refactored to ‘agile product development’? Some brave pioneers already doing this are re-learning that building good product is more opaque than simply getting work done. The land of product development is filled with holes, ambiguity and landmines of wrongness. Ideas that you are stone certain about often fizzle or change when you watch someone interact with your product. Being overly certain or focusing on ‘just getting work done’ to sustain velocity are mistakes that make matters worse.

Join me in an exploration of how to embrace wrongness, learn from it, and make it a vital part of our success. Our journey will explore the messy, sloppy and non-linear aspects of product development. Along the way, we’ll investigate how software construction is important, but courageously failing and learning in product is even more essential. We’ll look at how some teams are producing more real product value with less code. We will also peer into the world of program level development, where collections of teams produce better product by employing what might be called ‘test driven product.’

Who knows, toward the end of the journey, we might even rally to refactor the agile manifesto to read ‘Learning in Product over Simply Getting Things Done.’

David Hussman *teaches and coaches product discovery through iterative delivery. He has spent the last 10 years coaching agility, Lean practices and producing products for companies of all sizes around the world. For each engagement, David’s coaching is non-dogmatic, well-grounded, challenging and pragmatic. By focusing on really getting to know a project community, David seeds self-discovery and avoids falling into the expert trap of simply telling people what they “should do”. David spends most of his time pairing around code and tests, creating product ideas and roadmaps, and helping leadership teams pragmatically introduce the type of agility that fosters innovation and creates a competitive edge. David owns and guides DevJam (www.devjam.com), a composition of mentors who blend technology, people, and processes to create better products in competitive cycles. Visit his website at* ***www.devjam.com***

Poetry and Programming: Not as different as you think. One just pays a whole lot better.

Scott Poole, Columbia United Providers and Live Wire! Radio

Ever since Ada Lovelace, Byron's daughter, got back at him by becoming the world's first programmer, poetry and programming have been hopelessly related. In this humorous talk, poet and programmer Scott Poole takes a hard look at the love/hate relationship between the two halves of his brain. With the help of poetry and programming, he will explore warning signs and helpful tips for either avoiding or engaging your own artistic mania while maintaining the ruse of a normal citizen and productive developer of quality software.

Scott Poole *is most well known as the "House Poet" on the weekly Live Wire! public radio variety show, taped in Portland and broadcast nationally by Public Radio International. But he is also a software developer. Currently, he's the Senior Web Developer for Columbia United Providers in Vancouver, WA. He's been writing code for 9 years and poetry for 25. He is the author of three books of poetry,* The Cheap Seats*,* Hiding from Salesmen *and, most recently,* The Sliding Glass Door.

Testing in the Age of Distraction – Flow, Focus, and Defocus in Testing

Zeger Van Hese, Z-sharp

We live in curious times. Knowledge is available at our fingertips, everywhere. Social networks enable communication with peers from around the world. However along with these marvels of the information age come weapons of mass distraction. With so many things competing for our attention and with so little time to focus on real work, it's a wonder we get anything done at all.

What does all this distraction mean for testers? There is a common notion that only focused concentration leads to productive work and that distraction causes procrastination and hurts creativity. While it is important that testers are able to find flow and maintain focus, Zeger believes that a state of defocus – guilt-free play – can be helpful in testing. In this talk, Zeger shares tips, tricks and tools that over the years have helped him focus and defocus while testing. He explains how to benefit from distraction and how to find flow and focus when needed. Hopefully, this talk inspires people to make the most of their testing in these connected times.

Zeger Van Hese *(born and raised in Belgium) has a background in Commercial Engineering and Cultural Science. He started his professional career in the movie distribution industry but switched to IT in 1999. A year later he got bitten by the software testing bug (pun intended) and has never been cured since. Over the years, he developed a passion for exploratory testing, testing in agile projects and, above all, continuous learning from different perspectives. He was the 2012 Eurostar program chair and recently founded his own company, Z-sharp, dedicated to helping clients on the path to smarter testing. He is co-founder of the Dutch Exploratory Workshop on Testing (DEWT), founding member of the ISST, muses about testing on hisTestSideStory blog and is a regular speaker at conferences worldwide. Contact Zeger at zeger@z-sharp.be.*

Testing in The Age Of Distraction

The Importance Of (De)Focus in Testing

Zeger Van Hese

Z-sharp bvba

Opening Quote

"All ignorance toboggans into know
and trudges up to ignorance again:
but winter's not forever
even snow melts;
and if spring should spoil the game,
what then?"

- E.E. Cummings -

Biography

Zeger Van Hese has a background in Commercial Engineering and Cultural Science. He started his professional career in the motion picture industry but switched to IT in 1999. A year later he got bitten by the software testing bug (pun intended) and has never been cured since.

He has a passion for exploratory testing, testing in agile projects and, above all, continuous learning from different perspectives. Zeger considers himself a lifelong student of the software testing craft. He was a program chair of Eurostar 2012 and co-founder of the Dutch Exploratory Workshop on Testing (DEWT). He muses about testing on his TestSideStory blog and is a regular speaker at national and international conferences. In 2013, Zeger founded his own company, Z-sharp.

Prologue

The subject of distraction has kept me busy for quite some time now. To be honest, it kept me busy way longer than I anticipated, since I'm easily distracted and quite a big procrastinator. There is a lovely bit of irony in this all: in preparing and researching this paper and presentation about distraction, I was permanently distracted and ended up procrastinating like there's no tomorrow. In a way I became the very subject of what I was investigating. Come to think of it, being distracted while working on a piece about distraction does have a nice "meta" ring to it.

In one of the more useful corners of the internet I found out that "distracted" was once a synonym for "insane"[1,2]. In Shakespeare's Henry IV, a character named Falstaff utters the phrase "Poverty hath distracted her."[3] Falstaff didn't mean that the lack of funds made the woman in question absent-minded; he believed that it made her insane.

Distraction still has a bad name today. Many people see it as the ultimate enemy of getting things done, but I think that in reality distraction is often misunderstood. Over the years I developed strange relationship with it. Sure, it has worked against me, but it has worked wonders for me as well. In college, I noticed that unlike many of my friends I didn't need total isolation and silence. I was surprised to find I was able to study better with a radio on, or with a flickering TV in my line of sight. Without these permanent distractions, I just wasn't able to keep focus.

If you think that's awkward, you would get along fine with my parents.

This triggered me to further investigate the phenomenon. Gradually, this became a personal journey - I set out to learn about distraction, but I ended up learned a lot about myself.

86400

Imagine there is a bank that every morning deposits 86400 euros into your account. Every day this happens, over and over again. The only catch is that you cannot save that particular deposit until the next day. The 86400 euros you get in the morning are gone in the evening. What would you do? Would you think carefully about how you would use it each day?

The thing is, we all have this account available to us - except it's time we get to use. We all have 86400 seconds to spend every day. We cannot save them up. At the end of the day, they're gone.

Does that stress you out? Does this make you think of living differently?

We cannot hold on to these seconds, but we /can/ use them wisely. And by "wisely", I don't mean "being productive", at least not all the time. The real challenge is to manage our time so we can get the things done that really matter to us: we want to be happy, and at the same time be good partners, parents and friends. On top of that, we also want to excel professionally - we want to become better testers, coaches or managers

The Age of Distraction

"Our life is frittered away by detail... simplify, simplify."
- Henry David Thoreau -

We live in curious times. This is the Age of Information, but we might as well call it the Age of Distraction. Granted, humanity has never been free of distraction, but never have distractions been so overwhelming, so persistent as they are now.

At work, we have distractions coming from every direction: In front of us is the computer, with email and other notifications. In there, there is the addicting lure of the browser, which is nothing less than a black hole from which we can never escape - but it does offer unlimited opportunities for shopping, chatting and lolcat pictures. All the while, several new emails come in, waiting for a quick response. Several programs are open at once, the software under test one of them. And that is just in front of us. From the sides come a ringing desk phone, a ringing mobile, questionable music from coworkers, a colleague dropping by to ask a question, someone calling a meeting (and if we are lucky, someone offering a freshly baked cake).

On the way home, we are bombarded with advertisements, asking not only for our attention but also our money and desires. And all this continues well after we are parked on our driveways: television, home computer, iPad, kids and spouses - life in general.

With so many things competing for our attention, and so little time to focus on real work, it's a wonder we get any work done at all. So how exactly do we deal with this?

Some people claim to have found the solution: multitasking!

Multitasking

There is a slight problem with that, however: over the last twenty years, research has shown again and again that multitasking is a myth. When we think we are doing two things at once, we are almost always serial tasking - switching rapidly between tasks. We are not cruising along the information highway; we are stepping on the gas and then hitting the brakes, over and over. We are living in a state of continuous partial attention.

Our brain processes different kinds of information on separate channels - a language channel, a visual channel, an auditory channel, and so on - each of which can process only one stream of information at a time. If you overburden a channel, the brain becomes inefficient and prone to mistakes. If you have ever muted your car radio when being lost – I know I have - you will have experienced that the amount of attention we have is strictly limited, a zero-sum game. When all our attention is deployed to one modality – listening to the car radio - another modality suffers - in this case, the visual task of driving.

There are times when multitasking /does/ work, but only when following two conditions are met:

- At least one of the tasks is so well learned that it is almost automatic, like walking or eating.
- The tasks operate on entirely separate channels. For example, folding laundry (a visual-manual task) while listening to a test report (a verbal task). Which - I must admit - doesn't happen to me too often.

The figure below shows a model I created to help me to gain understanding of the forces at work in the system. It is by no means a scientific model - you will find that it is full of contradictions and paradoxes - but it does provide a good overview of the topics covered in this paper and the relation between them. I will cover the causes and different kinds of distraction, and highlight how distractions can be used to enhance focus and creativity. I will explain that phenomenon called procrastination and demonstrate that even that can be used in our favor. A good portion is reserved for focus and defocus, and tips and tricks to handle each one of them. I will conclude by explaining what all this means for testing.

Distractions are all around us. But why exactly is that a problem? Why are we so easily distracted? Can't we just ignore all things fighting for our attention? A it turns out, it is a little bit more complex than that. There are several factors - some human, some environmental - that cause us to be terminally distracted.

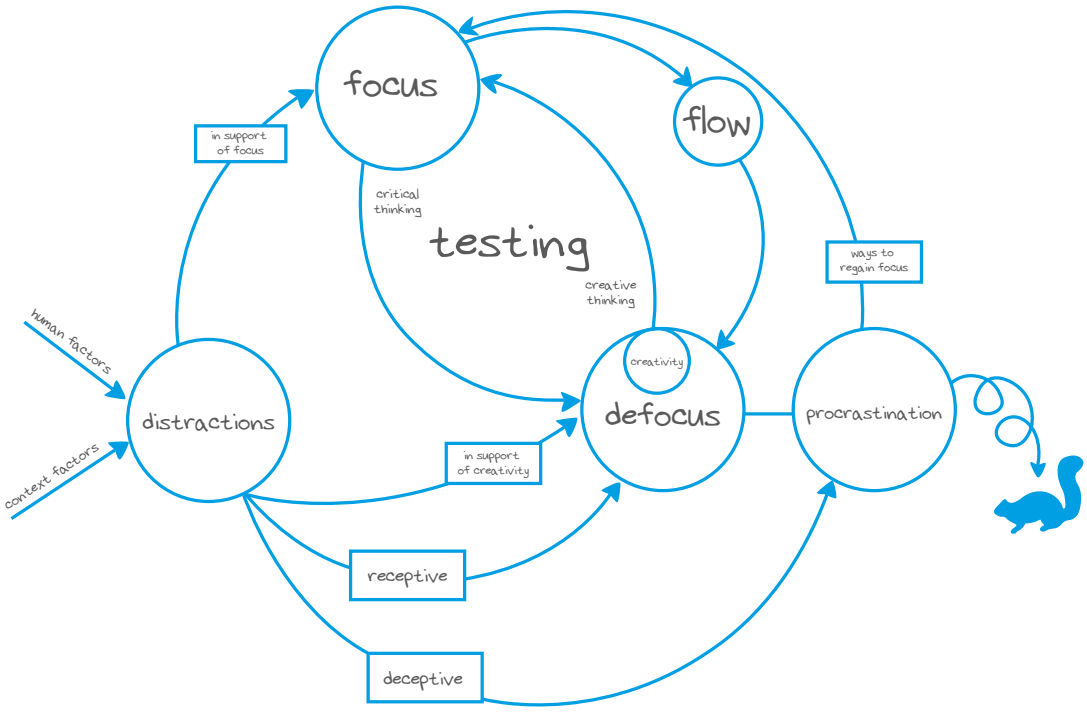


Fig. 1 - A Distraction Model

1. Human Factors

We are biologically wired for distraction.

We are biologically wired to be distracted[4]. When our ancestors were out hunting, and the bush next to them rustled, the ones who did not look up and see the lion coming at them – they are probably not our ancestors.

We are a daydreaming species.

According to a recent study by Harvard psychologists Matthew Killingsworth and Daniel Gilbert, people let their minds wander forty-seven percent of the time they are awake. In fact, the only activity during which we report that our minds are not constantly wandering is lovemaking – we are able to focus on that[5].

We make a lot of decisions in the course of a day. However, as the graph bottom left shows, making decision after decision comes with a biological price[6]. Decision fatigue is different from ordinary physical fatigue. You are not consciously aware of being tired; rather you become low on mental energy. The more choices you make throughout the day, the harder each one becomes for your brain, which eventually starts looking for shortcuts: you start acting impulsively and it becomes harder to resist urges (e.g. the urge to check your mail, twitter or the urge to eat the contents of your fridge).[7]

Fig. 2 - Decision Fatigue

This is also the reason why supermarkets place candy at their counters. By the time innocent shoppers get to the exit, they lose all resistance after the multitude of decisions they had to take in the aisles. Bereft of their willpower, they are an easy prey, especially vulnerable to candy and anything else offering a quick hit of sugar.

With decision fatigue in mind, there are some valuable lessons to be learned: if you want to stay in control, avoid scheduling meetings back to back, and taking important decisions at the end of the day.

It's an addiction.

I recently learned that the most profitable parts of a casino are the slot machines, because use a principle called "Variable-Ratio Schedule", also known as random payout. If you pull the handle on a slot machine and it pays you the same amount every hundredth time, you would quickly stop playing. But if it pays you a little bit some times, other times nothing and occasionally a lot, you are going to pull that handle a long time. Now think about text, email, twitter and Facebook messages in that setting: some are important, some are really trivial, and sometimes it's going to be something really urgent – they are random payout in your pocket.

2. Context Factors

Apart from these internal factors, there is also one big external factor that does that as well:

Jessica Hagy's graph on the bottom right[8] hits the proverbial nail on the head. There is nothing wrong with acquiring more information, which is all good and clarifying. When the amount of information coming at us rises however, we get more confused instead. This is a phenomenon also known as information overload, a term popularized by Alvin Toffler in his 1970 book Future Shock[9]. Information overload refers to the difficulty a person can have understanding an issue and making decisions that can be caused by the presence of too much information.

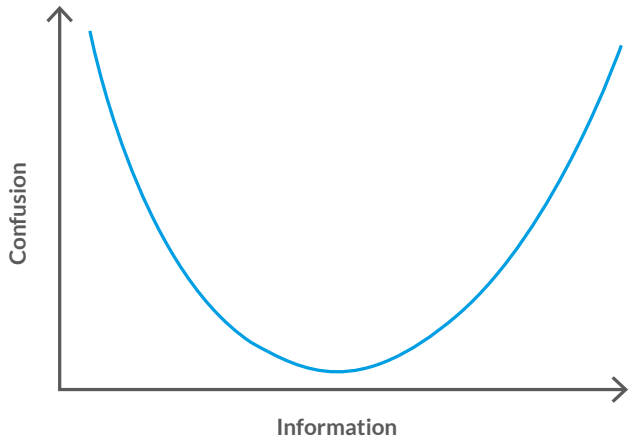


Fig. 3 - Information Overload

"It's not information overload. It's filter failure."
- Clay Shirky -

Actually this external factor could be considered a hidden internal factor. Maybe we shouldn't be blaming the information that is out there. It's us who are not able to filter the chaff from the wheat.[10]

Distractions come in all shapes and sizes - and in roughly two categories: receptive and deceptive ones. A receptive distraction is any sort of distraction that creates

mental space, one that relaxes you and helps you regain your focus in the longer term. This can be as simple as getting a glass of iced-tea, or a walk outside for a few minutes.

Deceptive distractions are all things that make you lose track of what you were working on and cause you to get immersed in all sorts of other issues. Examples: M&M's (managers and meetings), emails, phone calls, the internet in general. Of course, this is highly person-dependent: some people do get energized by watching forty variations of the Harlem Shake.

The likelihood of a distraction being receptive is tied to whether it engages a different set of skills than the task being distracted from. For example, reading/writing code and reading/writing emails or blogs are activities so similar that they are almost always deceptive. In general, deceptive distractions are the kind of distractions that facilitate procrastination.

Procrastination

The figure below may be painfully recognizable for many - it certainly is for me. We keep putting off stuff that needs to be done, and decide to get busy only when a deadline is breathing down our necks. Afterwards we proudly say "I did it again", but in reality these situations are draining our energy.

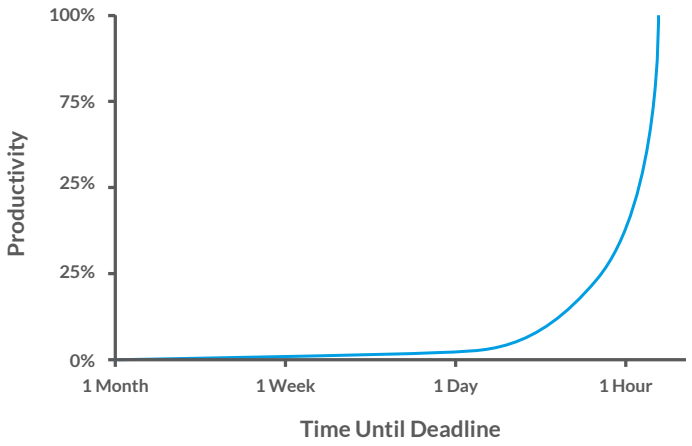


Fig. 4 - Procrastination in a nutshell

"Anyone can do any amount of work, provided it isn't the work he is supposed to be doing at that moment."
- Robert Benchley, 1949 -

This quote from Robert Benchley[11] summarizes procrastination well: in the face of a daunting task, all of a sudden everything else seems super appealing. Cleaning that desk? Oh, I need to do that to be able to work. And no way I can write without all my pencils sharpened. Yes, I know the work is urgent, but the lawn needs mowing.

Procrastination

There are a lot of misconceptions about procrastination. Some say it is plain laziness, others see it as attention deficit, but it is not. Procrastination does not mean doing absolutely nothing. When procrastinating you are actually doings all sorts of things, just not what you are supposed to be doing at that moment.

Procrastination is not a character defect, the main cause are underlying psychological problems that need to be addressed. We procrastinate to temporarily relieve deep inner fears:

- Perfectionism that paralyses you and keeps you from finishing stuff ("I need more time to finish that", "no way, this is not ready yet")
- Fear of failure, of not being good enough ("I am not an expert in this stuff, I'll make a fool of myself")
- Anxiety of starting and finishing ("If I manage to finish this, I'll even get more of this stuff to do and more responsibilities")

This causes a vicious circle of despair: we feel guilty because we haven't done something, which makes us feel more stressed. If we feel more stressed we procrastinate even more.

I should note that there is a thin line between procrastination and defocus - that's why it shows just that in the model. As James Bach mentioned in his 2009 book Secrets of a Buccaneer-Scolar, a lot of our procrastination leads to learning. Not always relevant for the task at hand, but possibly later on. Lots of our learning is a side effect.[12]

My Journey Into Focus

There I was, struggling in preparing this paper and the presentation to go with it, which was supposed to premiere at the Let's Test conference in May. At some point I was desperate enough to join a Facebook group about creative productivity, which - I must admit - sounds like buying a chair about jogging. But to be honest, some good tips came to surface there. One of them was to observe your procrastination habits. Take note of your procrastination habits and note down when and why it happens. If you do this for a couple of days, patterns will start to emerge.

Mid February - in full procrastination frenzy - I stumbled upon the book "The Now Habit" by Neil Fiore.[13] The book takes a look at the psychology behind procrastination and explains why and how we procrastinate. A couple of tools concepts in the book have stuck with me since:

Reverse Calender

The reverse calendar is a backwards schedule of the task ahead: you start from the end goal or deadline and list all the smaller activities that need to be done in order to reach that goal. This simple technique forces you to split a big, daunting task into

smaller, more achievable chunks of work. Mapping it in reverse helps you to keep the end goal in mind.

The figure right gives you an idea what my reversed calendar looked like in February, three months before my not very negotiable deadline. Much to my surprise it worked for me: the exercise eased my mind a great deal, and all of a sudden three months looked like the sea of time it actually is.

May 22, 2013	Presentation at **Let's Test**
May 17, 2013	First dry run
May 12, 2013	Presentation (Prezi) ready
May 04, 2013	First draft of paper finished
April 01, 2013	Rough outline finished
March 17, 2013	Research - Backlog of articles
March 15, 2013	Research - Checklist manifesto
March 08, 2013	Research - Focus manifesto
March 01, 2013	Start tool experiments
February 28, 2013	Research - Now Habit
February 19, 2013	Current day
February 11, 2013	Submitted for **EuroSTAR**
September 23, 2012	Submitted for **Let's Test**
January 2010	Initial idea 'Testing In The Age Of Distraction'

Table 1 - Reverse Calendar

Unschedule

Neil Fiore defines an unschedule as a week schedule, but with a twist. Instead of scheduling the work you have to do, you first fill in everything you want to do: first plan in time for social activities, play and hobbies and only then fill out the remaining gaps. This sounded relatively simple to do, so I decided to take a shot at it. I enrolled for a jazz initiation course, scheduled long overdue dinners with friends, even allocated some time for reading and watching television. I even put in some running during lunchtime breaks at a client site. When the fun slots were filled, I added the mandatory stuff (work times, meetings, other appointments, household matters). Only then I allocated time for the things I needed to work on.

The result of this change in perspective is that I felt refreshed and not guilty at all when not being productive, because it was all planned activity: guilt-free play. Furthermore, this changed my mindset from: "This week I have to work on my presentation for a minimum of 4 hours" to "I have a maximum of 4 hours to spend on my presentation". Subtle changes like this made me more eager to start the work.

Structured Procrastination

If you like science, but also like the sometimes delightfully absurd side of scientific research, The Ig Nobel Prizes are something worth looking into. They are handed out each year for unusual achievements in scientific research, and their stated aim is to "honor achievements that first make people laugh, and then make them think." In 2011, the Ig Nobel literature prize went to John Perry of Stanford University, for his Theory of Structured Procrastination.

Perry's Theory of Structured Procrastination[14] states:

"To be a high achiever, always work on something important, using it as a way to avoid doing something that's even more important."

Perry is convinced that you /can/ use procrastination to get loads of important stuff done, if you just make sure to have that one thing that seems even more urgent and important. The trick is of course to pick the right sorts of projects for the top of the list. These ideal projects have two characteristics: first - they seem to have clear deadlines (but really don't), and second - they seem awfully important (but really aren't).

After the initial obligatory laugh, I found out that I was doing just that in my personal life. I recently became independent and I absolutely need to work on my website, since there only is a temporary one-pager up there now. The website is always the top item on my list, and I definitely need to start working on that, but I tend to avoid it by doing other useful work instead. That company website seems awfully urgent and important to me (but it really isn't). And that deadline that I keep giving myself is rather imaginary. Self-deception for the win!

Using distractions as a way to focus sounds like a paradox if ever there was one. Aren't distractions supposed to be the enemy of focus? That used to be my conviction as well, but vividly colored flash-backs to the time when I was able to focus and memorize better with TV and radio on made me look into the phenomenon. I stumbled upon a couple of scientific studies that might explain what was going on there.

High Brain Load

We testers are very familiar with the phenomenon of inattentional blindness: when we focus intently on one task, we often fail to see other things in plain sight. A study by the Journal of Cognitive neuroscience[15] shed a different light on this. If we bring our brain under high information load, our processing becomes selective, and apparently we can ignore irrelevant distractions more effectively. Basically this is a way to use inattentional blindness to our advantage, when we are able to be blind for irrelevant details.

Unconscious Processing

When faced with a difficult decision, it is often suggested to "sleep on it". New brain imaging research from Carnegie Mellon University finds that the brain regions responsible for making decisions continue to be active even when the conscious brain is distracted with a different task[16]. The most intriguing part about this is that participants in the study did not have any awareness that their brains were still working on the decision problem while they were engaged in an unrelated task. With complex decisions, a brief period of distraction - while your brain can unconsciously process information - can be advantageous for your decision-making and your learning. This means that distraction can not only help us process information better, but also enables us to learn.

Focus

I mentioned before that distraction is often seen as the enemy of focus, and it certainly is if you're working on something that requires mostly logical thought. The problem here is that our brain has a "working memory" where the logical reasoning takes place. When our attention wanders, that working stack is dumped and it can take up to half an hour to get it back up to speed. Needless to say that the ability to focus is an important asset for a tester - here is a small selection of things that improved my ability to focus over the years.

Focusing Tips & Tricks

The Beauty Of Disconnection

If you feel that social media are the source of your interruptions, your best bet may be to disconnect from the internet. Keep in mind though that timing is essential. If you plan a focused test session and plan to disconnect before you start, do not post a tweet or Facebook message right before disconnecting. Don't tweet and quit. You will want to keep checking for reactions - that kind of random payout is highly addictive.

If your work requires access to the Internet, only turn off the things that risk distracting you. If you don't trust yourself to do that, there are plenty of blocking tools or apps available to do that for you. Here are two that worked wonders for me:

- SelfControl: blocks access to mail servers and websites for a predetermined period of time
- StayFocusd: a Google Chrome extension that helps you stay focused on work by restricting the amount of time you can spend on time-wasting websites

You might also want to look into using multiple desktops on your computer. The idea is to put the software under test on one desktop, and have possible distractions like twitter or Facebook quarantined on the other.

- For Windows, VirtuaWin is one such program
- Mac OS has a multiple desktop functionality built in in the form of "Spaces"

You Don't Need To Respond

This also means cutting down on all sorts of notifications. I made my personal mantra "You don't need to respond". We have developed a need to respond to so many things, and before we know our day becomes responsive rather than driven by conscious choices. Here is a convenient truth: you don't need to respond - you can get in control and decide when you respond.

The (10+2)*5 Procrastination Hack

Merlin Mann's (10+2)*5 procrastination hack has three simple principles:[17]

- 10 - Work for ten minutes with single-minded focus
- 2 - After ten minutes of dedicated work, take a 2-minute break to do whatever you want
- *5 - Iterate this four more times for a total of one hour

I first started using this nifty little technique intensively in the spring of 2012 when I was programme chair for EuroSTAR. I had to work my way through more than 420 submissions, and I soon learned that consuming large amounts of awkwardly formatted text brimming with buzzwords really threatens your sanity. My first attempts at making progress were not very successful, but from the moment I forced myself in the rigid structure of the procrastination hack, my productivity tripled. Many variations on this hack exist (the Pomodoro technique,[18] among others),but they all have the same core elements in common: strict time-boxing and short breaks.

In my experience, these short time-boxing techniques work really well with easily divisible logical/analytical tasks, like going through defect lists or reviews of any kind - but they don't match well with more creative, test design work. Creative people - designers, programmers, writers, engineers, thinkers and sometimes testers – typically need longer stretches of uninterrupted time to get their creative juices flowing. You cannot ask somebody to be creative in 15 minutes and really think about a problem. You might have a quick idea, but to be in deep thought about a problem and really consider a problem carefully, you need long stretches of uninterrupted time. Creativity needs time in order to unfold in unpredictable ways.

Test Approach: Time-Boxed Focused Test Sessions (SBTM)

This is also the reason why in exploratory testing sessions longer time boxes are usually a good idea. A typical format when doing exploratory testing is Session Based Test Management (SBTM),[19] with a time box (the session) and a defined scope - the charter. Here's how I usually manage my focus in such a session:

- I block off time for the session (1 to 2 hours), close email and Instant Messaging systems, let people know that I'm in a session (or arrange a meeting room if necessary)

- Before I start, I already open up everything I think I'll need to do my testing effectively: applications, databases, spreadsheets, tools, etc. This prevents me from getting distracted or slowed down mid testing
- Once the session started, I don't stop my testing to write up defects or to ask questions – I note them while I test, and I come back to the issues and questions later when the session is completed
- I plan faster tests and higher risks tests (or tests more important to the business) first
- I group features together to reduce context switching while testing

The To-Not-Do List

Another tool I started using is the to-not-do list. The concept is brilliant in its simplicity: at the start of every month, sit down and list all the things on your mind that you're not going to worry about doing that month. What could be more time-saving than that? As Jerry Weinberg stated in More Secrets of Consulting: "Anything not worth doing is not worth doing right."[20]

Flow

Have you ever found yourself in a state of totally being in the zone, so involved in an activity that nothing else seemed to matter anymore? That is flow, the ultimate state of being focused, the Walhalla of focus if you like. The term flow was coined by Mihály Csíkszentmihályi in his 1990 book "Flow: The Psychology of Optimal Experience."[21] His flow theory states that three conditions need to be fulfilled to achieve a flow state:

1. One must be involved in an activity with a clear set of goals and progress
2. The task at hand must have clear and immediate feedback
3. There is a good balance between the perceived challenges of the task at hand and our own perceived skills. It all boils down to having confidence that we are capable to do the task at hand

With this in mind, how could we as testers increase the chance of achieving a flow state in our work?

1. Clear set of goals and progress. This can be accomplished when we add direction to our testing: have a clear mission, and guide that mission with charters while testing
2. Clear & immediate feedback. There are ways to enable quick feedback during. Working in an exploratory fashion is a good way to make the feedback immediate: you design a theory, test for that theory and immediately see the result of that
3. The higher the skill level, the greater the confidence in our capability, and the higher the chance of getting in a flow state

But in all this looms a paradox for testers:

In order to achieve flow, Csíkszentmihályi says, people should be able to suspend their critical abilities for a while. But what happens when testers suspend their critical abilities? They probably will start missing important stuff. The feeling of flow can also work like an opiate, as it feels good to do things we are good at. Beware of the opiate of expertise: to improve, we sometimes have to seek discomfort and step out of our comfort zone.

If you ask people when or where they get their best ideas, the same similar answers come up. Most people will tell me “in the shower”, “in the car”, “while running”, “while walking in nature”, “when thinking about other stuff”. Personally, I have gotten my best ideas either when running or during long commutes in the car.

Mind-Wandering Promotes Creativity

Ideas typically don’t occur when we are focused on tasks. They happen when the mind starts wandering. You could say that mind-wandering promotes creativity: it is unconstrained and can go anywhere, which is the perfect condition for creative thought. In recent years, psychologists found that mind-wandering is an essential cognitive tool: daydreams seem to have a similar function as night dreams, facilitating bursts of creative insight.[22, 23]

Chance Favors The Connected Mind

In his 2010 book and TED talk on where good ideas come from, Steven Johnson explains that great ideas generally occur thanks to the combination of your hunch with someone else’s hunch. To get to that point, you need to place yourself in environments that foster good collaboration. Historically, he saw that happen in coffee shops, where scientists could meet face to face to exchange ideas. Nowadays, while the internet can often be a distraction, it can also be a fantastic collaborative environment.[24]

Social media are a double-edged sword. While they often are the portal to procrastination and oblivion, they can do wonders for testers as well. So far, twitter has proven to be the most valuable of them all. It has helped me to:

- Virtually attend testing conferences, even mingle in the discussions - all it takes is the right hashtag (e.g. try #esconfs, #LetsTest, #CAST201x or #AgileTD for starters)
- Bounce off new ideas on my followers. It is an instant-feedback sounding board. I often give other people feedback on their ideas too, which in turn inspires new ideas. Twitter’s function as an idea fertilizer / incubator cannot be underestimated

> Facilitate connection in real life. The Dutch Exploratory Workshop on Testing (DEWT) was formed by a couple of guys discussing on twitter and suggesting to meet up some time

During testing you can use Twitter to get in touch (and familiarize yourself) with the users of a product, and see how they perceive the service being delivered. Pradeep Soundararajan showed an eye-opening and at times hilarious example of something he called Twitter-driven exploratory testing[25] at the Øredev conference in 2011. Pradeep used twitter as a source of test ideas by entering the company name as a search term and looking for emotions associated with those tweets (ranging from sad smileys over #fail to a plain old #f**k). Being aware of your (and other people's) emotions is one of the most important oracles for a tester in order to find bugs.

Ambient Noise (Also Known As "The Coffee House Effect")

A paper in the Journal of Consumer research found that ambient noise can affect creativity. Results from experiments demonstrate that low (50 dB) to moderate (70 dB) levels of ambient noise enhance performance on creative tasks, while high levels of noise (> 85 dB) hurt creativity.[26]

Behold, the secret behind that productivity boost in coffee bars.

For the unfortunate few that don't have the luxury to go working in coffee bars, there is a website that will bring you in the right creative mood using ambient noise from coffee houses - minus the coffee: coffitivity dot com. This is great news for the agoraphobics among us: you can now get all cozy and creative without even having to leave the house.

Defocus

Dictionary.com defines the verb "defocus" as follows: "to lose concentration or awareness; become distracted". Since people cannot stay concentrated indefinitely, defocusing is a natural reflex for us. This is often fed by distractions. When receptive, they help us defocus from the tasks at hand and refresh our minds, which in the end - paradoxically - will help us to focus better.

A lot of people look down upon defocusing, thinking that staying focused is the only way to get things done. But if study professional athletes, you will notice that rest days are an explicit part of their training programs. Their coaches plan for them because an athlete's body absolutely needs to recuperate and recover in order to come out stronger.

Everyone accepts that athletes rest their bodies, since they are in such a physical line of work. Is it such a crazy idea that we, as knowledge workers, plan for rest as well to let our brains recuperate and to rejuvenate our thinking?

I think this addresses an important issue affecting testing. When solving testing problems, people always assume that you get more done when you are consciously paying attention to a problem. After all, that's what it means to be "working on something". But if you are trying to solve a complex problem, you need to give yourself a real break. Whenever creative thinking is needed, our mind performs the best when we're defocused.

There also is a lesson here: beware of going off on an appearance of work. Knowledge work doesn't always look like work. I made this mistake once: at a customer site there was this one person I didn't know too well who was constantly going outside for coffee and cigarettes. When I mentioned to someone that that person was probably not their most productive or motivated employee, they were surprised: "No, no, he is the mastermind behind our whole new program - he has to go outside - that's where he does all his thinking!" They knew, I didn't - I will never make that mistake again.

The "Follow Your Energy" Heuristic

While testing, sometimes you just got to follow your energy. It is okay and natural to let your mind wander. When you allow yourself to be distracted, your mind will start making connections. This is what James Bach calls the "Follow your Energy"-heuristic. If you want to be more in control of your learning, it is advised to combine the "Follow your Energy"-heuristic with the "Long Leash"-heuristic. Let your mind drift off, but in a controlled manner - keep it on a long leash. Every now and then, remind yourself that you are on a mission and gently pull on the leash to regain focus again.[27]

The "Get A Coffee" Heuristic

In the product I'm testing now, I came across some crashes that seemed random and were really hard to reproduce. After a while I noticed that the frequency of those bugs was higher right after lunch or after coffee breaks, when the application stayed open for a longer time. As it turned out, quite a lot of them were time-related, and not related to the actions done in the user interface.

From that moment on, I've been using the "Get-a-coffee" heuristic: take a break, go do something else for a while - works wonders for bugs that are impacted by time, but it is also great for bug finding in general: you tend to see things in a new light when defocused: fresh eyes find failure.

Last year I tested apps on digital TV. It were educational apps consisting of well-known children's stories with a karaoke-style text you could follow along - great for kids who are just learning how to read. One afternoon, I did a focused test session of the help pages that were there in four different languages. I found a major bug concerning the fact that some text was in the wrong language. After some intensive further investigation, I was convinced that had I found the most important bugs and

decided to stop testing. Until I took a coffee break, returned and noticed that the wrong screenshots were shown, something I failed to see for three hours in a row.

Notetaking

Work on your note taking skills. Always be ready to capture, everywhere. Inspiration strikes at strange times: right before you fall asleep, right after you wake up, in the shower, while running, in the car. You will forget most insights if you don't immediately write them down.

I keep a notebook on my night stall. I use Evernote on all my devices. It's available on different platforms and able to store snippets of text, websites, even spoken text. I tend to get lots of ideas while running or driving in the car, and I record spoken notes with my phone's voice recorder and mail them to Evernote. My Evernote library is what Steven Johnson would call a "spark file", a file that contains all your half-baked ideas in one place. The trick is to reread your stack every three or four months to get new insights and detect previously undiscovered relationships.[28]

Work With Your Own Circadian Rhythm

Your circadian rhythm is your internal biological clock. The trick here is to get familiar with your biological clock and plan accordingly. This sounds like common sense, but surprisingly this kind of common sense isn't too common: avoid scheduling an important meeting or a focused test session at a time when you will be operating on only one cylinder. And don't waste your peak work time at a doctor's appointment.[29]

Guilt-Free Breaks

As I mentioned above, taking regular guilt-free breaks is essential to being productive. The problem with that is that you need to get back to work at some stage. When you get back from a break, you cannot just restore your "working stack" from backup, you have to overcome some resistance to starting again. Here are two techniques for getting the best out of breaks: priming and leaving breadcrumbs.

Priming

Priming is very easy. Before you take a break, just identify the next problem that you face in your work and think about it for a few seconds. Just think "I'll need to have decision on this by the time I come back" and let unconscious processing do its work.

Leaving Breadcrumbs

Hänsel and Gretel used the breadcrumbs technique first, but it didn't really work for them. We can apply the same technique to mitigating the downside of having a break: simply dump your "working stack" on a piece of paper or a text editor. You don't need to write a whole novel, a few well chosen words can jog your memory will make it much easier to restore your working memory from its backup.

Closed / Open Mode Thinking

In a lecture on creativity in 1991, Monty Python's John Cleese contrasted "closed" with "open mode" thinking. Closed mode is the thinking mode we are typically in when at work: active, tense, anxious, purposeful, trying to get things done, stressed and a bit manic. This is when our mind has focus. In contrast, open mode is relaxed, not purposeful, more humorous, playful and curious, because we're not under pressure to get anything done.[30]

"This is the extraordinary thing about creativity:
If just you keep your mind resting against the subject in a friendly but persistent way, sooner or later you will get a reward from your unconscious."
- John Cleese -

Cleese argues that creativity is not possible in closed mode. Only when our mind is in open mode – which maps to a defocused state - creative magic can happen. In practice, there should be a time and place for each mode: developing ideas and reviewing concepts are best done in open mode, while deciding and carrying out a plan of action are typically done in closed mode.

Creative / Critical Thinking

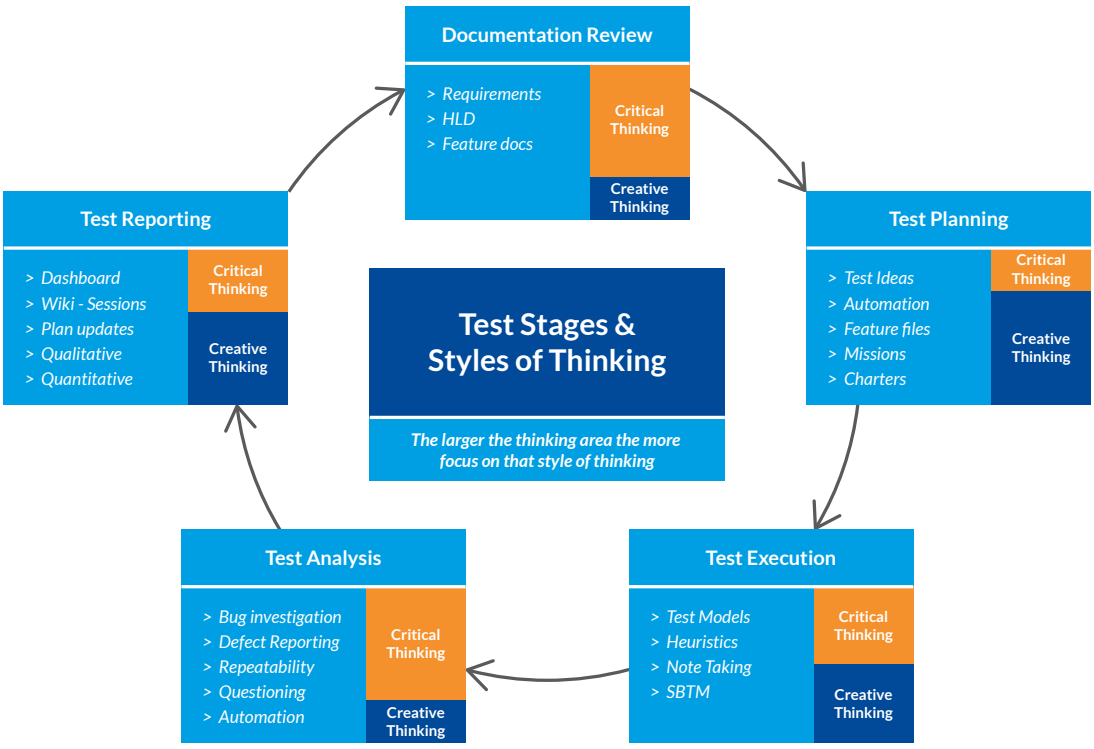


Fig. 5 - Test Stages and Styles of Thinking – John Stevenson

In a blog series earlier this year, John Stevenson wrote about creative and critical thinking, and which style of thinking to use in the different phases within testing.[31]

Creative thinking is the generation of new ideas. It is divergent – going in different directions. Critical thinking is thinking about thinking, with the intention of avoiding being fooled. It is convergent – bringing ideas and thoughts together.

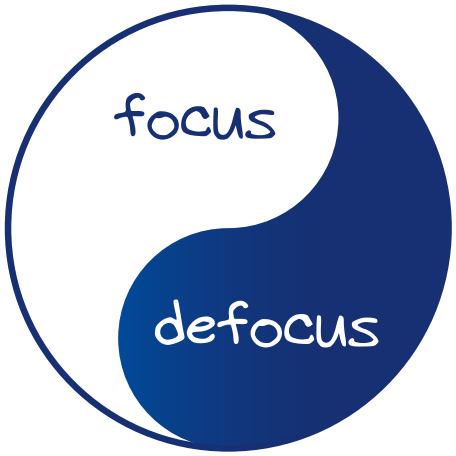
John came up with the diagram above, in which is clear that we need both styles of thinking in all of our testing efforts, but some activities rely heavily on critical thinking (e.g. reviews and analysis) while others mainly need creative thinking (e.g. generation of test ideas while planning). Activities like exploratory test execution (session based testing, note taking) need a good mix of both to be done well.

John Cleese's theory and John Stevenson's diagram illustrate the point I would like to make: to test effectively, we need to be able to switch between open and closed mode, between creative and critical thinking. In other words, managing our focus is a very important skill in testing. To think critically, we need to be focused. To think creatively, we need to embrace defocus. The trouble is that we often get stuck in tunnel vision when we would really need to step back and contemplate the wider view.

Focus is a paradox - it has distraction built into its very core. The two are symbiotic; they are like the yin and yang of consciousness. Focus and defocus are complementary (instead of opposing) forces interacting to form a dynamic system in which the whole is greater than the parts.

If humans are not designed to avoid distractions, why don't we harness, rather than fight, the power of distraction? That is the question I asked myself a while ago, and that was what I started doing over the last months. Thinking about how focus and defocus interrelate and how they can be used to our advantage gave me a better understanding of myself. It has changed the way I approach things. As a tester, it has made me more aware of traps and prejudice. Looking back, I now make more conscious choices in my work:

- I no longer see my procrastination as something bad. I'm now more aware when it happens, and it doesn't freak me out anymore. Much to my surprise I was able to start using it to my advantage.
- I now make conscious use of unconscious processing: instead of diving head over heels in a problem or a task, I study the problem, read up on it and then I decide to unwind, without feeling guilty about it.
- I started paying attention to the nature of the activities that need to be done. For the focused, logical tasks that require analytical thinking, I get disconnected and set time boxes with regular breaks. For tasks requiring a good amount of creative thinking, I make sure I'm connected and distractible.



References

1. In Defense of Distraction
   Sam Anderson, NY Mag, May 17, 2009
2. Shakespeare's false friends: distracted
   David Crystal, e magazine 24, 2004,
3. King Henry IV, Second Part
   William Shakespeare, Act 2, Scene 1, 1596 – 1599
4. Slow Tech
   Joe Kraus, TED among friends, 2012
5. A Wandering Mind Is an Unhappy Mind
   Matthew A. Killingsworth, Daniel T. Gilbert (Harvard University), Science 2012
6. Does decision fatigue affect your willpower?
   Maia Szalavitz, August 2011
7. Do You Suffer From Decision Fatigue?
   John Tierney, New York Times Magazine, August 17 – 2011
8. Needles and haystacks and such
   Jessica Hagy, ThisIsIndexed.com, October 9 - 2009
9. Future Shock
   Alvin Toffler, 1970
10. It's not Information Overload, it's Filter Failure
    Clay Shirky, New York Web 2.0 Expo keynote, 2009
11. How to Get Things Done: One Week in the Life of a Writing Man
    Robert Benchley, Chicago Tribune, February 2 – 1930
12. Secrets of a Buccaneer-Scholar: Self-education and the Pursuit of Passion
    James Bach, 2009
13. The Now Habit
    Neil Fiore, 1989
14. How to Procrastinate and Still Get Things Done
    John Perry, Chronicle of Higher Education, February 1996
15. Inattention blindness due to brain load
    Nilli Lavie, UCL Institute of Cognitive Neuroscience, July 2012
16. Neural Reactivation Links Unconscious Thought to Decision Making Performance
    J. D. Creswell, J. K. Bursley, A. B. Satpute (Carnegie Mellon University), Social Cognitive and Affective Neuroscience, 2013
17. Procrastination hack: '(10+2)*5'
    Merlin Mann, 2005
18. The Pomodoro Technique, 3rd Edition
    Francesco Cirillo, 2013

References

19. Session Based Test Management
    James Bach blog

20. More Secrets of Consulting: The Consultant's Tool Kit
    Gerald M. Weinberg, 2001

21. Flow: The Psychology of Optimal Experience
    Mihály Csíkszentmihályi, 1990

22. Why Great Ideas Come When You Aren'tTrying - Allowing the Mind to Wander Aids Creativity
    Matt Kaplan, Nature International Weekly Journal of Science, May 2012

23. Inspired by Distraction: Mind Wandering Facilitates Creative Incubation
    Benjamin Baird, Jonathan Smallwood, Michael D. Mrazek, Julia W. Y. Kam, Journal of the Association for Psychological Science, 2012

24. Where Good Ideas Come From – The Natural History of Innovation
    Steven Johnson, 2010

25. Twitter-driven Exploratory Testing
    Pradeep Soundararajan, Moolya blog, October 22 - 2011

26. Is Noise Always Bad? Exploring the Effects of Ambient Noise on Creative Cognition
    Ravi Mehta, Rui Zhu, Amar Cheema, Journal of Consumer Research, Vol. 39, No. 4, December 2012

27. Long Leash Heuristic for Critical Thinking
    James Bach, Michael Bolton, Rapid Software Testing course

28. The Writer's Room - The Spark File
    Steven Johnson, 201229. Power sleep: the revolutionary program that prepares your mind for peak performance James Maas, 1998

30. Five Factors to Make Your Life More Creative
    John Cleese, 1991

31. Creative and Critical Thinking and Testing
    John Stevenson, Blog article, 2013

Additional Reading

Focus (a simplicity manifesto in the age of distraction)
Leo Babauta, 2010

Forget Flow – The Secret to Skill Lies in Discomfort
Gregory Ciotti, February 2013

Why work doesn't happen at work - TED talk
Jason Fried, 2010

Getting Things Done: The Art of Stress-Free Productivity
David Allen, 2002

Dealing with digital distraction
Alex Talbott, September 2012

Your Brain: The Missing Manual
Matthew MacDonald, 2008

Shutting out a world of digital distraction
Carl Wilkinson, The Telegraph, 6 September 2012

Neuroscience: Exploring the Brain, 3rd Edition
Mark F. Bear et al., 2006

Do It Tomorrow and Other Secrets of Time Management
Mark Forster, 2008

Pragmatic Thinking and Learning: Refactor your Wetware
Andrew Hunt, 2008

Pomodoro Technique Illustrated: Can You Focus - Really Focus - for 25 Minutes?
Staffan Noteberg, 2009

Too Much To Know: Managing Scholarly Information before the Modern Age
Ann M. Blair, 2010

The Checklist Manifesto: How to Get Things Right
Atul Gawande, 2009

"The Paradox of Choice",
TED talk, Barry Schwartz, 2005

Creative Intelligence: Harnessing the Power to Create, Connect, and Inspire
Bruce Nussbaum, 2013

Weinberg on writing: the fieldstone method
Jerry Weinberg, 2005

Lifehacker
http://www.lifehacker.com

Coffitivity
http://www.coffitivity.com

"Does Luck Matter More Than Skill?",
Study Hacks blog, Cal Newport, 2013

Additional Reading

Noteboard
http://thenoteboard.com

More than Just 'Zoning Out – Psychological Science Examines the Cognitive Processes Underlying Mind Wandering
Association for Psychological Science, October 2012

Multiple media use tied to depression, anxiety
http://msutoday.msu.edu/news/2012/multiple-media-use-tied-to-depression-anxiety/

From procrastination, distraction or disengagement to purposefulness
Leanne Ansell-McBride, August 2012

Learning to be More User Centered

Eileen Forrester, Forrester Leadership Group

In your organization, is the quality function a real priority–or is it an outsider, an afterthought, a necessary evil, or a target for outsourcing? Whether we are talking about quality as testing, compliance, improvement, or brand perception, if you treat and manage quality like a service instead of a function, you can get more of what you want: better software, happier customers, smarter use of resources, and great business outcomes.

This approach works for both the provider and consumer of quality services, and for both in-house and outsourced quality. Making the shift to treating quality like a service means both provider and consumer have to be clear about what they need and expect, how those needs will be met, who is responsible for what, when, and for what cost.

In this talk, Eileen will present case examples showing the benefits of quality-related testing as a service, and also, somewhat surprisingly, how compliance and organizational improvement activities have benefited from a service mindset.

By using a service agreement approach, thinking of the various standards and frameworks as clouds from which you compose your quality system can be a better fit than wholesale adoption of any one quality framework—like a great coffee blend that gets you the exact taste, acidity, and finish you want.

Eileen Forrester *is a seasoned entrepreneurial leader, experienced in managing both for-profit and non-profit organizations in industry segments spanning technology, products, research, and training.*

Her company, Forrester Leadership Group, provides executive coaching, technology and product strategy, capability coaching and consulting.

Acceptance Test-Driven Development: Better Software through Collaboration

Ken Pugh

ken.pugh@netobjectives.com

Abstract

Defining, understanding, and agreeing on the scope of work to be done is often an area of discomfort for product managers, developers and quality assurance experts alike. Many of the items living happily in our defect tracking systems were born of the difficulty we have in performing said activities. Acceptance testing roots out these defects by reexamining the process we use to define the work to be done and how it is tested. Acceptance test-driven development helps with communication between the business customers, the developers, and the testers. The 5 W's are covered- *what* are acceptance tests, *why* you should use them, *when* they should be created, *who* creates them, and *where* they are used.

Biography

Ken Pugh (ken.pugh@netobjectives.com) is a fellow consultant with Net Objectives (www.netobjectives.com). He helps companies transform into lean-agility through training and coaching. His particular interests are in communication (particularly effectively communicating requirements), delivering business value, and using lean principles to deliver high quality quickly. He also trains, mentors, and testifies on technology topics ranging from object-oriented design to Linux/Unix . He has written several programming books, including the 2006 Jolt Award winner Prefactoring, Interface-Oriented Design, and his latest book Lean-Agile Acceptance Test Driven Development: Better Software Through Collaboration. He has helped clients from London to Boston to Sydney to Beijing to Hyderabad. When not computing, he enjoys snowboarding, windsurfing, biking, and hiking the Appalachian Trail.

Ken has a B.S.E from Duke University and an M.S.E. from University of Maryland

Copyright Kenneth Pugh, Net Objectives 2015

1. *What* Are Acceptance Tests?

Acceptance tests are from the user's point of view – the external view of the system. They examine externally visible effects, such as specifying the correct output of a system given a particular input, as shown in Exhibit 1. Acceptance tests can show how the state of something changes, such as an order that goes from "paid" to "shipped". They also can specify the interactions with interfaces of other systems. In general, they are implementation independent, although automation of them may not be. In addition, acceptance tests can suggest that operations normally hidden from direct testing (such as business rules) should be exposed for testing.

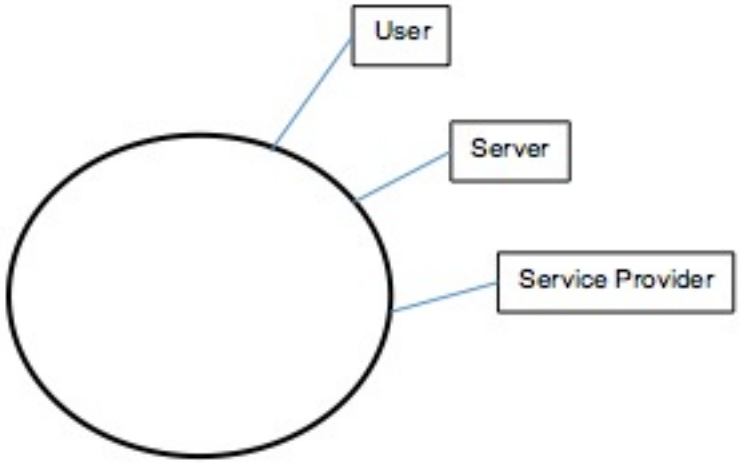


Exhibit 1
What Are Acceptance Tests

2. *Why* Do ATDD?

Teams that follow the acceptance test-driven process have experienced efficiency gains. In one case, rework went down from 60% to 20%. This meant that productivity doubled since the time available for developing new features went from 40% to 80%. In another case, the workflows were working the first time the system was brought up. Getting the business rules right, as in an example we shall see, prevents rework. Because the business customer, developer, and tester are involved in acceptance test creation, there is tighter cross-functional team integration. In addition, passing the acceptance tests visibly demonstrates that the story is complete.

3. *When* Are Acceptance Tests Created?

The value stream map for classical development is shown in Exhibit 2. After eliciting requirements, they are analyzed. A design is created and code developed. Then the system is tested. You can notice many loops go back from test to analysis, design, and coding. These loop backs cause delay and loss of productivity.

Why do these occur? Frequently the cause is misconstructions. In particular, it is misunderstanding the requirements. The loop backs are really feedback to correct these mistakes. There will always be a need for feedback, but quick feedback is better than slow feedback.

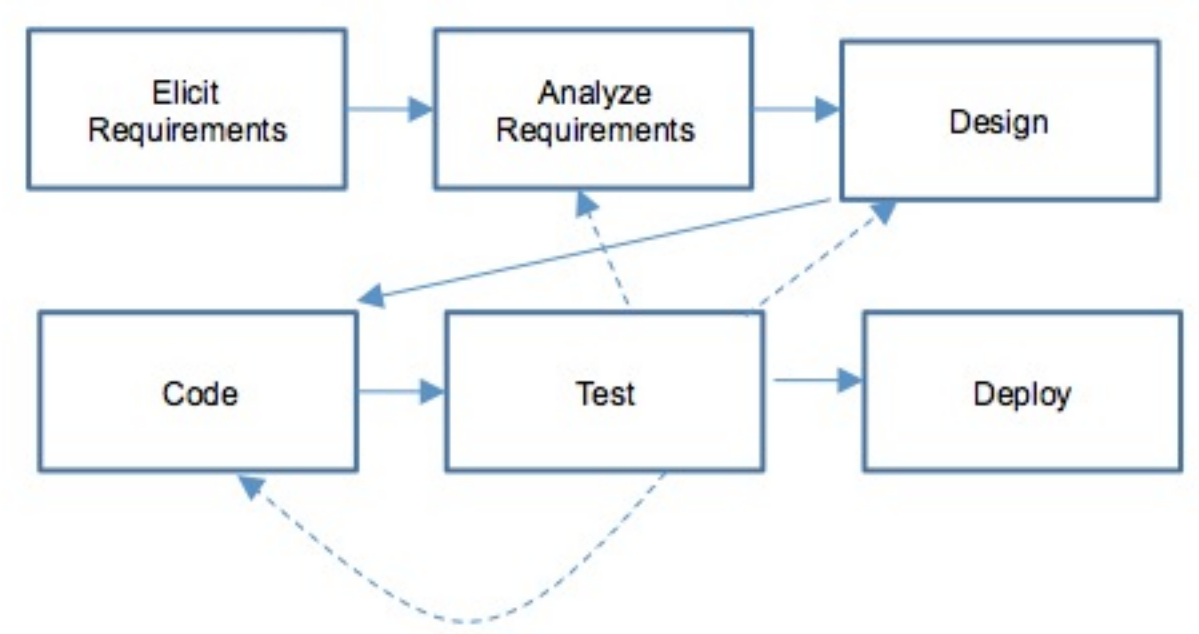


Exhibit 2
Classical Software Development

As you can notice in the revised value stream map in Exhibit 3, the acceptance tests are created when the requirements are analyzed. The developers then code using the acceptance tests. A failing test provides quick feedback that the implementation is not meeting the requirements.

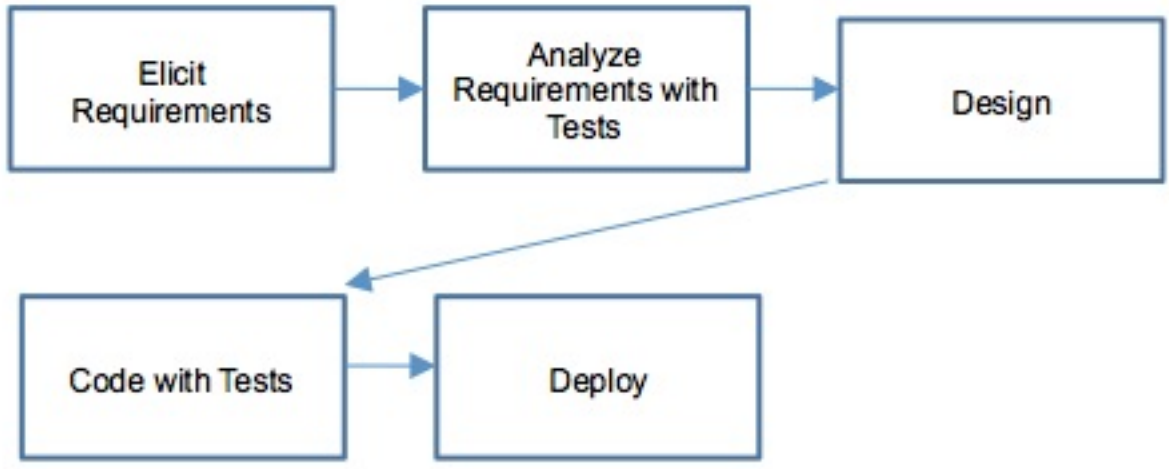


Exhibit 3
Software Development with Acceptance Tests

4. *Who* Authors the Tests?

The tests are authored by the triad – the customer, tester, and developer. At least one example used in the tests should be created by the customer working with the tester and developer. The tester and developer can then create more and have them reviewed by the customer.

The developers connect the test to the system by writing short bits of code. Anyone – customers, developers or testers can run the tests. Acceptance tests can be manual. However, automating acceptance tests allows them to run as regression tests to ensure that new features do not interfere with previously developed features.

Acceptance tests are not a substitute for interactive communication between the members of the triad. However, they provide focus for that communication. The tests are specified in business domain terms. The terms then form a common language that is shared between the customers, developers, and testers.

5. How Do Acceptance Tests Fit Into the Overall Testing Strategy?

Acceptance tests are only a part of the overall testing strategy, as show in Exhibit 4, a diagram adapted from Gerard Meszaros, Mary Poppendieck, and Brian Marick. They are the customer tests that demonstrate the business intent of a system, as shown in the upper left box of the exhibit. The component tests which are beneath them, are technical acceptance tests developed by the architect that help specify the behavior of large modules. Unit tests, the lowest left box, are partially derived from acceptance and component tests, and help the developer to create easy-to-maintain code. The tests on the right – usability, exploratory, and property – examine what are often termed the non-functional aspects. They also need to pass to have a high quality system.

	Per function	*Cross-functional*
Business Facing	Acceptance Tests	Usability Testing
Technology Facing	Component Tests	Exploratory Testing
	Unit Tests	Property Testing

Exhibit 4
Testing Strategies

6. Acceptance Test Example

Suppose you had a requirement which states:

> "As the marketing manager, I want to give discounts to repeat customers so that I can increase repeat business."

There are details that go with this story. For example, the customer discount is computed according to a business rule -- Customer Discount Rule. The details of this rule are:

If Customer Rating is Good and the Order Total is less than or equal $10.00,

Then do not give a discount,

Otherwise give a 1% discount.

If Customer Rating is Excellent,

Then give a discount of 1% for any order.

If the Order Total is greater than $50.00,

Then give a discount of 5%.

Now read the rule again and answer one question. For a customer who is Good and has an order of $50.01, what should the discount be?

Depending on how you read the rule, you may come up with one, five, or six percent. The rule is ambiguous. How do we make it clearer? The customer, developer, and tester come up with some examples, which turn into tests.

Suppose they come up with a table of examples, as shown in Exhibit 5. In third set of values, the discount for the case in question should be 1 percent. That's what the business customer wanted. Imagine if the customer had not been consulted and if both the tester and developer had thought it should be 6 percent.

Now these examples are used as acceptance tests. These tests and the requirement are tied together – the tests help clarify the requirement and the requirement forms the context for the tests.

Discount		
Order total	**Customer rating**	**Discount percentage?**
10.00	Good	0
10.01	Good	1
50.01	Good	1
.01	Excellent	1
50.00	Excellent	1
50.01	Excellent	5

Exhibit 5
Acceptance Test Examples

7. *Where* Acceptance Tests Are Implemented?

There are at least four ways to implement the tests. They are a testing script, which uses the user interface; a graphical or command line interface; a unit testing framework; and an acceptance testing framework. Let's take a brief look at each.

In the first case, the tester creates a testing script. For example, they logon as a Good customer, start up an order, and put items into it. When the order total is $10.01, they complete the order and make sure that it shows a $.10 discount. Now they repeat the process for the five other cases, increasing the possibility of carpal tunnel syndrome.

This script needs to be run at least once in order to ensure the discount is computed properly as part of the workflow. However there are three other ways to check for the other cases.

A graphical or command line interface could be created that accessed the module that computed the discount, as shown in Exhibit 5. The tester need only enter the customer rating and order total to determine if the discount is correctly computed.

Exhibit 6
Graphical Interface

The developer could create an Xunit test, as shown below. This automates the testing process. However, since the test is in the language of the developer, it can be more difficult to use as a communication vehicle and to ensure that changes in the business rule have been incorporated into the test.

```
class TestCase  {
   testDiscountPercentageForCustomer() {
     SomeClass o = new SomeClass();
     assertEquals(0, o.computeDiscount(10.0, Good));
     assertEquals(1, o.computeDiscount(10.01, Good));
     assertEquals(1, o.computeDiscount(50.01, Good));
     assertEquals(1, o.computeDiscount(.01, Excellent));
     assertEquals(1, o.computeDiscount(50.0, Excellent));
     assertEquals(5, o.computeDiscount(50.01, Excellent));
     }
   }
```

One could use an acceptance test framework, which allows the tests to be readable by the customer. The table shown in Exhibit 7 is from Fit – Framework for Integrated Testing -- but other frameworks, such as Cucumber and Robot Framework have similar tables.

Discount		
Order total	**Customer rating**	**Discount percentage?**
10.00	Good	0
10.01	Good	1
50.01	Good	1
.01	Excellent	1
50.00	Excellent	1
50.01	Excellent	5

Exhibit 7
Fit Test

The table becomes a test and is tied to the underlying system through glue code called a fixture. When the test is run, the results appear in the table – green is a pass, red is a fail.

8. Other Uses for Acceptance Tests

In addition to verifying requirements, acceptance tests can be used for other purposes. The number and complexity of the tests can help to estimate the relative effort required to implement a requirement. In the example test, there are six combinations. If there were thirty-six combinations, this would indicate that the effort to realize it would be greater.

The number of acceptance tests that pass relative to the total number of acceptance tests is a relative indicator of how complete is the implementation. In the example, if only one test passed, then the implementation is just started. If all but one test pass, then it is closer to completion.

9. ATDD from the Start

The acceptance test process actually begins at the definition of a feature or capability. For example, the user story about offering discounts is part of marketing initiative. There is a purpose in offering discounts – to increase repeat business. How do you measure the effectiveness of the discount? You need to create an acceptance test, such as "During the next year, 80% of customers will have placed an additional order over their average orders of the past three years." Often acceptance tests as this one are termed project objectives. Objectives should be SMART – specific, measurable, achievable, relevant, and time-boxed.

If this acceptance test passes, then the feature is a success. That is as long as there are no additional business features being added that might affect the outcome, such as providing a personal shopper for every excellent customer. If the acceptance test fails, it may be due to a variety of reasons such as an insufficient discount or a competitor's discount. Or it may be that the objective is not achievable –the economy is such that customers are not buying. In either case, you have a definitive measurement that suggests a course of action such as increasing the discount or abandoning the feature.

10. Review

Acceptance tests represent the detailed requirements for a system. They are developed by the triad--customers, developers, and testers -- as part of requirement definition. They are used by developer and testers during implementation to verify the system. Using acceptance tests can double the efficiency in producing new features.

References

Pugh.K. (2011) Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration. Boston: Addison-Wesley

Effectively Communicating with Acceptance Tests

Better Software Through Collaboration
Seminar Version

August 2015

Copyright © 2015 Ken Pugh, Net Object ives . All Rights Reserved.

Ken Pugh

ken.pugh
@netobjectives.com

- Fellow Consultant
- OOA&D, Design Patterns, Lean, Scrum, Test-Driven Development
- Over 2/5 century of software development experience
- Author of seven books, including:
  - *Prefactoring: Extreme Abstraction, Extreme Separation, Extreme Readability* (2006 Jolt Award)
  - *Interface Oriented Design*
  - *Lean Agile Acceptance Test-Driven Development: Better Software Through Collaboration*

No code goes in till the test goes on.
A journey of two thousand miles begins with a single step.

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Overall Rule

- There are exceptions to every statement, except this one

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Flow

- Introduction
- Acceptance Test Examples
- Test Anatomy
- Supplemental

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Introduction

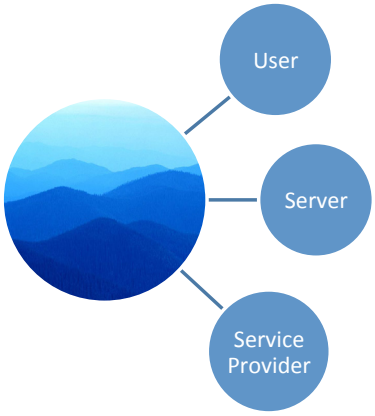
5 W's and an H

- ATDD
  - What are they
  - Why use them
  - Who creates and uses them
  - When are they created
  - Where are they used
  - How to create them

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

What Are Acceptance Tests?

- Acceptance Tests:
  - External view of system
- Examine externally visible effects
  - Inputs and outputs
  - State changes
  - External interfaces



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Definitions

- Acceptance criteria
  - General ideas
- Acceptance tests
  - Specific tests that either pass or fail
  - Implementation independent
- Triad – customer unit, developer unit, tester unit

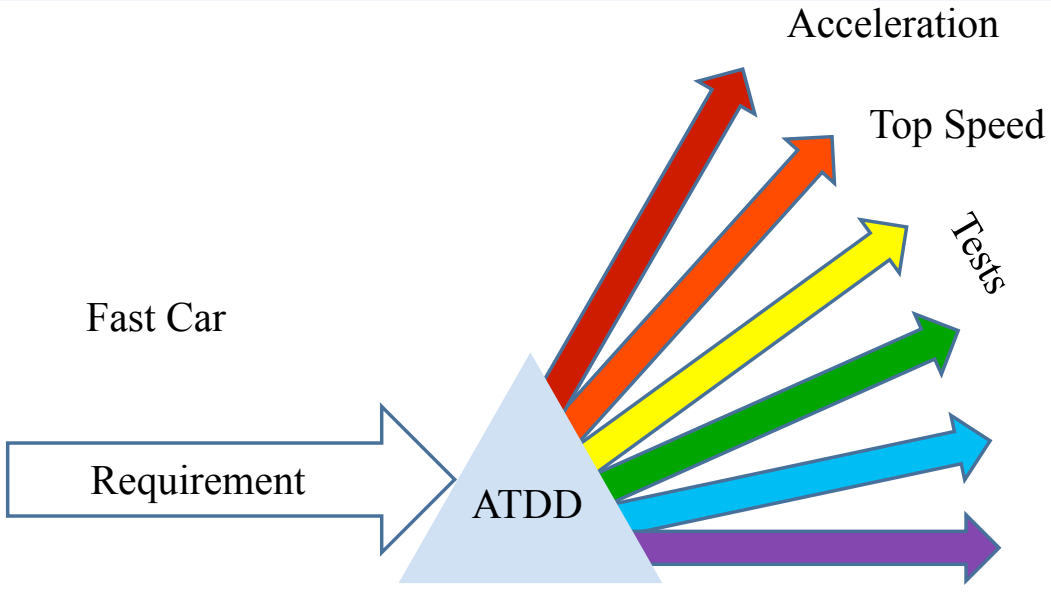
Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Fast Car Example

- Who wants a fast car?
- Criteria
  - Run on a closed course, measure acceleration
- Test
  - Detail acceleration (0 to 60 mph in X seconds)

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

ATDD as a Prism



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Why?

- Rework Down from 60% to 20%
- Workflows Working First Time
- Little Room for Miscommunication
- Saving Time
- Getting Business Rules Right
- Game Changing
- Tighter Cross-Functional Team Integration
- Crisp Visible Story Completion Criteria
- Automation Yields Reduced Testing Time

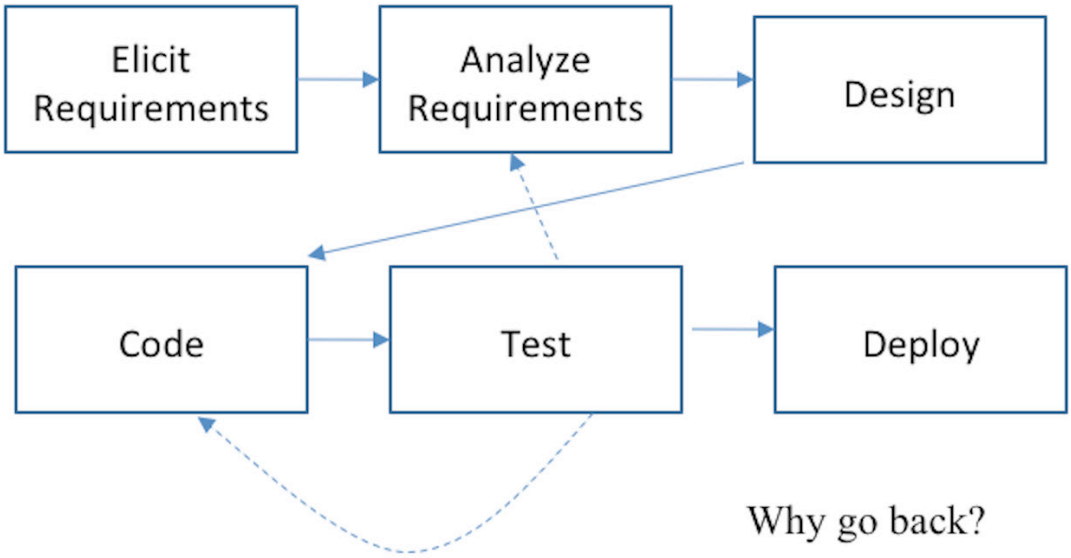
Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Requirements and Tests

- Requirements and tests are inter-related
  - You can't have one without the other
- A failing test is a requirement
  - Passing test denotes specification on how system works

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

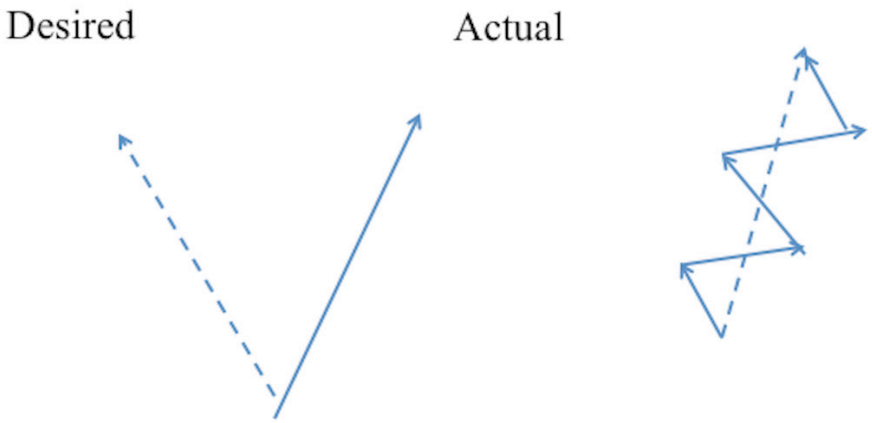
Value Stream Map – Classical Development



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

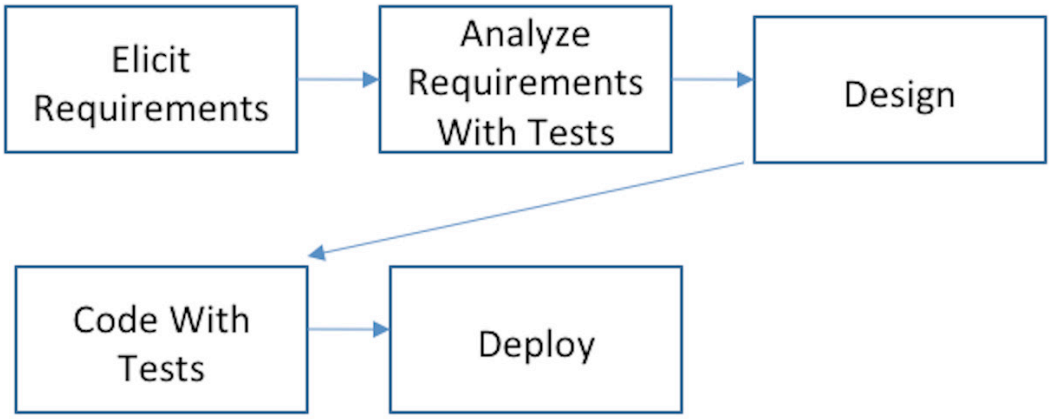
Why Mistakes?

- Misunderstandings, missed requirements, mis-other
- Feedback helps to correct misunderstandings
- Quick feedback better than slow feedback



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Value Stream Map – Agile Development



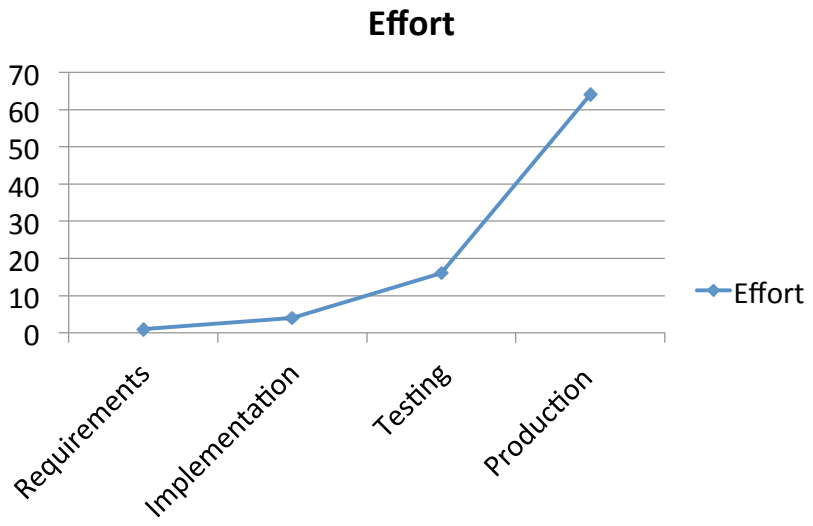
Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Move Testing Forward

- Two types of testing
  - Attempting to find defects – is WASTE
  - Attempting to prevent defects – is ESSENTIAL
- When are defects found?
  - Prevention is just early detection

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Cost of a Requirement Issue



Could be 1 to 64, 1 to 256, or something else

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Who Does What

- Author the tests (write)
  - Triad - customer, tester, developer
- Connect tests to system (bind)
  - Developer
- Run the tests (run)
  - Developers, testers, customers
  - Automated – part of build

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

ATDD, TDD, BDD

- ATDD
  - Tests written by triad prior to implementation
  - Implementation independent, can be automated or manual
  - Usually uses Given/When/Then template
- TDD – Test Driven Development/ Test Driven Design
  - Done by developer while coding
  - Implementation dependent, always automated
  - Write unit test, see it fail, implement code to pass
- BDD – Behavior Driven Development
  - Started as replacement for TDD unit testing framework
  - Usually associated with Cucumber-like languages (Given/When/Then)
  - Otherwise, like ATDD
- SPE – Specification by Example
  - Like ATDD, except uses "examples", rather than "tests"

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Acceptance Test Examples

First Example

- Input Temperature in Celsius, Output Temperature in Fahrenheit
- What tests would you run?

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Input and Output Example (continued)

Celsius	Fahrenheit	Notes
0	32	
100	212	How many needed?
-273.15	-459.67	2 digits precision
-273.151	Error	Below 0 Kelvin
500	932	Maximum – Needed?

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Input and Output – "Unit Tests"

Formula Tests

Celsius	Fahrenheit	Notes
0	32	
100	212	How many needed?

Precision Tests

Celsius	Fahrenheit	Notes
-273.15	-459.67	Precision

Limit Tests

Celsius	Fahrenheit	Notes
-273.15	-459.67	0 Kelvin
-273.151	Error	Below 0 Kelvin
500	932	Maximum – Needed?

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

A Business Rule Example

If Customer Rating is Good and the Order Total is less than or equal $10.00,

Then do not give a discount,

Otherwise give a 1% discount.

If Customer Rating is Excellent,

Then give a discount of 1% for any order.

If the Order Total is greater than $50.00,

Then give a discount of 5%.

Given a customer whose rating is Good and an order total of $50.01, what should be the discount?

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Business Rule Table = Test

Discount		
Order total	**Customer rating**	**Discount percentage?**
$10.00	Good	0%
$10.01	Good	1%
$50.01	Good	1%
$.01	Excellent	1%
$50.00	Excellent	1%
$50.01	Excellent	5%

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Ways To Implement Test

- Testing script
- Program interface
- Xunit framework
- ATDD framework

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Testing script

- Tester creates script (usually GUI based)
- Example:
  - Logon as Customer who is rated Good
  - Start order
  - Put items in the order until the total is exactly $10.01
  - Complete order
  - Check it shows a $.10 discount
- Repeat for other five cases

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

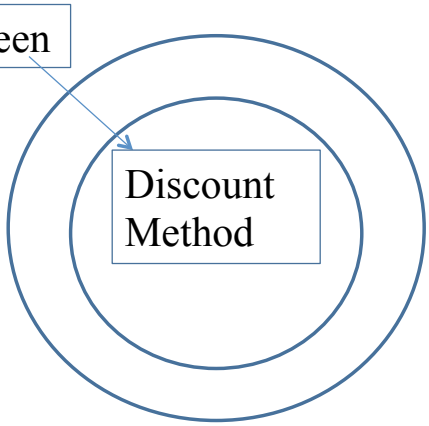
Program interface

- Create a command line or graphic user interface

Discount Percentage Screen

Customer Type: **Good**
Order Total: **10.01**

Percentage: **1 %**




```
C:>DiscountPercentage Good 10.01
Percentage: 1%
```


Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

XUnit Test

```
class TestCase  {
    testDiscountPercentageForCustomer() {
        SomeClass o = new SomeClass()
        assertEquals(0, o.computeDiscount(10.0, Good));
        assertEquals(1, o.computeDiscount(10.01, Good));
        assertEquals(1, o.computeDiscount(50.01, Good));
        assertEquals(1, o.computeDiscount(.01, Excellent));
        assertEquals(1, o.computeDiscount(50.0, Excellent));
        assertEquals(5, o.computeDiscount(50.01, Excellent));
        }
    }
```


Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Fit (Table = Test)

Discount		
Order total	**Customer rating**	**Discount percentage?**
10.00	Good	0
10.01	Good	1
50.01	Good	1
.01	Excellent	1
50.00	Excellent	1
50.01	Excellent	5

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Fit Test

Discount		
Order total	**Customer rating**	**Discount percentage?**
10.00	Good	0
10.01	Good	1
50.01	Good	Expected 1 Actual 5
.01	Excellent	1
50.00	Excellent	1
50.01	Excellent	5

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Tables As Requirement and Test

Requirement

Discount Rule		
Customer Rating	**Order Total**	**Discount Percentage**
Good	<= $10.00	0%
	Otherwise	1%
Excellent	Any	1%
	> $50.00	5%

Test

Discount Test		
Order total	**Customer rating**	**Discount percentage?**
$10.00	Good	0%
$10.01	Good	1%
$50.01	Good	1%
$.01	Excellent	1%
$50.00	Excellent	1%
$50.01	Excellent	5%

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

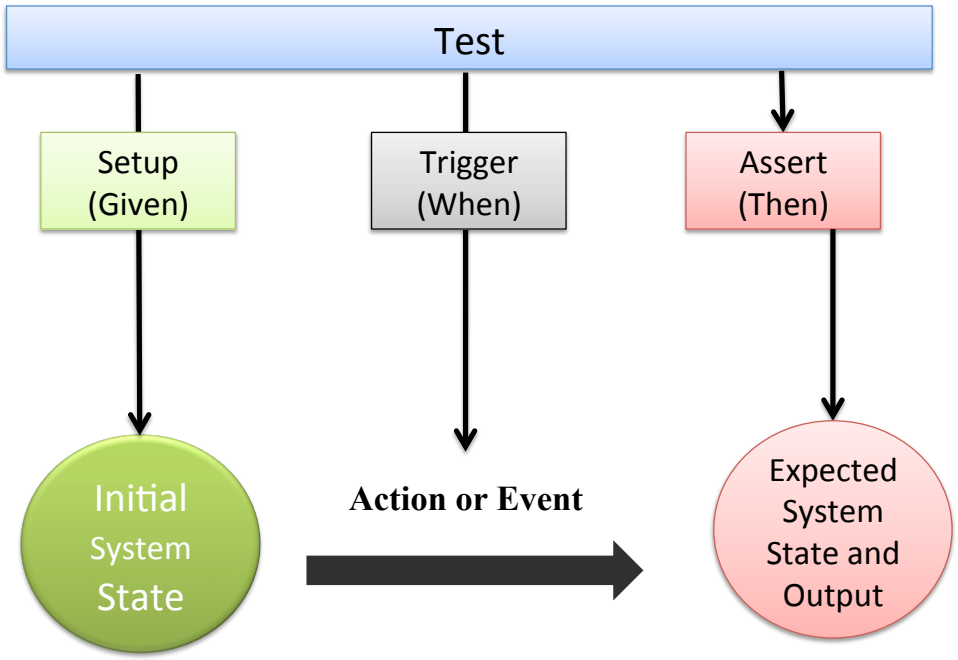
Cucumber Version

```
Scenario Outline: Compute discount
Given total is <OrderTotal> and rating is
     <CustomerRating>
When I compute discount
Then percent is <DiscountPercentage>
Examples:
|OrderTotal|CustomerRating|DiscountPercentage|
|10.00     |Good          |0                 |
|10.01     |Good          |1                 |
|50.01     |Good          |1                 |
|0.01      |Excellent     |1                 |
|50.00     |Excellent     |1                 |
|50.01     |Excellent     |5                 |
```


Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Test Anatomy

Scenario Flow



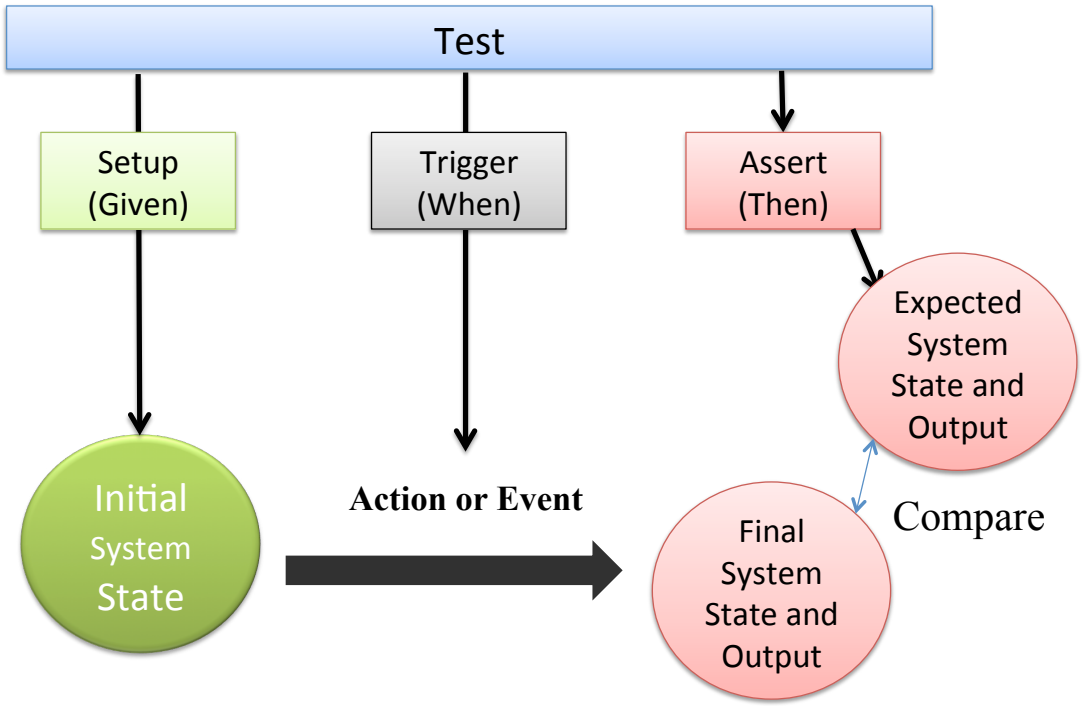
Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Example of Scenario

- Given (Setup)
  - Car is not moving
- When (Trigger)
  - Accelerator pressed
- Then (Assert)
  - 60 MPH reached before 4.5 seconds

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Test Flow



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Example of Test Against Scenario

- Given (Setup)
  - Car is not moving
- When (Trigger)
  - Accelerator pressed
- Then (Verify)
  - Check that 60 MPH is reached before 4.5 seconds

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Term Alternatives

- Expected system state and output = behavior
  - Expected behavior drives development →
    Behavior-Driven Development
- Tests that behavior is acceptable
  - Acceptance tests drive development →
    Acceptance Test-Driven Development

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Workflow Example

Given / When / Then Example

- Context: Sam's Lawn Mower Repair and CD Rental Store
- Given (Setup)
  Customer has ID (initial system state)
  CD has ID (initial system state)
  CD is not currently rented (initial system state)
- When (Trigger)
  Clerk checks out CD (action)
- Then (Verify)
  CD recorded as rented (final system state)
  Rental contract printed (output)

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Full Example (1)

Check Out CD

- Given Customer has ID
  and CD has ID
  and CD is not currently rented

Customer Data	
Name	**ID**
James	007

CD Data		
ID	**Title**	**Rented**
CD2	Beatles Greatest Hits	No

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Full Example (2)

- When a clerk checks out a CD:

Check Out CD		
Enter	**Customer ID**	007
Enter	**CD ID**	CD2
Execute	**Rent**	

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Full Example (3)

- Then the CD is recorded as rented and a rental contract is printed:

CD Data			
ID	**Title**	**Rented**	**Customer ID**
CD2	Beatles Greatest Hits	Yes	007

Rental Contract			
Customer ID	**Customer Name**	**CD ID**	**CD Title**
007	James	CD2	Beatles Greatest Hits

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Full Example – Extended

- Given

Rental Fee Business Rule
Fee
$3

Rental Time Business Rule
Time
2 days

- When a clerk checks out a CD on:

Today
1/1/2014

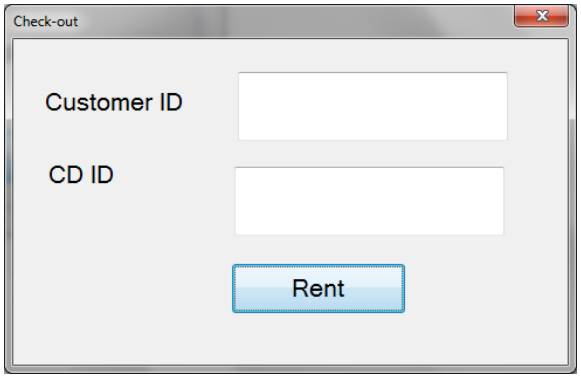
- Then a rental contract is printed:

Rental Contract					
Customer ID	**Customer Name**	**CD ID**	**CD Title**	**Due**	**Fee**
007	James	CD2	Beatles Greatest Hits	1/3/2014	$3

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

The Action

- Can drive a GUI



- Or a method

```
Rent(CustomerID aCustomer, CDID aCD);
```

- Or an Interactive Voice Response (IVR)
  - "Enter the customer id followed by the pound sign"

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Example of Business Rule

CD Rental Rates Business Rule
Regular $2 / 2 days plus $1 / day
Golden Oldie $2 / 4 days plus $ .50 / day
Hot Stuff $2 / 1 days plus $2 / day

CD Rental Rates As Table			
Type	**Base Rental Period Days**	**Base Rental Fee**	**Extra Day Fee**
Regular	2	$2	$1
Golden Oldie	4	$2	$.50
Hot Stuff	1	$2	$2

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Example of Business Rule Test

CD Rental Rates Business Rule
Regular $2 / 2 days plus $1 / day
Golden Oldie $2 / 4 days plus $ .50 / day
Hot Stuff $2 / 1 days plus $2 / day

CD Rates Test		
Type	**Days**	**Cost?**
Regular	2	$2
Golden Oldie	5	$2.50
Hot Stuff	6	$12
Hot Stuff	50	IGBTYOT

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Business Rule and Flow Tests

- Use cases / scenarios usually include business rules
- Test every business rule separately
- Flow test of scenario
  - Uses one variation of business rule

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Full Flow Example Revisited

Check Out CD

Given Customer has ID; CD has ID and not currently rented

Customer Data	
Name	**ID**
James	007

CD Data			
ID	**Title**	**Rented**	**Type**
CD2	Beatles Greatest Hits	No	Regular

and one variation of business rule for Rental Rates

CD Rental Rates As Table			
Type	**Base Rental Period Days**	**Base Rental Fee**	**Extra Day Fee**
Regular	2	$2	$1

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Full Flow Example – Extended

- When a clerk checks out a CD on:

Today
1/1/2014

Check Out CD		
Enter	**Customer ID**	007
Enter	**CD ID**	CD2
Execute	**Rent**	

- Then a rental contract is printed:

Rental Contract					
Customer ID	**Customer Name**	**CD ID**	**CD Title**	**Due**	**Fee**
007	James	CD2	Beatles Greatest Hits	1/3/2014	$2

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Guidelines

Tests

- Acceptance tests are not a substitute for interactive communication
  - They can provide focus for that communication
- Tests in business domain terms
  - Shared between customer unit and developer unit

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

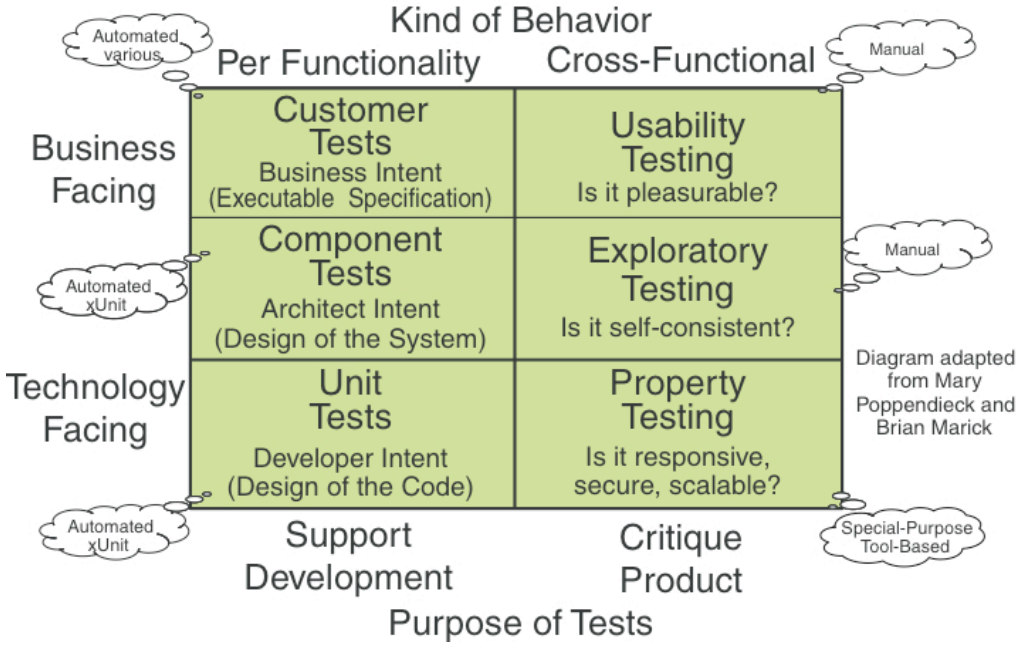
Guidelines

- Tests and automation should be developed separately
  - Understand the test first
  - Then explore how to automate it
- Automate tests for regression
  - Use in continuous build
- As much as practical, cover 100% of the functional requirements by acceptance tests
- Can break down stories by acceptance tests
  - One acceptance test per story

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

A Few Other Thoughts

Testing Strategies

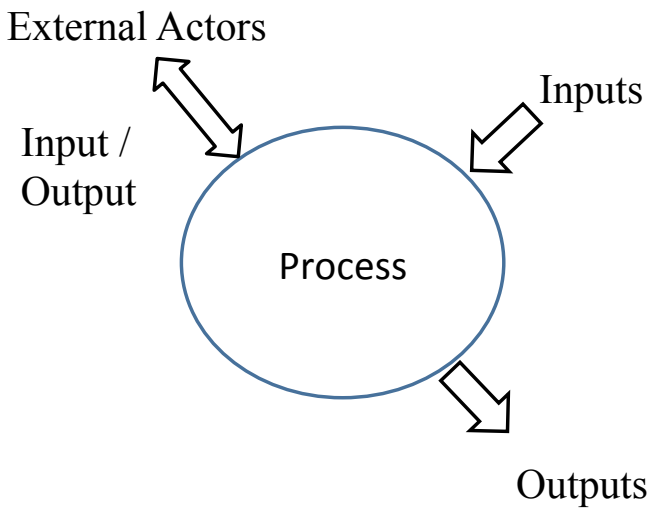


Meszaros, *XUNIT TEST PATTERNS: REFACTORING TEST CODE*, Fig 6.1 "Purpose of Tests" p. 51, © 2007 Pearson Education, Inc

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

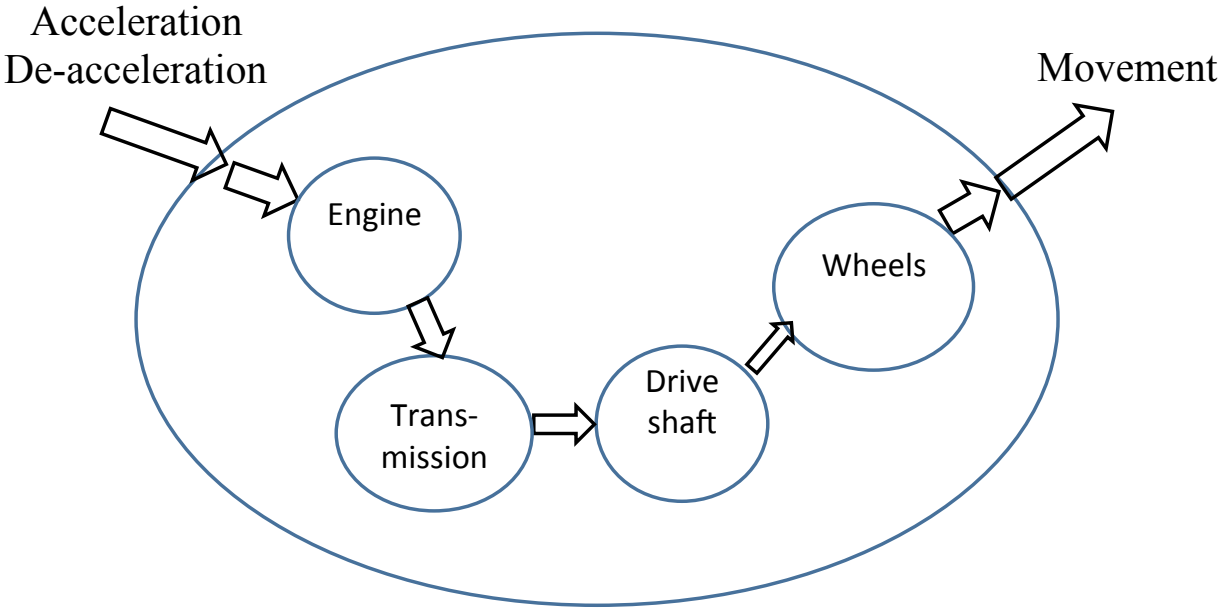
Context Diagram

- A context diagram shows scope



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Context of Automobile

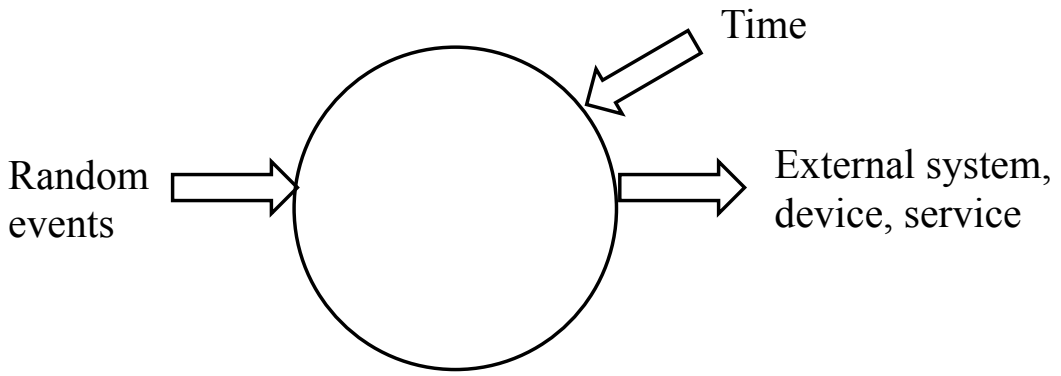


Component tests for the individual pieces

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

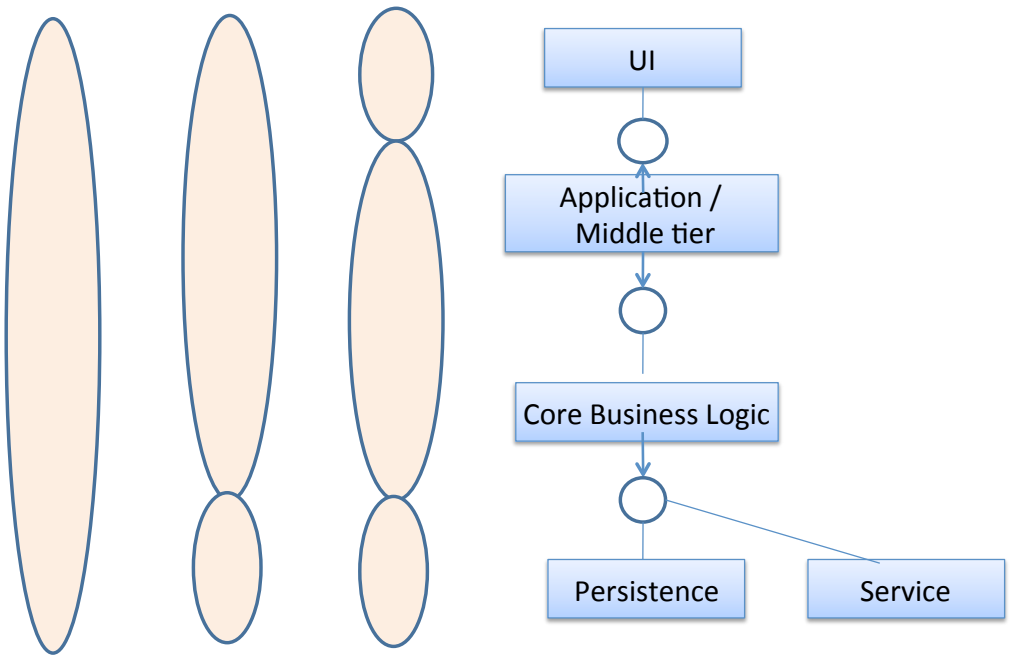
External Interfaces

- Connections to external systems need to have test doubles (mocks)
  - They are stand-ins for real system during testing
  - Random events may need to simulated
- Test doubles give repeatability and speed
  - Cheaper if have to pay for service



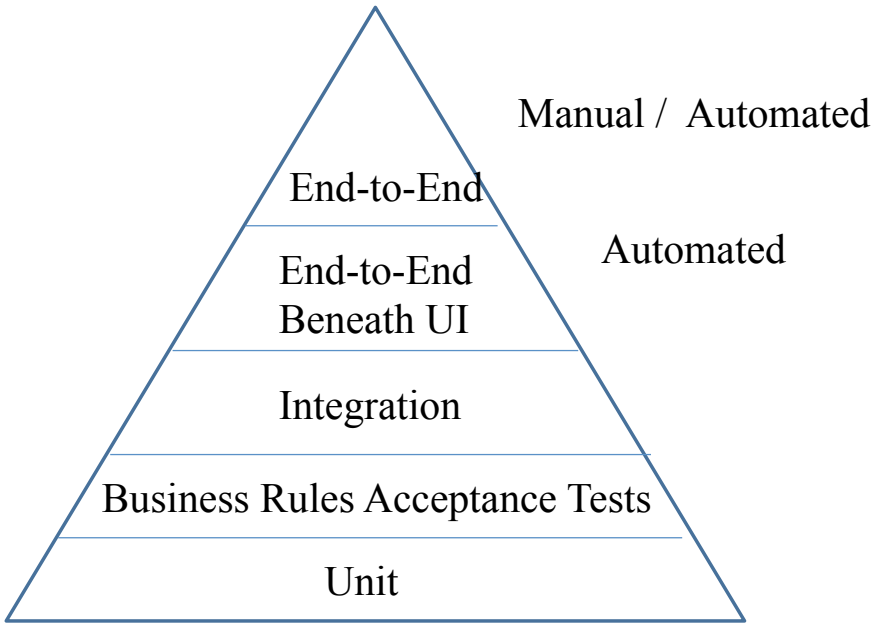
Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Tests to Cover Entire Flow



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Automation Testing Pyramid



Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Avoid Copy and Paste

In requirements, tests, code, documentation

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Not an Ending, But a Beginning

Recap

- Primary goals
  - Discover ambiguous requirements and gaps in requirements early on
  - Create a record of business/development understanding
- Secondary goals
  - Use acceptance tests as an executable regression test
  - Measure the complexity of requirements
  - Use the tests as a basis for user documentation

Copyright © 2015 Ken Pugh, Net Objectives. All Rights Reserved.

Copyright © 2015 Ken Pugh, Net Object ives . All Rights Reserved.

The Improvement Kata:
Annual Improvement Planning Meets Agile

Kathy Iberle
Iberle Consulting Group, Inc.
www.kiberle.com

Abstract

Many IT and high-tech organizations create yearly plans for advancement and improvements. All too often, new information quickly invalidates the plan, necessitating costly re-planning. Team-level Agile retrospectives avoid this "big design up front" problem, but retrospectives don't easily handle improvements that require coordinated effort across multiple teams.

The Improvement Kata solves this dilemma by providing an *agile* method for planning and executing large-scale improvements. Large-scale changes are systematically broken down into small pieces, and each of those pieces is treated as an experiment rather than a fixed plan. This enables the organization to navigate efficiently through the many surprises of process improvement, while implementing "the simplest thing that could possibly work".

This talk will discuss the fundamentals of the Improvement Kata and share some experiences with applying the Improvement Kata in development and IT organizations.

Biography

Kathy Iberle is the Principal Consultant at Iberle Consulting Group, Inc., where she helps clients improve their productivity and quality. Her extensive background in process improvement and quality methods enable her to blend classic quality control with the best of current Lean thinking.

Kathy started at Hewlett-Packard, where over the years she worked as a developer, quality engineer, test manager, and process improvement expert in software, firmware, and IT. Kathy's deep knowledge of Lean and Agile theory combined with a flair for pragmatic applications has enabled her to successfully extend Agile development methods into areas where most people think "Agile just can't work". She has published regularly since 1997 on quality and Lean topics. Today, she continues to explore new methods while helping her clients implement more effective processes.

Kathy has an M.S. in Computer Science from the University of Washington and an excessive collection of degrees in Chemistry from the University of Washington and the University of Michigan.

Copyright Iberle Consulting Group, Inc. 2015

Acknowledgements

Adam Light of SoTech Advisors (http://www.sotechadvisors.com/) introduced me to the Improvement Kata and was a key player in much of the work described in this paper. His wise advice also improved this paper considerably.

1 Introduction

Effective process change presents a difficult challenge. Far too often, we see the problems but don't arrive at successful solutions. Common causes of failure include:

- Starting so many change efforts that we can't finish most of them.
- "Analysis Paralysis". We can't decide where to start, so we keep analyzing instead of starting.
- Jumping to conclusions. We implement our favorite solution without knowing whether it is applicable to this particular problem.
- Insufficient support from the "powers that be".

However, the biggest obstacle to process change usually is the vast number of things that we *don't* know. We don't know how much our problems are costing the organization, we don't know exactly why they are occurring, and we don't know precisely what changes will resolve them. This makes it impossible to create a successful long-term plan. Each time we make a change, we discover more about our process. We uncover more problems and find better ways to solve the problem than those in our current plan.

Does this sound familiar? It's the same state of affairs which drove the invention of agile development methods – we can't know everything up front. Process improvement typically involves even more uncertainty than software development. There's little past data to help estimate how long a process change will take, and changes to processes often have large unexpected side effects.

The *Improvement Kata* is a process improvement framework which allows us to navigate this uncertain territory by taking one small step at a time. Just like agile methods, we make one small process change, evaluate the result, and adapt our plans in response to our learnings. The Improvement Kata steers the project using data from your real-life performance, rather than an industry-standard checklist.

In this paper, I will explore how the Improvement Kata works and share my recent experiences implementing the Improvement Kata

2 The Improvement Kata

2.1 Origins of the Improvement Kata

Kata is a Japanese word which means a small, structured practice routine or protocol. A Kata is practiced by the student, with feedback from a teacher, until the student has mastered the fundamentals and the pattern becomes second nature. A violinist practicing scales and a baseball player in batting practice are both following a *Kata*.

The Improvement Kata is based on the process improvement practices used at Toyota for manufacturing and related processes. The practices did not have a name at Toyota – they were simply "the way we do things here". American researchers eventually realized that these "kata" were a key part of implementing Lean practices within Toyota and named the practices "the Improvement Kata". Today, a leading researcher in this area is Mike Rother, the author of *Toyota Kata: Managing People for Improvement, Adaptiveness, and Superior Results* (2009), and a continually updated *Improvement Kata Handbook:* http://www-personal.umich.edu/~mrother/Materials_to_Download.html.

Most of the published work to date on the Improvement Kata involves, relatively predictable systems such as those in manufacturing. In manufacturing, *batches* of work are roughly homogeneous in size, arrive at a fairly predictable rate, and go through a fairly standard process. Health care also involves many predictable workflows such as appointment scheduling and lab tests. By comparison, software development and other forms of product development are highly variable. Batches of work arrive unpredictably, are of widely differing size and difficulty, and their sizes cannot be reliably known up-front.

As my colleagues and I have applied the Improvement Kata within IT departments, we've discovered the need for some modification of the original methods published by Mike Rother. As I explain the Improvement Kata, I'll point out some of the differences.

2.2 Steps in the Improvement Kata

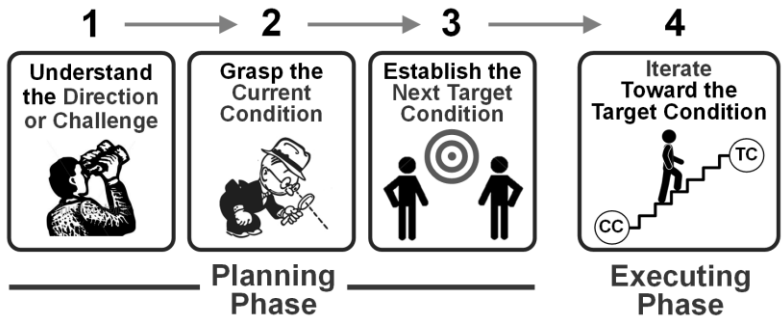


Figure 1: Basic Steps of the Improvement Kata. Rother 2015 A

1) **Understand the Direction**. Where do we want to go? The Improvement Kata starts with a long-term vision for the organization. Hakan Forss, who has been using the Improvement Kata in software development, gives some examples of typical visions for software development: (Forss 2014)

- Zero defects in production
- 100 percent value added
- Highest value first, on demand

Starting with the vision, leadership selects a single Challenge or "stretch goal". This focuses the energy spent by the organization on process improvement, rather than spending the energy on a random selection of whatever seems urgent at the moment. The Improvement Kata doesn't specify a particular timeframe for the Challenge, but it is a good fit with the annual improvement planning or annual "scorecard" planning performed in many large corporations.

The goal or Challenge specifies what and when, but not how. It must be stated clearly enough so that we can tell whether it is achieved or not. The classic example of a well-stated goal is John F. Kennedy's 1961 address to Congress: "**I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the Moon and returning him safely to the Earth. "**[Kennedy 1961][1].

2) **Grasp the Current Condition**. What is our starting point? This step is an assessment of the overall process and how it differs from the goal. This isn't meant to be a lengthy, detailed analysis, but rather simply captures what is currently known, focusing on aspects of the situation which are relevant to the future goal.

So far, the Improvement Kata is very similar to traditional annual planning for process improvements. There's a vision, a yearly goal, and an assessment of the current situation. Now, however, we "go agile".

3) **Establish the Next Target Condition**. Where can we go in the next few weeks? We define a target – a better future state which can be achieved in a *short timeframe*. We don't plan the whole year, or even the whole quarter. Instead, we choose one aspect of the Challenge which can be affected right away, and we set a measurable short-term goal or "Target Condition" for that aspect with an "achieve-by" date. Once the achieve-by date is reached, we stop, assess the Current Condition again, and set a new Target

[1] Kennedy's vision wasn't explicitly stated, but it appeared to involve a future state where the U.S. citizenry and military felt comfortably superior to the Soviet Union.

Condition based on what we now know. As in an agile sprint, we take a small bite, deal with it, and then choose the next bite.

4) **Iterate Toward the Target Condition**. How will we get from our Current Condition to our Target Condition? We don't know exactly how to get from where we are to where we want to be, so we take a small experimental step every day through the unknown territory between the Current Condition and the Target Condition. After each step, we reconsider the next step. This constant course correction minimizes the time spent on ineffective paths. Now let's look at each step in more detail.

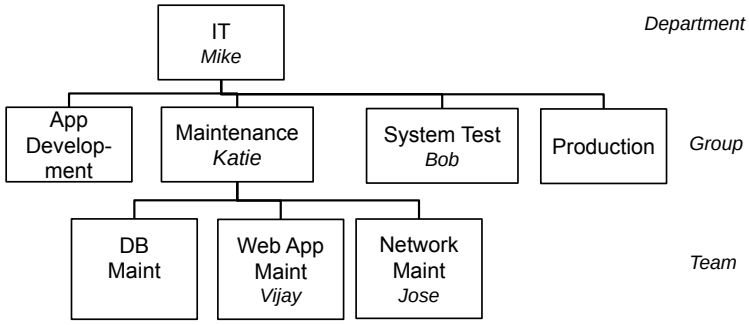
3 How Does the Improvement Kata Work?

3.1 The Situation

The Improvement Kata is easier to understand with an example, so we'll illustrate by walking through an imaginary annual improvement effort. This example takes place in a company of a few thousand employees. This company has both internal web applications to support business operations and customer-facing web applications. Both sets of web applications are handled by the same busy IT department, managed by Mike. Mike's department consists of four groups:

- App Development: creates new applications
- Maintenance: cares for applications in production
- System Test: serves both development and maintenance
- Production: manages networks, deployments, and so forth.

Each group consists of several teams.



This IT department has been struggling with patch failures. These patches fix defects in production systems, introduce minor new features, and install upgrades to deal with routine network and security issues. Sadly, about 20% of the patches fail in production.

Mike's boss, the VP of development, suggests that the IT department's annual improvement plan really should include getting on top of this problem.

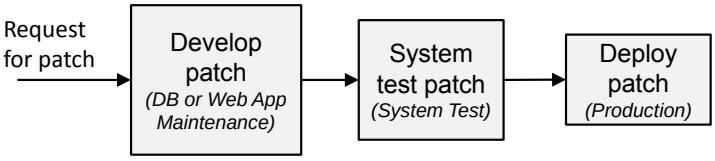
3.2 Planning: Set the Challenge

Mike and his managers agree that their vision is "Zero defects in patches to production systems." In their annual planning, they decide to aim for a 75% reduction in failures by the end of the year. Since 20% of patches currently fail, here's their challenge:

At the end of this calendar year, only 5% of patches fail in production.

Mike and his management team then look at the overall *value stream* for patches – the end-to-end delivery of value to the customer, beginning with the initial idea and ending with final deployment.

They sketch the value stream:



Mike sees that the end-to-end process isn't owned entirely by any one group in his department. Rather, it crosses several groups. This means Mike can't simply delegate the entire problem to the Maintenance group, because it might not be entirely solvable within that group. He has to keep an eye on the possible interactions between groups and teams. This "systems view" prevents unintentionally making the process worse by pushing work from one department into another.

3.3 Planning: Grasp the Current Condition

The next step is to understand what the situation is *now* with respect to the challenge. Rother's methods for measuring the Current Condition and Target Condition (Rother 2015 B) are not directly applicable to software development. We instead start with Rother's fundamental question "How is this process performing over time?" (Rother 2015 B, p. 75) and apply basic Lean principles:

- Identify the process and make it visible. We use a simple process map.
- Measure the *outcome* or bottom line of the process. How fast and well can it deliver value to the end users of the process' output?
- Look for the batches of work and measure them. User stories, anyone?
- Use data and measurements, not gut feel, opinions, or arbitrary assessment against an external standard.

We use metrics similar to what Forss recommends (Forss 2014 p 47-48).

- Overall *throughput* (# user stories completed per sprint)
- Average *Cycle Time* (end-to-end time to develop and deploy a story)
- Average Work in Process (# of user stories started but not finished at a specific point in time)
- # defects per story remaining after release.

Mike's management team currently measures the patch cycle time (average time to deliver a patch from request to deployment) and the failure rate, which are both outcome metrics. They use the value-stream map to capture the *Current Operating Pattern* - how the process works now.

The management team captures all this in a *Learner's Storyboard* (Figure 2). The *Learner* is the person who is responsible for learning about the process and changing it – in this case, Mike. The *Learner's Storyboard* is an Improvement Kata tool that is used at every level from top management to the people closest to the work. The Learner's Storyboard embodies several principles of Lean management artifacts:

- Limited space, to force a concise summary of the issue.
- Standard layout, so everyone knows where to look for different pieces of information
- Real data gathered by observing the current process.
- Visual and physical – usually a large sheet of paper on a wall.

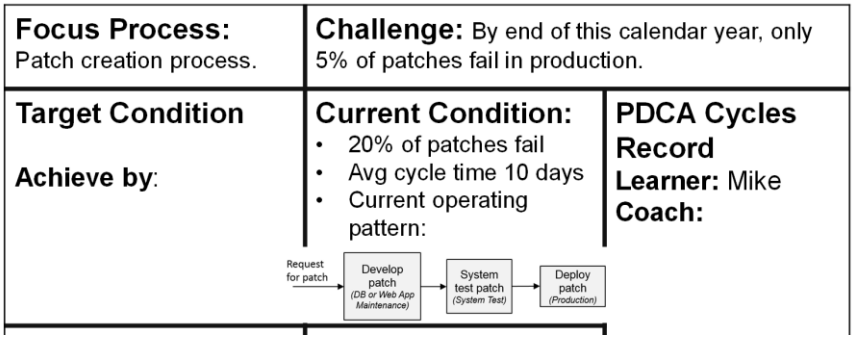


Figure 2 – Mike's Learner's Storyboard for the entire value stream

3.4 Planning: Set the Target Condition

The first steps of the Improvement Kata strongly resemble the first steps of more traditional annual planning methods. But instead of creating a detailed year-long plan, the Improvement Kata employs a cadence of short time boxes, just like Scrum sprints. Mike's company uses a three-week cadence, so Mike's next step is to define a Target Condition which he believes can be met in three weeks. Like the Challenge, a Target Condition is a description of the state you want to achieve, *not* a plan to get there.

Mike envisions a state in which one group of failures has stopped happening. To choose a group, he asks his managers "Why are the patches failing?". It quickly becomes apparent that there's no consensus as to what constitutes the biggest cause of failures, and no reliable data is available. The management team needs more information. Specifically, they need a profile of failures versus causes so they understand which causes create the most failures. Their first Target Condition is "A known profile of failures vs. causes". This is added to the Learner's Storyboard (Figure 3).

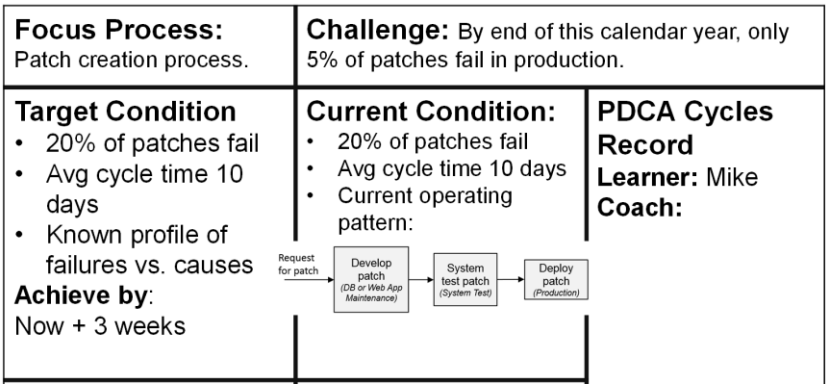


Figure 3 – Mike's Learner's Storyboard with first Target Condition

In real life, Learners often encounter this situation. They haven't been thinking about or measuring the entire value stream, so they don't have the data to understand their system. The first Target Condition often must be "We can see how our entire value stream performs." This implies adding measurements, but we don't prescribe a specific set. Instead, we let the measurements develop out of the problem at hand by adding one measurement at a time and checking whether that provides enough information for now. In other words, do the simplest thing that could possibly work!

3.5 Planning: Identify the Obstacles

Mike and his managers spend a quarter of an hour brainstorming the *obstacles* to the Target Condition. Why isn't there already a known profile? They quickly come up with a list of reasons.

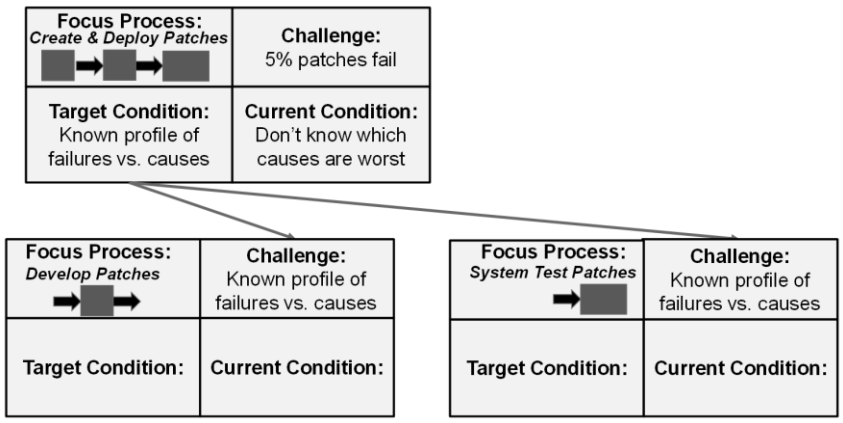
- Nobody is responsible for doing root cause analysis – it's not part of the regular process.
- We don't have an up-to-date list of patch failures to analyze.
- We're not sure how to do a root cause analysis.

Mike reminds the managers that the Target Condition is a profile of causes good enough to drive their improvement efforts, not a great root cause analysis process.

3.6 Decomposing the Problem: Where the Rubber Hits the Road

This is all very nice, but the problem isn't going to get fixed in management meetings. Processes get fixed at the *process level*, where people do the hands-on work. The managers need to decompose the problem into process-level pieces so individual Learners can tackle each piece. The decomposition is done in a fractal manner, where the Target Condition at the manager's level becomes the Challenge at the process level. As shown below, the problem is broken down by along the boundaries in the process map, which

are not necessarily the same as the boundaries in the organizational chart. A process may be co-owned by two departments, or languish in the "white space" between departments.



The "Develop Patch" process is owned by the Maintenance department headed by Katie.

The "System Test Patch" process is owned by the creatively named "System Test Group" headed by Bob.

Mike's Target Condition "Known profiles of failures vs. causes" becomes the Challenge for both of the subprocesses.

In order to get a hands-on view of the patch development process, Katie asks her technical lead, Mark, to act as the Learner. Katie and Mark work together to prepare the Learner's Storyboard shown below. First they copy the Target Condition in Mike's Storyboard into the Challenge of their Storyboard. They then describe the Current Condition, considering obstacles relevant to their group.

Katie asks Mark to take a small step by defining a Target Condition that is only one week away. Mark points out that they need a root cause analysis method before they can do the analysis. Developing a method seems like a realistic one-week goal, so they write the Target Condition as a future state "We have a method to classify defects by causes." Katie asks Mark, "How will we know whether the method is successful?" They agree on two criteria 1) different people using the method on a given failure come to similar conclusions about the cause, and 2) the causes tell us something about how the failures were created. These criteria play the same role that acceptance criteria play for a user story.

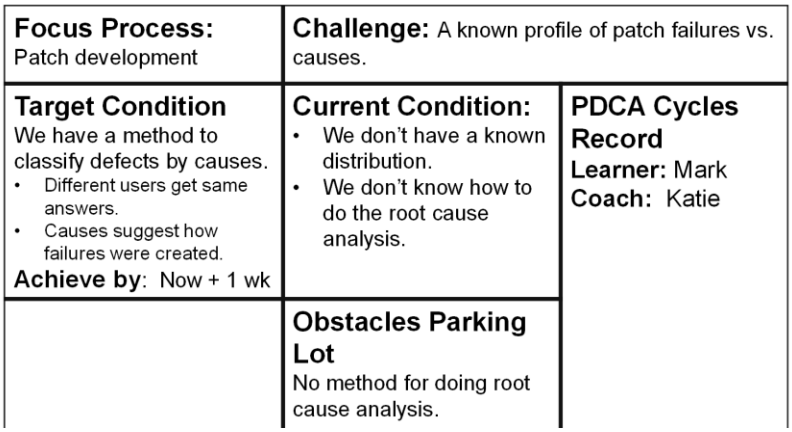


Figure 4 – Mark's Learner's Storyboard

3.7 Plan-Do-Check-Act

Next, Mark attempts to make an actual process change by creating a method for root cause analysis. In the Improvement Kata, the day-to-day activities of process improvement are performed in very small steps called Plan-Do-Check-Act cycles. We experiment towards the Target Condition in small, inexpensive, safe increments.

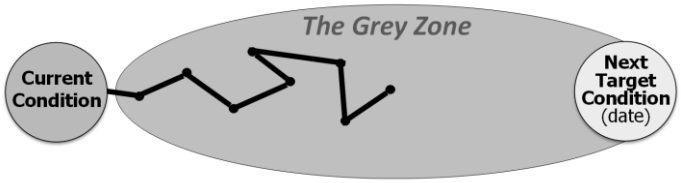


Figure 5 – Iterating towards the Target Condition. Mike Rother 2015 A.

Plan-Do-Check-Act[2] (PDCA) essentially describes the Scientific Method as applied to process improvement:

- **Plan**: Make a testable prediction (hypothesis)
- **Do:** Test your hypothesis with an experiment. (Go do it and find out what happens.)
- **Check** (or Study): Review the facts and data resulting from the experiment.
- **Act:** Adjust your plan based on what you have learned

Katie asks Mark to start a PDCA sheet in the standard Improvement Kata format. Katie and Mark meet every other day to review the progress. The PDCA sheet, like a lab notebook, captures both the results of the experiments and the learnings. This PDCA sheet shows how Mark progresses in small steps.

PDCA CYCLE RECORD					
Process: Patch development		**Challenge:** Known distribution of patch failures over causes. **Current Target Condition:** A method to classify defects by causes. **Achieve-By Date: Jan 25.**			
Learner(s): Mark	**Coach:** Katie				
Start / End	**#**	**What Will You Do? (Step)**	**What Do You Expect?**	**What Actually Happened**	**What We Learned**
1/03	**1**	Send blanket email to IT asking who knows root cause analysis.	Somebody on some other team in IT has done this before.	Nobody has experience, although Rosa sent me a paper from a conference.	We don't have in-house capability.
1/04	**2**	Surf the web looking for a reference.	I will find some good descriptions of how to do root cause analysis.	The descriptions are very technical. Many are for manufacturing, not software. A book by Johanna Rothman looks promising.	Root cause analysis is done differently in different fields. In software, root cause analysis is also called defect cause analysis.
1/05	**3**	Order *Corrective Action* by Johanna Rothman.	It will have examples of what to do and some categories of causes.	The examples on pp 32-33 seem relevant, although more complicated than we need.	Some people do really detailed analyses of root cause, down to different types of coding logic.
1/10	**4**	Adapt the checklist on p. 32 to use for our analysis.	I can create a root cause analysis checklist today.	Done. I dropped the irrelevant categories e.g. bad prototypes can't be a root cause because we don't do prototypes for patches.	Not much.

[2] The PDSA cycle was formulated by Walter Shewhart in the 1930s. (Best 2003)

3.8 The Coaching Kata

Katie and Mike know that the Plan-Do-Check-Act cycle doesn't come naturally to most people. Most people can Plan and Do, but many find the Check or Study step challenging. At Toyota, there is a formal Kata for practicing Plan-Do-Check-Act, which Mike Rother refers to as the *Coaching Kata*. Let's see how Katie and Mark practice the Coaching Kata to speed their way through the PDCA cycles.

Each Learner has a mentor or *Coach*, whose role is to help the mentee or Learner use the scientific method and create change. The Coaching Kata consists of a daily standup meeting in front of the Learner's Storyboard and PDCA sheet, where the Coach asks the Learner the *Five Questions*.

1. What is the Target Condition?
2. What is your Actual Condition today?
    a. What was your last step?
    b. What did you expect?
    c. What actually happened?
    d. What did you learn?
3. What obstacles are preventing you from reaching the Target Condition? Which *one* are you addressing now?
4. What is your next step (next experiment)? What do you expect to see?
5. How quickly can we go and see what we have learned from that step?

Let's listen to a coaching session between Mark and Katie.

Katie: "What is the Target Condition?"

Mark: "We have a method to classify defects by causes."

Katie: "What is your actual condition today?" (See Table 1).

Mark: "I finished the checklist yesterday, so I guess we have a method now."

Katie: "Will the answers from this method be useful for reducing patch failures?"

Mark: "Yes. The categories are kinds of mistakes, like Missing Requirements or Coding Errors."

Katie: "Will different users get the same answers from this method?"

Mark: "They should."

Katie: "How could you find out quickly if they will?"

Mark: "We'll see when people use it to do the root cause analysis."

Katie: "A full analysis will be quite a bit of work. Is there a way to find out sooner?"

Mark: "I guess I could test it on just a few defects with one or two other people."

Katie: "That would be a good next step. When you do that step, what do you expect to see?"

Mark: "It will work."

Did the new process work? Here's what happened next:

1/11	**5**	Try checklist with a couple of example defects.	It will work.	I didn't find a category for defect #652. It was caused by bad data.	We need to add "bad data in database" to the possible causes.
1/12	**6**	Give checklist to Rosa to try.	Rosa will put the defects in categories and we will see a pattern.	Rosa put all the defects into just one category: "coding mistake".	Need to help developers understand that not all defects are coding errors.
1/13	**7**	Talk to team then try checklist again.	Defects will be in more than one category.	Defects sorted into five categories.	This process is workable and gives us some useful info.

Mark's experience is typical of what we observe with our clients. The first attempt at measurement or process change often doesn't work, but the very small steps allow people to quickly and safely learn from the failure and rapidly proceed in a more productive direction. The conscious reflection required by "What did you learn?" changes the perspectives of the Learners. For instance, simply asking "What have you learned?" can quickly turn grumbling about difficult people who just don't want to adopt a new process into thoughtful inquiry about the difficulty of the process being proposed. (See Light 2015.)

3.9 Success Card

Back to our story! Mark has reached his first Target Condition. Katie asks him to write a Success Card, which she posts on the wall next to Mark's Learner's Storyboard.

While success cards didn't originate at Toyota, we find that they help Learners and teams recognize and celebrate progress. Because change happens in such small increments, it can be hard to see the progress without a physical record.

Learner: Mark	**Coach:** Katie
Target Condition: A method for root cause analysis that works.	**Results:** We developed a process that fits our types of defects & we can train people to use it.
Elapsed time for the PDCA cycles: ~10 days	**Benefit:** Now we can do root cause analysis and find out why our patches fail.

Figure 6 - Success Card

3.10 The Next Iteration

Now Mark meets with his Coach, Katie, to set a new Target Condition. The Challenge remains the same, but the Current Condition has changed. Mark and Katie set a new Target Condition and brainstorm obstacles.

Focus Process: Patch development	**Challenge:** A known distribution of causes for patch failures.	
Target Condition We have a known distribution of causes. **Achieve by**: Now + 1 wk	**Current Condition:** • We don't have a known distribution. • We know how to do the root cause.	**PDCA Cycles Record** **Learner:** Mark **Coach:** Katie
	Obstacles Parking Lot No list of failures to analyze. Nobody assigned to do the analysis.	

Figure 7 - Mark's second Learner's Storyboard

Katie notices that one of the obstacles – "No list of failures" – was previously assigned to Bob and his system test group. She contacts Bob, who provides a list of failures found in the previous month complete with defect identifiers. The Improvement Kata structure ensures that managers of different teams maintain alignment so that they are able to help each other.

By the following week, Mark's team has used the root cause analysis method to produce a concise report on the causes of the failures found in the previous month. (Figure 8).

Figure 8 – Patch Failures vs Causes

3.11 The Chain of Coaching

The Improvement Kata is designed to bubble up data from the process level to the levels above. Mark and his team have just provided an important piece of new data. Mike and his management team see that the largest category of issues is "unexpected user actions". There were no user stories for these actions, the developers didn't design or code for them, and the various levels of testing didn't notice the gaps. Since this category is quite large, addressing it should reduce patch failures substantially.

Once again, more information on the Current Operating Pattern will be needed. Why are the user stories missing? Who might be able to predict the users' real-life actions? (Hint: System testers often are able to predict the odd things users do.) Both Katie and Bob will ask their groups to find out more. With this additional data, Mike and his managers can define a new Target Condition to prevent the gaps or detect them before release. This could include changes in the Development process, or in System Test, or both.

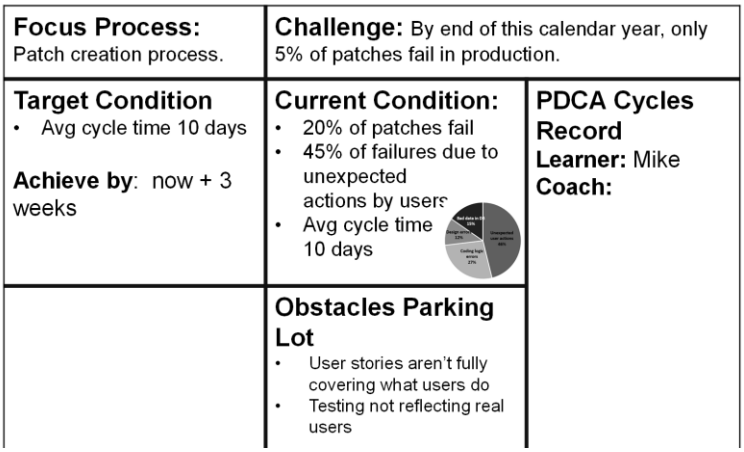


Figure 9 - Mike's second Learner's Storyboard

The teams are now driving changes based on data about their own process, and the types of failures they encounter. There is a natural forum to discuss whether these changes will affect more than one team and if so, how to prioritize. The Improvement Kata gives Mike and his team a concise and current view of the probable costs and benefits of changes being proposed, and a good view of the improvements made.

4 Reflections on the Coaching Kata

The other parts of the Improvement Kata bear some resemblance to other process improvement methods such as Hoshin planning, but the Coaching Kata is unique. When I was first introduced to the Improvement Kata, I thought that the Five Questions and the daily meetings seemed unnecessarily formal and a bit silly. After all, PDCA has been taught and practiced in many other process improvement

methods without all this folderol. However, after observing numerous coaching sessions, I've come to appreciate the value of the Coaching Kata.

Learners discover things for themselves, through rapid experiments. The hands-on experience and rapid feedback makes their discoveries stick in their heads. For instance, I've seen Learners discover key points of process improvement for themselves, such as the advisability of testing processes before disseminating them. In an amazingly short time, Learners were actively applying these learnings and expanding on them as they created useful change in their own processes. I've taught the same concepts in a more traditional way to people who were trained in PDCA as part of Hoshin Planning and/or Six Sigma. They didn't progress up the learning curve nearly as fast as did people using the Improvement Kata.

Here are some of the things I observed over a year of watching Coaching sessions.

- Although most people remembered the "scientific method" from their school days, at first very few could compose a good hypothesis. It took practice to start recognizing their assumptions as assumptions rather than facts.
- Most people wanted to go directly from "what actually happened" to "what will I do next" without asking "what did I learn?" When they were required (via the Five Questions) to explicitly consider what they had learned, the next step often changed.
- If the meetings were not at a regular time and kept short, the participants stopped meeting. Once the Learner and Coach stopped meeting, progress usually ground to a halt.
- When "How quickly can we go and see" was not asked by the coach, PDCA Cycles slowed down and Learners tried to take larger steps. The sense of urgency seemed to be lost.
- Steps requiring half an hour or so of effort over a day or two fit into people's workloads.
- Coaching meetings held 2 or 3 times a week maintained momentum, but once a week did not.
- The Coach doesn't always need to know "the right answers". The Five Questions will push both Coach and Learner in the right direction.

The Coaching Kata reminds me of pair programming. Two people look at the same problem simultaneously, with the Coach watching for the common pitfalls (such as jumping to conclusions) while the Learner is thinking about the specific problem at hand.

Mike Rother says, "The Improvement Kata pattern is the fundamental way of working at Toyota, and there are several Toyota practices through which this pattern gets utilized and reinforced. The research found the Improvement Kata pattern underlying all of them: Daily Management, Daily Problem Solving, A3, Improvement Events, Standard Work, Quality Circles…This means copying visible Toyota activities – such as the A3s – without bringing along the enabling coaching environment is unlikely to change much." (Rother 2015 A, Ch. 1). The Coaching Kata is used all the time, not just initially to teach PDCA.

5 The Improvement Kata and Annual Planning

The Improvement Kata strongly resembles many other process improvement methods historically used in annual improvement planning. The initial assessment, setting of targets, and the hierarchical breakdown is characteristic of Hoshin planning (Kenyon 1997, ISixSigma 2015). The "systems view" approach was a fundamental part of the Rummler-Brache methods which were popular in the 1990s (Rummler 1995), although those methods didn't always identify the value produced by the value stream. PDCA-style thinking is used in both Hoshin and Six Sigma, albeit with considerably larger steps.

The Improvement Kata differs substantially from the more traditional methods in its use of:

- Iterative or rolling-wave planning. Rather than cascading down yearly targets to each department and each team, the first cascade sets targets only a few weeks out. Results are rolled back up to the top, and new targets cascade down, on a regular and frequent cadence.
- The Coaching Kata. Very small steps and significant emphasis on the experimental nature of PDCA.

- Active daily management involvement. Managers are a key part of the Coaching Kata and are responsible for seeing and managing the big picture across teams.

Implementing the Improvement Kata in its entirety is challenging because it changes the way process improvement is done from top to bottom in the organization. I did observe individual teams independently employ the Target Condition "sprints" and the Coaching Kata to effectively solve problems identified in their team-level retrospectives. But these teams still struggled with problems that stretched across multiple teams or departments. They couldn't see the whole system accurately from their vantage point, so they couldn't pinpoint the causes of the issues precisely. I saw the biggest "bang for the buck" come when the Improvement Kata was implemented across a group or department. Once managers could see flow between teams, simple but non-obvious improvements came to light. (Light 2015)

In an organization without any standard process improvement method, it would probably work to start by adopting the Coaching Kata within a single team. Once several teams know how to state a Current and Target Condition and use PDCA to get from one to the other, the teams' managers could start iterative planning together, building the Chain of Coaching from the bottom up.

6 Conclusion

The biggest problem with annual improvement planning is that it's, well, annual. You are expected to plan all your improvements for a whole year despite not knowing what the changes will require. You make guesses in order to plan, but then the plan has to be changed again and again. All this re-planning is very costly as well as frustrating.

Team-level agile retrospectives avoid all the "big planning up front", but they struggle with changes that require alignment across the organization. A single team can't see all the nuances of how work flows through multiple different teams, let alone influence all those other teams. Without a view of the entire value stream, individual teams are also prone to sub-optimizing the system, so their team runs faster but the overall delivery of value has not improved and may even have gotten worse.

The Improvement Kata provides the focus and organizational alignment expected in an Annual Plan without all the re-planning. Teams work on *different* obstacles to achieving the *same* goal at the *same* time, allowing them to build on one another's work and assist one another, while taking advantage of each specific team's unique perspective on the problem at hand.

As we've seen, the Improvement Kata also incorporates many of the other advantages of agile:

- It uses Lean and agile methods that have been proven to deal well with uncertainty: cadence, time boxing, and iterative planning.
- PDCA steps split the work into very small batches, which provide the quick feedback crucial to working efficiently in the face of unknowns.
- Learner's Storyboards and PDCA sheets act as "information radiators", providing visibility to status in a lightweight manner.
- Managers have enough visibility into the system so they can see where help is needed to make improvements happen, but the actual decisions happen at the process level where people are most familiar with the process.
- The choices in the Improvement Kata are driven by data taken from specific processes in specific organizations. The focus is on making *your* process deliver your value faster and better, not on achieving a checklist of practices dictated by experts.

I see the Improvement Kata as a promising alternative to the schedule-driven methods often used in annual improvement plans.

References

Aulinger, Gerd. Rother, Mike. Rosenthal, Mark. 2013. “The Coaching Kata: Chain of Coaching”. http://www.slideshare.net/mike734/the-coaching-kata-chain-of-coaching Downloaded June 15, 2015.

Best, M and Neuhauser, D. 2003. “Walter A. Shewhart, 1924, and the Hawthorne Factory.” Quality and Safety in Health Care. 2006 April. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2464836/ (accessed July 25, 2015)

Forss, Hakan. 2014. “Toyota Kata – Habits for Continuous Improvement”. MIX-IT 2014 Lyon France. https://hakanforss.wordpress.com/2014/04/29/toyota-kata-habits-for-continuous-improvements-mix-it. (accessed June 15, 2015).

Forss, Hakan. 2013. “Stop doing Retrospectives and Start your Toyota Kata”. http://www.slideshare.net/HkanForss/stop-doing-retrospective-and-start-your-toyota-kata

Iberle, Kathy. 2013. “Lean in the Software Test Lab”. *Proceedings of the 31st Annual Pacific Northwest Software Quality Conference*.

ISixSigma. 2015. “Hoshin Planning: Making the Strategic Plan Work”. http://www.isixsigma.com/methodology/hoshin-kanri/hoshin-planning-making-strategic-plan-work/ (accessed June 15, 2015).

Kennedy, John F. (May 25, 1961). *Special Message to Congress on Urgent National Needs* (Motion picture (excerpt)). Boston, MA: John F. Kennedy Presidential Library and Museum. Accession Number: TNC:200; Digital Identifier: TNC-200-2. Retrieved August 1, 2013.

Kenyon, David. 1997. “Strategic Planning with the Hoshin Process”. Quality Digest. May 1997. http://www.qualitydigest.com/magazine/1997/may/article/strategic-planning-hoshin-process.html# (accessed July 15, 2015).

Kwinn, Kathryn Y. 2003. “Enhancing the Total Customer Experience through HP-UX Patch Quality”. *Proceedings of the 21st Annual Pacific Northwest Software Quality Conference*. Pp. 235-246.

Light, Adam. 2015. “Managing for Continuous Improvement Using the Improvement Kata and Coaching Kata”. Cutter IT Journal. Volume 28, No. 6. June 2015.

Reinertsen, Donald G. *The Principles of Product Development Flow: Second Generation Lead Product Development*. Celeritas Publishing. Redondo Beach, California. 2009.

Rother, Mike. 2010. *Toyota Kata: Managing People for Improvement, Adaptiveness, and Superior Results*. New York: McGraw-Hill.

Rother, Mike. 2015 A. “The Improvement Kata Handbook, version 31.1.” The Toyota Kata Website. http://www-personal.umich.edu/~mrother/Materials_to_Download.html (accessed June 15 2015).

Rother, Mike. 2015 B. “The Improvement Kata Practice Guide, version 10.” The Toyota Kata Website. http://www-personal.umich.edu/~mrother/Materials_to_Download.html (accessed July 23 2015).

Rummler, Geary A., Brache, Alan P. 1995. *Improving Performance: How to Manage the White Space on the Organization Chart, 2nd edition.* Jossey-Bass, San Francisco.

Brewer's Yeast: The Product Owner's Influence

Ronald Thompson, Eiscon Group

Two brewers can use the exact same ingredients, but two different types of yeast, and the results will be two very different tasting brews. Like yeast in brewing, the Product Owner's influence has one of the strongest impacts on the "flavor" – and the success — of the final systems product.

In this talk, Ron will define a very specific role for the Product Owner in the delivery of valuable systems, going beyond the often vaguely described task of "merely" providing a prioritized backlog. The role that Ron defines has four fundamental parts:

1. Defining the most valuable requirement to work on next
2. Defining implementable (releasable, that is) units of value
3. Establishing context by being Guardian of the Vision
4. Ensuring that process, product, and quality feedback loops are in place

The way the Product Owner approaches each of these tasks will, like brewer's yeast, influence the identity and quality of the resulting product.

Ronald Thompson *started his career as a programmer and enjoyed solving technical problems. Soon, however, he decided that he liked solving business problems better. For the past 30 years, Ron has worked and consulted on various aspects of delivering quality systems solutions, primarily by facilitating better communication between business and technical folks. He is currently serving as product owner for a long-time agile team at ePlan Services in Denver.*

Eiscon Group
Guides for the Information Age

Brewer's Yeast: Product Owner's Influence

PNSQC 2015

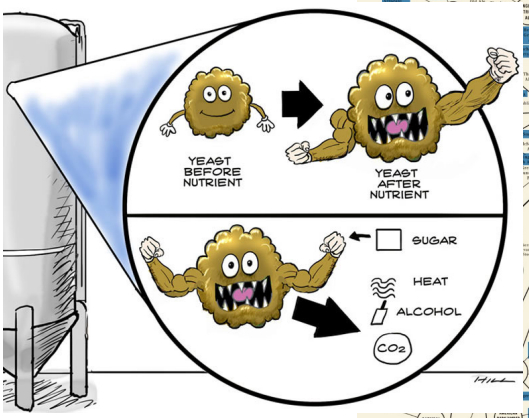
Ronald E. Thompson III

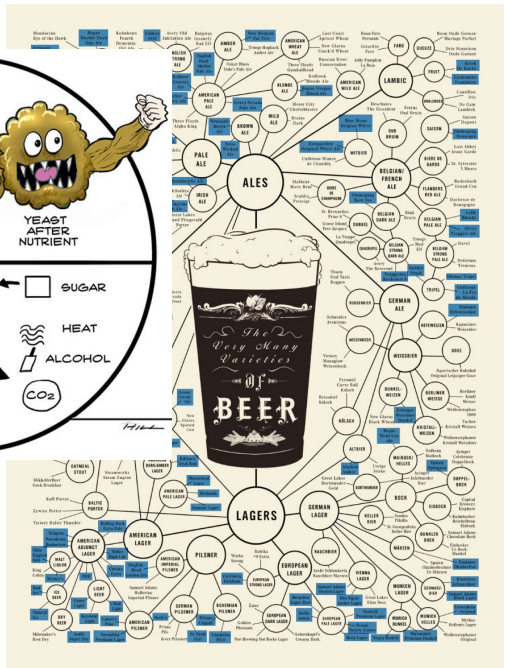
As a Product Owner, I want to have more impact, leading to increased success in delivering systems products

Key Questions

- What makes a good product owner?
- How do they influence the product?
- What does that mean to quality?

Brewer's Yeast

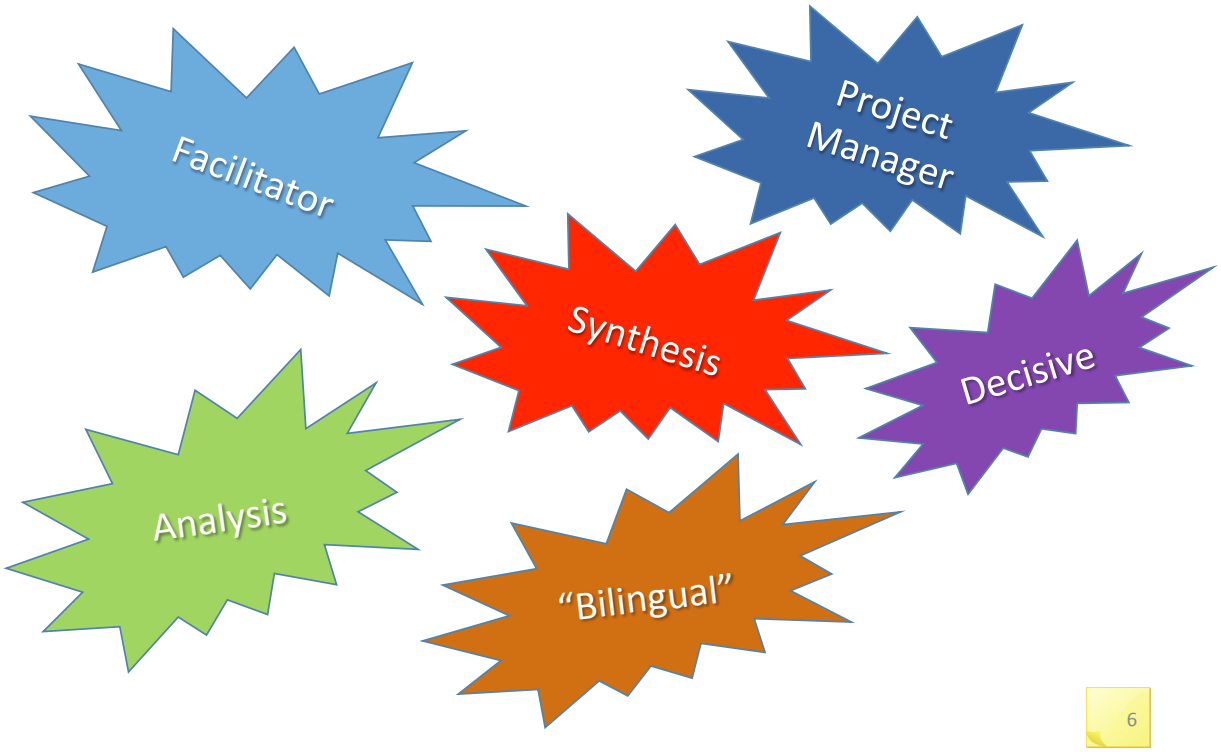




Product Owner Issues

- "Just" prioritized backlog
- Separate from team
- Same as Product Manager
- Underpowered
- Proxy
- Remote
- Too busy

Good Product Owner



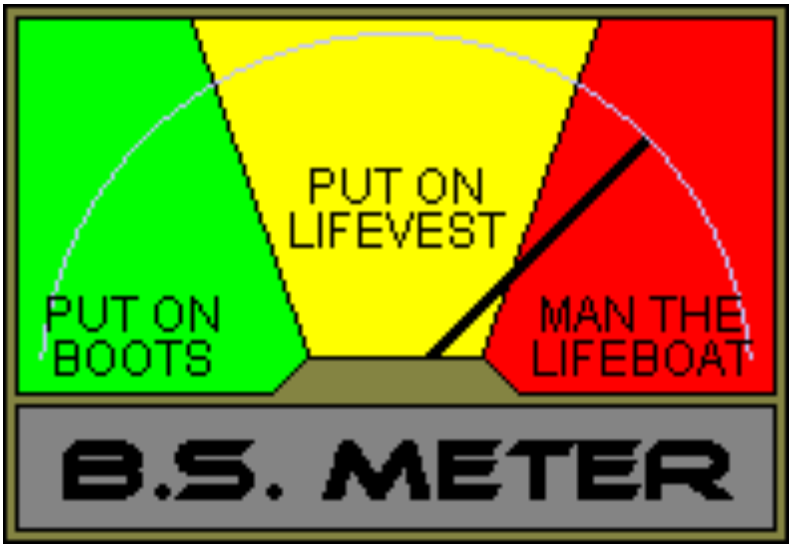
Agile Scrum Half

Predictable and sustainable delivery of highest business value

Agile Scrum Half

- Facilitates stakeholders
- Value-add translation
- Shapes the work of the team
- Introduces the product

Most Critical Skill



Product Ownership vs. Product Management

Product Owner

- Inward facing
- Customer and User
- Near-term plans
- Product use
- Product delivery

Product Manager

- Outward facing
- Market
- Long-term roadmap
- Product direction
- Profit and loss

Product Owner Influence

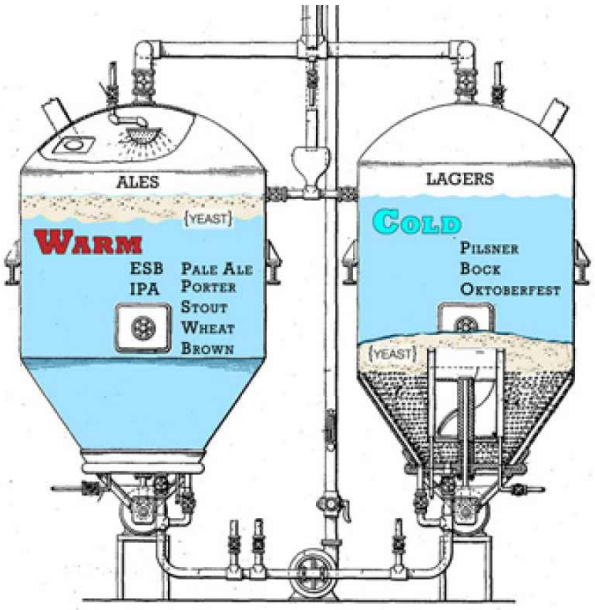
- Determining Business Value
- Defining Release Boundaries
- Establishing Context
- Creating Feedback Loops

Fundamentals vs. Styles

Determining Business Value

- Get the right requirements, rather than get the requirements “right”
- Quantify and verify value
- Preserve business value throughout
- Eliminate low value, high effort
- Get desired business value

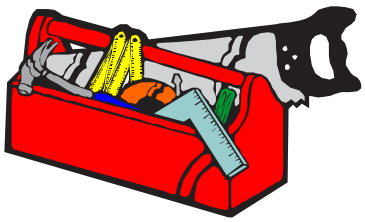
Ale vs. Lager



http://www.popsci.com/science/article/2013-01/beersci-what-difference-between-lager-and-ale

Tools: Determining Business Value

- Theme/epic/story/requirement/feature
- Lean startup experiments
- Negotiating Minimum Viable Product (MVP)
- Estimates of value and effort
- Numbers v. “Feel”
- Identify assumptions

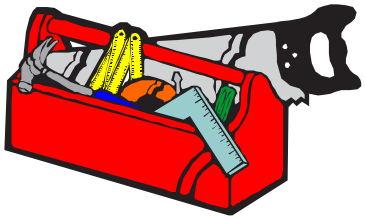


Defining Release Boundaries

- Implementable unit of value
- Long and short-term view
- Boundary, not scope
- What fits?
- Priorities
- When is it “good enough”?
- Ready to release?

Tools: Defining Release Boundaries

- Charters
- Theme Road Maps
- Story Maps
- Project Milestones
- Priorities
- Sprint/project planning
- Boundary adjustments
- Negotiating MVP



Establishing Context

- Who uses the product?
- What do they want it to do?
- Why do they want that?
- What is the priority?
- Guarding the vision
- Guiding to the vision

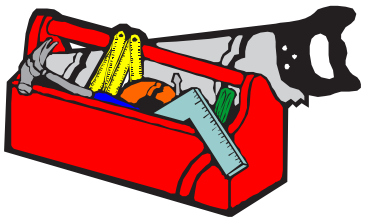
Yeast adapts to its environment

So, using a “Belgian” yeast strain will not make your beer Belgian.
Only one thing will make your beer Belgian: brewing it on Belgian soil.

-- from beerandbrewing.com

Tools: Establishing Context

- Vision
- Charter – boundaries and objectives
- Themes – coarse-grained stories
- Answer questions (or get answers)
- Make or facilitate decisions
- Business procedures
- Training material

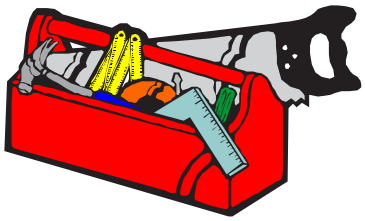


Creating Feedback Loops

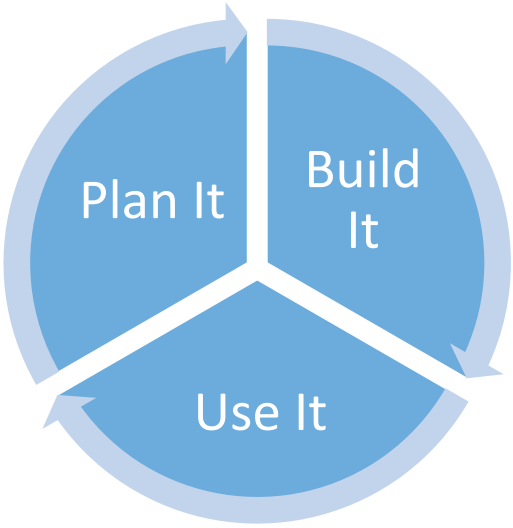
- Process
- Product
- Quality
- Leading and trailing metrics

Creating Feedback Loops

- Acceptance Tests
- Value v. effort tradeoffs
- Metrics
- Experiment Results



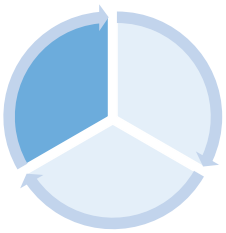
Quality Throughout the Lifecycle



Plan It

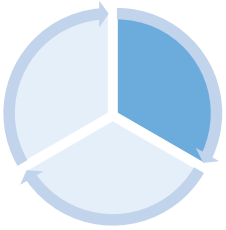
Context feeds Quality

Value
Priority
Risk
Importance
Sequence



Build It

- Create Acceptance Tests
- Evaluate Acceptance Tests
- Discuss exploratory testing results



Use It

- How is the product really used?
- Where are the problems?
- Is something not working?

Yeast is good for beer

Product Owners are good for developing products

Questions

Eiscon Group
Guides for the Information Age

Ronald Thompson
Eiscon Group, Ltd.
303-470-6922
agilescrumhalf@eiscon.com

TestOps in a DevOps World

Simon Howlett

simon.howlett@workiva.com

Abstract

A DevOps team's remit is that of 'frictionless development'. This is a desire for a development ideal driven by a 'need for speed' and the conscious & intentional removal of hold ups in taking an idea from concept to customer. Frictionless development allows product teams to concentrate on innovation instead of being beholden to often-arbitrary processes. This movement poses an interesting question for Quality Assurance Engineers as the 'need for speed' mindset places challenges them to do more with less (time, mostly), and the challenge to sync the two sides of the conversation is often fractious.

Focusing on increasing the speed of delivery often brings issues of quality into focus. Careful relationship management and compatible tooling coupled with a quality driven mindset across an organization is required to attempt to ensure that 'need for speed' is taken as a metaphor for efficiency not just 'fast'.

Aligning both DevOps and TestOps teams is a technique that can speed up the product release and support a robust and reliable quality focused product, two areas that often comprise each other.

Biography

Simon Howlett is a Senior Product Development Manager in Test Engineering at Workiva. Focusing on providing Test Frameworks and Solutions with an emphasis on reliability combined with ease of use/support. Simon currently manages a team of engineers based mostly in Ames Iowa, though he himself resides in Portland, Oregon.

Copyright Simon Howlett, 2015

1 Introduction

Workiva provides a cloud-based SaaS platform for collaborative reporting across multiple industries. They have been successful in part through creating a customer first culture throughout the whole organization. To Quality Assurance (QA) team members, that culture means striving to ensure everything released to customers reaches a level of quality and reliability the customer can count on. Workiva's Wdesk platform handles many large complex documents for customers working in often highly regulated industries and this demands a focus on reliability, security, accuracy all of which provide our engineering teams with some quite challenging opportunities. Workiva also like to release every day, go fast and in most cases rewrite conventional wisdom, particularly in terms of people's roles in producing a 'quality' product.

Any Agile software organization looks to release product updates often (in some cases more than daily) and see benefit from putting their teams in charge of their own releases. DevOps teams are often the focal point of this process, creating tooling for build and deployment environments allowing teams to go as quickly as they desire but in the testing world there is often a disconnect in this ideal due to checks and gates created as part of a QA process. This process creates an impression that QA and testing speed is a bottleneck to the organizations goals. An equivalent discipline, 'TestOps' exists, that when working alongside DevOps can be vital in ensuring this is not the case, and can more importantly assist in team empowerment by creating a seamlessly deployed testing environment alongside DevOps build and deployment infrastructure.

A 'TestOps' team in essence concentrates on availability of infrastructure and platforms required for testing at all levels, from functional testing through integration testing to lower level unit and API tests. This is a change from the traditional Test Engineer role that can often be the person responsible for writing test fixtures and maintaining test frameworks. Through making all test tooling available to all of the product team as part of the same workflow they already use allows a change in mindset which helps to move the responsibility for quality (through test coverage and process) onto the whole team (instead of the traditional 'tester'). When combined correctly with a DevOps 'frictionless development' mindset a very robust clean development, testing and release process can occur.

Where a DevOps team would focus on build, configure and deploy, a TestOps team can interface with the build and configure stages to communicate with this system provide testing feedback mechanisms, that are defined in testing specifications owned by product teams themselves. This gives a seamless commit-tested workflow that allows for fewer bottlenecks in the development process. This flexibility extended with continuous Integration removes hours of manual intervention to setup testing environments, reducing the time it takes to get quality, tested code in front of the customer. This can only be successful if DevOps and TestOps platforms are in sync, and the two teams work closely together both on products and process.

2 Case Study

In early 2010 Workiva had a team of 30 development engineers and a QA team of 5 people, all working on Wdesk, a cloud based application aimed at enabling companies to create and file accounts, pre-releases and other documents with the Securities and Exchange Commission (SEC).

Pre-release testing was completed on a manual exploratory basis and very quickly the ability to cover the application with just manual testing became an issue and graduating from manual test scenarios to an automated testing process became a project for a small group of QA engineers.

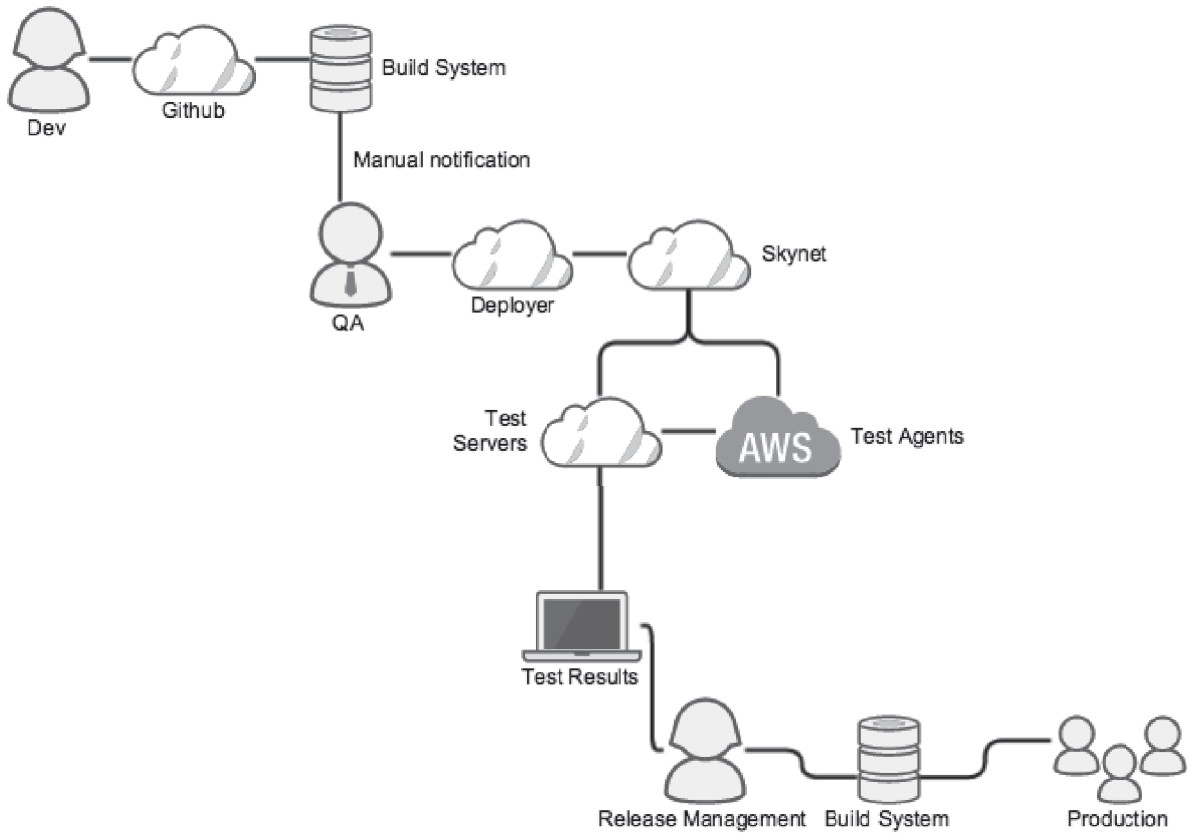
The initial goal for this new team was to automate test coverage for the 800 manual test scenarios the QA team worked through prior to each release (at this point Workiva were on a release schedule of every two weeks, with up to 2000 updates in each release). These test scenarios included mostly end to end tests ranging from document creation and editing to format translation to confirm that documents content and formats were correct when translated to formats defined by the SEC for corporate accounting.

An initial challenge to this plan was the platform on which the testing needed to occur was built using Adobe's Flex language. This only provided a small number of off the shelf testing options, really only one that was deemed suitable for the task, SmartBears 'TestComplete', and its test runner, 'TestExecute'. To make things simple for testers to create coverage we created an easy to use drag and drop interface to create tests built using business language driven test fixtures. This separation of test runner and tests meant that rapid progress could occur on gaining test coverage for our product without the need for testers to write code or incur large license costs for test complete.

Over the next two years test coverage grew from the 800 tests per release that covered the initial manual test cases (the initial goal for the project), to 7000 full stack tests. These tests were now running 24 hours a day, full runs on release candidates, customized components for product team builds and small hourly smoke test builds.

Over time the supporting infrastructure also increased from running tests on a small farm of physical machines to a virtualized platform of 40 Google App Engine Servers, 400 Amazon ec2 machines replicating the users, and running through around 1,000,000 test scenarios run per month. Somewhere along the way the framework adopted the name 'Skynet'.

Figure 1 - Initial Test and Release Process



2.1 Initial Testing Success, future scaling issues

The initial goal of providing test coverage had been met and had found hundreds of bugs prior to release (and continues to), though it was quickly becoming apparent that these tools are very 'tester' focused. Although functional test coverage could be created by developers at the same time as putting products and features together, the tooling to do so lived outside of the environments and repositories developers worked in each day. This was not going to be scalable as the organization moved quickly to a model of multiple product teams and services releasing independently through a streamlined release infrastructure championed by DevOps. Other time consuming issues arose around required manual intervention (as seen in figure 1) a manual handoff between development and QA post build, and the same to Release Management, post QA results review, and these needed to be mitigated going forward.

In some ways this testing approach had been too successful. Functional tests were used for all testing, in some places where much faster lower level tests could have been used. Particularly in the area of service testing (such as document translation and comparison), where over 1000 functional tests existed. These tests took around 4 minutes each performing the same user actions that were not actually the point of the test – the translation itself took seconds, so this was a costly problem for everyone

This was not the only challenge to testing in the existing workflow;

- **Flaky tests.**
  Release Candidate runs average around 99.1% passing rate on over 7000 tests. This means that on any test run QA has to research 1-30 failed tests. There are some tests that are unreliable given the frequency of failure due to latency or odd infrastructure conditions. As a result the aim to have 100% passing tests runs on our main customer product is made unrealistic by the platform it is testing – there has still to this day never been a full functional test run on this part of our application with 0 failed tests!
- **Lots of maintenance**
  Rather than a product team owning and updating tests a secondary team has to update fixtures and mappings for the flex based tests due to prohibitive costs of TestComplete. This also introduces risk by having a team not familiar with the product under test, investigating the reasons for failing tests – often coding around bugs to resolve issues without realizing.
- **Time**
  To complete a full test run takes somewhere in the region of 3 hours. This is mostly restricted by the number of TestExecute licenses, and our server platforms ability to scale to handling a given number of test tasks at any one instance. Going forward working in a much more distributed model, test runs could not possibly take this length of time
- **Cost**
  Each test run stands up the full application stack. This involves storing, copying, amending data on 40 servers and around 300 user machines. This proves expensive (much more than service level/integration tests) though when compared against the cost of those bugs caught getting passed to the user the effect is still positive – even though seen as 'revenue protection' it is clearly not the most cost effective way to do this. Costs for testing approached around $80,000 per month at their most expensive point.
- **Tests could not be dependent on each other, no collaborative testing**
  Each test is run in isolation and state is not preserved between tests. Also collaboration scenarios could not be tested which was a large problem given a large focal point for our customer was robust collaboration on documents.

As the organization moved to break up into discrete teams each contributing its own micro-service outside of the monolithic codebase it became important to re-evaluate this model with the entire product team in mind, and the new DevOps delivery model.

3 Scaling with new DevOps and TestOps teams

With DevOps teams taking responsibility for build and deployment architecture allowing engineering organizations to scale as quickly as their business demands, its vitally important that TestOps teams are able to interface accordingly and grow alongside DevOps. Having a test infrastructure that is not as easily maintainable and usable as a nimble build and release mechanism will lead to delays in software releases as QA gets backed up in checking for release confidence in its prescribed test activities. This cohesion is made more important as Agile teams move to releasing smaller release more frequently to customers. In turn, with many cloud-based vendors available to customers, quality must be upheld even with the ability to update applications at a much greater frequency.

In early 2014 Workiva created a DevOps team to rethink release infrastructure to support more products, with an intention to move to a model of interoperating product teams and micro-services, allowing each team to release independently and much more frequently – more than once per day.

What had previously been an application platform based on a monolithic codebase was being broken into individual products, modules and micro-services. This changes the release model from one that can be managed in one place by a Release Management team, to one where teams could benefit from being able to release independently as and when they choose.

The DevOps team was to build and support a new architecture and workflow enabling product teams to release software at a time and frequency their own choosing. Building in support for container based application deployment (using Docker) and creating an eco-system that supported all technologies that product teams saw the best fit for their products/services.

Alongside DevOps, the Test Engineering team (prior to being known as TestOps) was already owners of a test framework and this team had already began to realize that the model for testing that was currently in use could not scale, certainly in an organization that was moving quicker and wanting to deliver software much faster than before.

Ensuring that the DevOps and TestOps teams and tooling are in sync is a challenge, competing priorities, differing customers and outlooks mean there needs to be a very simple interface between the two, and an understanding that this interface must always be available. Common REST API's proved to be an efficient choice, allowing the TestOps team insight into what was building/deploying in DevOps world, and in return, Test Results and Reporting from TestOps to support releases were provided back to the build system (which in turn needed to be provided to Release Management to support questions from external auditors).

All of this must be close to seamless as any manual intervention or delays to development reduces buy in from teams and becomes a perceived bottleneck or flakiness in the process. So communication on dependencies between teams was established at the outset, with an architect from each team in place to ensure the vision of both teams, whilst focused in different areas leads to something that appears to be the same outcome.

Another interesting component in making all these changes work together is a Release Management team. Responsible previously for managing every part of every release of updates to customers, in the new model teams are empowered to be able to release on their own at any time they feel ready to do so.

The Release Management focus moves to building tools to assist in quicker release of products through the removal of checks that we previously done by humans, providing a suite of tools such as test coverage reports and automated release process checks (checking code reviews and testing sign offs are as expected on pull requests, for example) to check for auditable reviews and test artifacts and streamlining teams release process to allow them to move faster.

This new found team autonomy poses an interesting challenge for a Quality Assurance Analyst/Engineer as the model they are used to working in of testing one 'ball' of code where all assumptions can be clearly

seen, changes to one where they must know the status of each of these component parts at time testing occurs – and where any interoperability occurs.

The QA Analysts role itself is also adapting to become more of Quality Coach, or in some cases, a Quality Engineer (more accurately, the 'Software Development Engineer in Test', or SDET) and the role of a traditional tester was seen as possibly becoming obsolete. The idea of a separate entity solely responsible for the quality of an application does not fit the agile paradigm of the 'whole team' being responsible for quality. Whilst the QA person on a team was responsible for ensuring quality, they should not be seen as the only person responsible for building it into a product and process.

Critically at this point, the established Test Engineering team focus needs to move to a more 'TestOps' like outlook to ensure we could provide the tools and support to ensure the high standards upheld in terms of quality were maintained at scale, and where teams are now accountable for their whole release process, but kept in sync with the 'frictionless development' path being championed by DevOps and engineering as a whole.

3.1 DevOps & 'Frictionless Development'

DevOps teams are the providers of 'frictionless development'. Allowing teams to innovate and not be hampered by environmental or workflow issues, putting new code into production and respond to feedback quicker. In an Agile environment teams themselves are empowered and accountable for making what they need to happen to release quality software that customers love, without any noticeable drop in quality. This means the DevOps team undertaking the task to build and support an infrastructure to support a wide ranging build, test and deployment system on whatever platform teams are needed by team to build and deploy their new products.

3.1.1 DevOps Tooling

Continuous Integration (CI) systems are the backbone of DevOps success. A good CI system enables quick feedback on build stability, quicker artifacts into QA, and ultimately quicker to production code.

Typically DevOps will own the build system, deployment tools (assuming cloud/server based products), production monitoring systems (occasionally referred to as CloudOps) and often release consistency measuring tools such as code coverage reporting. All of these tools can also be leveraged by TestOps, to deploy test builds, report on performance and consistency etc. providing feedback to product teams on exactly the same infrastructure used for production deployment reducing time and cost to setup and maintain separate test resources for deployment and also give a much more real world feedback loop for release confidence.

3.2 Build and Test Infrastructure Cohesion

DevOps and TestOps infrastructure need be in sync. It should appear as one smooth system/workflow to any engineer, committing code (and test coverage) should be all that they need to be concerned with, transparency and reliability is key for success anything that adds to the time taken to gain feedback on features loses the momentum 'frictionless development' strives to provide...

A quick, reliable build system is the cornerstone of any DevOps infrastructure, Travis Ci, Jenkins, Bamboo etc. are well trusted build systems, though Workiva chose to build a new in-house build system was created with an emphasis on speed and reliability.

A new build system *'Smithy'*

Smithy is instrumented to run unit, integration or functional tests, defined in a yaml configuration file at build time. Failed unit or integration tests prevent a build artifact for release being created (again

increasing the focus on quality and accountability on the product team as a whole). Test coverage and result metrics are recorded within Smithy to support a teams' release, and numerous API's are available for use in other applications for release management, auditing, documentation etc.

These build system API's are the key to working with other systems as they provide the view into the state of any application and its teams requirements at any time. Using triggers like file changes, GitHub pull requests, release branch merges/tags, it can be possible to send notification to TestOps that testing resources are required due to the type of build in progress and at completion tests can be commenced. This makes the experience smooth for the engineer committing code as if they are aware of a given trigger then can start tests without any intervention from others. No requirement for a QA engineer to deploy a build manually to a test server for example.

A new deployment architecture;

A dependable deployment architecture is the secondary DevOps platform piece, again making getting applications out to the customer for feedback / into production simple. It should be configurable to define set standards where deployment is allowed (no drop in unit test coverage, no functional test failures etc.), but these should be safeguards not bottlenecks to a team.

Based on the concept of isolating applications in and environments in Docker containers Workiva two further applications for deployment;

- ***'Shipyard'***: Workiva's Docker container build tool
- ***'Harbor'***: Workiva's package and container release service

Both of these are also needed by TestOps for deployment of builds for testing, so API's etc. where needed should be available to consumers/users.

4 TestOps

TestOps as a concept revolves around ensuring product teams have access to any required test infrastructure, platforms and frameworks they require without needing large amounts of time consuming configuration before commencing tests. Any benefit from use of a CI system will be lost if the QA process takes days to complete due to environmental setup and tear down on the part of the QA person on a team.

A DevOps/TestOps model is intended to focus the whole team on quality and as such anything TestOps does needs to be seamless and reliable. A simple example is any test frameworks must be reliable enough to very rarely have downtime or connectivity issues. Any time lost evaluating flakiness, or delaying feedback on release build will create a bad perception of the usefulness of the system. This makes the mindset of the TestOps team critical.

4.1 Test Coverage, and a Zero Bug Policy

Any TestOps team has to be highly functioning, highly accountable team to get this right. It must be understood what impact is incurred if any TestOps items are out of service or not functioning correctly. Two techniques that can be employed to help with this are high unit test coverage for in house built testing tools, and a 'Zero bug policy'.

The unit test policy is essentially any drop from 100% line and statement coverage fails a build. This means we can rely on our test coverage at a lower level and engrained in the team is a mindset of writing better code. Understanding the impact of loss of confidence in a testing system is great incentive here.

The zero bug policy essentially means any bug raised on production systems at any point during a team sprint must be evaluated and either worked on at the expense of a lesser item, or if deemed not customer impacting, closed and explained as to why.

The result of introducing these policies at Workiva has ensured a highly accountable TestOps team that has a high uptime rate and great respect in the engineering organization. This assists with the communication with DevOps and other teams as we talk as equals, as well as internal customers.

4.2 TestOps Tooling

For a TestOps team, the most important activity is providing exactly what a product team needs to test and receive feedback. For agile product teams and the advent of micro-services a need for new types of test infrastructure arise, instead of a traditional model where an environment for testing depends on the whole application stack being available some more considered requirements emerge:

- What tests do is entirely within the teams control, test governed and managed by an external team promotes risk to product quality
- Testing should need as little setup as possible – any dependent services should automatically be available (or mocked)
- Test results reported automatically along with any generated artifacts, preferably using a defined format (such as xUnit, jUnit, Rich Test Format etc.)
- Tests are executed within an environment that can be mutated in any fashion desired without impacting other tests (such a virtual environments, containers etc.).
- Interoperability, backwards-compatibility, and well-defined APIs have never been more important. Careful attention to API and service level testing is a must.

Most importantly, newly formed TestOps team needed to plan a workflow where teams can own their test frameworks and leverage DevOps infrastructure to enable simple to setup test solutions.

4.2.1 Integration with DevOps infrastructure

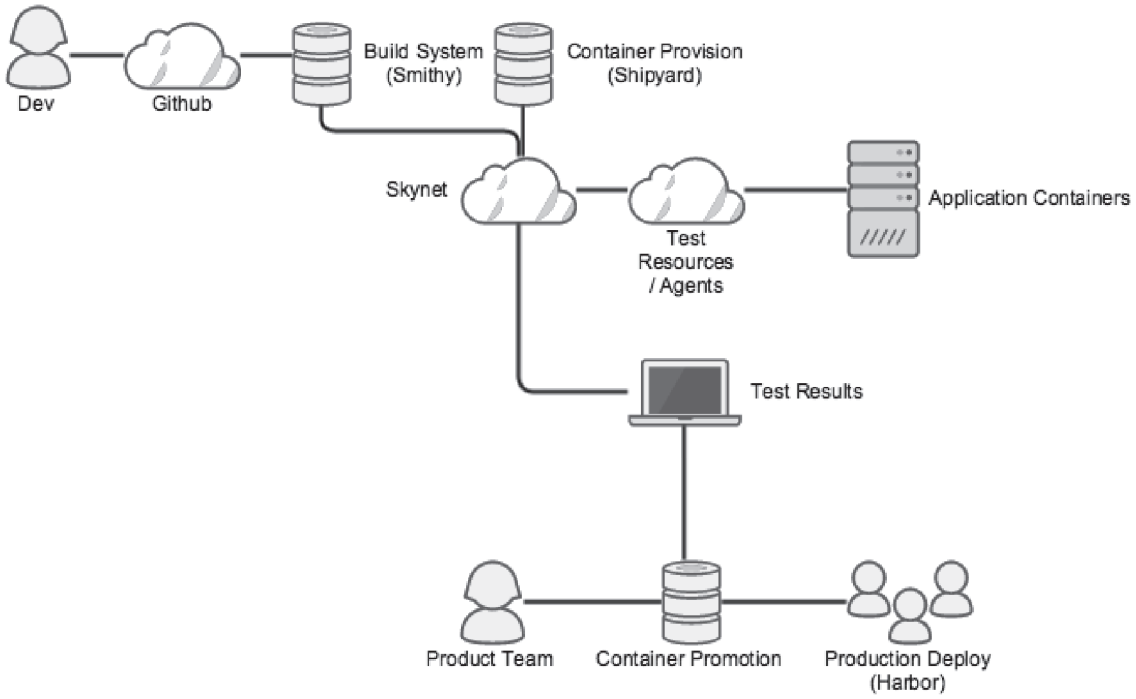
For DevOps the build system is the central application for all teams to create release artifacts and perform lower level tests. In terms of functional testing workflow, given the long running and often brittle nature of these tests preventing build completion on failing tests is a problem, with this in mind functional testing is kicked off as a post-process of the build system, and as such requires TestOps to build an infrastructure that not only takes it cue from DevOps systems, but integrates with them as much as possible to allow for transparent workflow for teams

The connection from a build system to a test system should be as simple as checking an API for notice of build completion, readily for testing. No human interaction to start up functional tests should be required. In Workiva's workflow completed builds are marked as 'success' in the API. Each build in is noted via a listener application in TestOps on starting and when it is completed the API shows a value for the build of "status": "success", 'Skynet' can look to the build itself to see if testing is needed.

To make the call for testing to start Skynet checks for the presence of a skynet.yaml configuration file in the repository from which the code came and if one is found, Skynet reads in its content to pick up requirements & configurations for testing.

This API watching creates the important seamless transition fro DevOps infrastructure to TestOps. The build artifact is available, its low level tests have passed and it is off into any selected functional Testing.

Figure 2 - New Test and Deploy workflow, syncing DevOps and TestOps tooling



4.2.2 Enable teams to easily specify what tests to run and when

To fully realize the value of this seamless workflow, teams must specify what they want testing and when. A specification must exist that TestOps can programmatically understand removing the delay caused by human intervention to configure testing, containing details such as;

- Definition of when the tests are run – every commit, or whenever a specified identifier (in this case Github tag/release is present) and any subset of these
- Where the tests for the given build/module/feature are located
- What test runner commands are needed
- What upstream and downstream tests need to be located and run, again based on the same tag identifier as above
- What service template is to be used for these test – essentially which resources and environment such as servers, Docker images etc. are needed for these test to be executed

TestOps can then translate these requirements to begin testing as soon as a build is complete, testing can commence with no human intervention and at this point QA's task is to review test results.

4.2.3 ‘You don’t have to stand up the whole system to test’

As teams are creating and consuming other teams products, modules and micro-services it is inefficient to need to recreate the whole platform in testing for all testing. Standing up dependent services should be enough as long as it is implied there is continuity between services (consistent APIs etc.).

To test a service simply would involve sending a message from the originating service, and then asserting that the message is received by the message broker. As long as the service that is receiving the message maintains their API, then the secondary and downstream services do not always need to be tested at this point.

For example, when a teams service depends on Vessel, a message broker, to send messages to another service, which depends on a dozen other things all that is needed to be tested is the linkage between their service, and vessel itself, not any other services further downstream

Proper versioning is a necessity here as If an API changes the type of message it receives failure will occur unexpectedly. To ensure consumers can continue to function until the update is available a given standard for API versioning must (and does) exist

4.2.4 Enable teams to also test their changes with upstream and downstream consumers

Most team’s features are not lucky enough to work in isolation and many depend on external services, or contribute to other products/features. Being able to ensure release confidence on both the changes under test and any consumers is a challenge especially when those consumers are often not in the same location or codebase. TestOps should provide a mechanism for configuring what other modules tests can be utilized post build to ensure consistency and root out any breaking changes to downstream consumers without manual intervention. This configuration is best placed in the same location as the other configurations for testing, the testing specification as noted in the previous sections. This specification is another item that removes risk, as the product team should in most cases know best what parts of an application/platform their changes will interact with.

4.2.5 Testing Specification Example

Keeping testing specifications as simple as possible is key to success, it should be human readable and of a simple format to ensure consistency. For Workiva the YAML format works perfectly and it is also widely used already across the organization in other products. The skynet.yaml file provides a working test specification which defines when and what testing should occur and if another product or features tests need to be run. This allows teams to ensure that their release does not break another team’s work and/or functions in the way that is expected once consumed by others.

TestOps track builds from the DevOps API list of in-flight items to identify that we may need to test. Examining the repository for a skynet.yaml on these builds triggers an event if test criteria is satisfied, queuing the build in skynet ready for tests to commence as soon as the package is available

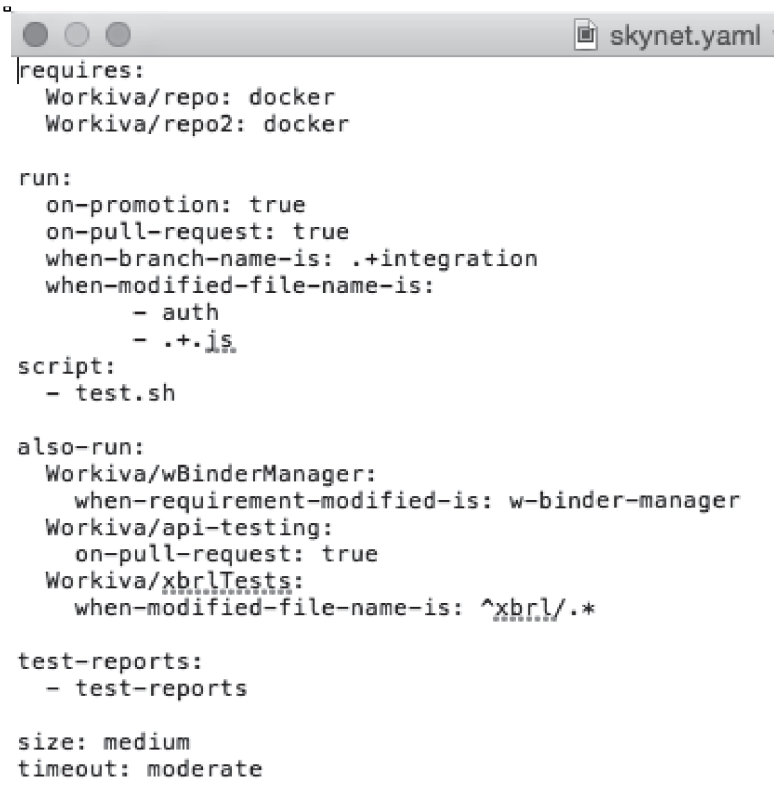


Figure 3 - Example skynet.yaml file indicating what testing requirements exist for an application

This above example skynet.yaml file specifies that;

- when a pull request is made from a branch with ‘integration’ in its title (or any of the other specified criteria)

- Using the specified Docker containers run the tests, using any commands noted within test.sh
- run the tests in the also-run section if other criteria are met
- output the reports to the test-reports directory for submission
- a timeout value, if exceeded will cause tests to be failed

Further configurations can include:

- Environment variables needed at run time, such as cloud service credentials
- An over-ride to allow container promotion to production even with test failures
- ‘App-id’ – a request for a GAE Server to specify to deploy a build to and run tests (similar to the original testing model)
- ‘Artifacts’ – a location to store artifacts such as test files created during testing
- ‘Env’ - A list of strings or maps that declare the environment in which the scripts run.
- ‘Image’ – specify a Docker image to use
- ‘Requires’ - Can be set to a map of source keys and artifact values. This creates an explicit dependency on a build artifact for these tests to execute and all build artifacts will be downloaded and made available in the test execution environment before the tests are started. If the build triggering the test run is listed as a dependency, then that version of the artifact will be obtained. Any other dependencies will obtain the latest released version of that artifact.
- ‘On-promotion’ - Execute tests when an application is being promoted to the production application repository.

This ability to set configuration options at the end of build time syncs TestOps work on test frameworks and platforms, with DevOps build and release tools. If all of the pre-build tests pass, the build artifact is passed to Skynet, the tests configured in the skynet.yaml are run, and if these return without error the build is passed for promotion to production, and placed in DevOps deployment queue.

5 Team Investment

With the two sides of the infrastructure in line between DevOps and TestOps, the remaining piece is the cultural change to ensure the benefits of this system are seen by all the product team and they all invest in the quest for quality. The traditional model of separation of duties in testing between development and QA team members can be seen in part as a reason for the slow down in a release process. Developers had the friction removed in their workflow, only to get hung up in labored QA who also had the task of providing test coverage. This slowdown removes any benefit from aligning DevOps and TestOps

The model intentionally puts the test tooling in everyone's hands to counteract this perception, developers could write tests whilst coding features at all levels of the stack and could run them all on commit, rather than waiting for feedback from a QA resource after a round of manual testing. Each build gives them the artifacts required for release or items to be resolved. This allowed the QA resource to assist in test writing, exploratory test with less of a release pressure from the rest of the team who viewed their part as 'done', and also time to concentrate on deeper investigations, such as performance.

This also removes any element of blame. The often heard 'well, did QA test it' is no longer a valid question because it is the whole team's responsibility to support their release, not just the QA or Release Engineers say so. This increases accountability on the team and encourages participation and pride in gaining solid releases. The ideal place to end up is to be able to release software without QA intervention, because the whole team is so invested in test coverage and quality, the DevOps/TestOps tooling allows for quick accurate feedback on the state of a release.

6 Evaluation

Whilst this model is relatively new, Workiva has already seen benefit of this model:

- Because of changes to testing and release tooling we have moved away from an irregular release schedule, where we released an average of twice per week. We now consistently release to production four times per week.
- Moving builds to be deployed automatically to test environments reduced the average time from build completion to testing starting down from an average delay of 1 day between builds completing and tests starting, to the delay being zero, and the tests start as soon as the build completes. This puts tested builds in a ready to release state 1 day earlier on average.
- With this and other efficiency gains from Development, Testing and Release Management tooling since fall 2013 we have decreased the time from when a pull request is created (after code review and testing) to production, from an average of one week to one business day
- Moving translation tests into a service level test instead of part of the functional test run enables them to test in around 1 hour, outside of the current Skynet queue. Previously running the 335 translation tests took 100 minutes (duration), which equates to 50 instance hours (100 minutes on 30 test servers) for each test run that used these tests. This would be every release candidate run (once per day) and any other test runs utilizing translations, reducing costs and increasing throughput in the test queue. This workflow will be replicated for at least 3 other features in the coming months.
- Teams moving off the old flex based test frameworks no longer have to wait in line for an available test server and 'tester' machine, Docker containers are immediately available to run tests and test feedback is available in an order of minutes vs. hours (or even days)
- We have a 20% reduction in server costs for automation since starting this work, this comes in the form of instance costs, data write/deletes etc. as less needless test scenario setup is being done as tests move to a more appropriate place in the testing stack

- Added to this 20% we have achieved a 50% cost reduction overall in 12 months (a saving of around $40,000 per month), some of this through moving tests into test frameworks that test just the service under test, reducing test times and the number of scenarios run for each release. For our flex application, the number of total tests per run has reduced by 15% over a six-month time frame, and with the introduction of new test frameworks and test tools we would expect this to continue at a similar rate over the next year.

One relatively simple example of progress is where previously a team may or may not opt to run our full battery of test against their development builds, and an issue downstream in a consumer of that changes feature does not get exposed. In the previous model, we would 'luckily' catch that error in our Release Candidate test run, and it would have to be reverted from the release and through another round of development then release testing. In the model where each team releases its own updated container for its own codebase, that Release Candidate test run simply not occur, left unchecked that bug gets out into the wild, and breaks the consumers feature, yet with test specifications in place, if a test fails in a consumers tests when the parent is updated, the release is flagged straight away as no good. It does not take another release run of all of our tests to catch that, nor is there an extra risk of it getting into the customers hands broken

The challenge of keeping up with DevOps frictionless development really is centered around working with them and the product teams to see where TestOps can fit in the tools needed for testing without hampering development workflow. So far at Workiva we have had some success with a dramatically changed model for Testing and Release, but have managed to avoid separating a development organization empowered by DevOps tooling and process, from a QA Team using 'TestOps' tooling by ensuring the two are in sync. Ensuring teams are empowered to release at will through coordinated work through DevOps, TestOps, Release Management (and no shortage of available API's) ensures we can go fast and intentionally remove gates in the release process, whilst keeping the level of quality in our products our customers expect, and keep our resources happy and engaged.

Glossary

CI – Continuous Integration
DevOps – Development Operations
Ec2 – Amazon's Elastic Computing virtualization platform
GitHub – Code Repository and Version Control
REST API - A standard for externalizing cloud services to consumers
ROI – Return on investment
SaaS – System as a Service
TestComplete – A Front End Test Automation Application, made by SmartBear
TestExecute – The Test Runner for TestComplete
TestOps – Testing Operations
Skynet – The name given to Workiva's Test Framework Controller
YAML - a human-readable data serialization format

References

Web Sites:

Zellermeyer, Gal, 2013, '0 Bugs Policy'
http://galzellermayer.blogspot.com/2013/05/0-bugs-policy.html

DevOps: Are You Pushing Bugs to Your Clients Faster?

Wayne Ariola

wayne.ariola@parasoft.com

Abstract

Now that rapid delivery of differentiable software has become a business imperative, software development teams are scrambling to keep up. In response to increased demand, they are seeking new ways to accelerate their release cycles—driving the adoption of agile or lean development practices such as DevOps. Yet, based on the number of software failures now making headlines on a daily basis, it's evident that speeding up the Software Development Lifecycle (SDLC) opens the door to severe repercussions.

Organizations are remiss to assume that yesterday's practices can meet today's process demands. There needs to be a cultural shift from *testing an application* to *understanding the risks associated with a release candidate*. Such a shift requires moving beyond the traditional "bottom-up" approach to testing, which focuses on adding incremental tests for new functionality. While this will always be required, it's equally important to adopt a top-down approach to mitigating business risks. This means that organizations must defend the user experience with the most likely use cases in the context of non-functional requirements—continuously.

In order for more advanced automation to occur, we need to move beyond the test pass/fail percentage into a much more granular understanding of the impact of failure: a nuance that gets lost in the traditional regression test suite. Continuous Testing is key for bridging this gap. Continuous Testing brings real-time assessments, objective go/no-go quality gates, and continuous measurements to refine the development process so that business expectations are continuously met. Ultimately, Continuous Testing resets the question from "are you done testing?" to "is the level of risk understood and accepted?" This paper is designed to help software development leaders understand how a new perspective on "test" can help them accelerate the SDLC and release with confidence.

Biography

Wayne Ariola, Chief Strategy Officer at Parasoft, leads the development and execution of Parasoft's long-term strategy. He leverages customer input and fosters partnerships with industry leaders to ensure that Parasoft solutions continuously evolve to support the ever-changing complexities of real-world business processes and systems. Ariola has contributed to the design of core Parasoft technologies and has been awarded several patents for his inventions. He has a BA from UC Santa Barbara and an MBA from Indiana University.

Cynthia Dunlop, Lead Technical Writer at Parasoft, authors technical articles, documentation, white papers, case studies, and other marketing communications—currently specializing in service virtualization, API testing, DevOps, and continuous testing. She has also co-authored and ghostwritten several books on software development and testing for Wiley and Wiley-IEEE Press. Dunlop holds a BA from UCLA and an MA from Washington State University.

1 Introduction

In response to today's demand for "Continuous Everything," the software delivery conveyer belt keeps moving faster and faster. However, considering that testing has been the primary constraint of the software delivery process, it's unreasonable to expect that simply speeding up a troubled process will yield better results. (I Love Lucy fans: Just think of Lucy and Ethel at the candy factory, struggling to keep pace as the conveyer belt starts putting out chocolates faster and faster.)

Now that rapid delivery of differentiable software has become a business imperative, software development teams are scrambling to keep up. In response to increased demand, they are seeking new ways to accelerate their release cycles—driving the adoption of agile or lean development practices such as DevOps. Yet, based on the number of software failures now making headlines on a daily basis, it's evident that speeding up the SDLC opens the door to severe repercussions.

Organizations are remiss to assume that yesterday's practices can meet today's process demands. There needs to be a cultural shift from *testing an application* to *understanding the risks associated with a release candidate*. Such a shift requires moving beyond the traditional "bottom-up" approach to testing, which focuses on adding incremental tests for new functionality. While this will always be required, it's equally important to adopt a top-down approach to mitigating business risks. This means that organizations must defend the user experience with the most likely use cases in the context of non-functional requirements—continuously.

In order for more advanced automation to occur, we need to move beyond the test pass/fail percentage into a much more granular understanding of the impact of failure: a nuance that gets lost in the traditional regression test suite. Continuous Testing, which involves automatically executing a set of tests designed to verify whether business expectations for functionality, security, reliability, and performance are satisfied, is key for bridging this gap. It shifts the paradigm from a pure bottom-up approach to a hybrid model in which top-down testing is leveraged to protect the expected user experience from changes introduced as the application evolves. Ultimately, Continuous Testing resets the question from "are you done testing?" to "is the level of risk understood and accepted?"

2 Testing: The Elephant in the Room

As organizations begin to accelerate the SDLC, process bottlenecks will become evident. One of the bottlenecks that continues to plague SDLC acceleration is testing. At best, testing has been considered inevitable overhead—a "time-boxed" event that occurs sometime between code complete and the target release date. At worst, organizations have pushed quality processes to post-release, forcing a "real-time customer acceptance test."

Testing has always been the elephant in the room. Psychologically, the malleable nature of software has given organizations an excuse to defer investment in testing. However, this deferral results in technical debt. Over time, technical debt compounds, escalating the risk and complexity associated with each release.

Another obstacle to SDLC acceleration is the lack of a coordinated, end-to-end quality process. If trustworthy and holistic quality data were collected throughout the SDLC, then more automated decision points could optimize downstream processes.

3 Continuous Testing: How it Helps

Continuous Testing provides a real-time, objective assessment of the business risks associated with an application under development. Applied uniformly, it allows both business and technical managers to make better trade-off decisions between release scope, time, and quality.

Continuous Testing is NOT simply more automation. Rather, it is the reassessment of software quality practices—driven by an organization's cost of quality and balanced for speed and agility. Ultimately, Continuous Testing can provide a quantitative assessment of risk and produce actionable tasks that will help mitigate these risks before progressing to the next stage of the SDLC.

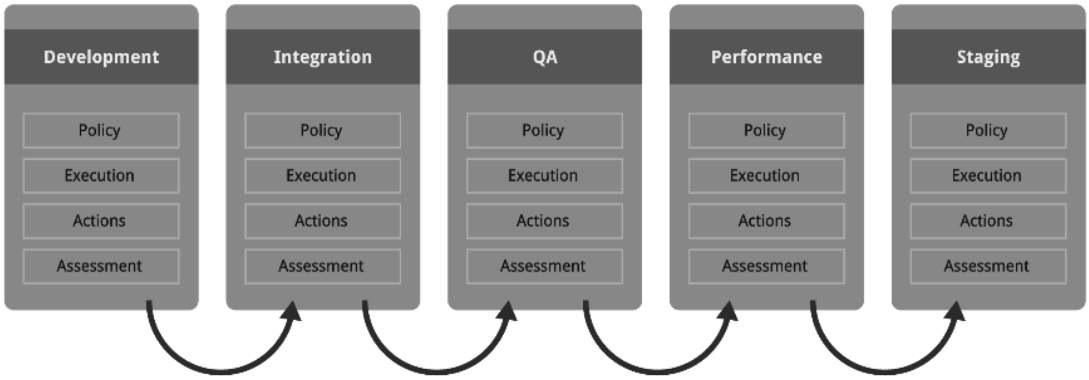


Figure 1: Mitigate risks before they progress to the next stage of the SDLC

For example, assume that a retailer knows that very specific user experience factors related to the application make the consumer more likely to add items to their e-commerce shopping cart. These metrics, which are defined in business language and measured automatically, are the exact same benchmarks used to accept a software release candidate. The question that the test suite should answer is how application modifications impact this set of business metrics. The goal is to bring the business expectations and business language to the forefront of the validation process—ultimately, protecting the user and the brand.

When it comes to software quality, we are confronting the need for true process re-engineering. Continuous Testing is not a "plug and play" solution. As with all process-driven initiatives, it requires the evolution of people, process, and technology. We must accommodate the creative nature of software development as a discipline, yet we must face the overwhelming fact that software permeates every aspect of the business—and software failure presents the single greatest risk to the organization.

4 What's the Business Value of Continuous Testing?

Given the business expectations at each stage of the SDLC, Continuous Testing delivers a quantitative assessment of risk as well as actionable tasks that help mitigate these risks before they progress to the next stage of the SDLC. The goal is to eliminate meaningless tests and produce value-added tasks that truly drive the development organization towards a successful release.

Continuous Testing—when executed correctly—delivers three major business benefits.

First, continuous tests are the artifacts that drive a central system of decision for the SDLC. Continuous tests place a bracer around core business functionality as expressed and implemented in software. The assessment of the validity or stability of these core business artifacts provides a real-time, quantifiable assessment of the application's health.

Second, Continuous Testing establishes a safety net that allows software developers to bring new features to market faster. With a trusted test suite ensuring the integrity of dependent application components and related functionality, developers can immediately assess the impact of code changes. This not only accelerates the rate of change, but also mitigates the risk of software defects reaching your customers.

Third, Continuous Testing allows managers to make better trade-off decisions. From the business' perspective, achieving a differentiable competitive advantage by being first to market with innovative software drives shareholder value. Yet, software development is a complex endeavor. As a result, managers are constantly faced with trade-off decisions between time, functionality, and quality. By providing a holistic understanding of the risk of release, Continuous Testing helps to optimize the business outcome. We'll cover this in greater detail in section 6.1 of this paper.

5 Re-Evaluating the Cost of Quality

A critical motivator for the evolution towards Continuous Testing is that business expectations about the speed and reliability of software releases have changed dramatically—largely because software has morphed from a business process enabler into a competitive differentiator.

For example, APIs represent componentized pieces of software functionality which developers can either consume or write themselves. In a recent Parasoft survey about API adoption, over 80% of respondents said that they have stopped using an API because it was "too buggy." Moreover, when we asked the same respondents if they would ever consider using that API again, 97% said "no." With switching costs associated with software like an API at an all-time low, software quality matters more than ever.

Another example is mobile check deposit applications. In 2011, top banks were racing to provide this must-have feature. By 2012, mobile check deposit became the leading driver for bank selection (Zhen, 2012). Getting a secure, reliable application to market was suddenly business critical. With low switching costs associated with online banking, financial institutions unable to innovate were threatened with customer defection.

This sea change associated with the quality expectations of software is also reflected in the manner in which software failures are reported. Today, software failures are highlighted in news headlines as organizational failings with deep-rooted impacts on C-level executives and stock prices. Parasoft analyzed the most notable software failures in 2012 and 2013; each incident initiated an average -3.35% decline in stock price, which equates to an average of negative $ 2.15 billion loss of market capitalization. This is a tremendous loss of shareholder value. Additionally, looking at organizations that endured multiple news-worthy software failures, it is clear that the market punishes repeat offenders even more aggressively. Repeat offenders suffered an average -5.68% decline in stock price, which equates to an average of negative $ 2.65 billion loss of market capitalization.[1]

The bottom line is that we must re-evaluate the cost of quality for our organizations and individual projects. If your cost of quality assessment exposes a gap in your quality process, it's a sign that now is the time to reassess your organization's culture as it relates to building and testing software. In most organizations, quality software is clearly the intention, yet the culture of the organization yields trade-off decisions that significantly increase the risk of exposing faulty software to the market.

6 What's Needed for Continuous Testing?

As you begin the transformation to Continuous Testing, the following elements are necessary for achieving a real-time assessment of business risks.

[1] From Parasoft equity analysis of the most notable software failures of 2012-2013.

Figure 2: Elements of Continuous Testing

6.1 Risk Assessment—Are You Ready to Release?

As we review the elements of Continuous Testing, it's hard to argue that one element is more important than the rest. If we present our case well enough, it should become obvious that each element is critical for overall process success. However, we need a place to start— and establishing a baseline to measure risk is the perfect place to begin as well as end.

One overarching aspect to risk assessment associated with software development is continuously overlooked: If software is the interface to your business, then developers writing and testing code are making business decisions on behalf of the business.

Assessing the project risk upfront should be the baseline by which we measure whether we are done testing and allow the SDLC to continue towards release. Furthermore, the risk assessment will also play an important role in improvement initiatives for subsequent development cycles.

The definition of risk cannot be generic. It must be relative to the business, the project, and potentially the iterations in scope for the release candidate. For example, a non-critical internal application would not face the same level of scrutiny as a publically-exposed application that manages financial or retail transactions. A company baseline policy for expectations around security, reliability, performance, maintainability, availability, legal, etc. is recommended as the minimum starting point for any development effort. However, each specific project team should augment the baseline requirement with additional policies to prevent threats that could be unique to the project team, application, or release.

SDLC acceleration requires automation. Automation requires machine-readable instructions which allow for the execution of prescribed actions (at a specific point in time). The more metadata that a team can provide around the application, components, requirements, and tasks associated with the release, the more rigorous downstream activities can be performed for defect prevention, test construction, test execution, and maintenance.

6.1.1 Technical Debt

Core to the management of SDLC risk is the identification of technical debt. A Development Testing Platform will help prevent and mitigate types of technical debt such as poorly-written code, overly-complex code, obsolete code, unused code, duplicate code, code not covered by automated tests, and incomplete code. The uniform measurement of technical debt is a great tool for project comparison and should be a core element of a senior manager's dashboard.

6.1.2 Risk Mitigation Tasks

All quality tasks requested of development should be 100% correlated to an opportunity to minimize risk or technical debt and related to a defect prevention policy (discussed in section 6.2). Policies can be formed around any aspect of the development process. For example, policies might cover non-functional requirements related to security, reliability, performance, compliance, etc. To be effective, policies must be definable, enforceable, measureable and auditable.

A developer has two primary jobs: implement business requirements and reduce the business risk associated with application failure. From a quality and testing perspective, it is crucial to realize that quality initiatives generally fail when the benefits associated with a testing task are not clearly understood.

A risk mitigation task can range from executing a peer code review to constructing or maintaining a component test. Whether a risk mitigation task is generated manually at the request of a manager or automatically (as with static code analysis), it must present a development or testing activity that is clearly correlated with the reduction of risk.

6.1.3 Coverage Optimization

Coverage is always a contentious topic—and, at times, a religious war. Different coverage techniques are better-suited for different risk mitigation goals. Fortunately, industry compliance guidelines are available to help you determine which coverage metric or technique to select and standardize around.

Once a coverage technique (line, statement, function, modified condition, decision, path, etc.) is selected and correlated to a testing practice, the Development Testing Platform will generate reports as well as tasks that guide the developer or tester to optimize coverage. The trick with this analysis is to optimize versus two goals. First, if there is a non-negotiable industry standard, optimize based on what's needed for compliance. Second (and orthogonal to the first), optimize on what's needed to reduce business risks.

Coverage analysis is tricky because it is not guaranteed to yield better quality. Yet, coverage analysis can certainly help you make prioritization decisions associated with test resource allocation.

6.1.4 Test Quality Assessment

Processes and test suites have one thing in common: over time, they grow in size and complexity until they reach a breaking point when they are deemed "unmanageable." Unfortunately, test suite rationalization is traditionally managed as a batch process between releases. Managing in this manner yields to sub-optimal decisions because the team is forced to wrangle with requirements, functions, or code when the critical details are no longer fresh in their minds.

Continuous Testing requires reliable, trustworthy tests. When test suite results become questionable, there is a rapid decline in how and when team members react. This leads to the test suite becoming out-of-sync with the code—and ultimately out of control.

With this in mind, it is just as important to assess the quality of the test itself as it is to respond to a failing test. Automating the assessment of the test is critical for Continuous Testing. Tests lie at the core of software risk assessment. If these risk monitors are not reliable, then we must consider the process to be out of control.

6.2 Policy Analysis—Keep up with Evolving Business Demands

Policy analysis through a Development Testing Platform is key for driving development and testing process outcomes. The primary goal of process analysis to ensure that policies are meeting the organization's evolving business and compliance demands.

Most organizations have a development or SDLC policy that is passive and reactive. This policy might be referenced when a new hire is brought onboard or when some drastic incident compels management to consult, update, and train on the policy. The reactive nature of how management expectations are expressed and measured poses a significant business risk. The lack of a coordinated governance mechanism also severely hampers IT productivity (since you can't improve what you can't measure).

Policy analysis through a Development Testing Platform is the solution to this pervasive issue. With a central interface where a manager or group lead defines and implements “how,” “when,” and “why” quality practices are implemented and enforced, management can adapt the process to evolving market conditions, changing regulatory environments, or customer demands. The result: management goals and expectations are translated into executable and monitor-able actions. One such example of a Development Testing Platform is the Parasoft Development Testing Platform, which monitors policies using a number of software testing practices (e.g., static analysis, unit testing, coverage analysis, runtime error detection, etc.) across distributed teams and throughout the SDLC.

Policy analysis is the measurement of the policy itself. For example, if a particular security policy such as "validate every input as soon as it is received" is enforced via static analysis, measuring that policy might involve considering:

- How is the static analysis rule violated?
- How many times is it violated?
- Is there quantifiable evidence that following the related coding standard actually helps the application meet business expectations associated with security?

Ultimately, policy analysis is part of an infrastructure that promotes continuous process improvement.

The primary business objectives of policy analysis are:

- Expose trends associated with the injection of dangerous patterns in the code
- Target areas where risks can be isolated within a stage
- Identify higher risk activities where defect prevention practices need to be augmented or applied

With effective policy analysis, “policy” is no longer relegated to being a reactive measure that documents what is assumed to occur; it is promoted to being the primary driver for risk mitigation.

As IT deliverables increasingly serve as the “face” of the business, the inherent risks associated with application failure expose the organization to severe financial repercussions. Furthermore, business stakeholders are demanding increased visibility into corporate governance mechanisms. This means that merely documenting policies and processes is no longer sufficient; we must also demonstrate that policies are actually executed in practice.

This centralization of management expectations not only establishes the reference point needed to analyze risk, but also provides the control required to continuously improve the process of delivering software.

6.3 Requirements Traceability—Defending the Business Objective

All tests should be correlated with a business requirement. This provides an objective assessment of which requirements are working as expected, which require validation, and which are at risk. This is tricky because the articulation of a requirement, the generation or validation of code, and the generation

of a test that validates its proper implementation all require human interaction. We must have ways to ensure that the artifacts are aligned with the true business objective—and this requires human review and endorsement. Continuous Testing must promote the validation of testing artifacts via peer review.

A Development Testing Platform helps the organization keep business expectations in check by ensuring that there are *effective* tests aligned to the business requirement. By allowing extended metadata to be associated with a requirement, an application, a component, or iteration, the Development Testing Platform will also optimize the prioritization of tasks.

During "change time," continuous tests are what trigger alerts to the project team about changes that impact business requirements, test suites, and peripheral application components. In addition to satisfying compliance mandates, such as safety-critical, automotive, or medical device standards, real-time visibility into the quality status of each requirement helps to prevent late-cycle surprises that threaten to derail schedules and/or place approval in jeopardy.

6.4 Advanced Analysis—Expose Application Risks Early

6.4.1 Defect Prevention with Static Analysis

It's well known that the later in the development process a defect is found, the more difficult, costly, and time-consuming it is to remove. Mature static analysis technologies, managed in context of defined business objectives, will significantly improve software quality by preventing defects early.

Writing code without static code analysis is like writing a term paper or producing a report without spell check or grammar check. A surprising number of high-risk software defects are 100% preventable via fully-automated static code analysis. By preventing defects from being introduced in the first place, you minimize the number of interruptions and delays caused by the team having to diagnose and repair errors. Moreover, the more defects you prevent, the lower your risk of defects slipping through your testing procedures and making their way to the end-user—and requiring a significant amount of resources for defect reproduction, defect remediation, re-testing, and releasing the updated application. *Ultimately, automated defect prevention practices increase velocity, allowing the team to accomplish more within an iteration.*

At a more technical level, this automated analysis for defect prevention can involve a number of technologies, including multivariate analysis that exposes malicious patterns in the code, areas of high risk, and/or areas more vulnerable to risk. All are driven by a policy that defines how code should be written and tested to satisfy the organization's expectations in terms of security, reliability, performance, and compliance. The findings from this analysis establish a baseline that can be used as a basis for continuous improvement.

Pure "defect prevention" approaches can eliminate defects that result in crashes, deadlocks, erratic behavior, and performance degradation. A security-focused approach can apply the same preventative strategy to security vulnerabilities, preventing input-based attacks, backdoor vulnerabilities, weak security controls, exposure of sensitive data, and more.

6.4.2 Change Impact Analysis

It is well known that defects are more likely to be introduced when modifying code associated with older, more complex code bases. In fact, a recent FDA study of medical device recalls found that an astonishing "192 (or 79%) [of software-related recalls] were caused by software defects that were introduced when changes were made to the software after its initial production and distribution"(FDA, 2002).

From a risk perspective, changed code equates to risky code. We know that when code changes, there are distinct impacts from a testing perspective:

- Do I need to modify or eliminate the old test?

- Do I need a new test?
- How have changes impacted other aspects of the application?

The goal is to have a single view of the change impacts from the perspective of the project as well as the perspective of the individual contributor. Optimally, change impact analysis is performed as close to the time of change as possible—when the code and associated requirements are still fresh in the developer's or tester's mind.

If test assets are not aligned with the actual business requirements, then Continuous Testing will quickly become unmanageable. Teams will need to spend considerable time sorting through reported failures—or worse, overlook defects that would have been exposed by a more accurate test construction.

Now that development processes are increasingly iterative (more agile), keeping automated tests and associated test environments in sync with continuously-evolving system dependencies can consume considerable resources. To mitigate this challenge, it's helpful to have a fast, easy, and accurate way of updating test assets. This requires methods to assess how change impacts existing artifacts as well as a means to quickly update those artifacts to reflect the current business requirements.

6.4.3 Scope and Prioritization

Given a software project's scope, iteration, or release, some tests are certainly more valuable and timely than others. Advanced analysis techniques can help teams identify untested requirements, tasks, and code. Advanced analysis should also deliver a prioritized list of regression tests that need review or maintenance.

Leveraging this type of analysis and acting on the prioritized test creation or maintenance tasks can effectively prevent defects from propagating to downstream processes, where defect detection is more difficult and expensive. There are two main drivers for the delivery of tasks here: the boundaries for scope and the policy that defines the business risks associated with the application.

For example, the team might be working on a composite application in which one component is designed to collect and process payment cards for online transactions. The cost of quality associated with this component can be colossal if the organization has a security breach or fails a PCI DSS[2] audit. Although code within the online transaction component might not be changing, test metadata associated with the component could place it in scope for testing. Furthermore, a policy defined for the PCI DSS standard (as well as the organization's internal data privacy and security) will drive the scope of testing practices associated with this release or iteration.

6.5 Test Optimization—Ensure Findings are Accurate and Actionable

To truly accelerate the SDLC, we have to look at testing much differently. In most industries, modern quality processes are focused on optimizing the process with the goal of preventing defects or containing defects within a specific stage. With software development, we have shied away from this approach, declaring that it would impede engineering creativity or that the benefits associated with the activity are low, given the value of the engineering resources. With a reassessment of the true cost of software quality, many organizations will have to make major cultural changes to combat the higher penalties for faulty software. Older, more established organizations will also need to keep up with the new breed of businesses that were conceived with software as their core competency. These businesses are free from older cultural paradigms that might preclude more modern software quality processes and testing practices.

No matter what methodology is the best fit for your business objectives and desired development culture, a process to drive consistency is required for long-term success.

[2] PCI DSS is the Payment Card Industry Data Security Standard

Test optimization algorithms help you determine what tests you absolutely *must* run versus what tests are of lower priority given the scope of change. Ideally, you want intelligent guidance on the most efficient way to mitigate the greatest risks associated with your application. Test optimization not only ensures that the test suite is validating the correct application behavior, but also assesses each test itself for effectiveness and maintainability.

6.5.1 Management

Test optimization management requires that a uniform workflow is established and maintained associated with the policies defined at the beginning of a project or iteration. A Development Testing Platform must provide the granular management of queues combined with task workflow and measurement of compliance. To achieve this:

- The scope of prescribed tasks should be measurable at different levels of granularity, including individual, team, iteration, and project.
- The test execution queues should allow for the prioritization of test runs based on the severity and business risk associated with requirements.
- Task queues should be visible and prioritized with the option to manually alter or prioritize (this should be the exception, not the norm).
- Reports on aged tasks should be available for managers to help them determine whether the process is under control or out of control.

6.5.2 Construction

With a fragile test suite, Continuous Testing just isn't feasible. If you truly want to automate the execution of a broad test suite—embracing unit, component, integration, functional, performance, and security testing—you need to ensure that your test suite is up to the task. How do you achieve this? Ensure that your tests are…

- **Logically-componentized**: Tests need to be logically-componentized so you can assess the impact at change time. When tests fail and they're logically correlated to components, it is much easier to establish priority and associate tasks to the correct resource.
- **Incremental**: Tests can be built upon each other, without impacting the integrity of the original or new test case.
- **Repeatable:** Tests can be executed over and over again with each incremental build, integration, or release process.
- **Deterministic and meaningful**: Tests must be clean and deterministic. Pass and fail have unambiguous meanings. Each test should do exactly what you want it to do—no more and no less. Tests should fail only when an actual problem you care about has been detected. Moreover, the failure should be obvious and clearly communicate what went wrong.
- **Maintainable within a process:** A test that's out of sync with the code will either generate incorrect failures (false positives) or overlook real problems (false negatives). An automated process for evolving test artifacts is just as important as the construction of new tests.
- **Prescriptive workflow based on results:** When a test does fail, it should trigger a process-driven workflow that lets team members know what's expected and how to proceed. This typically includes a prioritized task list.

6.5.3 Test Data Management

Access to realistic test data can significantly increase the effectiveness of a test suite. Good test data and test data management practices will increase coverage as well as drive more accurate results. However, developing or accessing test data can be a considerable challenge—in terms of time, effort, and compliance. Copying production data can be risky (and potentially illegal). Asking database administrators to provide the necessary data is typically fraught with delays. Moreover, delegating this

task to dev/QA moves team members beyond their core competencies, potentially delaying other aspects of the project for what might be imprecise or incomplete results.

Thus, fast and easy access to realistic test data removes a significant roadblock. The primary methods to derive test data are:

- Sub-set or copy data from a production database into a staged environment and employ cleansing techniques to eliminate data privacy or security risks.
- Leverage Service Virtualization (discussed later in this resource) to capture request and response traffic and reuse the data for subsequent scenarios. Depending on the origin and condition of the data, cleansing techniques might be required.
- Generate test data synthetically for various scenarios that are required for testing.

In all cases, it's critical to ensure that the data can be reused and shared across multiple teams, projects, versions, and releases. Reuse of “safe” test data can significantly increase the speed of test construction, management, and maintenance.

6.5.4 Maintenance

All too often, we find development teams carving out time between releases in order to "clean-up" the test suites. This ad-hoc task is usually a low priority and gets deferred by high-urgency customer feature requests, field defects, and other business imperatives. The resulting lack of ongoing maintenance typically ends up eroding the team’s confidence in the test suite and spawning a backlog of increasingly-complex maintenance decisions.

Test maintenance should be performed as soon as possible after a new business requirement is implemented (or, in the case of TDD-like methodologies, prior to a requirement being implemented). The challenge is to achieve the optimal balance between creating and maintaining test suites versus the scope of change.

Out-of-sync test suites enter into a vicious downward spiral that accelerates with time. Unit, component, and integration tests that are maintained by developers are traditionally the artifacts at greatest risk of deterioration. Advanced analysis of the test artifact itself should guide developers to maintain the test suite. There are five primary activities for maintenance—all of which are driven by the business requirement:

- Delete the test
- Update the test
- Update the assertions
- Update the test data
- Update the test metadata

6.6 Service Virtualization—Eliminate Test Environment Access Issues

With the convergent trends of parallel development and increasing system complexity/interdependency, it has become extremely rare for a team to have ubiquitous access to all of the dependent applications required to execute a complete test. By leveraging Service Virtualization to remove these constraints, an organization can gain full access to (and control over) the test environment—enabling Continuous Testing to occur as early and often as needed.

Want to start testing the component you just built even though not much else is completed? Don't have 24/7 access to all the dependencies involved in your testing efforts—with all the configurations you need to feel confident that your test results are truly predictive of real-world behavior? Tired of delaying

performance testing because access to a realistic environment is too limited (or too expensive)? Service Virtualization can remove all these constraints.

With Service Virtualization, organizations can access simulated test environments that allow developers, QA, and performance testers to test earlier, faster, and more completely. Organizations that rely on interconnected systems must be able to validate system changes more effectively—not only for performance and reliability, but also to reduce risks associated with security, privacy, and business interruption. Service Virtualization is the missing link that allows organizations to continuously test and validate business requirements. Ultimately, Service Virtualization brings higher quality functionality to the market faster and at a lower cost.

7 Conclusion

Admittedly, moving from automated testing to the level of business-driven Continuous Testing outlined above is a big leap. But it is one that yields significant benefits from a DevOps perspective. Through objective real-time validation of whether software meets business expectations at various "quality gates," organizations can automatically promote software from one phase of the SDLC to the next.

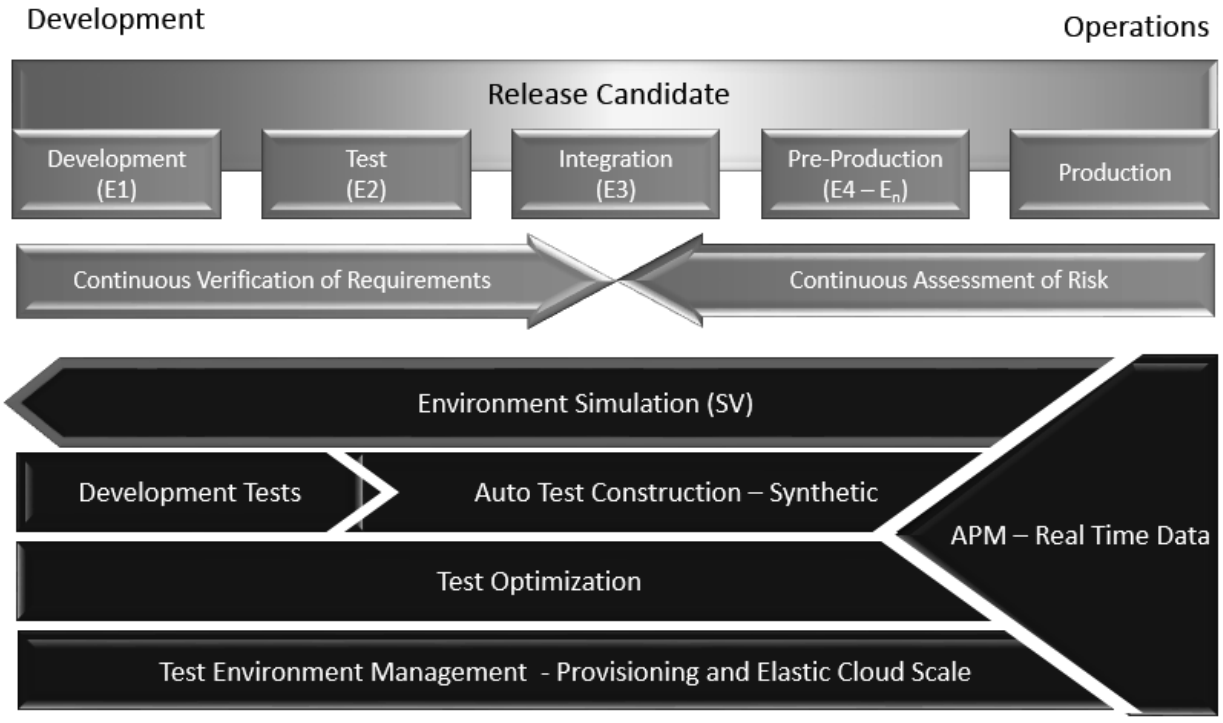


Figure 3: The Next Generation Software Quality 'System'

The benefits of this automated assessment include:

- Throughout the process, business stakeholders have instant access to feedback on whether their expectations are being met, enabling them to make informed decisions.
- At the time of the critical "go/no go" decision, there is an instant, objective assessment of whether your organization's specific expectations are satisfied—reducing the business risk of a fully-automated Continuous Delivery process.
- Defects are eliminated at the point when they are easiest, fastest, and least costly to fix—a prime principle of being "lean."

- Continuous measurement vs. key metrics means continuous feedback, which can be shared and used to refine the process.

References

FDA. "General Principles of Software Validation; Final Guidance for Industry and FDA Staff." http://www.fda.gov/RegulatoryInformation/Guidances/ucm085281.htm (accessed June 8, 2015).

Zhen, Simon. "Take it to the Bank: Mobile Check Deposit is the Next-Gen Standard," MyBankTracker, entry posted March 13, 2012, http://www.mybanktracker.com/news/2012/03/13/bank-mobile-check-deposit-next-gen-standard/ (accessed June 8, 2015).

Version Control Your Jenkins Jobs with Jenkins Job Builder

Wayne Warren

wayne@puppetlabs.com

Abstract

Puppet Labs uses Jenkins to automate building and testing software. While we do derive benefit from this automation, there is a maintenance burden involved in keeping Jenkins configured correctly. The Jenkins Web UI can be tedious to use, even when managing a small number of jobs (Jenkins' unit of configurable work). The complexity of managing the system continues to increase as the number of projects, versions, and developers increases.

To address this burden, Puppet Labs has invested significant engineering time and effort into understanding, improving, and applying Jenkins Job Builder (JJB) to manage job configurations. JJB is a Python tool developed by the Openstack Infra developers whose features include: modular support for Jenkins plugins, support for parameterized job templates, and deletion / creation of managed jobs. Using it has helped mitigate configuration drift between jobs and pipelines, identify and decommission forgotten jobs, and establish a review process for Continuous Integration (CI) changes.

This paper explains how to use JJB to manage and version control Jenkins configuration. Readers should expect to walk away with an understanding of the following key concepts:

- Generic job/pipeline types (unit, packaging, component integration, system integration)
- Language and framework specific implementations of generic job/pipeline types
- Job template name guidelines for optimal reuse
- Configuration testing and deployment strategies

I will conclude with a brief preview of the future direction of Jenkins Job Builder development.

Biography

Wayne is a Quality Engineer dedicated to improving and streamlining Puppet Labs' approach to automating common test and build tasks related to the validation and release of Puppet Enterprise and its component projects. He has experience with embedded Linux kernels and file systems, developing functional test frameworks for embedded systems, and automating Jenkins configuration using JJB.

Copyright Wayne Warren 2015

1 Introduction

Jenkins is a continuous integration (CI) tool that automates various build, test, and packaging steps. Because managing configuration for Jenkins jobs can be tedious, repetitive, and error-prone when using the Jenkins Web UI due to its click-heavy interface.

What might we look for in an alternative system for managing Jenkins job configuration? At a high level, an alternative must include changes that can be version controlled, tested out-of-band, and reviewed by teammates before being deployed. We should also be able to optimize configuration reuse between projects with similar CI needs as well as between Jenkins instances since we run multiple public and private Jenkins instances.

Jenkins Job Builder (JJB) is a set of Python scripts created and maintained by the OpenStack Infra developers. It takes an arbitrary number of YAML files that are processed according to Domain Specific Language (DSL) rules to produce a list of Jenkins job dictionaries. These dictionaries are used to generate XML in a modular fashion for the Jenkins job they represent. Modularity, in this context, refers to the way in which XML is generated on a per-plugin basis and makes it possible to add support for additional Jenkins plugins. Once all dictionaries have been converted to XML, JJB then uses Jenkins' HTTP API to POST these configurations to the specified Jenkins server to either create new jobs or update existing ones.

This paper aims to describe the JJB DSL, propose a review process, and introduce conventions for optimizing reuse of job configuration.

2 Background

2.1 Continuous Integration

Continuous Integration in software development refers to the practice of continuously merging feature code from developer branches into a designated “mainline” branch of a software project. A continuous integration service is one that provides automation to facilitate continuous integration by building and testing whenever a commit is merged into a software project. This automation ensures that builds and tests are run in a uniform environment at a consistent cadence that allows developers to discover regressions before software is shipped to users.

2.2 Jenkins

Jenkins is an open source, continuous integration service with a master/slave architecture. The “master” is used to manage configuration and queuing of jobs and “slaves” are used to execute the actionable content of those jobs. A “job” is a reusable, configurable unit of work that can be scheduled to run using different types of “triggers”. Jobs can be configured to perform various types of tasks including shell scripts, rake tasks, maven tasks, or many other build/test/integrate-oriented snippets of automatable behavior.

Jenkins provides various types of configuration to modify its behavior:

- **global**
  system configuration that manages the overall behavior of the Jenkins master
- **security**
  security-sensitive configuration for the Jenkins master
- **view**
  all configuration specific to a particular Jenkins view
- **job**
  all configuration specific to a particular job

Jenkins “core” is a daemon providing pluggable interfaces that can be used by auxiliary software (aka “Plugins”) to implement additional behavior for either the service (master and slaves) as a whole, or more commonly for individual jobs.

Jenkins is primarily designed to be configured through its web UI, which is a manual process requiring navigation of web forms. It also provides a command-line tool and an HTTP API. As of Jenkins 1.615, the HTTP API has documented support for configuration of **jobs** and **views**. Jenkins also has endpoints that accept JSON for **global** and **security** configuration that are used by its web UI, but there is no documented support for these endpoints to be used programmatically.

2.2.1 Problems with the Jenkins Web UI

Users typically configure Jenkins for the sake of modifying specific job behavior. This can include job timeout, log retention, Jenkins slave pinning, task execution, setting relationships between jobs, software configuration management (SCM) behavior, and many other types of behavior. These changes may need to be made for different reasons at different stages in an organization's growth or in a specific project's lifecycle. Because of this, the scope of the changes being made may not be limited to a single job—they may be a matter of policy or of differentiation between new and previous development cycles. It may be that an entire “pipeline” of jobs may need to be duplicated and modified to suit a slightly different need.

Whatever the reason for configuring Jenkins jobs, its Web UI is considered by many to be outdated and difficult to navigate. Problems we have found include:

- Difficulty navigating configuration web elements when attempting to investigate and modify a single job (let alone many jobs that constitute a pipeline of test, build, and packaging steps).
- Configuration drift between pipelines for projects with very similar build and testing requirements is inevitable since each job must be modified individually.
- Changes made by individual team members may lead to configuration or automation policy violations since the rest of the team does not have the opportunity to review and approve of changes before they are made.

2.2.2 Seeking Alternatives

So what might we look for in an alternative system for managing Jenkins job configuration? At a high level, there are a number of workflow and technical requirements that needed to be met to reduce the pain of interacting with Jenkins.

Workflow

- Must be able to version control our Jenkins configuration.
- Testing of configuration must be able to happen without affecting our production systems.
- Changes must be reviewed and approved by our team the same way that developers review code before merging.

Technical

- Must be able to optimize configuration reuse between projects with similar CI needs.
- Should be able to share configuration between Jenkins instances.

3 Jenkins Job Builder

After evaluating several alternatives side-by-side, comparing the pros and cons of each, we found that Jenkins Job Builder (JJB) met our requirements without departing from the workflows our developer and QA teams were accustomed to.

JJB is a command-line tool designed to simplify Jenkins interactions by enabling users to configure **job**s using human editable YAML. The sequence of events for updating jobs with JJB is roughly as follows:

1. Read all specified YAML files in as Python dictionaries.
2. Apply JJB DSL rules to the given dictionaries:
    1. Expand job templates
    2. Apply defaults to all jobs
    3. Interpolate variables specified at the project, job-group, and job-template levels.
3. For each resulting job dictionary:
    1. Interpolate variables and expand macros in-place
    2. Generate XML using JJB's Jenkins plugin modules
4. POST each XML string to the Jenkins HTTP API to either create a new job or update an existing job.

The key feature that makes this a viable alternative to the Jenkins Web UI is the use of YAML to store job configurations. The YAML can be placed under version control, allowing it to be reviewed using, for example, the Github Pull Request (PR) model of code review. The use of Jenkins' HTTP API to perform the configuration update means relatively few changes were necessary to our Jenkins deployments to implement this new approach to configuring jobs. It also means that configurations can be easily shared between Jenkins instances since deploying to a different instance only requires a slightly different configuration be passed to JJB. Finally, the hierarchical organization of jobs using JJB's “job-group” and “project” concepts, in combination with the “job-template” concept, means that configurations can be reused between jobs with similar CI requirements.

JJB does not currently support **global**, **security**, or **view** configurations. The following sections explore some of JJB's features in more depth.

3.1 Modular XML Generation

JJB's most prominent feature is its modular approach to XML generation in which support for each Jenkins Plugin is implemented as a function that generates the necessary XML snippet. Such XML snippets are inserted in the appropriate place in each job's XML configuration structure. Supporting additional plugins involves the following steps:

1. Add a function to the appropriate Python module in JJB's source; the module corresponds to the type of behavior (ie, “builder”, “publisher”, “trigger”) this plugin supports and the function corresponds to the plugin itself.
2. List that function in JJB's setup.cfg entry points. This enables the XML generating code to know about it at runtime.
3. Document, via doc strings on the function, the dictionary keys necessary to configure that particular plugin.

3.2 Plugin Version Detection

Complementing JJB's modular XML generation is its use of the Jenkins HTTP API to query the targeted Jenkins instance for a list of installed plugins and their versions. This allows JJB modules to conditionally generate XML structures based on the versions of plugins installed on the target Jenkins instance.

3.3 The JJB DSL

The JJB DSL provides the ability to reuse configuration snippets between jobs and pipelines. It provides CI configuration abstractions that allow projects and their jobs to be parameterized, hierarchically organized, and editable in any plaintext editor.

- **project**
  JJB “project” is the top-level object that instantiates all job templates and groups.
- **job-template**
  JJB “job-template” is a YAML dictionary (aka “hash” or “map” depending on the programming language) containing keys that map to Jenkins plugins and values that may include template strings.
- **job-group**
  JJB “job-group” is similar to a “project” except that it is reusable between different projects
- **defaults**
  JJB “defaults” is where default JJB variables can be set.
- **<macro>**
  Macros are top-level JJB elements which can be referenced within specific sub-elements of a job-template.

3.3.1 Templates

JJB utilizes the concept of “job-template” to define parameterized jobs that can be reified at run time with variables passed to them by either the “job-group” or “project” where they are used. The “job-group” and “project” concepts are DSL elements which provide the ability to group lists of jobs and job templates, and to override JJB variables used by job templates.

3.3.2 Macros

Macros are snippets of JJB code that get expanded in place where used in “job-template” and “job” objects. Like job-templates, these provide a DRY (Don't Repeat Yourself) element to job configuration, reducing the number of places where changes need to be made when modifying the behavior of many jobs at once (for example, setting build log retention policy).

3.3.3 Defaults

JJB defaults are a way of setting default values for variables that are interpolated into job templates at run time. Having a single place to set these values is important because it reduces the need to unnecessarily set default values on each job-template. Defaults have the lowest precedence when variable overrides occur between defaults, job-group, project, and job-template items in the JJB DSL.

3.4 Job Deletion

It is possible to use shell glob syntax to delete jobs managed by JJB from the command line. Combined with a thoughtful approach to naming job-templates this can be used to target pipelines specific to projects, branches of projects, or even categories of jobs.

3.5 Test Output

The JJB command line tool provides a flag that will cause it to run in “test” mode (as opposed to “update” mode). In test mode, JJB will either:

- emit all of its generated XML to stdout
- or given a directory name, create one file per job generated whose contents is the XML that would be used to update that job in update mode and whose name is the name of the job as it is referred to in Jenkins itself.

This test output mode is crucial to both development of JJB itself, where we use the output to ensure consistent output between one revision of JJB and the next for OpenStack's project configs, and for reporting changes between one revision and the next of an organization's Jenkins configs.

4 A Version-Controlled JJB Workflow

This section assumes some familiarity with general version control system (VCS) concepts such as “branches” and “commits”. Examples use the Git VCS, in particular the pull request model of review and code merge used by Github.

The basic idea is that since JJB enables jobs to be defined using YAML rather than XML, these files can be edited by humans and stored in a VCS repository in the same way that developers version control source code for software projects. The primary benefit of version controlling Jenkins Jobs in this way is that a team can then automatically synchronize live Jenkins jobs with their VCS copy--thus mitigating the potential for configuration drift between similar jobs edited asynchronously through the Jenkins web UI. If a team member wishes to make a permanent change to a particular set of jobs, they must follow the same process the team uses for making changes to source code for a given project.

4.1 Pull Requests (PR)

Pull Requests are a Github-specific development workflow item in which a user “submits a PR” containing their intended changes as a series of commits against the main line of development of a project. This PR workflow alerts other developers working on the same project that there is a change waiting to be merged and allows them to view comment on patches against various files in the PR. Github also provides an API by which external software may react against particular events on repositories of interest. For example, Jenkins can listen using this API for PRs against a repository and trigger CI jobs to run tests before changes are merged. This general PR workflow is exactly what Puppet Labs uses to allow our CI team to review changes before merging.

4.2 Testing

Puppet Labs stores its JJB configuration in a single monolithic git repository. Whenever someone makes a change to this repo, they submit a Github PR which triggers a Jenkins job that tests this PR and reports back to the PR with a SUCCESS/FAILURE message and a link back to the Jenkins job that ran. The only test that we currently use to validate our changes performs the following steps:

For each Jenkins instance managed by the config repo:

- Generate Jenkins job config XML from the master branch, output to a local directory using JJB’s “test” subcommand.
- Generate Jenkins job config XML from the PR branch, output to a local directory using JJB’s “test” subcommand.
- Use the “diff” command line tool to recursively compare the two output directories, save this information in a file named *<jenkins-instance-name>.diff*
- Archive as artifacts on the jenkins jobs all files that match the pattern “*.diff”.

These “diff” output files allow us to compare the differences in XML generated between revisions of the configuration repo as well as to detect when jobs are deleted or created.

4.3 Deployment

As with any version-controlled software, the end goal of JJB configuration changes is to be merged into the mainline branch of development. Once this happens in Puppet Labs' JJB configuration repository we have automated deploy jobs in Jenkins that are triggered to push changes out to all affected JJB-managed Jenkins instances ensuring that new jobs are created and existing jobs are updated to reflect the new state of the JJB config master branch.

In addition to this per-merge deploy, we also run JJB twice-daily to ensure a high degree of consistency between merges. This allows developers to make changes to jobs asynchronously during the day to troubleshoot or debug CI prior to proposing more permanent changes with additional PRs against the JJB configuration repo.

4.4 Benefits

So the theory seems sound, but how does it hold up in the real world? JJB's templating system and ability to group jobs together has led to an increase in individual CI Automation engineer productivity by allowing CI pipelines to be designed once and deployed many times. The initial design time for a pipeline tends to take a bit longer than if everything were done using the Jenkins Web UI, but once that pipeline template exists it is fairly trivial to deploy a copy of that template for new projects as long as those projects follow the conventions expected by the CI jobs.

5 Related Work

5.1 Job DSL & Workflow Plugins

The Job DSL Plugin makes available a new type of build step when configuring jobs. This build step allows the creation, configuration, and triggering of other jobs—all using the Groovy scripting language. This essentially enables the use of “meta” style jobs that manage the workflow of an entire “pipeline” of jobs. While this did meet some of our technical criteria—we could easily reproduce groups of jobs for different branches or projects—it didn't meet our workflow needs. Specifically, this plugin does not allow us to version control our Jenkins configuration or review changes made by teammates before those changes go into production.

The Workflow Plugin is similar to the DSL Plugin in that it provides a sort of “meta” facility of organizing jobs in a pipeline. It also uses Groovy to describe this organization. However, it suffers from the same drawbacks as the DSL Plugin, namely that does not allow version control or review of changes made by teammates before those changes go into production.

It's also worth mentioning that neither of these plugins takes us outside the Jenkins Web UI even if they do make interactions with the web ui more productive.

5.2 Templated XML Files

Prior to investigating existing alternatives to managing jobs purely through the Jenkins we had for some of our CI pipelines implemented a series of templated XML files that we could interpolate with variables and POST to the Jenkins HTTP API. This allowed us to relatively easily reproduce this same pipeline exactly for different projects. It also allowed job configuration to be kept in version control.

However, it lacked generalizable templates that could be reused for different purposes or within the context of different types of CI pipelines. It also moved the complexity of interacting with the Jenkins Web UI into the complexity of dealing with templated XML.

6 Conclusion

Jenkins is an automation service that improves the quality of continuous integration work flows for software projects by allowing them to automate testing, building, packaging, and in some cases deploying changes as they are merged into version control repositories. However, the Jenkins user experience and scalability to more than a few simple pipeline setups suffers from its unintuitive and tedious web interface.

Ideally, developers working on new software projects should not have to expend significant time or energy understanding the complex Jenkins machinery to set up CI when other projects have similar test/build/packaging needs—it should be a simple matter of naming the pipeline type, submitting a PR to

a configuration repo, and obtaining approval from teammates with more Jenkins knowledge. Additionally, when all that is necessary is to make a simple change to an existing pipeline, the behavior affected by that change should be represented by a well-documented set of parameters that can be changed without that developer having to understand all the workings under the hood.

Jenkins Job Builder allows us to achieve the necessary level of abstraction to consistently reproduce and maintain CI pipelines for specific project types. We also avoid the problems of configuration drift, unapproved changes, and the tedium of managing all our jobs with the Jenkins Web UI. Many of these benefits follow directly from being able to version control our configuration templates using the Git source control management tool.

7 Future Work

The JJB version 1.x and earlier library is only really intended to be used for the JJB command line tool. However, there is an effort within the Openstack Infra developer group to clean up the library with the intent of producing a programmatic API to allow JJB users to produce their own tools as well as to configure Jenkins directly using Python if they wish. This is the main development effort for JJB targeting the 2.x version series.

Our Road to Continuous Delivery @ Tango

Amit Mathur

amit@tango.me

Abstract

Tango is a free mobile communications service with 300+ million registered members available on iPhone, iPad, Android phones and tablets. Tango's large user base, broad platform support, frequent releases, and organizational design pose challenges to achieving quality.

As a follow on from last year, this paper discusses specifically the investments that we made in test and deployment automation to enable the transformation from a traditional software delivery model to a continuous delivery model. This new model has enabled us to continue to have agility despite a rapid growth of customers, employees, and functionality.

Continuously releasing software may sound great in theory but is riddled with challenges. This paper will help you learn the tools and techniques we have used during this evolution as well as lessons we learned along the way.

Biography

Amit has over 15 years of experience in software quality assurance as a tester, test architect, test manager, and Director of Technology at Motorola, Veritas, Symantec, Microsoft, and now Tango. He has presented in numerous testing conferences including CAST, Star West, and PNSQC.

Amit holds a B.S. in Computer Science from the University of Illinois at Urbana Champaign.

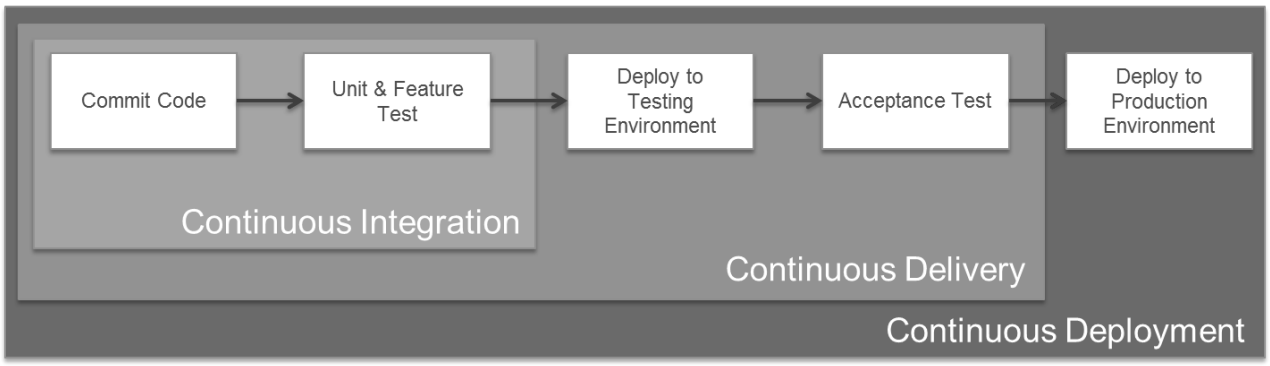
Copyright Amit Mathur 2015

1 Introduction

Mobile communications is hot and Tango is at the center of the mobile social communications race. It has been described as more of a sprint than a marathon. This has led us at Tango to converge towards a monthly release cycle.

As if releasing every month isn't challenging enough, we also try hard to give engineering teams a lot of autonomy to get their features out which poses interesting challenges when we attempt to integrate all the pieces together.

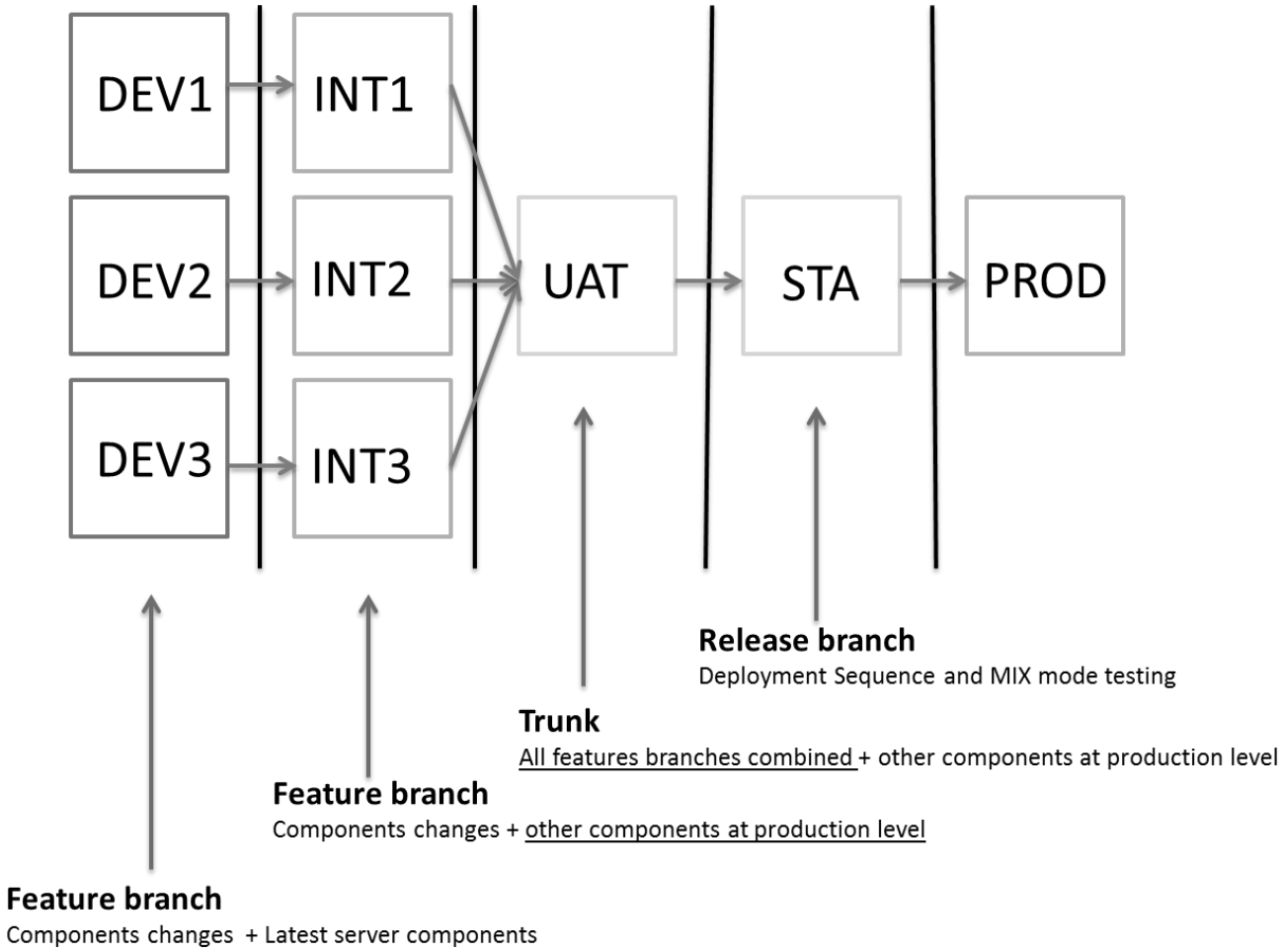
2 Continuous Delivery



Tango has enjoyed the benefits of continuous integration (automated unit and feature tests) after commits for many years. To continue to increase our velocity we shifted focus to automating our deployment to our integration testing environments and automatically executing acceptance tests.

Our next step will be Continuous Deployment all the way to production! It has been an exciting ride and the rest of the paper will discuss how we implemented the systems the benefits achieved by each.

3 Release Process



Our high level goal is to release a new Tango client (iOS and Android) every month. To achieve this goal we believe that the best approach is to conduct as much testing in production as possible. For this to be feasible, we need to accelerate getting our server infrastructure into production quickly.

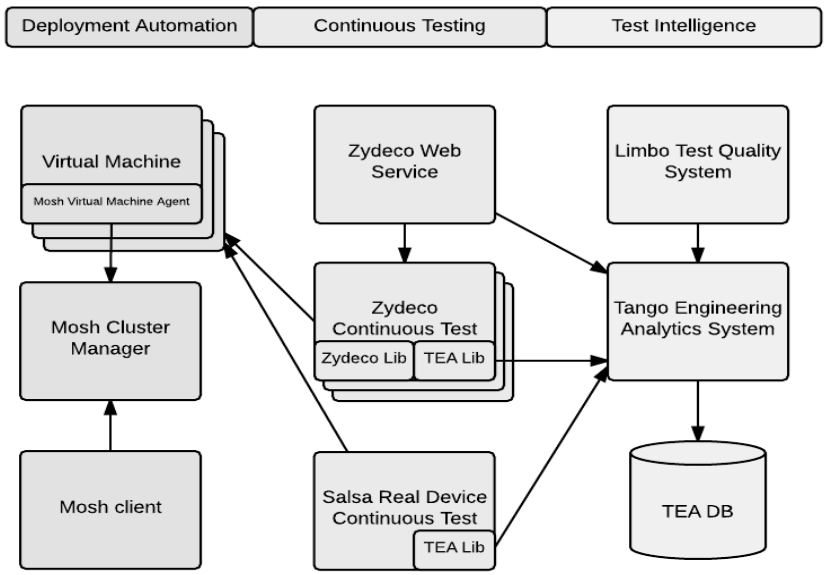
Here is the high level release process:

- Test feature branches
    - Feature branches contain all of the client and server changes for a new Tango feature. The feature branch is tested in a development environment.
- Integrate changes to trunk
    - Once the features are verified in the feature branches the branch is approved to integrate changes to trunk. Once landed to trunk, the changes are deployed to an integration

environment and tested together. All of the features that are targeted for a given release land to trunk in the same way.

- Create release branch
    - Once all of the features for a given release have been landed to trunk and tested in the integration environment, a release branch is created.
- Certify release branch
    - The release branch is then deployed into a staging environment for final certification.
- Deploy release branch to production
    - Once certified the release branch is deployed to production.
- Beta testing of clients pointing to production
    - Once the necessary services are present in production, new clients can now point to production. Tango employees are encouraged to update their clients to the new version for early testing.
    - After a period of internal testing, we release clients to select beta testers outside of Tango.
- Staged Android rollout
    - Android allows us to have a staged rollout for the client. We steadily increase the % of customers that will receive the new client and monitor for any adverse impact to our core business metrics as well as crashes.
- iOS rollout
    - Finally, once Android is 100% rolled out, we release our iOS client. We do this as we want to maintain parity between the client versions and iOS does not allow us to do staged rollouts.

4 High Level System Overview



At a high level there are three different pillars that support our continuous delivery efforts. First and foremost there is deployment automation. We have a **deployment automation** system that can automatically deploy various topologies onto AWS called Mosh.

The second pillar is **continuous testing**. We have several test frameworks that we use to ensure that the code is of high enough quality to release to production.

The third pillar is **Test Intelligence**. This is important as it allows us to determine the state of our tests to ensure that test failures are analyzed in a timely manner.

5 Deployment Automation

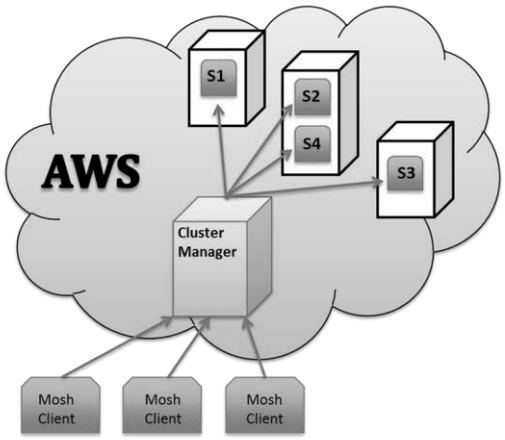
5.1 Mosh

One of the key challenges that we faced as we evolved as a company was that there were multiple different ways that servers were deployed to dev, test, and production environments. This difference was not only inefficient, it also was the cause of many escapes to production as the environment drift surfaced issues in production that were not found in dev and test environments. Traditionally, all of the dev, test, and production servers were hosted in a datacenter. There were many manual steps required to provision virtual machines, install application dependencies, and update application code and configuration.

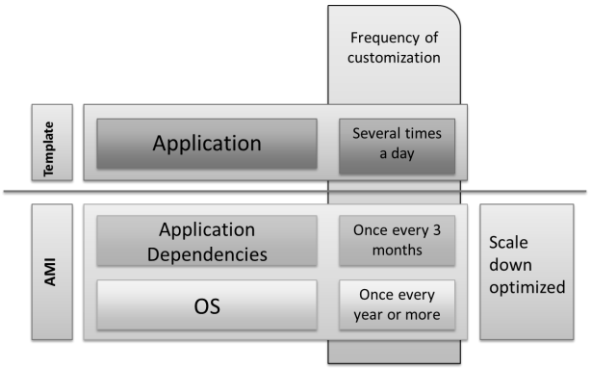
If we were going to release code quicker to production we realized that something had to be done to bring order out of chaos. We decided to build a system that would help us automate these steps. We named this system *mosh* to acknowledge the reality of the state of deployments at that time—like navigating through a mosh pit. Also, we like to name things after different kinds of dances.

5.2 Mosh Layers

Mosh is a distributed system which lives in the Amazon cloud that has a client, cluster manager, and virtual machine agents that run on all managed virtual machines. The client is how the operator interacts with the system. The Cluster Manager is the *brains* of the system and decides where applications should be allocated. The Virtual Machine Agent runs on each virtual machine is the component responsible for updating application builds and updating application configurations.



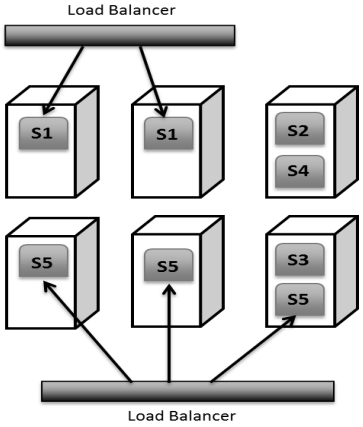
Mosh has two key layers. The first layer contains the operating system and application dependencies. The second layer is the application and application configurations. For us the first layer does not change that often so we decided to include both parts into the base virtual machine image that we use (AMI).



5.3 Balancing environment requirements with cost

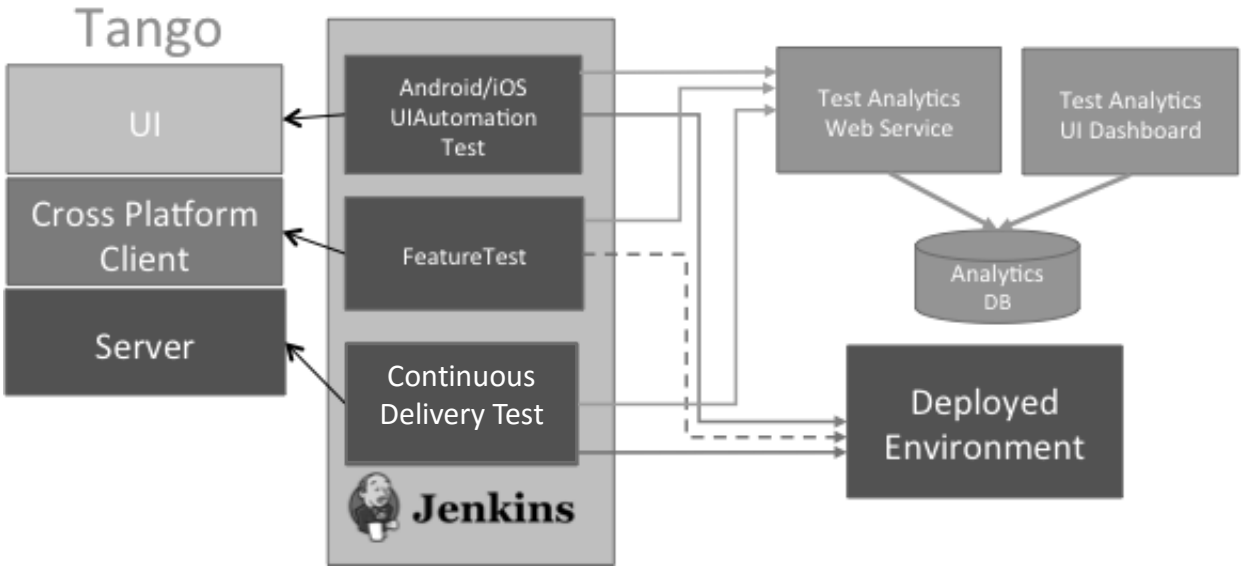
Mosh is designed to allocate services both in a scale out mode and a scale down topologies. Scale out mode (service S1 and S5 below) allows services to have multiple instances allocated behind a load balancer. Mosh also allows services to be deployed in a scaled down mode (service S2, S3, and S4 below).

The reason why mosh supports both topologies is that different environments have different requirements. Our production environment has a requirement for all the server components to be scaled out. Certain testing environments have a requirement for some components to be scaled out to allow for mixed mode (testing n and n+1 server builds in parallel) testing but not all of them. Development environments do not require scale out. The ability to support different topologies allows us to balance our testing needs with cost.



6 Continuous Testing

6.1 Overview



At a high level, we have three test frameworks with different goals. The UI automation frameworks are designed to test the UI layer and below (all layers) for end-to-end coverage. The Feature Test framework is designed to cover the client cross platform layer to server layer. The continuous delivery tests are designed to specifically target servers.

All frameworks are designed so that test cases can be annotated and results sent to a central analytics server to provide a high level view of all tests and the ability to drill down to specific test failures to reduce risk associated with product and test issues.

6.2 UI Automation

Our original UI automation strategy was to leverage Android and iOS UI automation to cover our new client UI during a release. The problem is that our UI changes very frequently (daily) and our automation ends up continuously broken. We decided that instead of spending tons of time during the release fixing the functional tests we would focus our efforts on two fronts:

1. Performance/Reliability – The performance and reliability of our clients are critical as they have a direct impact on the user perception of quality and their engagement. We realized that we simply could not do performance and reliability without automation since it requires running in loops and taking time measurements. Towards the end of the release, we run performance automation for core scenarios as part of the final sign off to ensure there is no performance regression for new clients.
2. Server Regression Validation – Although we get good targeted server coverage via ServerAPI tests and client/server coverage via the Feature Tests that can be targeted towards server environments, we still like to ensure that when we release new server code that our existing production clients do not break. This is accomplished by continuously running the current

production client release against our server environments to ensure that core user scenarios are working end to end. This does not suffer from the problem of client UI constantly changing since we are keeping clients fixed and changing the servers.

6.3 Feature Test Framework

The Feature Test Framework is a hermetic[1] system (self-contained server system with no network connection) utilized by developers to author end to end tests that can be run on their development machine. The system consists of a test client that is a light wrapper on top of the cross platform client layer that contains the bulk of the client code that gets released with the exception of the user interface layer.

In addition to unit tests, developers are required to author feature tests to ensure that they have end-to-end coverage. The key benefit of feature tests is that developers can understand the impact of changes throughout the ecosystem on their components. For example if they have a dependency on a downstream component that changed, they may witness their feature tests breaking as a result and quickly get the issue resolved with the offending party.

Finally, a subset of feature tests (indicated via the dashed line above) can point to deployed environments. They serve as tests for environment stability and some of them are turned into production server monitors. They serve as good monitors since they simulate user behaviors very well as they contain the key parts of the clients that users actually use.

6.4 Continuous Delivery Tests

Continuous Delivery Tests are test cases that specifically target direct server interactions. As our services are RESTful, the tests perform http requests and verify the correct http responses are received.

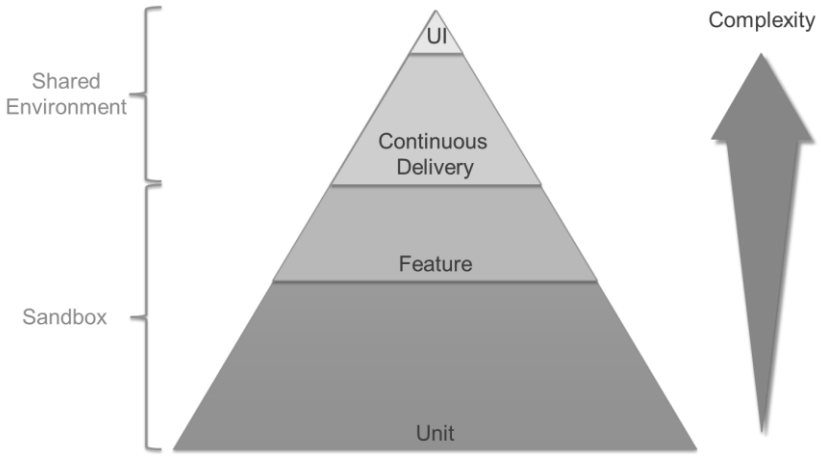
The idea behind having targeted server tests is that they can be developed and executed without the client being present. In particular, it allows us to target servers directly that only interact with other servers.

Here are the requirements for continuous delivery tests:

- Environment Mobility -- Should be able to run in any environment including localhost.
- Non-intrusive -- Should clean up all garbage created by the test and should not interfere with other test instances (including itself).
- Security -- Must not expose environment secrets as test could be run in production environment.

One of the key realizations we had when defining continuous delivery tests is that since they exercise a relatively small amount of the server code there need to be many more targeted tests that have been run already before these get run. We tell our development teams that they need to think in terms of the testing pyramid. Without solid unit and feature tests, the continuous delivery tests are not very useful since there is a high risk that something is broken that the continuous delivery tests did not exercise. When combined together they are very powerful as we get increased confidence as the code flows through different environments.

The testing pyramid below depicts visually the quantity of expected tests of each type. As we get higher in the pyramid the complexity of the test increases since it is exercising a larger area of the system. One key to continuous delivery is to have enough high quality tests that can be run against a shared environment instead of being constrained to run only in a sandbox.



7 Test Intelligence

7.1 Team Testing Dashboard

	Test Coverage *Actual vs. Desired Automated Tests*	Test Execution *Outcome of Tests Run in Last 7 Days*	Test Quality *Defects in Automated Tests*	Product Quality *Defects in Product*	Total Score *Weighted Average of All Scores*
MAD	95.24	61.9 ▽	95.24	95.24	**86.91** ▽
GEM	44.0 △	70.0 △	96.67 △	96.67 △	**76.84** △
EVIL	42.0 △	81.82 ▽	86.36	90.91	**75.27** ▽
T1	87.5	57.4 ▽	54.25	85.71	**71.22** ▽
Dragon	40.0 △	69.51 ▽	85.06 △	90.24 △	**71.2** △
ROCK	34.44	60.23 ▽	22.35	92.42	**52.36** ▽

We want the vertical teams to 'own' the quality of what they built so we built a Test Testing Leader board which helps the entire company visualize where each team is at with their test automation (coverage, what was being run, product issues found, and test issues).

The idea is to gamify testing so that teams feel motivated to execute and keep their test automation clean. This encourages a bit of healthy competition between teams. All test automation is run continuously via our Jenkins continuous integration system and fed in real time to this dashboard.

One nice element of the dashboard is that it is updated in real time and provides a stock ticker like up/down symbol providing a continuous feedback loop to test teams.

7.2 Test Quarantine

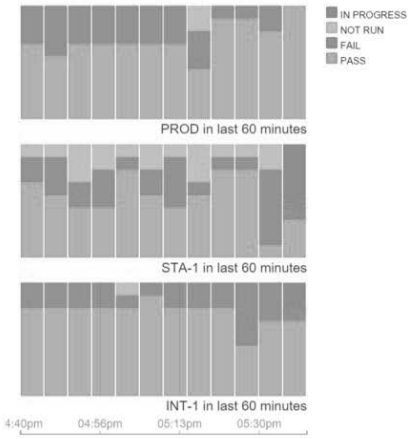
We also have a server component by component status that indicates the health of each component as well as a quarantining system that blocks certifications if tests are not analyzed of fixed in a timely manner. A test gets quarantined (depicted by the toxic icon below) when it has had a sufficient amount of

unanalyzed failures or test issues. This allows us to distinguish between issues where we have a high confidence and those that we have lower confidence.

Scenario Description	PROD				STA-1				INT-1		
Channels can be subscribed.	☣	23.8	3.0		☣	100.0	66.0		✓	100.0	97.0
Contact filtering after adding a Tango User	☣	0.0	100.0	0.0	☣	50.0	100.0	22.0	☣	9.1	8.0
Read Receipt functionality	⚠	93.3	87.0		☣	77.8	72.0		⚠	92.3	80.0
SDK Authentication process	☣	23.8	3.0		⚠	88.9	98.0		✓	100.0	100.0
Photo upload to TC	⚠	86.7	100.0	100.0	☣	50.0	73.0		☣	85.7	39.0
Feed functionality.	✓	100.0	95.0		☣	0.0	34.0		☣	92.3	78.0
Making a video call to the Android robot.	✓	100.0	97.0		☣	55.6	75.0		✓	100.0	83.0
Registration Flow on Android.	✓	100.0	100.0		⚠	93.5	99.0		✓	100.0	100.0
Discovery service	✓	100.0	99.0		⚠	96.6	99.0		☣	51.5	66.0

* TEST IN PROGRESS

Availability in last 60 minutes
As of Thu Jul 16 2015 - 17:33:26



8 Conclusion

Based on these (and other) test investments, we have been able to scale up the company dramatically while steadily improving the quality of our client software and servers. We still have a lot more work to do to ensure that nasty bugs don't escape to production but are happy that we are fully focused on continuous delivery to help us achieve our goals.

References

1. Hermetic Servers, Chaitali Narla & Diego Salas, http://googletesting.blogspot.com/2012/10/hermetic-servers.html

Pre-Mortems

Julie Green

julie.a.green@gmail.com

Abstract

Post-Mortem or Retrospective Meetings can be too little, too late. These kinds of exercises are sometimes too focused on the project in question. That level of specificity makes it difficult to generalize the output enough to apply it to the next batch of work. As a result, many lessons learned are filed away and never implemented. Even when a team intends to use what their new-found knowledge, it becomes extra work that is eventually abandoned. Instead of Post-Mortem or Retrospective Meetings, your ability to predict failures can be increased up to 30% by holding a Pre-Mortem.

Pre-Mortems are meetings that happen before the project, iteration or sprint has ended. A Pre-Mortem meeting creates a safe environment where the sole purpose is to predict failure. Instead of a gripe session, the Pre-Mortem is structured so that attendees are asked for a few areas where they think issues will occur. Then everyone is asked if there is one thing they can do to stop the problem from happening. The list of action items is generated from each person's "one thing that they can do."

Using the Pre-Mortem strategy you can stop these failures from occurring.

Biography

Julie Green has been a QA Engineer for 10 years and has experience using Agile and Waterfall methodologies. Julie was the Scrum Master in an Agile environment for a team that worked in web development. In addition to web projects she has also performed testing in API integration, middleware, and mobile apps.

After providing consulting services as a subcontractor to Cover Oregon Julie decided that there had to be a better way of communicating potential failures. She heard about the idea of Pre-Mortems on the Freakonomics Radio Podcast episode titled "Failure Is Your Friend." This idea was presented to her QA team at Con-way Enterprises Inc. with a positive response.

Julie reached out to the inventor of the Pre-Mortem idea, Gary Klein, who put her in contact with others using the technique. In her discussions with other groups using Pre-Mortems she has documented their suggestions. Julie has presented the idea of Pre-Mortems at the Software Quality Assurance User Group (SQAUG). Two groups have reached out to try the Pre-Mortem technique.

Julie lives with her husband, son and the cutest dog ever, Floyd, in Forest Grove, Oregon. Julie is currently looking for groups willing to try a Pre-Mortem discussion. She can be contacted at julie.a.green@gmail.com.

1 Introduction

If you've ever had a Post-Mortem or Retrospective meeting you know that it can be an awkward and/or combative event. We air our dirty laundry from the previous project and put on our armor to ready our defenses. Before starting, we mentally rehearse the major issues we encountered during the project and pre-prepare our responses.

Whether or not the project was a disaster this meeting just seems like extra work or a family therapy session. The facilitator needs a Ph.D. in Psychology in order to help everyone process their feelings. After the meeting we may come up with a list of changes we want to make next time. How many of these are worked on? Lessons learned from the past can't always be applied to the future. When we think back on that project we just accept it for what it was.

Why do projects turn out badly? Isn't there a way for us to stop the failure before it happens?

The following ideas came from the Freakonomics Radio Podcast episode called, "Failure Is Your Friend." The episode describes a method of reducing the incident of failure by 30% by holding a Pre-Mortem meeting.

2 Go Fever

Go Fever is a term coined by NASA. It means insisting on moving forward despite being told that there is a potential failure. When you are striving towards quality, like many of us do every day, you are constantly warning or informing your teams and managers about potential failure. Most of the warnings given are risk assessed. At the very end, with our stomach in knots, a decision is made. It is either a "Go" or a "No Go".

How many of us worked with colleagues who did not listen when we made warnings, opened defects, and generally disapproved of the quality we were sending to our customers? I've never encountered anyone in a Quality Assurance or Quality Control field that didn't have a run-in with Go Fever. In my case I would have manager after manager insist that the urgent defect wasn't urgent. I seemed to always test right up to the last minute still finding defects. So many times the date that the project was due was more important than the quality of the project itself.

2.1 Examples of Go Fever

2.1.1 Apollo 1 Spacecraft

When NASA coined the phrase "Go Fever" it was after a life and death situation turned to death. On January 27, 1967 the Apollo 1 space mission team conducted a test. A flash fire occurred during that test in the command module. Lt. Col. Virgil I. Grissom, Lt. Col. Edward H. White, and Roger B. Chaffee died in the fire.

A comprehensive investigation found many lethal designs with the Apollo 1 spacecraft. Someone was in a hurry. NASA had two years before they were scheduled to land on the moon in 1970 yet the paced was forced so that they would leave two years early.

2.1.2 Shuttle Challenger Disaster

Go Fever was evident again after the Challenger Space Shuttle disaster in 1986. Allan McDonald, engineer for contractor Morton Thiokol explains in his book Truth, Lies, and O-Rings: Inside the Space Shuttle Challenger Disaster," that he was concerned about the freezing temperatures in Florida the day of the Challenger launch. He knew that there wasn't enough testing at this temperature.

Before the launch there was a teleconference where McDonald recommended a delay. He was challenged at every recommendation of a delay. He aired on the side of safety. When challenged, McDonald was asked to prove that the launch would fail. He could not do that because he did not have data from a test at a low temperature to support his claim.

McDonald refused to sign off for the launch. Instead his boss signed off. Like all of us who work in quality, McDonald was not happy. He expressed this to his boss.

After the tragedy Physicist Richard Feynman reviewed the case and pointed out that the O-Rings when tested at low temperatures failed to perform. Since NASA had a history of success between Apollo 1 and the Challenger there was a sense of arrogance leading up to the Challenger disaster. Arrogance that lead to the Go Fever of NASA officials.

2.1.3 Cover Oregon

In providing consulting services as a sub-contractor on the Independent Verification and Validation Team (IV&V) for Cover Oregon I kept hearing the message, "We have to go live on October 1st." Our team was required by the Federal Government to make sure tests were carried out the way they were specified by the Quality Assurance team at Cover Oregon. Each state had an IV&V contractor. We were required to be a contractor, or independent party of some kind.

I didn't hear about Go Fever until after the work I had done on the IV&V team. Right away I thought of the message "We have to go live on October 1st." Really? If individual's privacy was at stake, or if you couldn't even log in the Cover Oregon web site had to go live? And now we know it didn't go live on October 1st. They was nothing safe enough to put into production. I knew in September that this was a no go, and I said so. But this was not a message you could state. This was $200,000,000 plus in Go Fever.

2.2 Courage – the antidote to Go Fever

Courage is the way to stop Go Fever. This is a hard thing to do for most. When we speak up we can seem like we are not a team player. Most bosses want their teams to be optimistic about the outcome of the project. Instead of calling it like it is we have to say things differently in order to still be part of the team.

Momentum of the project can keep it going on its trajectory to disaster. The politics of our offices make it seem impossible to stop. We are told that this is how the culture of our company works. The egos in our leadership and teams can silence those who are looking out for our quality.

There is a way to boost our courage and make it safe to talk about potential failure.

3 Pre-Mortems

Pre-Mortems are a meeting, a strategy, where the team imagines that a project has failed. Holding a Pre-Mortem reduces the overconfidence of the bosses and team. During this meeting the whole point is to talk about failure so everyone is a team player. Calling out potential failures is safe during the Pre-Mortem.

3.1 Gary Klein Ph.D.

Imagining potential failure is called Prospective Hindsight. Dr. Gary Klein Ph.D. found that when Prospective Hindsight is used it increases your ability to identify future outcomes by 30%. Dr. Klein, who is a cognitive psychologist, practices experimental psychology. He and his team use the Pre-Mortem strategy on their own projects to predict how their outcomes can fail. Dr. Klein has written several books including "Seeing What Other's Don't" which discuss ways to increase our insight and encourage failing.

3.2 Sterling Wiggins – Aptima.com

Being extremely curious about Pre-Mortems, I reached out to Dr. Klein for more resources. He put me in touch with Sterling Wiggins from Aptima.com. Sterling works on flight simulation software that is used by the United States Air Force. Sterling uses Pre-Mortems with his projects.

Sterling explained his success with Pre-Mortems and is working on a book about it. He has consulted with nurses who use the Pre-Mortem strategy with patients before surgery. In these life and death situations a Pre-Mortem proves itself by providing better patient outcomes.

3.3 Dave Thomas – Forest Fire Containment

Dr. Klein also put me in touch with Dave Thomas. Dave works on projects to contain forest fires. His work is also live and death. Dave performs Pre-Mortems with his team, sometimes in the helicopter on the way to the fire. Dave knows the Pre-Mortem stops preventable mistakes by forcing everyone to think about them beforehand.

4 When to have a Pre-Mortem

I asked Dr. Klein, Sterling Wiggins and Dave Thomas questions about Pre-Mortems. The question on top of the list was when? At what point in the project is it most valuable to have a Pre-Mortem?

Sterling was the most knowledgeable about software processes. We discussed different methodologies and agreed that with a Waterfall methodology it would be the design phase. For Agile the best time would be after the planning meeting. It also could help to have another Pre-Mortem half-way through the project.

Dave only has the criteria that they hold a Pre-Mortem before sending firefighters into a particular space.

4.1 Why not hold a Retrospective?

I've held retrospective meetings while acting as Scrum Master at ADP Dealer Services (now CDK Global). These meetings have turned nasty on me. It often seems too late because the sprint it over. Many times in a retrospective the improvement we find is too specific to the project we were working on. We don't have an opportunity to implement the improvement we thought up.

Also with a retrospective the product now is done. It is what it is. It is now exponentially harder to change.

5 Pre-Mortem Experiment

My other questions for Dr. Klein, Sterling Wiggins and Dave Thomas centered around how to conduct the Pre-Mortem. It helps to imagine a project we are all working on. We just had our planning meeting and now it's time to have the Pre-Mortem.

5.1 Pre-Mortem Meeting Agenda

The Pre-Mortem consists of 5 steps. The first 4 are:

1. Imagine the project has completed and failed.
2. Write down 1 or more reasons why it failed. (2 minutes)
3. Share your reasons for failure.
4. Compile reasons

5.2 Imagining the failure

Dr. Klein suggests everyone get into a relaxed state when activating their imagination. A state that is little dreamy but not sleeping. You may want to close your eyes and take a deep breath.

Next the facilitator imagines and discusses looking into a crystal ball or some such fictional predictor of the future. They see that the project has failed. The failure was so bad we don't want to look at each other in the hallway. We are embarrassed that we were part of this team. The failure was legendary.

5.3 Reasons the project failed

The facilitator asks everyone to write down 1 reason why the project failed. If they have more they are encouraged to do so. This activity is usually timed for 2 minutes.

5.4 Share the failure reason

Each person shares 1 failure reason. If they wrote down more than one they should share the most important or the one that they felt would mostly lead to a failure. The facilitator should make sure that their reasons are clearly understood. Only the person who's turn it is to share should be sharing, any rebuttals should be discouraged. We want to hear any failure no matter whether or not someone knows it should not cause failures.

The facilitator will compile the list of reasons. Whiteboards or large easels of paper work for this task. The data gathered needs to be captured later in a format that can be used for follow-up. Some people may mention an outlandish scenario. The "Asteroid falling from the sky" scenario should also be written down along with the others. Sterling explained that playing along with this and moving on helps the team move forward.

5.5 One thing you can do

The fifth and final action in a Pre-Mortem is a 2 minute session where everyone writes one thing they can do to stop the failure. Anyone can choose any failure item. If they don't feel they can change the item they wrote down perhaps they can help with an item someone else wrote.

The team should be encouraged to work on an item where they can make a difference.

5.6 Share the one thing you can do

As with the failures each person then shares the one thing they can do to stop the failure. This is where we find there is nothing we can do about the asteroid falling from the sky. But there are many things we can do about other issues.

The facilitator should compile this list again. This list becomes the action items for the team to work on. The team should then prioritize which items are most important. Those are the things to work on. All other items can be listed as risks.

Someone should become the owner of this list. This could be the facilitator, project manager, product owner, scrum master or team member.

At the end of the sprint or project each item should have been checked off or a plan created to deal with the issue.

6 Challenges

I have yet to convince a team to hold a Pre-Mortem. The comfort of doing what has always been done or what the Agile Consultants say is too overwhelming. There is nothing to lose except a possible half hour to one hour.

Another challenge is team cooperation. That asteroid scenario is a funny mask to cynicism. The cynic has no suggestions for change and does not reflect the entire team.

Follow-up on action items is another challenge. These items can be seen as unimportant to the Boss with Go Fever. It might be better to assign this task to the scrum master or facilitator.

References

Garber, Steve. 2014. “Apollo 1 (204).” NASA History Homepage posted February 3, 2014. http://history.nasa.gov/Apollo204/ (accessed May 31, 2015).

Demaret, Kent. March 24,1986. “'Group Think' and 'Go Fever' Brought the Shuttle Down, Says Ex-Astronaut Donn Eisele” http://www.people.com/people/archive/article/0,,20093213,00.html (accessed June 1, 2015).

Atkinson, Joe. October 5, 2012. “Engineer Who Opposed Challenger Launch Offers Personal Look at Tragedy” http://www.nasa.gov/centers/langley/news/researchernews/rn_Colloquium1012.html (accessed June 3, 2015).

“Failure Is Your Friend: A New Freakonomics Radio Podcast”, Freakonomics Radio Podcast, Freakonomics.com, June 5, 2014.

Improving Quality Team Coding Skills with Code Clubs

Authors: Dwayne Thomas, Kevin Swallow

Email addresses: dthomas@crowdcompass.com, kswallow@crowdcompass.com

Abstract

As CrowdCompass has matured its agile processes, pressure has increased on the company's testing team to become more efficient at delivering high quality products. Testers actively sought coding knowledge to contribute more to software development discussions. Testers also sought to automate more of the testing workflow. The need for the code discussions arose because very few of the testers were experts with programming. Moreover, most of us had less than 2 years of experience at CrowdCompass and did not know how else to speed our adoption of coding best practices.

Code Clubs are recurring coding discussions that often feel like a brewing stew of ideas. They give us the nutrients to strengthen our strategic effort of leveraging code in our testing efforts. CrowdCompass leadership wholly encourages the clubs, because learning and initiative are company values --and that is reflected in our quality engineering (QE) practices. We believe the organization has benefited from these low investment meetings. No direct causal link can be established between product improvements and the clubs, but, since the CrowdCompass Code Clubs started, many of the test staff have started new scripting projects and even some of our most code-anxious testers have contributed to the automation effort.

This paper shares the experiences of our testing team with these code discussions. We will provide the information necessary for the reader to start their own Code Club and some of the programming lessons we learned.

Biography

Dwayne Thomas is a quality engineer for CrowdCompass by Cvent. He's mainly tested in the television/software space, before coming into mobile conference software. Dwayne caught the learning bug when he taught middle school math for 4 years in New York City.

Kevin Swallow was an Oregon licensed professional mechanical engineer before moving into software quality and automation. He graduated magnum cum laude from CSU-Sacramento and is currently working at CrowdCompass, where he promotes software best practices and helps to implement automated testing.

Copyright Dwayne Thomas and Kevin Swallow 2015

1. Introduction

As Crowdcompass has matured its agile processes, pressure has increased on the company's testing team to become more efficient at delivering high quality products. Testers actively sought coding knowledge to contribute more to software development discussions. Testers also sought to automate more of the testing workflow. The need for the code discussions arose because very few of the testers were experts with programming. Moreover, most of us had less than 2 years of experience at CrowdCompass and did not know how else to speed our adoption of coding best practices.

Code Clubs are recurring coding discussions that often feel like a brewing stew of ideas. They give us the nutrients to strengthen our strategic effort of leveraging code in our testing efforts. . CrowdCompass leadership wholly encourages the clubs, because learning and initiative are company values --and that is reflected in our quality engineering (QE) practices. We believe the organization has benefited from these low investment meetings. No direct causal link can be established between product improvements and the clubs, however, since the CrowdCompass Code Clubs started, many of the test staff have started new scripting projects and even some of our most code-anxious testers have contributed to the automation effort.

This paper shares the experiences of our testing team with these code discussions. We will provide the information necessary for the reader to start their own Code Club. As a storytelling device, we at times compare process of starting a code club to cooking processes. We include pieces of code in 'single' quotation marks in order to make audiences familiar with the content of our discussions. Electronic links are underlined throughout the paper and listed fully in the references section at the end of the paper.

2 The CrowdCompass Code Club

2.1 History

The idea for the CrowdCompass Code Club started simmering after Dwayne Thomas determined that a Code Club was an excellent opportunity to fill the QE team's skills gaps. QE manager Lloyd Bell enthusiastically supported the idea. Most of the testing team joined in. As news of our efforts wafted out to the rest of our organization, some developers started contributing to it as well.

2.2 How to start your own Code Club

Start your own Code Club to start experimenting with the benefits of it. The following paper is a collection of ideas highly relevant within the CrowdCompass context. However, everyone in your organization could join the Code Club. Simply put, the clubs are informal gatherings of 3-6 testers discussing 100 lines of code for one hour. We call our club "Codetry." Sessions proceed by testers questioning any new terms or practices in the scripts. Internet searches help explain new terms. An experienced software developer is vital to understanding best practices.

We scheduled meetings in Microsoft Outlook and kept them at about 85% regularity. We created an ongoing instant message room to capture fleeting, but essential, learning points. The notes helped folks that were not able to attend the meeting. We had varying success using Google Docs for planning ideas for future sessions. We found the meetings were easiest to organize on a week by week basis, since we wanted to make the meetings highly relevant for those who could attend.

We also think it is important to share efforts early and often to the rest of your organization. Sharing helps to recruit new members and get new resources.

2.2.1 Early Reading Code Good recommendations

Our Code Club draws on many sources--starting with the foundation of offerings on www.readingcodegood.com. These sources were great for beginning since they were readable and had

engaging applications. The Python and Ruby programming language scripts often met our criteria and we were quickly able to look at the behavior they implemented. Nevertheless, we tried to be inclusive of many languages in our meetings.

For example, the Google Books repository helped us understand how to pull information from the Google Books application programming interface (API). The interfaces enable the exchange of information among systems that use different technologies. They can skip the presentation layer of data transfer. Since CrowdCompass has branches on the East Coast and we share their API's, API functions had enough relevance to us testers. For example, API discussions broached how the Ruby language includes external libraries (it uses the require keyword). Other folks at the meeting offered that Python uses the 'import' keyword to accomplish the same task.

Another Readingcodegood recommendation, the Country and Region select script, showed us how to programmatically translate country names and introduced us to internationalization and UTF-8 character concerns (Sstephenson, 2014). Our testing group decided pieces of code snippets like this Joel on Software (2003) one could be relevant if and when our company expands its applications to non-US markets. As states the article states we certainly would want our application pages to translate as more than a string of question marks.

2.2.2 Keeping it simple, but not too simple

When we started the Code Club effort, our goal was to keep things simple and comprehensible. At first we avoided file-spanning object oriented concepts such as inheritance, encapsulation, polymorphism, and abstraction. These concepts had especially nonlinear logic for the tester group. Also these concepts might have hidden code implementation from testers. In later meetings we tackled these topics (see appendix).

2.2.3 Seeking breadth

We needed familiarity with finding code on GitHub to expand the topics to review. Github is an online community where developers share more than 24 million code projects. This search finds code repositories of less than 100 kilobytes (Github, 2015). Since code files are text based, this search for small files proved more than adequate for the first few sessions of Code Club. Participants tended towards ruby and python repositories, the languages of our testing harness and our company's applications. Files with the Ruby (.rb) and Python (.py) extensions were most fruitful for us: they contained good working code examples. Python code is so readable, for example, that this code will loop and print the 3 strings that the reader might expect it to:

```
'for x in my_range(1, 10, 0.5):     print x'
```

Still, we found that different languages had enough idiosyncrasies that we could shop among them and learn their conveniences and handicaps.

Github also introduced us to project file structure on the internet. We found that lib directories (folders named lib) often contained scripts that were appropriate for our review. We learned to ignore other files, such as vagrant files (.vag), as they seem to only contain lower level information that was more understandable to computers than people. We still have to revisit this kinds of files to learn more about them.

2.2.4 Discussing the Python script implementations

At Code Club we asked questions about implementations not specific to our business logic. One attendee asked "what does `if __name__ == "__main__":` do?" The online programming question-and-answer forum Stackoverflow helped us answer that before executing the code, Python will define a few special variables. MAIN is an entry point for the language. By doing the main check, the code only executes when needed.

2.2.5 Less perfect scripts

We made a point to learn code from scripts of varying levels of expertise. The Rayel program showed us how Java's print statements ('System.out.println') are more complicated than Python and Ruby (Amanoske, 2015). Java's implementation made sense to us since Python ('print') and Ruby ('puts') are more interactive languages and Java has to be compiled before being used. During these exercises, having a software developer in the group was especially helpful for going through the obtuse code. The developers helped us discuss code quality as well as its structure, syntax, function and implementation.

3 Complex ingredients and flavors

As we learned, we became more confident and knowledgeable. Early scripts led us to more involved ones which could be combined with our company's code stew. We more confidently analyzed some of the more complex code previously mentioned.

3.1 Design patterns

Discussions about scripting, best practice, and application design led to our covering design patterns explicitly. These are frequently-used code structures for object creation and method heuristics. We focused on patterns that were more important for interactive languages such as Python or Ruby and/or relevant to our work. We looked at the Strategy Pattern, the Model View Controller pattern, the Template Method Pattern and decorators (Faif, 2015).

As an example, for mainly his own practice, one server developer at our company concocted a math polynomial formula to explain the template method pattern. He showed that it is possible to reuse the math distance operation with an assortment of polygons, for example from linear to triangles. He used the template method pattern to implement the DRY (do not repeat yourself) mantra. The pattern allows us to change specific steps of methods without changing the method's structure. The same pattern was especially instructive for the testing team's understanding of our testing harness. The operation of logging in and out of an app could similarly be reused. Notably, the template pattern is the key concept in inheritance, which we initially shied away from.

3.1.1 Model view controller pattern

The model view controller, common in applications, was a discussion topic for the testing team. Figure 1 of the pattern shows that between the computer and the browser there are several layers of script that typically deliver the expected user experience. Each layer of the application is separate and can even be written in a different programming language.

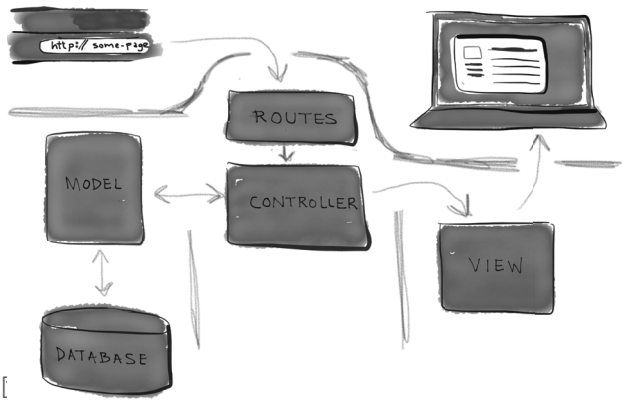


Figure 1: Model view controller diagram

3.2 Configuring servers for deployment

Eventually, Code Club acquainted us with difficulties in deploying servers. Deploying servers is a common practice of members of the CrowdCompass operations team. The INI file format, often found in repositories, is an informal standard for configuration files for some platforms or software. INI files are simple text files with a basic structure composed of sections, properties, and values. In MS-DOS and 16-bit Windows platforms up through Windows ME, the INI file served as the primary mechanism to configure operating system and installed applications features (Wikipedia, 2015). The Ansible script is a radically simple IT automation platform that makes your applications and systems easier to deploy (TRBS, 2015). It avoids writing scripts or custom code to deploy and update your applications—automates in a language that approaches plain English, using text based shell sessions to install on remote systems.

4. Family recipes: learning the company product

Kevin Swallow recognized one of the great benefits of the Code Club was to give the participants insight into our own applications, and motivated them to further study in that direction. Outside of Code Club, testers have installed the company's app onto their local computers under the guidance of developers and preview development features. Testers might not start with the company's app since setting it up requires using development tools.

4.1 Ruby on Rails app walkthrough

4.1.1 Design principle

We began with a Code Club meeting that reviewed one of our Ruby on Rails app repositories. We also referred to the online Install Rails (2015, Griffel) documentation. Attendees learned about convention over configuration. When the convention implemented by the tool matches the desired behavior, it behaves as expected without the developer having to write configuration files. A URL can describe a program entry point, a controller, a method, perhaps an object. Controllers pull data from similarly named models and the output cascades to similarly named views. Only when the desired behavior deviates from the implemented convention is explicit configuration required.

4.1.2 Organization of app

We also learned about the default structure of a rails application. When we see errant behavior in an interface with an https URL, we know to look for config>routes.rb. This file helps us follow web requests into the application and to controllers. This knowledge is sure to help testers identify bugs.

4.2 Tests in Rails

At another session, one of developers demonstrated how unit tests were developed and where they could be found in our application. By understanding the tests that the developers include in the app, we expect to be able to limit our own work to avoid redundancy, and find areas that we might want to include in additional testing. The first step is to find and understand them.

4.3 Debugging

Some of our more focused debugging sessions are still in the works. We have tentative arrangements for developers to tell us how they start with a bug ticket, find the error in the code, and fix it. We hope this knowledge will improve how testers write bug tickets for developers.

4.4 Demoing C and iOS unit tests

Code club provided a venue for programmers and testers to discuss the C language and unit tests. In the course of learning about getters and setters of the C language, we learned about the percentage of the test coverage on our iOS devices. Getters and setters are needed to prevent the need to modify every file in the codebase with variable changes. The also allows for validation in one location and a hidden internal representation elsewhere. For example 'getAddress()' could actually be getting several fields. Getters also prevent setting temporary variable names. This knowledge could also help test engineers learn how to structure their tests so that they make defect isolation efficient.

5 Juggling knives: unexpected tools, new skills

5.1 Jira automated notification of ticket statuses

Code Club trained us to write scripts to facilitate the administration of our software development. Some of these scripts were very exciting and quickly useful. One of these was a script that would automatically execute a Jira query, and then publish a notification to the appropriate chat room. Some testers improved their script to a twice a day notification about Jira tickets that needed to be reviewed by product managers. If product managers do not get to the Jira tickets in a timely manner, testers eventually get blocked on their testing. Therefore our release team devised a script to automatically notify the Product team about potential blockers. The Jira Notifier script is just a few steps away from being a full-fledged bot, which would listen to the Hipchat dialog for commands and execute scripts on a time and calendar basis as well. We'll have a follow-up Code Club session when the first QA Bot comes on line, and it's predicted that each of our sprint teams will have their own.

5.2 The more you know

We reviewed the code for the automated notifier with its author which, in turn, introduced us to the open source python library CLICK(2015). CLICK is the Command Line Interface Creation Kit and is a neat tool for testers who might drown in their own command line creations without appropriate help documentation at their own fingertips This library has become a staple of most of the testing team's scripts.

5.3 Impacts on the testing team and the company

Code Club seems to have had significant impacts. Discussion of coding practices among the quality team members has become more frequent and more informed. At least one quality team member who once adamantly refused to have anything to do with automation is now writing automated tests. Attendance at code-centric brown bags regularly includes a core of 4 attendees and a rotating cast of about 15 participants. In a response to Kevin's suggestions to push the Code Club more solidly toward improving CrowdCompass testing tools, Dwayne installed a local development environment on his computer, and now several other testers have followed suit. There is talk of establishing local environments for everyone on the testing team.

None of this is happening in a vacuum. One suite of automated API tests is nearly complete, and has had measurable impacts on quality. In fact, it was used to make a major upgrade to one of its endpoints rapidly and with confidence. We help address the testing team's skills gap which allowed us to write more automated tests.There is a definite new sense among all the testers we should code. Developers are also more frequently completing reviews on their own work as well. There is no way to separate which parts of this progression are thanks to Code Club and which parts are simply due to the company zeitgeist.

6 Sumptuous deserts - personal projects

We reviewed some code just for personal projects and plan to do more in the future.

6.1 Webscraping

For life's mundane or stressful moments there might be a script to help things out. The Beautiful soup library (a collection of functions) has proven useful for scraping the web. Web Scraping code potentially helped a tester save money on an Amazon purchase (Shorey, 2015). This Python script uses a government api to search for jobs that do not restrict employment based on criminal records (Shorey, 2015). Another code sends a text when a new Craigslist posting matches a given keyword or phrase (Gjreda, 2015). Individual testers planned to set up a task that runs automatically at timed intervals and saves manual effort.

6.2 Code script for making life easier

Dirmon is a utility which keeps track of changes in a user's current directory (Ayancey, 2015). It is useful for monitoring folders that are not on the web. This script would be useful for telling when to sync folders. The use case for this script might be to regularly update any personal versions of larger shared projects.

Conclusion

Our Code Club originally centered on developing general code fluency for testers. As we narrowed our focus to emphasize code in test, we were glad to see more developers dropping in. We think this is a good thing, and we plan to continue to promote test automation and better testing through knowledge and understanding of coding in general and our company's products in particular. At the same time there is plenty of good material to review outside that scope, and we plan to stray from the narrow path whenever there is a need.

We believe that Code Clubs has been a worthy inclusion in CrowdCompass practice that has benefitted everyone at all levels. For the company, it has been a low-investment training opportunity. Thankfully, many co-workers stepped forward to help us. Understanding code and code principles leads to more involved discussions about code changes. Such discussions likely help prevent bugs from happening or else help bugs to be found sooner. The participants have been provided opportunities for professional development and knowledge sharing among the team and across the company. The authors found that even, with the varied coding experiences, most of the testing team benefited from the discussions in reviewing important practices and concepts. The discussions also confirmed learning programming can be very empowering, allowing users and companies to manipulate an existing product to serve their own interests.

Recommended readings

Hartley, Brody. I Don't Need No Stinking API: Web Scraping For Fun and Profit. https://blog.hartleybrody.com/web-scraping/ (accessed June 1, 2015).

Real Python. "Model-View-Controller (MVC) Explained -- With Legos" https://realpython.com/blog/python/the-model-view-controller-mvc-paradigm-summarized-with-legos/ (accessed March 10, 2015).

Suggested code for review

Amanoske. "Rayel." https://github.com/amanoske/Rayel/blob/master/Rayel.java (accessed Nov 8, 2014).

Ahawker. "Appthwack." https://github.com/appthwack/appthwack-python/blob/master/appthwack/appthwack.py (accessed July 30, 2015)

Ayancey. “Dirmon.” https://github.com/ayancey/dirmon (accessed February 23, 2015).

Github.

https://github.com/search?utf8=%E2%9C%93&q=size%3A%3C100+&type=Repositories&ref=searchresults

(accessed May 30, 2015).

Ivanoats. “FizzBuzz.” https://github.com/codefellows/FizzBuzz/blob/master/Ruby/fizzbuzz.rb (accessed December 9, 2014).

Faif. “Python-Patterns.” https://github.com/faif/python-patterns/blob/master/strategy.py (accessed May 30, 2015).

Shorey, Rachel. “Buscando.” https://github.com/aliyarahman/buscando/blob/master/db_populate.py (accessed January 13, 2015).

Shorey, Rachel. “Second_Chance_Employers.” https://github.com/aliyarahman/second_chance_employers/blob/master/usajobs.py (accessed January 20, 2015).

Sstephenson. “Country_and_Region_Select.” https://github.com/basecamp/country_and_region_select/blob/master/lib/country_select.rb (accessed December 16, 2014).

TRBS. https://github.com/ansible/ansible/blob/devel/examples/scripts/yaml_to_ini.py (accessed January 27, 2015).

Waldherr, Simon. “IRCLogger.” https://github.com/SimonWaldherr/ircLogger.go/blob/master/ircLogger.go (accessed February 3, 2015).

Wikipedia. “INI files.” http://en.wikipedia.org/wiki/INI_file (accessed July 30, 2015).

Zeantsoi. “GoogleBooks.” https://github.com/zeantsoi/GoogleBooks (accessed December 2, 2014).

References

Griffel, M. “Install Rails.” http://installrails.com/ (accessed May 12, 2015).

Richardson, Leonard. Beautiful Soup Documentation.

http://www.crummy.com/software/BeautifulSoup/bs4/doc/ (accessed June 1, 2015).

Ronacher, Armin. $Click. http://click.pocoo.org/4/ (accessed June 1, 2015).

Stackoverflow. “What does if name do.” http://stackoverflow.com/questions/419163/what-does-if-name-main-do (accessed June 1, 2015).Yitbarek, Saron. “Reading Code Good.” http://www.readingcodegood.com/ (accessed June 1, 2015).

Glossary

Inheritance is when classes in the lower hierarchy get variables (static attributes) and/or methods (dynamic behaviors) from the higher hierarchies.

Encapsulation is a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. See figure 2 where organs and systems of the cat have been abstracted away.

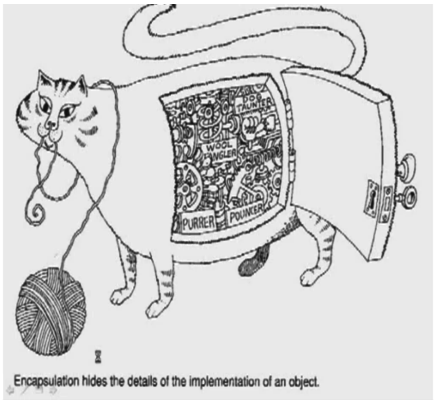


Figure 2: Encapsulation diagram that hides the systems of the cat

Polymorphism takes advantage of inheritance and allows objects to behave differently in different contexts, for example the + (plus) operator in C++: provided integers,

for example, 4 + 5, it performs integer addition to produce 9. Provided with floating point numbers like 3.14 + 2.0, it performs floating point addition, which is a very different operation for a computer, and returns 5.14. Finally given string values, s1 + "bar" it will do string concatenation and return foobar (assuming s1 = “foo”)!

Abstraction is what software developers use to decompose complex systems into simpler components. For example, imagine a video game where we have chosen to abstract a person to their height and their ability to run as a class.

Appendix

Abstraction example: Java code that makes the person run as a function of height, ignoring the other complications of personhood.

```
Public Class Person
   Private _height As Int16
   Public Property Height() As Int16
      Get
         Return _height
      End Get
      Set(ByVal Value As Int16)
         _height = Value
      End Set
   End Property
 Public Sub Run()
   End Sub

End Class
```

How to Develop High Quality Software

John J. Paliotta

john.paliotta@vectorcast.com

Abstract

When organizations design a physical product, a significant amount of time is spent designing the manufacturing process and on the quality assurance testing at each stage of production. If the product cannot be manufactured or tested efficiently, it will be redesigned.

In contrast to this, testing is often an afterthought for software development groups, with little thought given to the test and manufacturing process until late in the development cycle. During the design and development phases, the focus is almost always on functionality and performance. While these items are important, they are meaningless if the application is buggy and difficult to maintain.

This paper will define a software development process that spreads testing responsibilities across the entire organization and life-cycle, and improves quality by: ensuring the completeness and correctness of requirements, measuring the effectiveness of test activities, and implementing a continuous and repeatable test process.

Biography

John J. Paliotta co-founded Vector Software in 1990. In 1994, he and William McCaffrey built the first version of the VectorCAST products, VectorCAST/Ada. This product was initially sold to customers building Avionics, Military and Space applications. Currently Paliotta serves as the Chief Technology Officer and oversees all Engineering. He received his AB in Math from Boston College.

Copyright John J. Paliotta 2015

1 Overview

On a yearly basis, organizations commit themselves to key objectives, often achieved by improved processes and measured via metrics for performance, quality, revenue, and profitability. Measuring the performance involves testing and reporting. Savvy employees know to ask for a list of what they will be judged on before being reviewed. However when it comes to developing new software products, identifying goals for quality and testing is rarely considered.

When organizations design a physical product, a significant amount of time is spent designing the manufacturing process and on the quality assurance testing at each stage of production. This allows manufacturing and test issues to be solved prior to the design being complete. If the design cannot be manufactured consistently and economically, it will be reworked.

Much has been written recently about the Internet of Things where nearly all electronic devices in the world will be web enabled. No one is sure of how the architecture will evolve, but it seems obvious to me that it must involve the migration of intelligence to these endpoints. Clearly it will not scale to have trillions of sensors transmitting raw data back to centralized data centers.

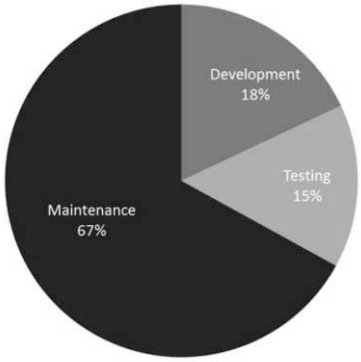
Based on this fact, it seems incumbent on the developers of this new wave of connected devices, to prove the robustness of the software that will control these endpoints. It might be a great feature if your refrigerator can be used to order groceries but not a good side effect if the compressor shuts down because the internet connection locks up.

2 The Current State of Quality

According to Cambridge University research[1], the global cost of debugging software has risen to $312 billion annually. Companies are clearly spending incredible amounts of money treating the symptom, but not nearly enough solving the root cause.

Multiple studies have shown that the largest component of software cost is not the original development, test, and manufacturing cost, but the post release maintenance cost.

As shown in this chart, the maintenance costs are generally double the original development costs. Is there any other industry that could support this cost structure? If it costs an auto company $10,000 to produce a car, do they expect the warranty costs for that car will be $20,000?



3 How did we get here?

Software is interesting for a variety of reasons, but one of its best and worst features is its malleability. It is so easy to change software that most applications live in a constant state of flux, with patches for bugs and new features constantly being bolted on, and often times breaking things in the process.

The malleability of software has led to a "we can fix things later" attitude among software developers. "During the next round of changes, we'll make it better." This attitude is so pervasive that a term was coined by Ward Cunningham for the cost associated with shortcuts taken; he calls this "Technical Debt[2]".

---

[1] Financial Content: Cambridge University study states software bugs cost economy $312 billion per year

[2] http://en.wikipedia.org/wiki/Technical_debt

The fact is that most all applications have a pretty large technical debt, and as with financial debt, at some point the debt must be repaid.

In the past technical debt was often "paid off" via a complete application re-write, kind of a declaration of bankruptcy. As recently as 15 years ago, it was common to build an application, patch it for a number of years, and then do a complete rewrite once it got too slow and hard to maintain. In fact the first system I worked on out of college went through three implementations in six years, moving from 100% assembly language, to a high-level language, to new hardware and a second high-level language.

As software has become larger and more complicated, there is less green field development. Most new development relies on legacy code bases, and the code being built today will have a longer life-cycle than ever before.

The famous Heart Bleed bug from 2014 is a great example; the bug was introduced by an engineer in 2011 and existed un-detected in the OpenSSL code base for three years. By the time it was detected, the bug created an exploitable vulnerability in more than 17% of all secure web servers.[3]

4 There are no easy Answers

I recently bought a new piece of networking equipment, and found the following in the user guide:

"As with many electronic devices many errors can be resolved by turning the power off and back on to reset the device."

So is that the solution? As more and more devices that we use every day contain software, will we be running around cycling power to everything in our houses? Clearly not. Consumer and market pressure will eventually force all vendors to improve quality or perish.

The debt analogy, as mentioned above, is perfect for software, because with Technical Debt, just like financial debt, there are no easy answers. Just as a lottery ticket is not a likely solution for financial debt, there are no "silver bullet" solutions to Technical Debt.

[3] http://en.wikipedia.org/wiki/Heartbleed#Affected_OpenSSL_installations

5 The Quality Improvement Process

Over the last 30 years, there have been many new technologies and development paradigm changes aimed at improving software quality (the lottery ticket approach). While all of these advances have provided significant productivity improvements, there has not been an associated improvement in quality.

Continuing with the financial analogy, the solution to financial debt requires you to: create a budget (set goals), lower spending (change behavior), pay down debt (reap benefits). Improving the quality of software requires the same measured approach.

When organizations think about improving software quality for legacy code bases, they are often overwhelmed by the scope of the problem. Technical Debt has been accrued over many years with many developers contributing to the problem, so it is unreasonable to hope for an immediate improvement. Quality Improvement must be seen as a process that is measured by incremental improvement, not a single activity that magically transforms a code base.

The first step is to develop a quality plan to minimize new technical debt as legacy applications are modified and new development is undertaken. The rest of this paper will focus on actionable ideas for developing such a plan.

5.1 Best Practices from Industry

Interestingly, many industries that build safety-critical software such as avionics, automotive, medical device, aerospace, railway, and industrial controls, have already formalized best practices to produce dependable software.

These best practices are formalized in development standards such as: DO-178B/C (Avionics), ISO 26262 (Automotive), IEC 615108 (Industrial Controls), FDA and IEC 62304 (Medical Devices) and the CENELEC standard (Rail Applications). While there standards do not ensure bug-free software, they certainly provide a framework for a repeatable process that improves quality.

If you look at these standards, they have three main objectives:

- Tests should prove that an application complies with its formal requirements
- Code coverage should be measured to ensure that testing is complete
- Each deployed version should undergo complete testing

5.2 Best Practices from Experience

From my 30 years building software and software tools, I would add the following objectives to the list:

- Requirements must be complete and correct
- Coding style and architecture should be easy-to-understand
- Testing must be part of everyone's responsibility
- Testing must be Automated and Continuous

6 A Practical Approach to Software Quality

6.1 Ensure Requirement Completeness

Many defects will be prevented by ensuring software requirements are complete. Take for example, writing the specifications for a square root function. While this is a simple function within a math library, it can be very poorly implemented if the requirements are not complete.

Example:

> *The square_root() function shall return the square root of its input for all valid values.*

This requirement seems simple, but what range of values should be supported? Will the inputs be 32 or 64 bit values? What do I do with out-of-range inputs? Should I log any errors? How do we log errors?

A better specification would be:

> *The square_root() function shall return the square root of its input for all valid inputs, and 0 for all invalid inputs. Valid inputs are positive 32-bit floating point numbers, zero and positive infinity. Invalid inputs are negative numbers, negative infinity and NaN. In the event of an error, the type of error and system time should be logged using the common error logging system.*

Based on this revised specification, an architect is able to build a set of low-level requirements and test cases that will validate correct performance, and the developer will have a clear understanding of the implementation required.

6.2 Write Simple Code

For code to have improved testability, it must be flexible and easy to understand. In the early days of software development, developers were held in high regard if they could make software small and fast. This was in a time when computer systems had limited CPU clock speed and limited memory. Engineers needed to be efficient to fit code onto the device. Programming tricks abounded, as did really hard to understand code!

As application size and lifecycles become larger and longer, there should be a premium on building code that is easy to understand and maintain. Fortunately, making code easy to understand dramatically improves testability.

Bob Gray of consulting firm Virtual Solutions said that “Writing in C or C++ is like running a chain saw with all the safety guards removed.”[4] Some of these “safety guards” for coding style that will improve testability are:

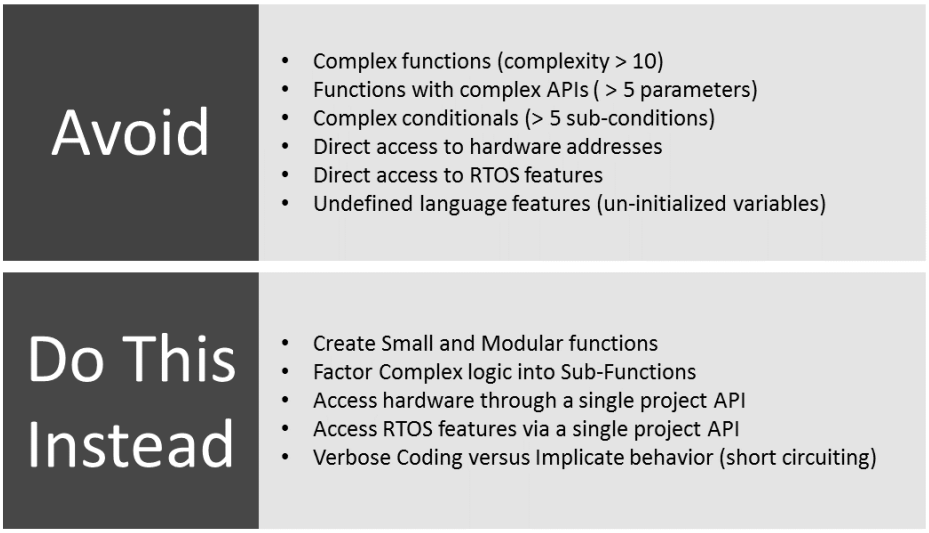


Figure 2: Follow these tips to help you write easily testable code

Every engineer will have their own idea of what simple means, so it is important that development groups implement their own set of rules, and it is critical that these rules be formalized with automated checks or at least peer review checklists.

[4] Byte (1998) Vol. 23 Nr 1-4. P. 70

6.3 Implement Quality Gates

Creating a culture of quality requires documented workflows for each team member that addresses his or her part of the quality challenge. For example, developers should have a set of prerequisites to meet prior to making commits to Configuration Management (CM). Without any restrictions, or "gates" on behavior, code bases will very quickly become buggy. On the other hand, if there are too many restrictions, a new feature might never get released.

The following diagram shows a reasonable set of quality gates to be met before code changes are committed to CM:

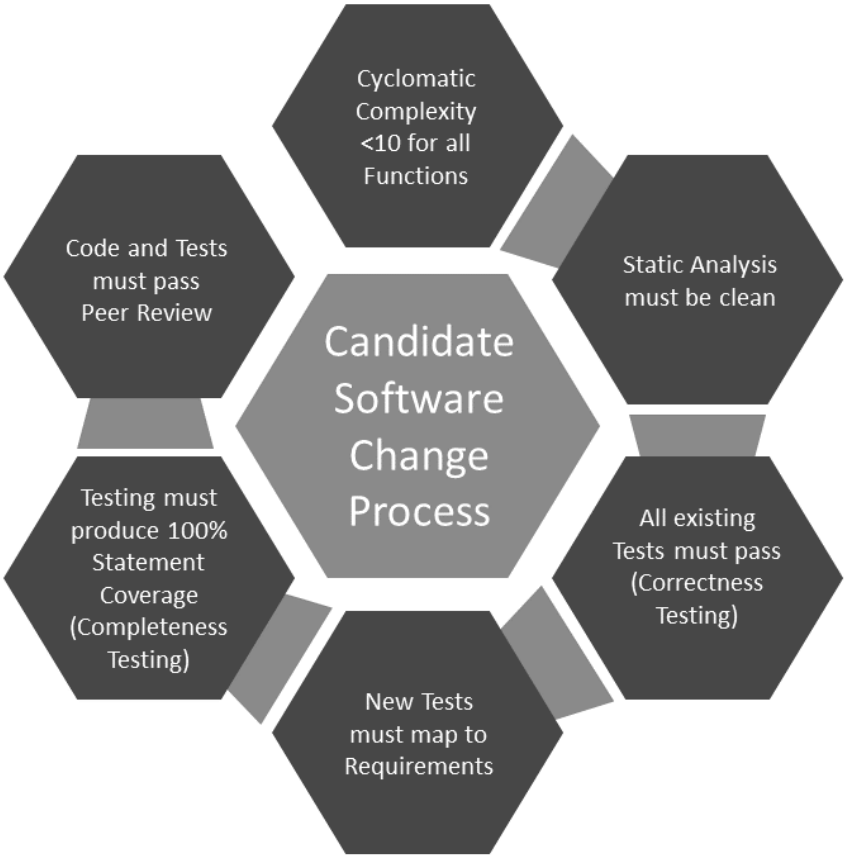


Figure 3: Candidate software change process

It is ideal if the quality gates can be automated with a tool, but it is not critical, and it is certainly not a reason to delay implementing any gates. As a first step, document a process, implement a checklist that reflects the process, and apply the checklist via peer review.

6.4 Develop Achievable Goals and Communicate the Rationale

Many people will look at Figure 3 and think that implementing this workflow is impossible. "We don't have tools." "We don't work that way." "We'll never get anything done." These excuses cannot be tolerated if you want to improve quality.

Whether you have a 20 year old code base, or all new code, you must create a workflow that makes sense for the organizations goals, and one that will be embraced by the team. One size does not fit all. For example, with a legacy application that has no formal requirements, limited test cases, and no test automation, it doesn't make sense to choose a goal of 100% code coverage. But a goal of 100% coverage for all "new" code does make sense.

Transitioning to a quality culture may be difficult for developers who consider the new "gates" to be more work and time consuming. As you start to introduce new processes, mentoring and training can help ease the transition. Create a group of senior team members to lead the transition, leverage training and tools, and validate the new workflow by comparing quality metrics for projects that adopt this new workflow against historical trends.

Hint: Start with team members who have an early adopter mentality.

Additionally it is critical to communicate the reasons for change with the whole team. A simple and honest message is best:

> *"We just lost a long term customer because they're unhappy with our buggy software. If we don't change we'll be out of business in a year."*

Well I hope things aren't that dire for your group, but the point is people need to understand the motivation for changes, and that improvement requires change.

6.5 Publish Metrics

The biggest thing that you can do to speed adoption is to publish data to the whole team that show tangible results. Focus on Good, Fast, and Cheap (from the project management triangle). The following are some metric ideas:

Good: Bugs found by Customer, QA, and Developers
Fast: Average time for a branch to get through QA
Cheap: Percentage of staff time used for bug fixing versus new feature development

More important than the metrics that you choose to publish is the story that they tell, choose metrics that will move over short periods of time and publish trend data.

Consider an application that has been deployed to customers, but is still being enhanced with a three-month release cycle. Publishing the number of bugs found by customers compared to those found by developers or internal testing would provide actionable data for your team.

The following graph shows a reduction in customer reported bugs on each sequential release. The reduction in customer reported bugs tells your team that their quality efforts are working and will encourage them to continue.

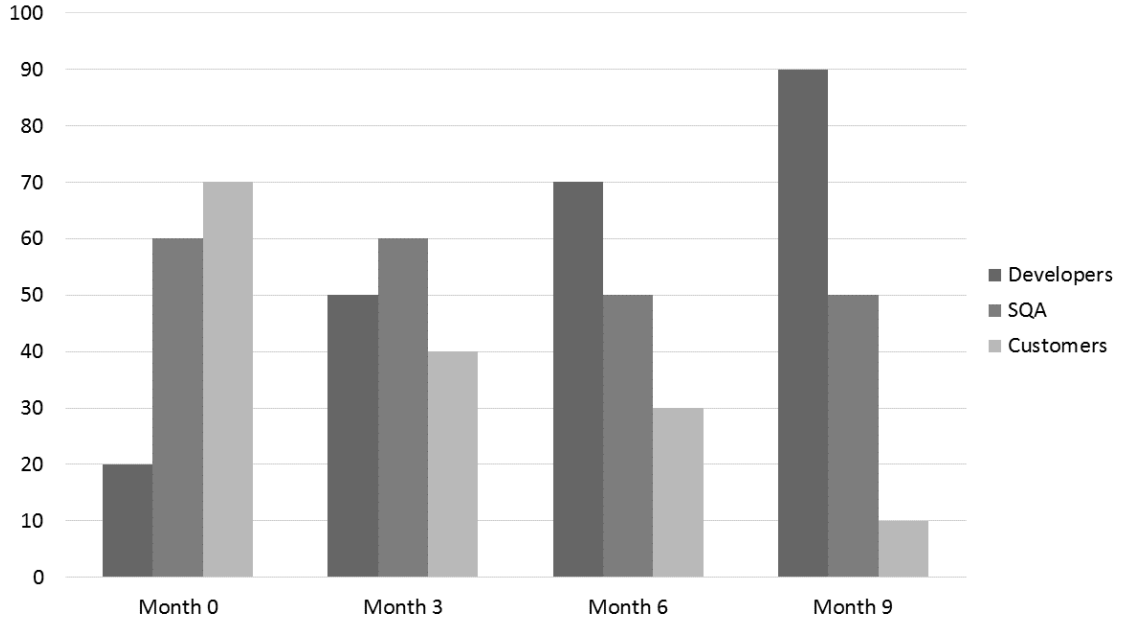


Figure 4: Fixing bugs before they are shipped is the most cost-effective way to deliver quality

If the data shows a negative trend this is also useful, as it helps the team understand the scope of the problem and accept that change is needed.

6.6 Test to Requirements

In the earlier section on requirements, I discussed the benefit of well written and complete requirements to ensure that a developer has all of the information to create a quality implementation. The second benefit of good requirements is that they provide a framework for testing. In fact the software development standards such as DO-178 (for avionics) and ISO 26262 (for automotive) specifically require that tests, and code, be mapped to requirements.

Mapping tests to requirements provides several benefits:

- **It ensures that tests validate what the software is supposed to do, not what it does do**. One of the big problems with developer-based testing is that the test values are often chosen by looking at the implementation rather than the requirements.
- **It ensures that there are not any forgotten cases**. If the requirement says that there are 100 different values for a message ID, shouldn't we have tests for all 100 values?
- **It promotes the testing of functional boundary cases**. Everyone agrees that software most always fails at boundary values, not at nominal values. In addition to testing at machine boundaries: (e.g. max Integer) it is important to test functional boundaries (e.g. max Speed)
- **It allows managers to understand which tests need to be re-run when requirements change**.

6.7 Measure Code Coverage

Code coverage analysis might have the highest return on investment of any development process change. Modern tools make it easy to implement code coverage as part of the build process, allowing coverage measurement with no changes to the test process.

One of the most misunderstood issues with code coverage is its relevance to software quality. Even within groups that must adhere to software development standards such as DO-178 (for avionics) and ISO 26262 (for automotive) there is confusion. So let me state it simply:

100% Code Coverage should not be the goal of software testing;
it should be the result of complete testing.

What this means is that Code Coverage Analysis goes hand-in-hand with the Requirements Based Testing described in the previous section. The only way to judge the sufficiency of your requirements based tests is by measuring the resultant code coverage.

Gaps in the code coverage might point to poorly implemented tests, which don't adequately test the requirements, but the gaps might also be the result of "extra" code (e.g., the requirement indicates there are 100 message IDs that should be handled, but the developer added 10 new ones and the requirement never got updated).

Code coverage data is also a great choice as an initial metric to publish to your team. As with everything I have mentioned so far, it is not important what the initial value is, but it is important to show coverage trending up to validate the "extra work" your team is doing. If you are not measuring code coverage, then you are testing "blind".

6.8 Speak the Language of Test

Everyone working on a software project has something to contribute to software quality. The system architects understand the big picture about how the application should function. The developers understand the implementation decisions made and how they will affect performance. The QA team understands the correctness of the application and the interdependencies between functions that might not be obvious to other team members.

It is important to involve all of these team members in developing the tests which will prove the correctness of the application.

Written language has been a boon to the development of the human race, but it is a really inefficient way to describe software bugs. One of the largest inefficiencies in the average development team is communicating efficiently. Read through any random bug report in your system, and it is likely that you'll often see the following timeline:

1. Bug found: QA sees something that is broken
2. Bug documented: A description of the bug is entered into the bug tracking system
3. Developer gets bug: Developer tries to duplicate the bug, can't, asks for more details
4. More details added: QA adds more details
5. Bug duplicated: Developer duplicates the bug
6. Developer fixes bug: Changes committed to fix the bug
7. Fix sent to QA: Fix provides a partial solution or has negative side effects.
8. Bug sent to developer: …

You might get dizzy following this ping pong match of bug assignment.

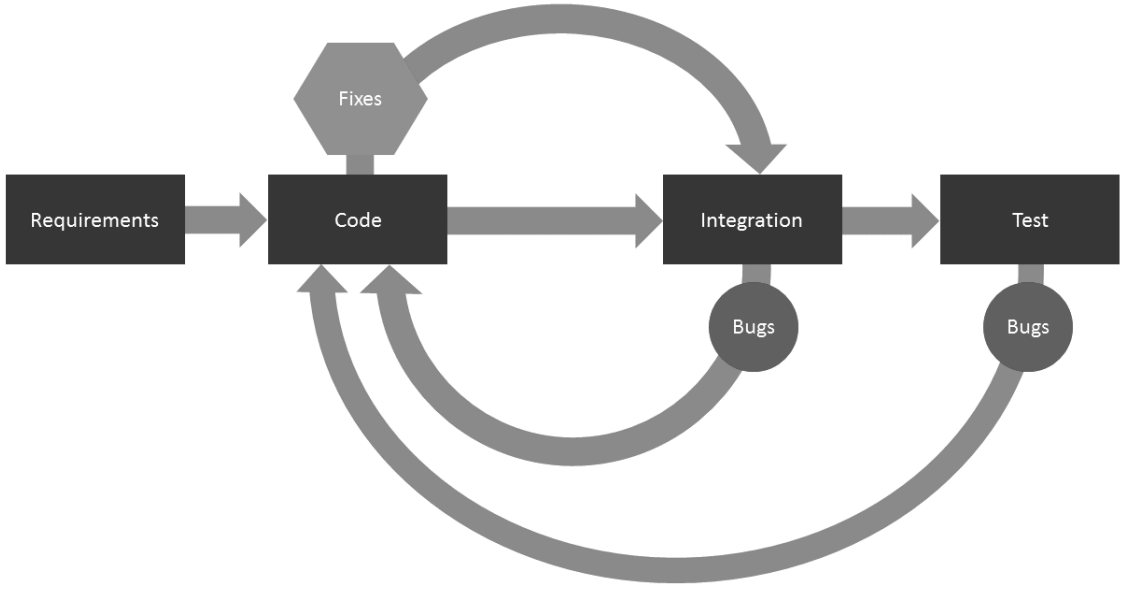


Figure 5: The back and forth game that many bugs experience.

It is much more efficient if test cases are used to document the initial bug rather than a text description. It's even better if the developer can easily run that same test. And it's perfect if the developer can run the test under control of the debugger to find the problem.

This workflow is what I call *Speaking the Language of Test*. It removes all of the inefficiencies that result from team members trying to describe a problem with words, and it gets rid of the back and forth Ping-Pong game that many bugs experience.

6.9 Implement Test Automation

In order for software testing to improve the quality of applications, it has to be thorough, easy and fast. This enables developers to have the ability to run any test, at any time, on any version of the code.

Every organization has developed a software build system allowing for unattended incremental application building, but most have not implemented a repeatable incremental testing infrastructure. Too often, testing is performed periodically with manual processes required, rather than constantly and incrementally with complete automation.

Each developer often implements their own testing methodology, and these methods are often nothing like what QA is using to test. If there is no organization-wide platform for testing, there is a significant lag between when bug is introduced and when it is found. The longer this lag time, the more difficult it is to find the cause, and the harder it is to fix. Ideally, each change to the software runs all tests that are affected by that change before the change is integrated.

6.10 Change Based Testing

Even with complete test automation it is likely that running "all tests" will take hours or days. Faced with this situation, it is reasonable that most groups run tests periodically rather than constantly. In fact, telling a developer that changed one line of code that they need to run a week of testing is a great way to ensure developers see testing as the enemy.

Change based testing solves this exact problem. It ensures that the quantity of testing is in proportion to the source changes by automatically choosing the sub-set of tests that are affected by a change and running only those tests. For small to mid-sized changes, test cycles take minutes rather than days.

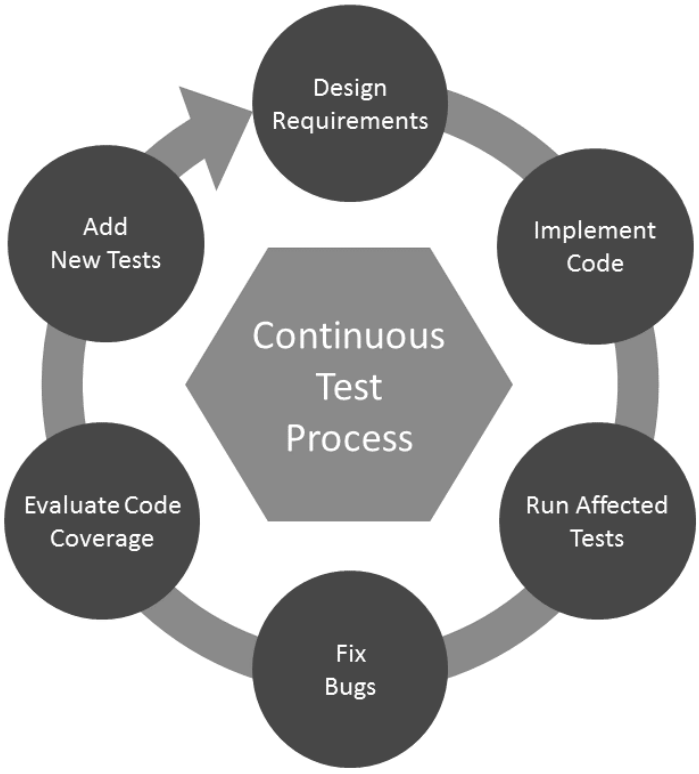


Figure 6: A continuous test process reduces development time

6.11 Invest in Development Tools

Over the last several years, there has been a steady flow of new tools for software testing. In order to create a quality culture, there are several of these tools that should exist within an organization's software toolbox.

The first is a **static analysis tool.** These tools are able to parse code to find problematic patterns within the source code that commonly result in bugs. In fact several coding standards have been developed by the industry to identify these patterns. For example, the MISRA standard[5] adopted by the automotive industry provides a set of rules that prohibit certain code constructs that are either ambiguous in the language standard or error prone in implementation.

The second is a **code coverage tool**. These tools add instrumentation bread crumbs to a code base during testing to document the statements and branch outcomes that have been tested. They make it easy to focus new test efforts on inadequately tested portions of an application.

The third is a **test automation platform**. If your team is going to *Speak the Language of Test* it is critical that they have a common platform to allow them to communicate. A test platform should support the easy creation and execution of black-box and white-box testing to ensure application correctness. These tools, among others, will help ease and ensure the process of better quality software.

6.12 Throw Away Your Source Code

Historically development teams produced source code as their work product. Emphasis has always been on the implementation details. While clearly the implementation is critically important, I would suggest that the emphasis should be placed on the Applications Programming Interface (API) and test cases. If a robust and well documented API is created and test cases are built to formalize the correct behavior of this API, then the implementation is really secondary.

In fact, I would argue to use your most senior developers for API and test design and utilize lower-cost, junior developers for the implementation. Changing the emphasis to the API will increase software reuse and decrease life cycle maintenance costs.

7 Business Advantages of Quality Improvement

There are many compelling business reasons for improving application quality. In our hyper-competitive global economy customer satisfaction is being driven like never before by software. Software is the primary controller of the human interface experience with the majority of electronic devices, and for many devices software is the brand.

Was the success of Apple's iPhone driven primarily by the hardware choices they made or the incredible ease-of-use provided by their ground breaking iOS software? Clearly the software played a huge role.

Improving software quality will result in the following business benefits:

- Fewer bugs go to integration, which shortens integration time
- Shorter integration time means faster release cycles
- Fewer bugs go to customers, leading to happier customers
- Happier customers lead to increased revenue and brand loyalty

[5] http://www.misra.org.uk/

8 Summary

If you are in the business of building software, you have a quality problem. It's that simple. There are no bug free applications. As with any problem, the first step to improvement is to recognize the problem, and then identify a plan for improvement, implement that plan and measure results.

In this paper, I have tried to provide a variety of process improvement initiatives that can be easily adopted by any development group, big or small, but in addition to a process change it is critical to invest in an attitude change. Quality requires a team effort, not just some magic dust from the QA department.

There are two key take-away items from this discussion:

- There are no silver bullets to improving quality
- If everyone on the team has a stake in testing and quality, a higher quality product will be the end result

Building high quality software is not an easy task; it requires a constant engagement in process improvement and metrics measurement. But creating a process that emphasizes testing throughout the development process will result in applications that are released faster with quality "baked in."

References

Byte 1998 Vol. 23 No 1-4: 70

Cambridge University Study States Software Bugs cost Economy $312 Billion per Year. Financial Content. January 8, 2013. http://www.jbs.cam.ac.uk/media/2013/financial-content-cambridge-university-study-states-software-bugs-cost-economy-312-billion-per-year/

Information is Beautiful, "Codebases," http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

McCormick, John. "Software Quality – By the Numbers." *eWeek. March 4, 2004.* http://www.eweek.com/c/a/Web-Services-Web-20-and-SOA/Software-QualitymdashBy-The-Numbers

Misra, "Misra Home," http://www.misra.org.uk/

Wikipedia, "Heartbleed," https://en.wikipedia.org/wiki/Heartbleed#Affected_OpenSSL_installations

Wikipedia, "Technical Debt," https://en.wikipedia.org/wiki/Technical_debt

House of Quality for Complex Software Development Decisions in Mobile, Wearables and IoT

Manini Sharma, Shruti Chandna

manini.sharma@intel.com

Abstract

The process of decision making for complex software such as that for mobile devices, Wearables and IoT (Internet of Things), is a challenging task. The software design complexity and high level of interdependencies, along with crunched schedules and market pressures, make decision making even harder. How can we take decisions quickly that are technically accurate, address market needs and also communicate effectively to management?

House of Quality (HoQ) is a popular tool in the manufacturing world that enables quick decision making, better product definition, and strategically prioritizing efforts while also being an effective communication tool among stakeholders including marketing, engineering and senior management. At Intel, while developing software for mobile features, optimization of power-and-performance (PnP) are given high importance. We decided to validate HoQ for one of the software trade-off decisions that was highly complex and took significant time and resources to arrive at.

The paper explains how the product development team built the House of Quality to verify the decision taken by making a single picture, covering several dimensions and comparing the ease of taking this complex decision through HoQ. We find that the learnings are apt for decision making in such high complexity and schedule constrained world of mobile devices, Wearables and IoT.

Biography

Manini Sharma is a Platform Software Quality Engineer at Intel, based in Bangalore, India. She works on software for Tablets and Internet of Things based on Intel Architecture. She's worked in a variety of roles in the past decade including Product Marketing, Software Development and Application Engineering. She's been focused on Software Quality for the past four years.

She has an M.Tech. in Information Technology from IIIT-Bangalore and is an MBA candidate from IIM Kozhikode.

Shruti Chandna is an Engineering Manager at Intel India (Bangalore) with extensive experience in product development along with expertise in system architecture, operating systems and embedded software development related roles. She is currently managing teams which are responsible for optimizing power and performance of Intel (Core & Atom) products across all market segments (such as phones, tablets, laptops, desktops, etc.)

Copyright Manini Sharma June 10, 2015

1 Introduction

Quality Function Deployment (QFD) has been around since 1960s having multiple manifestations (Quality Function Deployment 2015). However, it's based on a single unambiguous concept; that of keeping customer needs central to product decisions. "House of Quality" (HoQ) is a matrix tool to implement QFD. It was first developed by Mitsubishi Heavy Industries in 1972 and been used by several manufacturing organizations, such as AT&T, HP and GM (Hauser and Clausing 1988). HoQ, in itself has gone through changes. Modern QFD recommends non-matrix tools, such as the Blitz QFD® (Blitz QFD (R) 2015).

For this paper, we apply the HoQ to a power-and-performance (PnP) scenario to demonstrate the benefits that can be derived for mobile devices and discuss its applicability to Wearables and Internet of Things. The paper is structured as follows: in section 2 we elaborate on the problem of complexity in decision making and section 3 details the House of Quality concept. Section 4 is dedicated to the power and performance case study, in section 5 we discuss the key benefits and limitations of the HoQ and finally in section 6 we elaborate on applicability of HoQ to Wearables & IoT.

2 Problem: Large amount of unstructured data slowing down decision making

With new innovations in technology happening alongside rapid go-to-market cycles (Stamford 2014), organizations are aggressively competing to get to the market first. New breakthroughs require companies to morph quickly and accept new realities or else wilt away (Kocher, Chris 2014). In such fast paced environment, strategic and product design decisions also need to be made fast.

At the same time, information overload makes it difficult to understand a problem and to make decisions. Some cognitive scientists highlight the distinction between raw information and information in a form we can use. Decision makers performing complex tasks would need excess cognitive capacity to process the raw or disorganized information. Hence information overload may be better viewed as organization under-load (Wikipedia 2015). Therefore, there is need for a structured and intuitive way to organize key information such that relevant information becomes apparent and enables accurate decision making.

3 Structuring data using House of Quality

To structure the information for decision making, we investigate the ability of HoQ tool to organize the information and accelerate the decision making process. While there are slight variations in the way the HoQ is created (N. R. Tague 2005) (Lowe 2000) (Carroll 2007), for this case study, we have used it mostly as per the Harvard Business Review article on HoQ (Hauser and Clausing 1988) with some optimizations elaborated later in this paper.

To create this HoQ, first the customer needs are identified and prioritized. This can be achieved using quality tools such as affinity diagram, matrix diagram, etc. (N. R. Tague 2014) The result is a structured list of customer attributes, often in the words of the customers. Each of the customer attributes have a priority value assigned, representing its relative importance to the customer. Another list is created for engineering characteristics or specifications of the product, as measured by the organization. A matrix is then developed for the interrelationship between these "Customer Attributes" (CAs) and "Engineering Characteristics" (ECs). The ECs in this matrix are picked based on the impact that they have on the identified CAs. Customers' perception of the product and its competitors is documented to the right of the matrix, which helps in understanding the product's relative positioning. Technical complexity index and competitive specification are added below the matrix to decide on the final engineering targets. A roof is added on top of the ECs for inter-relationships between them. Finally, the matrix is filled with weighted value of the interrelationship between the CA and EC. See Figure 1 below for the structure. Decisions are taken by identifying trade-offs. (Hauser and Clausing 1988)

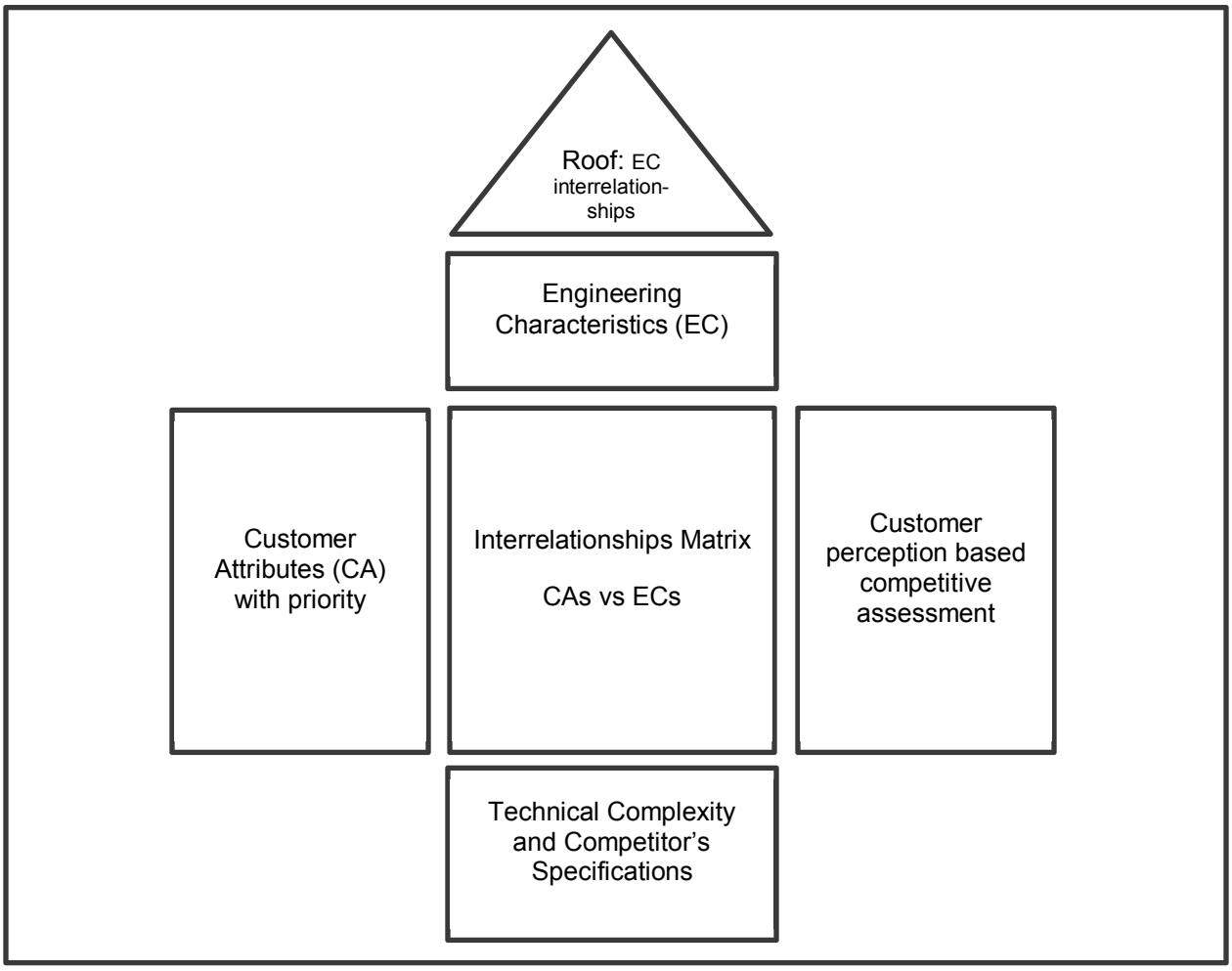


Figure 1: Framework of the House of Quality (Hauser and Clausing 1988)

4 Case study: PnP Trade-offs

4.1 Background

Battery life and high performance are both top considerations for mobile devices (Atluri, et al. 2012). Manufacturers of mobile devices constantly work towards adding new features, improving overall performance and battery life of their products.

It has been established that battery life is inversely proportional to power consumed, which in turn is directly proportional to the performance of the device (Puttaswamy, Choi and Park 2002). Hence, tradeoffs need to be made between power, performance and product features. A poor design choice without considering quantitative impact of interdependent engineering aspects, customers, marketing feedback and competitive data, can lead to failure of the product in the market (Atluri, et al. 2012).

On one of the Intel mobile device platforms, the 'Video Record' use case had a significant gap between the desired and actual power utilization. The excess power consumption reduced battery life of the device by 20%. Since the product launch date was fast approaching, a quick decision needed to be taken on possible trade-offs to close this gap in battery life. For decision making, inputs were needed from multiple teams - engineering, program management, product validation and marketing. With such a large set of

stakeholders, it took weeks to go over complex interdependencies and get consensus. The decision was finally reached just in time for product launch.

4.2 Building the HoQ

For enabling faster decisions in the future, the team decided to validate their decision using HoQ. This was done to pilot the use of HoQ in software product development, as well as to evaluate if HoQ could speed up the decision making process.

The process of building the HoQ was as below:

1. Finding what the Customer Attributes of this context are: The marketing team provided this list for the Video Record use case and helped prioritize the CAs.
2. Identifying the dependencies and building the EC list: This refers to the correlation between SoC (System-on-Chip) and components with respect to CAs supported by the platform. The EC list was built based on this. This was an engineering input. The identified CAs and ECs were placed as headers in the interrelationship matrix.
3. Entering the CA data: CA data is entered on the right and shows the customer's perception of the CAs for the Intel and competitor product. These were taken from the competitive analysis data which was already done.
4. Entering the EC data: EC data is entered at the bottom, showing the technical specifications between Intel and the competitor for each identified EC.
5. Technical difficulty grading: Complexity of developing or changing the given ECs was graded on a 10 point scale. This helped in gauging the effort that would be needed to trade-off a given EC. These were placed at the bottom as well.
6. Creating the roof with ECs correlations: The interrelationships between the ECs were created by the Engineering team.
7. Filling in the Matrix: This was done based on joint discussion between the Technical-Marketing and Engineering team.

The HoQ that emerged is shown in Figure 2.

The time taken to build the matrix was couple of hours' activity from Engineering as well as Marketing teams each. Comparing that to the number of meetings and discussions that were originally done, the speed gained was significant – from weeks of meetings and email exchanges to just few hours. The final step was to discuss the HoQ and decide how to attain the additional battery life.

4.3 Deriving the trade-off decision based on HoQ:

There is a direct correlation between battery life and the power utilization. Power, in turn, is dependent on various SoC and platform components on a mobile device. Looking at the HoQ, it became clear that there were several ways to reduce the power utilization:

1. Power is directly (positively) correlated to the number and frequency of various SoC components such as processor cores, Graphics Processing Unit (GPU), and Image Signal Processor (ISP) and platform components such as memory frequency. Therefore, in order to reduce the power utilization, we could reduce the frequency at which these components run.
2. Power is also impacted by the resolution of the camera sensors. Higher the resolution, more the power utilization. Hence, in order to reduce power utilization, we could reduce camera resolution.
3. Another way to address this problem is to increase the battery size. Increasing the battery size would offset the higher power consumption, but increase the cost and also the dimensions of the device.
4. The display also uses power. Higher resolution displays consume higher power. Hence, lowering the display resolution could reduce power consumption.

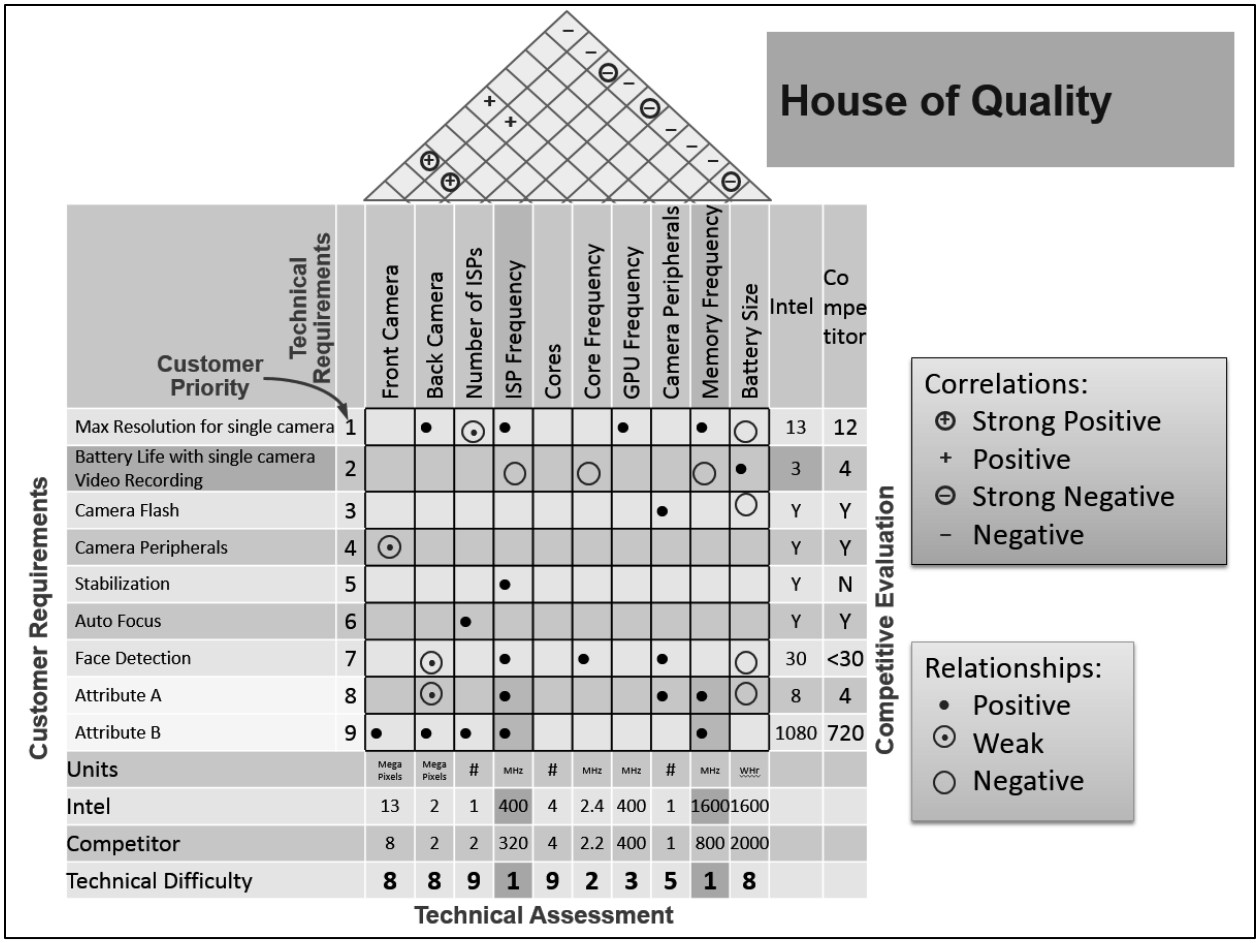


Figure 2: House of Quality built for the PnP Scenario

The following were inferred by taking a quick glance at the HoQ:

1. The CA - 'Battery life during video recording', is strongly correlated with ISP frequency, Core Frequency and Memory Frequency. Reducing these ECs would improve the battery life.
2. The changes to ISP Frequency and Memory Frequency ECs are the lowest in technical difficulty, hence would be easier to modify in software given the crunched schedule.
3. The lowest priority features of Attribute A and Attribute B have a strong correlation with ISP Frequency and Memory Frequency. These features are driving the high ISP Frequency and Memory Frequency values.
4. Based on competitive analysis, Intel's product is significantly higher in ISP Frequency (400MHz vs 320MHz) and Memory Frequency (1600MHz vs 800MHz). This gives Intel leeway to trade-off these characteristics to improve battery life.
5. Removing the low priority features like Attribute A and Attribute B (masked for confidentiality reasons) would remove the need for running ISP and Memory at high frequencies and thus reduce overall power.
6. A quick check of the impact of these changes is done by reviewing the other dependent CAs, such as Stabilization, Face Detection and Max Resolution for Single Camera.

Hence, in one discussion, the decision of this trade-off was validated. Had the team used the HoQ for the original decision making process, this decision could've been reached much faster and with same accuracy.

One of the key stakeholders, who approves these decisions, and runs the Change Control Board, is the Program Office. Using this visual tool to explain the rationale behind the decision as well as the technical and marketing impact led to quick approval and execution of the proposed modifications.

5 Optimizations, benefits, and limitations

5.1 Optimizations to "Slim Down" the HoQ:

Critics of the HoQ often prescribe to the rigidity of its structure and need for comprehensive information (What Software is Best for QFD 2015). However, as seen in the case of PnP scenario, these can be overcome by focusing on the essential and discarding the non-essential aspects of the HoQ based on context.

One may ask, why build the matrix/house at all, if we will discard a large chunk of the data? The answer is, we need to visually and mentally be aware of the customer attributes and engineering characteristics while making the trade-off decisions. The decision-making process includes making sure, with reasonable confidence that the dropped attributes or characteristics are in fact, low priority and low risk. Keeping the context of customer need central to the building of the matrix guides the way to creating a suitable HoQ.

To apply the HoQ to the PnP case, some optimizations were made. These were departures from the originally referenced HoQ diagram (Hauser and Clausing 1988):

1. Although display resolution has a positive correlation with power consumption, it was fundamental to the product specification and could not be altered; hence it was excluded from the HoQ EC dimension.
2. While battery size was included in the EC dimension, it was ruled out as an option because it would've led to increased cost, larger dimensions and additional weight.
3. Some of the EC changes such as number of cores, GPUs, ISP, etc. that would need complex hardware changes, were also ruled out immediately.
4. ECs that had dependencies on other ECs impacting high priority CAs, were kept in the HoQ. CAs or ECs that customers find critical or are central to the product offering were also kept in the HoQ.
5. The comparison of low priority CAs (Attributes A & B) dependent on low technical complexity ECs (ISP frequency & memory frequency) helped find the right trade-off between product features and power, which was acceptable to both customers and engineering teams without compromising on project timelines.
6. The interrelationships diagram uses "Positive", "Weak" and "Negative" nomenclatures instead of High, Weak, Low. This was found to be more appropriate for this context. Note: Here Weak implies there is almost no relationship.
7. We didn't use all the elements of HoQ, like directions on the ECs (which direction improves the product – higher/larger versus lower/smaller). These were not relevant to the problem at hand and thus, were unnecessary.
8. Mathematical weight calculations became unnecessary since the decision making didn't require it.

Such optimizations were made to achieve focus and speed in decision making. Similar optimizations can be applied to "slim down" the HoQ relevant to the context in which it is being applied.

5.2 Key Learnings and Benefits Derived:

There were several learnings and benefits from building the HoQ in the PnP scenario:

1. The use of HoQ significantly reduces the effort to bring out the complex interdependencies by creating a structured and objective view of the problem which helps in making quick decisions.

2. If the data needed for building the basic structure of the HoQ is available, the decision making process can be accelerated. Hence, assigning priorities to customer requirements and including market survey results to the existing product development documentation would be beneficial.
3. If the data needed for building the HoQ is not available, trying to build the HoQ will make this immediately obvious: the decision cannot be made in a fact-based way at this point. Without the clarity of the HoQ, this could go unnoticed in weeks of meetings and email chains, or worse, lead to gut-based, random decisions.
4. The Engineering teams are most suited to drive this process. They can establish the list of ECs and take responsibility of filling in the interrelationship matrix between ECs and CAs. This helps in accelerating the decision making process.
5. Using HoQ in day-to-day processes like software qualification can help in quickly assessing whether missing a particular goal is acceptable or not. It can also help teams streamline efforts towards working on high importance tasks.
6. This single visual method of organizing the complex information and decision making makes it easy for key stakeholders to understand the problem and rationale behind the decision.

5.3 Criticism and Limitations of HoQ

The HoQ tool has received a fair amount of criticism. The diagram of HoQ and the amount of information it holds can increase exponentially. For a 1000x1200 dimension matrix, the number of intersecting cells is 1.2 million. It has been found that typically 95% of the data that is filled in the intersecting matrix is unimportant. (What Software is Best for QFD 2015) Hence, spending one's energy on comprehensiveness of this tool may become time-consuming and exhausting. The complexity of its output may not yield the best decision easily.

At the same time, reducing the dimensions and oversimplification of its use, can lead to poor decisions; defeating the entire purpose of the tool (Modern QFD and Traditional QFD 2015). The challenge is thus to find the right balance between over simplification and exploding complexity.

Another criticism is that the use of ordinal scale in customer input may not reflect the correct mathematical interpretation of actual need. E.g. If customers are asked to rank features in order of importance, something that is ranked 4 may not be twice as important as something that is ranked 2. (Modern QFD and Traditional QFD 2015).

These criticisms are founded in sound logic. However, by intelligent and balanced use of the HoQ, making sure that the tailoring being done meets the need of the situation at hand, without oversimplifying or overcomplicating, one can arrive at decisions quickly and with high accuracy. As with any tool, the key to success lies in the user's skills and knowledge of the tool. This will be the case when the users are stakeholders who have full context knowledge and know what areas need attention and which are unimportant (N. R. Tague 2005). Similarly, when using statistical techniques, the numbers generated from the HoQ can be correctly interpreted only when the stakeholders provide correct values.

At the same time, one must remember that HoQ is only a tool for *getting to* a decision, not the decision itself. It structures and visualizes input data, and helps identify important interrelations that might go unnoticed otherwise. It helps organize the key information elements in an easily interpretable way. In the end, a human must make the decision. For example, in the PnP scenario, increasing the device size was not an option at all, however the numbers may look.

6 Extending the application of HoQ to Wearables and IoT

As seen in the case study, the application of HoQ can save considerable time in decision making in mobile devices context. The pilot showed a reduction in decision making time from weeks to few hours with the same accuracy. As in the case of mobile devices, Wearable product design and development process is complex, which is exaggerated by rapidly changing technology, society trends, life style and user behavior. This complex combination of triggers, including signs, novelty, technology, aesthetics,

and culture, interact with one another, leading to customers' desire for fashionable Wearables (Kao, et al. 2013). Here the importance of customer preferences cannot be overstated. The HoQ can be effectively used to make design and trade-off decisions, keeping the evolving fashion in mind. If there are conflicting customer requirements, the customers can be segmented and then prioritized based on target marketing strategy. The final priority of a customer attribute in the HoQ can be weighted based on relative priority of the customer segment (N. R. Tague 2005).

In the larger context of Internet of Things (IoT), companies are understanding the need to be nimble. Rapid evolution in IoT is expected to consume unanticipated resources that will diminish an organization's ability to add new core functionality. Slow adoption will slip schedules and slow down revenue generation. Evolving architecture, protocol wars and competing standards necessitate the need to evolve and adapt quickly (Kocher, Chris 2014). New use cases are emerging and products under development need to be modified on-the-fly to be competitive when they launch. Devices are expected to get smaller, less expensive and more integrated than before. This will have implications on Security, Privacy, and Complexity of platforms that need to be supported (Kocher, Chris 2014). Here, HoQ can enable quick decision making for adding functionality when products are under development. Trade-offs can be identified using HoQ to make the product attractive at the time of launch. Intelligent "slimming down" of the HoQ, based on the context of the product decision would help a team focus on the relevant information. By structuring the complex interdependencies through HoQ, we can enable the stakeholders to arrive at and align on a decision quickly.

7 Conclusion

The Intel mobile platform development team spent weeks of effort in dealing with complex interdependencies and meetings for getting consensus with stakeholders. This inspired a search for a tool that would speed up decision making in such scenarios. House of Quality tool was piloted to validate and verify whether the same or better decision could be arrived at, faster. The result from HoQ tool was perfectly in line with that of the rigorous lengthy process. The time taken was reduced from weeks to few hours.

We believe that the HoQ can be easily extended to other scenarios within mobile platform development and also to software development in general, especially for complex products like Wearables and IoT.

Acknowledgements

The authors would like to thank the reviewers for their valuable comments and suggestions to improve the quality of the paper. We also thank our organization, Intel Corporation, for the support and funding for this paper. We especially want to thank

- Rajesh Yawantikar – Manager, Intel Software Quality, Intel Corporation
- Ove Armbrust – Software Quality Engineer, Intel Corporation
- Amith Pulla – Program Manager, Intel Corporation

Vipul Batta – Program Manager, Intel Corporation

References

Atluri, Venkat, Richard Lee, Umit Cakmak, and Shekhar Varanasi. 2012. *Making Smartphones Brilliant: Ten Trends.* McKinsey & Company.

2015. *Blitz QFD (R).* QFD Institute. 04 20. Accessed 06 07, 2015. http://www.qfdi.org/what_is_qfd/blitz_qfd.html.

Carroll, Sue. 2007. "An Analytical Approach to Software Metrics Management." In *Fundamental Concepts for the Software Quality Engineer*, 181-197. Milwaukee, Wisconsin: ASQ Quality Press.

Hauser, R. John, and Don Clausing. 1988. *The House of Quality.* HBR. June. Accessed 06 07, 2015. https://hbr.org/1988/05/the-house-of-quality/ar/1.

Kao, Ching-Han, Chun-Ming Yang, Chen-Hao Hsieh, and Yu-Shan Hung. 2013. "Decision Making in the Design Process of Wearable IT Products." *International Association of Societies of Design Research (IASDR).* Tokyo, Japan.

Kocher, Chris. 2014. *The Internet of Things: Challenges and Opportunities.* Sand Hill Group. 11 17. Accessed 07 20, 2015. http://sandhill.com/article/the-internet-of-things-challenges-and-opportunities/.

Lowe, Dr. A. J. 2000. *QFD.* Webducate. Accessed 07 21, 2015. http://www.webducate.net/qfd/qfd.html.

2015. *Modern QFD and Traditional QFD.* QFD Institute. 03 14. Accessed 06 07, 2015. http://www.qfdi.org/newsletters/modern_qfd_and_traditional_qfd.html.

Puttaswamy, Kiran, Kyu-Won Choi, and Jun Cheol Park. 2002. "System Level Power-Performance Trade-Offs in Embedded Systems Using Voltage and Frequency Scaling of Off-Chip Buses and Memory." *ISSS 02.* Kyoto, Japan.

2015. *Quality Function Deployment.* Wikipedia. 04 06. Accessed 06 07, 2015. https://en.wikipedia.org/wiki/Quality_function_deployment.

Stamford. 2014. *Gartner Says a ThirtyFold Increase in InternetConnected Physical Devices by 2020 Will Significantly Alter How the Supply Chain Operates.* Gartner. 03 24. Accessed 07 20, 2015. http://www.gartner.com/newsroom/id/2688717.

Tague, Nancy R. 2005. "House of Quality." In *TheQuality Toolbox, 2nd Edition*, 304-314. Milwaukee, Wisconsin: ASQ Quality Press.

Tague, Nancy R. 2014. *Seven New Management and Planning Tools.* ASQ. Accessed 06 07, 2015. https://asq.org/learn-about-quality/new-management-planning-tools/overview/overview.html.

2015. *What Software is Best for QFD.* QFD Institute. 03 14. Accessed 06 07, 2015. http://www.qfdi.org/newsletters/what_software_is_best_for_qfd.html.

Wikipedia. 2015. *Information overload.* Wikipedia. 07 18. Accessed 07 20, 2015. https://en.wikipedia.org/wiki/Information_overload.

Perils of Legacy Design Description in an Increasing Agile World

Ray Miller, Solution Design Quality and Assurance SME
NTT DATA Federal
Ray.Miller@nttdata.com | Ray.Miller@jhu.edu

Abstract

Producing design documentation is a fact of life for systems and software development. But why is this documentation generated? And how much documentation is enough? If you query project managers, the response is likely to be contractual/regulatory requirements to the former and a verbose 20th century template to the latter. Ask software developers the same questions and you'll likely elicit a litany of tirades not suited for publication. There must be a better reason to allocate significant resources to something that is often a source of aggravation and typically relegated to shelfware.

The United States Citizenship and Immigration Services (USCIS), a Department of Homeland Security component at the forefront of the Agile and DevOps movement in the federal space, has embarked on a concerted effort to address these and other issues impacting post-legacy SDLC environments. The agency is committed to employing Lean Architecture, Lean Software Engineering and Lean Manufacturing techniques across the enterprise to ensure that maximum benefit is derived from its Agile and DevOps investments. This commitment includes a “Three Bears” approach to system and software documentation: not too little, not too much, but just enough.

This paper describes a broad Voice-of-the-Customer effort conducted by USCIS that resulted in an innovative paradigm shift to documenting software design; one grounded in Lean principles and Systems Thinking concepts that exceeded the expectations of most ardent Agile practitioners while satisfying applicable regulations.

Biography

Ray Miller is a Lean practitioner with significant and specialized experience supporting business and technical leadership to ensure that IT products and services are efficiently designed to meet or exceed consumer expectations. Ray’s versatile engagement history ranges from boutique firms to a Big-4 consultancy supporting military, federal and state agencies in 1) formulating architecture/engineering quality strategies and measures across the enterprise, 2) facilitating adoption of Lean Manufacturing techniques in DevOps environments, and 3) mentoring management and development teams in Lean Architecture and Lean Software Engineering best-practices.

In addition to quality (CMQ/OE), information security (CISM) and project (PMP) management proficiencies, Ray possesses practitioner-level expertise in solution architecture (TOGAF), software quality engineering (CSQE) and quality auditing (CQA). Ray is a contributing-member of the American Society for Quality (ASQ), the Information Systems Audit and Control Association (ISACA) and the Project Management Institute (PMI).

A featured speaker on emergent design quality and assurance topics and a recognized instructor in software quality engineering best-practices, Ray’s previous venues have included the Washington Press Club, DC Metro (WMATA) Headquarters and the International Software Quality Conference. Lastly, Ray is currently pursuing a graduate degree in a recently established Quality Management concentration at Johns Hopkins University.

1. Introduction

Software development methodologies and approaches have evolved significantly over the past decade. This is clearly evident in the rapid adoption of Agile and most recently, Lean techniques within United States Citizenship and Immigration Services (USCIS) and the commercial software development industry as a whole.

However, has essential documentation kept pace with this SDLC revolution? In the project management realm, we've seen the legacy Project Management Plan supplanted by a minimalist Project Oversight Plan to accommodate Lean/Agile methodologies. But what about the ubiquitous System Design Document (SDD)? Within the USCIS Delivery Assurance Transformation Group, the SDD is considered one of four essential artifacts in documenting a *software-based* technology solution:

1. System Design Document
2. User Guide
3. Operations Guide
4. Test Plan/Test Cases

2. SDD Voice-of-the-Customer

Based on the frustration voiced by Agile/Lean project teams during artifact reviews coupled with a corresponding escalation in SDD rework, an appraisal of the current (Legacy) SDD Template was deemed a priority by the USCIS CIO as well as Delivery Assurance Branch (DAB) leadership.

To address senior leadership's concerns, the DAB promptly formed a design team to conduct a broad Voice-of-the-Customer (VoC) effort to ascertain the Legacy SDD Template's *fitness for intended purpose or use* in modern software development environments. The VoC effort concentrated on three areas:

1. SDD Content
2. SDD Structure
3. SDD Platform

It is important to note that SDD process (who, what, when) and tooling (how) were intentionally excluded from this VoC effort as both are the purview of other USCIS branches.

2.1 VoC Findings

The Cause-and-Effect (fishbone) diagram **Figure 1** below depicts the findings of this comprehensive VoC effort:

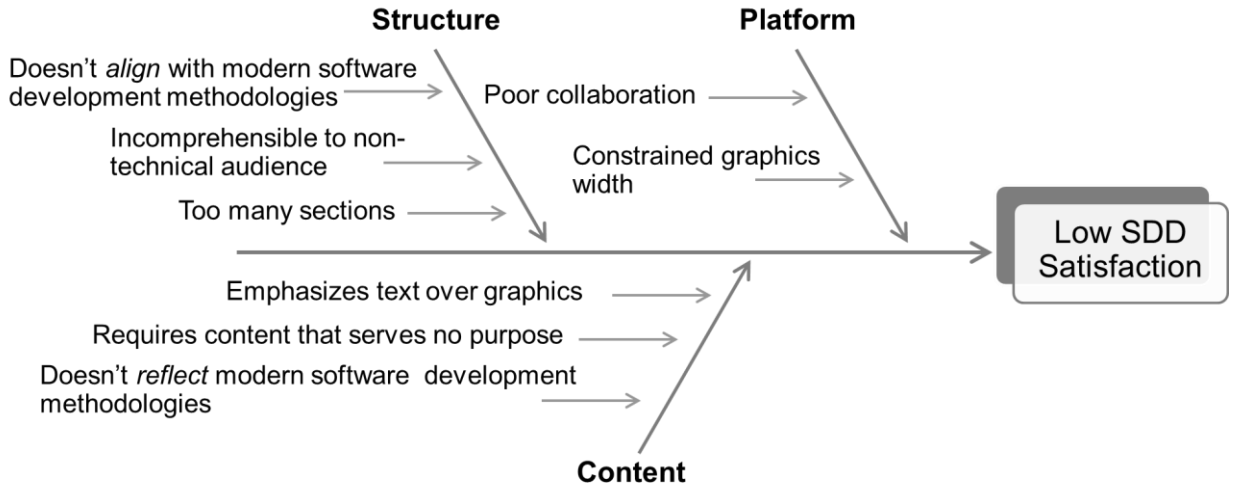


Figure 1: Legacy VoC findings

2.2 Key VoC Takeaways

Based on the in-depth analysis of VoC data collection artifacts, the Design Team surmised four principal takeaways:

2.2.1 The Legacy SDD Template does not embrace Agile values

The Agile Manifesto is an important milestone in software development methodologies. Of the four Agile Manifesto values, "Working software over comprehensive documentation" is most applicable to SDD development. Unfortunately, the Legacy SDD Template runs counter to this value statement as it employs a *kitchen sink* approach; a tailor-down strategy incorrectly assuming that its deeply layered structure of verbose sections subsections will accommodate any design description effort.

2.2.2 The Legacy SDD Template does not apply Lean principles

While the concept of Lean Software Engineering is fairly new to USCIS and commercial software development as a whole, its adoption is rate quickly increasing within mature and disciplined software development teams. Here too, the Legacy SDD Template does not embrace the seven Lean principles with particular emphasis on the first (eliminate waste) and the last (see the whole). Quite the contrary, the content expectation of the Legacy SDD Template is laden with waste and the structure does well to obscure "see the whole."

2.2.3 The Legacy SDD Template is monolithic

There are three generally-accepted methods to content inclusion in a design description artifact:

1. By Reference – Content is incorporated by URL, full path statement, or physical location identifier.
2. By Attachment – Content is incorporated to the rear of the artifact as one or more appendices.
3. By Embed – Content is incorporated into the appropriate main artifact section.

The structure of the Legacy SDD Template strongly encourages attachment or embeds, resulting in a populated artifact that has ranged to thousands of pages in size. Such resulting work product is promptly relegated to shelfware as few modern software developers would consider the artifact as having any practical value. Add to this, the difficulty of continually updating a document of such behemoth proportions and the result is an exercise in futility.

2.2.4 The Legacy SDD Template incorporates hardware into a software description

Modern solution architectures are partitioned into three domains or "layers":

1. Business – A description of the structure and interaction between the business processes and information needs.
2. Application – A *software*-engineered *capability*, described through viewpoints and *independent of any infrastructure*, that supports business process and information needs.
3. Infrastructure – A description of the structure and interconnection of compute and storage resources *at the virtual or physical device level*, provisioned to support the *resource needs* articulated in one or more Application domains.

The scope of any 21st century SDD should be solely the *application* layer. In the case of the Legacy SDD Template, it readily incorporates Infrastructure elements into the application domain. The result is a tightly bound hardware/software solution that fails to take into account the following reality:

> *In an era of cloud computing, modern solution architectures at the application layer must be described as* ***independent of virtual and physical devices****.*

Lastly, enterprise architecture purists may question the absence of the data architecture layer [1] within the above domain model. In DAB solution architecture: 1) Information architecture is incorporated into the business domain, and 2) Data architecture is incorporated into the application domain.

3. Ramifications of Legacy SDD Template Continued Use

The Design Team evaluated the Legacy SDD Template utilizing *fitness for intended purpose or use* as the measurement criteria. In the case of USCIS, the principal purpose and use of an SDD is threefold:

1. A contract document utilized in procurement of software development and maintenance services;
2. An essential onboarding artifact for team members new to the program or project;
3. A documented reflection of the system's current design state at a specific release cycle.

The Legacy SDD Template fails these requisite expectations on all counts; when used to describe a modern software-engineered *capability*, the Legacy SDD Template's predilection to hardware, antiquated structure, and extraneous content all combine to convolute any rational comprehension of system structure and behavior. To paraphrase a USCIS Manager, *to understand the (legacy) SDD, you must first understand the system,* which is analogous to putting the cart before the horse.

4. Advent of the Lean SDD

Based on the aforementioned VoC analysis and intended purpose/scope of a USCIS SDD, the DAB Manager directed the Design Team to *effect sufficient SDD change* which would result in a product that meets *consumer* needs and expectations while satisfying applicable regulations and ancillary stakeholders. Acting on this direction, the Design Team approached the SDD solution as one that would:

1. Adhere to Agile values and Lean principals,
2. Seamlessly integrate into Agile and Lean environments,
3. Remain backward compatible with legacy environments,
4. Require no specialized tooling to populate, and
5. Be *readily comprehendible* by management, practitioners, and ancillary stakeholders.

As a result of the discussion of the aforementioned five guiding principles, a consensus among the Design Team was reached to follow a Lean Engineering approach that would emphasize “eliminate the waste” and “see the whole” principles *in the context of an SDD*. This approach would satisfy the architectural description expectations of the Scaled Agile Framework [2] while remaining in compliance with DHS SDD content expectations.

4.1 Eliminate the Waste

From the “eliminate the waste” perspective, the Design Team resolved to model on *sufficiency criteria*, more commonly known as a “good enough” approach.

To “eliminate the waste,” the Design Team started with the SDD's largest areas of concern and worked until the expected benefit no longer warranted the effort expended (benefit/cost ratio). The four-step process is described in **Figure 2** below:

Figure 2: SDD ‘Eliminate the waste’ process

4.1.1 Properly scope the SDD

The first step was to properly *scope* the SDD as a solution architecture *application* domain artifact void of virtual or physical *device* references. The Design Team accomplished this by 1) *refining* the definition of the application domain, and 2) *unbinding* the application domain from the infrastructure domain while retaining a *loose-coupling*. The textual additions below in bold describe how this was accomplished:

1. Business – A description of the structure and interaction between the business processes and information needs.
2. Application – A software-engineered capability, described through **context, composition, structure, interaction, and information viewpoints**, independent of any infrastructure, that supports business process and information needs.
3. Infrastructure – A description of the structure and interconnection of compute and storage resources at the virtual or physical device level, provisioned to support the **processing node and execution environment** needs articulated in one or more Application domains.

By eliminating the Infrastructure layer from the SDD and enhancing the application and infrastructure domain definitions, the Design Team:

1. Removed extraneous content that provided no value to software development teams, and
2. Facilitated the provisioning of staging and production environments irrespective of platform (physical/virtual/cloud)

4.1.2 Establish the SDD as a control document

The next step was to jettison *redundant* content (Lean practitioners recognize this form of waste as *overprocessing*) from the SDD. **Figure 3** below depicts the SDD’s relationship to external documentation and the description of content these external artifacts provide *by reference* to the Lean SDD:

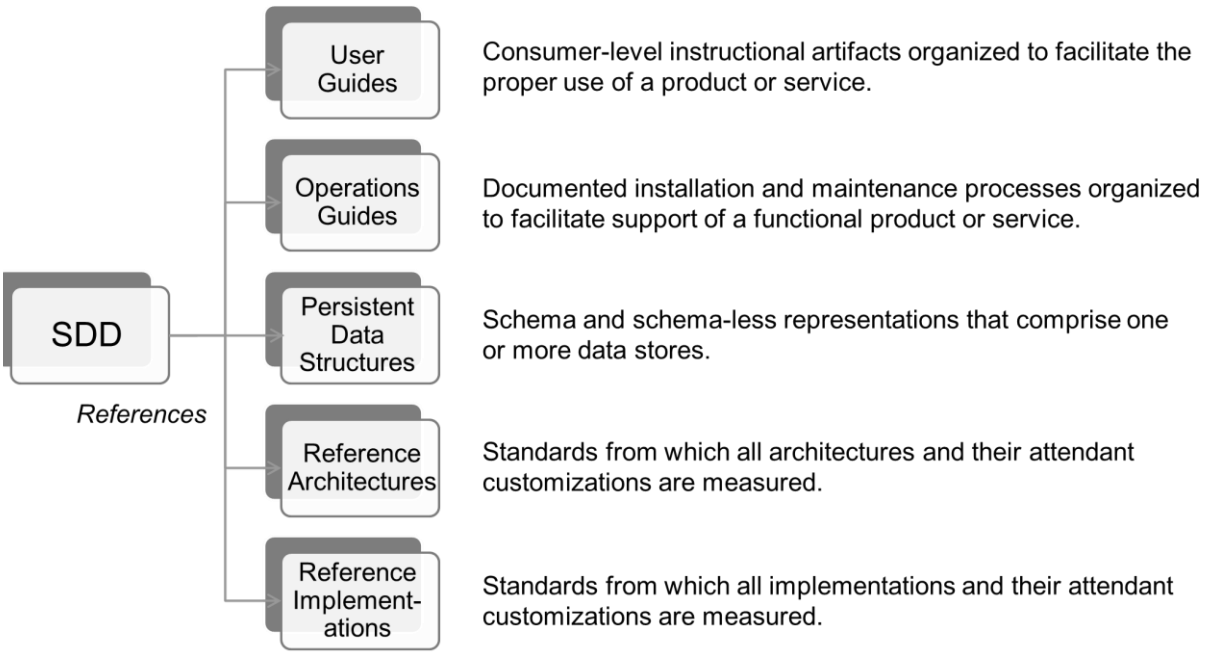


Figure 3: SDD as a control document

By eliminating redundant content from the SDD the Design Team:

1. Removed content that provided no value to the software development team, and
2. Identified the authoritative content sources.

4.1.3 Assemble the SDD as a product suite

The next step was to further refine redundant content elimination. **Figure 4** below depicts and describes the artifacts that comprise the SDD *product suite*:

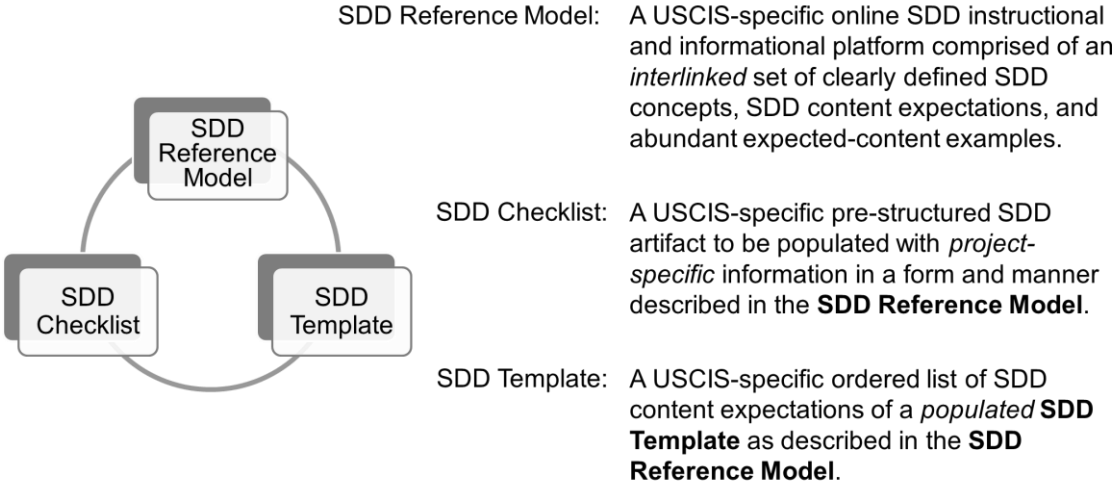


Figure 4: Artifacts comprising the Lean SDD

By relocating instructional content from the SDD Template to an SDD Reference Model, the Design Team was able to:

1. Develop a highly effective instructional platform, free from concern of overwhelming the SDD Template, and

2. Eliminate the mass-deletion of instructional content subsequent to populating the SDD template.

4.1.4 Adopt a tailor-up approach to the SDD

The final step in eliminating waste is to implement a tailor-up approach. In utilizing this method, project teams *build-up* the Lean SDD from a baseline of content, which is *always* required. Thus, content that provides no value is not initially in the SDD Template and added content that provides no value or is redundant is more easily detected.

4.2 See the Whole

From the "see the whole" perspective the Design Team adopted a synergistic approach: *Systems Thinking* and graphics-first.

4.2.1 Systems Thinking

As an essential concept in "see the whole", the two complementary definitions of Systems Thinking [3] adopted by the Design Team were:

1. "A discipline for seeing wholes ... a framework for seeing interrelationships rather than things ... a process of discovery and diagnosis."

2. "The art of simplifying complexity. It is about seeing through chaos, managing interdependency, and understanding choice."

As the Design Team quickly deduced, in embracing a "see the whole" concept, *what constituted a system* within USCIS was problematic:

> *For reasons unknown to the Design Team, USCIS software project teams had adopted the same system boundaries as the System Owner, that is,* ***the functional/technical boundaries of a system were synonymous with boundaries of system funding****. These artificial borders served to contravene the most fundamental of systems science tenets, effectively concealing true system boundaries which are of paramount importance in "see the whole."*

To rectify this situation, three definitions relating to systems were established which would address *true* functional/technical boundaries without infringing on the system boundary definition adopted by USCIS System Owners:

1. System-of-Interest – The top-level system in the system structure, containing **subsystems** and user groups/classes that **directly** interact with the **principal subsystem**. *Note that the subsystems may be systems in their own right and may exist within their own environment.*

2. Subsystem – A collection of people, processes and technologies organized to accomplish a specific function or set of functions **within** a **system-of-interest**. *Note that while a subsystem may be a formally declared USCIS system, it is still considered a subsystem in the Lean SDD.*

3. Principal Subsystem – A **subsystem** within the **system-of-interest** that is decomposed into constituent parts and aspects. *Note that the boundaries of a principal subsystem are synonymous with the boundaries recognized by USCIS Systems Owners (system funding).*

The importance of these definitions in describing true system depth and breadth will become apparent in section *4.3.1 Define and organize SDD viewpoints*.

4.2.2 Graphics-first

The axiom “A picture is worth a thousand words” is applicable when describing a solution architecture *application* domain. Research has proven time-and-time-again that complex ideas can be conveyed with a single still image, making it possible to absorb large amounts of data quickly. To exploit this fundamental human trait, the Design Team adopted a graphics-first approach where *for each Design View*:

1. A *single sentence* states what the view intends to convey,
2. Followed by the primary view graphic,
3. Followed by *textual elaboration* of the view graphic.

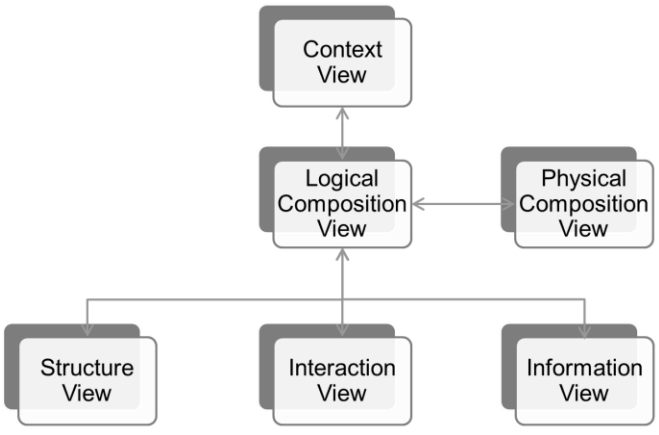
4.3 ‘See the Whole’ Process

In “see the whole”, the Design Team started with the SDD's largest areas of concern and worked until the expected benefit no longer warranted the effort expended (benefit/cost ratio). The four-step process adopted by the Design Team is described in **Figure 5** below:

Figure 5: SDD ‘See the whole’ process

4.3.1 Define and organize SDD viewpoints

Modern design descriptions utilize a viewpoint/view [4] paradigm. While this model can lead to extremes (DoDAF [5] and IEEE-1016 [6]), the Design Team's adherence to Agile values and Lean principles ruled out anything nonessential. Working with project teams, the Design Team settled on the following six views identified in **Error! Reference source not found.** below that would sufficiently describe any solution architecture *application* domain within USCIS:



View	Description
Context	A *top-level* “black box” perspective and scope of the design subject on the **system-of-interest**.
Logical Composition	A high-level “white box” perspective of the **principal subsystem** on the **system-of-interest**.
Physical Composition	The *provisioning* of the **principal subsystem** within the **system-of-interest**.
Structure	**Principal subsystem** static structures and transient data aspects.
Interaction	**Principal subsystem** human-machine and machine-machine behavior and interplay aspects.
Information	**Principal subsystem** persistent data aspects.

Figure 6: Application Domain view hierarchy

4.3.2 Distinguish SDD architecture from SDD engineering

Modern design descriptions differentiate architecture from engineering; a particularly important distinction for those at USCIS implementing Scrum and Scrum-ban [7] flavors of Agile. **Error! Reference source not found.** below depicts the delineation and description of architectural and engineering views:

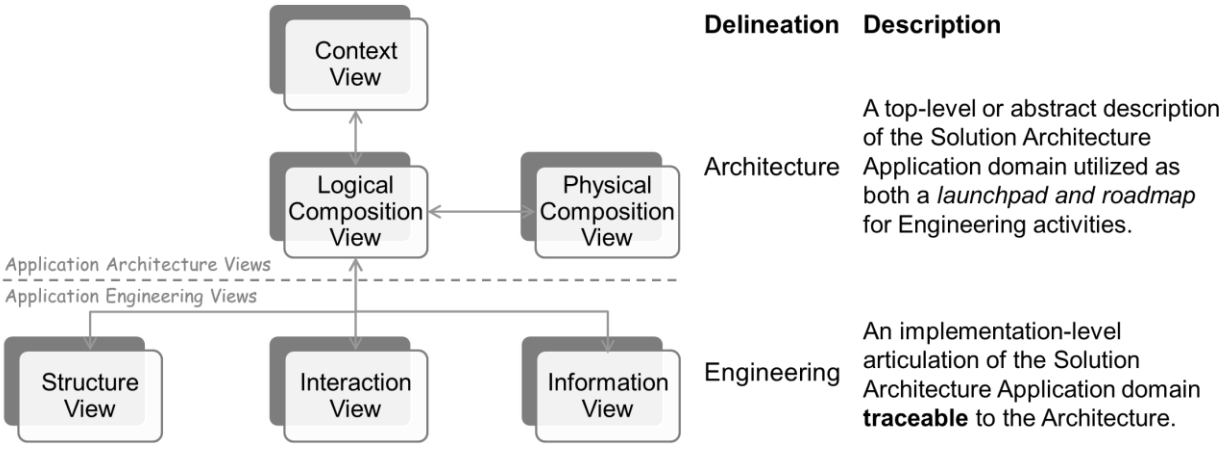


Figure 7: Application Domain view delineation

This delineation between architecture and engineering views bodes well with the Scaled Agile Framework as:

1. Architecture is an iteration (sprint) 0 initiated activity within each increment (release cycle), and
2. Engineering is an iteration 1..n activity within each increment.

4.3.3 Establish SDD symbology

While the Design Team has satisfied three of the five SDD solution guiding principles:

1. Adhere to Agile values and Lean principals;
2. Seamlessly integrate into Agile and Lean environment;
3. Remain backward compatible with legacy environments.

Two guiding principles had yet to be addressed:

4. Require no special tooling to develop the SDD;
5. Be *readily comprehendible* by management, practitioners, and ancillary stakeholders.

These principles were easily satisfied through the adoption of block diagramming [8] techniques. Utilizing a *box-and-line* notation allows stakeholders not proficient in architecture domain symbology to easily articulate and comprehend a design description. The addition of requisite block-*stereotyping* facilitates a consistent understanding of each element across every view. Finally, block diagramming requires no specialized drawing software; all example diagrams contained in the Lean SDD Reference Model views were developed utilizing Microsoft PowerPoint.

4.3.4 Format and publish SDD artifacts

Finally, being *readily comprehendible* in “see the whole” can be a difficult proposition when the form-factor is restricted to 8½ x 11 inches and the platform is a Microsoft Word document. Here too, the Design

Team elected to abandon this 20th century relic and instead format and deliver Lean SDD artifacts utilizing the web page capabilities of USCIS ECN (SharePoint) platform. The advantages of this platform change were fourfold:

1. Project teams can review the Lean SDD Reference Model, template, and checklist *at any time utilizing any device* connected to the USCIS Intranet.

2. Project teams can copy and paste the XHTML-validated Lean SDD template and checklist from the SDD SharePoint site to the project team's SharePoint site *without content or formatting loss*.

3. Project teams can develop diagrams and populate the Lean SDD template *unfettered by the 8½ x 11 inch form-factor*.

4. Project teams can archive an approved Lean SDD from their respective SharePoint sites to the appropriate Information Technology Document Library (ITDL) utilizing the MHTML [9] file format supported by Internet Explorer, Chrome, Firefox, and Opera web browsers.

5. Benefits of Utilizing the Lean SDD

By adhering to Agile values and Lean principles, the Design Team was able to successfully address the principle purpose and use of a USCIS SDD:

1. A contract document utilized in procurement of software development and maintenance services;

2. An essential onboarding artifact for team members new to the program or project;

3. A documented reflection of the system's current design state at a specific release cycle.

The Lean SDD fulfills these requisite expectations on all counts; when used to describe a modern *software*-engineered capability, the Lean SDD's application-domain scoping, systems-thinking approach, graphics-first content, and exponential reduction in volume all combine to advance the comprehension of system structure and behavior at a rate previously thought unattainable. When deployed on a modern content management system, the result is a collaboratively-developed SDD that will seamlessly integrate into any Agile, Lean, or legacy environment.

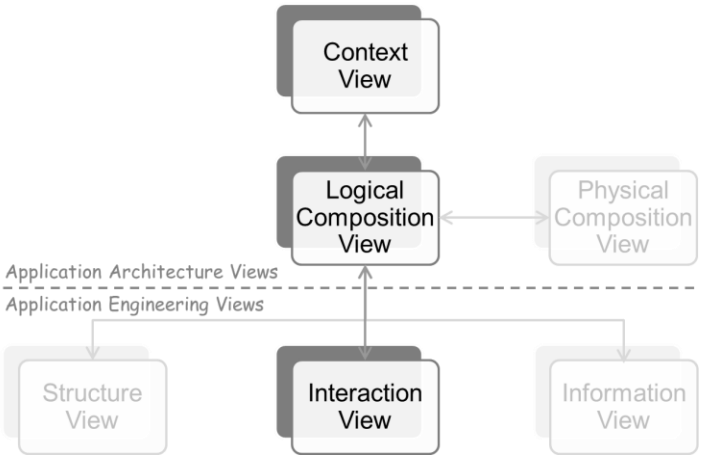
6. Lean SDD Implementation

The Design Team believes that *documentation is a part of software development, not a separate activity*. This runs counter to how the project teams currently generate design documentation. As change is always a difficult proposition, especially in demanding environments with high expectations, the Design Team collaborated with multiple project teams of varying size to determine the best course of action. The Design Team and project teams jointly concluded that a wholesale transition to the Lean SDD was not a practical solution due to project team workloads and tight delivery schedules. Instead, "evolution, not revolution" would rule the day.

The Design Team, in concert with the project teams, ascertained that this approach would entail a two-step process:

6.1 Identify Minimal Lean SDD Views

Upon review of the three principal intents of an SDD previously articulated in section three *Ramifications of Legacy SDD Template Continued Use*, and section five *Benefits of Utilizing the Lean SDD*, the Design Team focused on intent number two: "An essential onboarding artifact for team members new to the program or project." The Design Team, in collaboration with the project teams, determined that the three views identified in **Figure 8** below are considered imperative for new team member *initial* understanding of application structure and behavior:



View	Description
Context	"10,000 foot view" of application **structure** *and* **behavior**
Logical Composition	"5,000 foot view" of application **structure** *traceable* to the Context View
Interaction	"2,500 foot view" of application **behavior** *traceable* to the Logical Composition View

Figure 8: Application Domain imperative views

The three remaining views were *initially* omitted for the following reasons:

1. Some information provided by the Physical Composition View can be abstracted from Infrastructure Architecture diagrams.
2. Some information provided by the Structure View can be abstracted from existing UML Class and Package diagrams.
3. Some information provided by the Information View can be abstracted from existing Entity Relationship Diagrams (ERDs).

6.2 Phase-in Lean SDD Adoption

Active USCIS systems are extensive in number and range from the very small to the extremely large. Once again, the Design Team collaborated with the project teams to integrate Lean SDD development *without impeding software development*. As USCIS release increment velocity typically ranges from two to eight weeks for all projects, two adoption approaches would be utilized based on project size and complexity:

1. Small or medium size projects of normal complexity – Context, Logical Composition, and Interaction Views are to be completed *prior to the release of an increment*. The Physical Composition, Structure and Information Views would be completed prior to the subsequent release increment.
2. Large or extremely complex projects – One view is to be completed and incorporated into the Legacy SDD (supplanting any duplicate content) *per release increment*. At the end of six increments, the Lean SDD will completely supplant the Legacy SDD.

The above approaches allow the project teams to incorporate the Lean SDD into the software development cycle without disrupting iteration velocity and subsequent increment releases.

7. Conclusion

When the Design Team embarked on this SDD *fitness-for-use* endeavor, it did so with a collective open mind. The Design Team correctly reasoned that any attempt to modernize the Legacy SDD Template would be tantamount to *putting fenders on a carriage*. A new approach was clearly needed, one that would *complement* 21st century software development methodology and techniques.

The new approach adopted by the Design Team, *one grounded in Lean principles and Systems Thinking concepts*, ably achieves the scope and purpose expectations of a modern SDD while adhering to the five guiding principles established earlier in this paper.

USCIS project team feedback of the Lean SDD product suite (reference model, template, and checklist) is overwhelmingly positive; to paraphrase agency adopters, *the (lean) SDD conveys system structure, function, and intent in a manner that is immediately understandable to those with little or no system knowledge*, which was the DAB's mission from the very beginning.

References

[1] The Open Group, "TOGAF 9.1 > Part II: Architecture Development Method (ADM) > Phase C: Information Systems Architectures - Data Architecture," [Online]. Available: http://pubs.opengroup.org/architecture/togaf9-doc/arch/chap10.html. [Accessed 19 July 2015].

[2] Scaled Agile, Inc., "Scaled Agile Framework (SAFe)," [Online]. Available: http://scaledagileframework.com/. [Accessed 19 July 2015].

[3] Systems Engineering Body of Knowledge (SEBoK), "What is Systems Thinking?," [Online]. Available: http://sebokwiki.org/wiki/What_is_Systems_Thinking%3F. [Accessed 19 July 2015].

[4] The Open Group, "TOGAF 9.1 > Part IV: Architecture Content Framework > Architectural Artifacts > Views and Viewpoints," [Online]. Available: http://pubs.opengroup.org/architecture/togaf9-doc/arch/chap35.html#tag_35_04. [Accessed 19 July 2015].

[5] U.S. Department of Defense, "DoDAF 2.0 Architectural Models-Views and Descriptions," [Online]. Available: http://dodcio.defense.gov/Portals/0/Documents/DODAF/DoDAF_2-0-Arch_Models-Views_Descriptions.docx. [Accessed 19 July 2015].

[6] IEEE Standards Association, "1016-2009 - IEEE Standard for Information Technology--Systems Design--Software Design Descriptions," [Online]. Available: http://standards.ieee.org/findstds/standard/1016-2009.html. [Accessed 19 July 2015].

[7] C. Ladas, "Scrum-ban," [Online]. Available: http://leansoftwareengineering.com/ksse/scrum-ban/. [Accessed 19 July 2015].

[8] A. Kossiakoff, W. N. Sweet, S. Seymour and S. M. Biemer, Systems Engineering Principles and Practice, 2nd Edition, John Wiley, 2011.

[9] FileInfo.com, ".MHT File Extension," [Online]. Available: http://fileinfo.com/extension/mht. [Accessed 19 July 2015].

Improving Forecasts Using Defect Signals

Paul Below

Paul.below@qsm.com

Abstract

On large software development and acquisition Programs, testing phases typically extend over many months. It is important to forecast the quality of the software at that future time when the schedule calls for testing to be complete. Shewhart's Control Charts can be applied to this purpose, in order to detect a signal that indicates a significant change in the state of the software. This signal is then used to evaluate and modify forecasts.

Biography

Paul Below has over 30 years of experience in technology measurement, statistical analysis, estimating, Six Sigma, and data mining. He is a Principal Consultant with Quantitative Software Management, Inc. (QSM) where he provides clients with statistical analysis of operational performance, process improvement and predictability.

He has written three articles for Crosstalk (Journal of Defense Software Engineering), and he is co-author of the IFPUG Guide to IT and Software Measurement (CRC Press, 2012). He has developed courses and been an instructor for estimating, Lean Six Sigma, metrics analysis, function point analysis, and also taught metrics for two years in the Masters of Software Engineering Program at Seattle University. He has presented papers at a dozen industry conferences.

Paul is a Certified SLIM Estimation Professional, and has been a Certified Software Quality Analysis and a Certified Function Point Analyst. He is a Six Sigma Black Belt, and has one US Patent. He is a member of IEEE, the American Statistical Association (ASA) and the National Defense Industrial Association (NDIA).

Copyright Paul Below August 2015

1 Introduction

Every process displays variation. Some processes display controlled variation and others display uncontrolled variation.

In the 1920's, Control Charts were invented by Walter A. Shewhart. In most of the rest of the 20th century the concepts were popularized by people such as W. Edwards Deming. The driver for this development as explained by Shewhart:

> "The engineer desires to reduce the variability in quality to an economic minimum. In other words, he wants (a) a rational method of prediction that is subject to minimum error, and (b) a means of minimizing variability in the quality of a given product at a given cost of production."[1]

Over several decades, QSM has found that software metrics commonly follow a Rayleigh curve[2]. This results in a very different situation from the typical use of control charts, where the process being measured is expected or desired to have a consistent output each time, every time.

In this paper, I describe the use of control charts during testing phases of software development projects. This use is not to determine if the testing is in control, nor is it in order to improve product quality (although that has also been done[e.g.3,4]), but rather to determine when there has been a shift in quality. This is in order to improve mapping of project progress to forecast curves and thereby improve estimates of project schedule.

Therefore, let us look at a typical example.

2 Using Control Charts to Detect Signals

In order to improve defect forecasts, I use Individuals and Moving Range charts (*XmR).* This is a type of control chart that is suitable for most real time situations, including the collection of periodic data such as defects detected in a given time period (such as week or month). The Individuals chart has each value plotted in time order. The Moving Range chart, on the other hand, plots the short term variation from one period to the next.

While most signals denoting a significant change in the underlying situation, such as stabilization of the product reliability, appear on the individuals chart it is good practice to look at the moving range chart as well, as some signals will only show up on it.

Control charts are based on the long term average value as well as the average moving range value of one point to the next. It is important to calculate control limits correctly in order to not miss valid signals. The appropriate formulas can be found in select books [e.g.5,6] and are also built into statistical tools such as *SPSS*, *Minitab* and *SAS*.

Control limits provide a signal of sporadic or chronic problems. For tracking defects, however, the signal we are looking for is a change in the underlying quality of the software product. Hopefully, this will be a signal of an improvement and not a signal of a problem!

3 Rules

There are a number of rules that are used to detect signals. The number of rules used and the definitions of the rules vary slightly from one source to another. However, the traditional use of control charts is best met by keeping the number of rules to a minimum, thereby reducing the chance of obtaining a false signal. In this paper, I am using all the rules because I do not want to miss a signal (erring on the side of a false positive).

All uses of control charts walk this decision line. Shewhart originally used 3 sigma limits because he wanted to minimize false signals, which would incur the unnecessary cost of researching a problem that didn't exist. In other words, when he saw a signal he wanted to be almost completely certain it was real.

In IBM *SPSS 22*, for example, there are 11 possible rules that can be turned on or off:

- One point above +3 Sigma, or one point below -3 Sigma
- 2 out of last 3 above +2 Sigma, or 2 of 3 below -2 Sigma
- 4 out of last 5 above +1 Sigma, or 4 out of 5 below -1 Sigma
- 8 points above center line, or 8 below center line
- 6 in a row trending up, or 6 trending down
- 14 in a row alternating up and down

4 Example Control Charts

In Figure A, weekly defects detected are plotted. All the SPSS rules are turned on. If the defect detection rate has changed significantly, that would show up as a special cause signal in the control chart. In this example, the balance between testing and fixing has not remained constant. Five of the points show up as circles, meaning they violated one of the rules (see Table 1).

Figure A: Control Chart Example

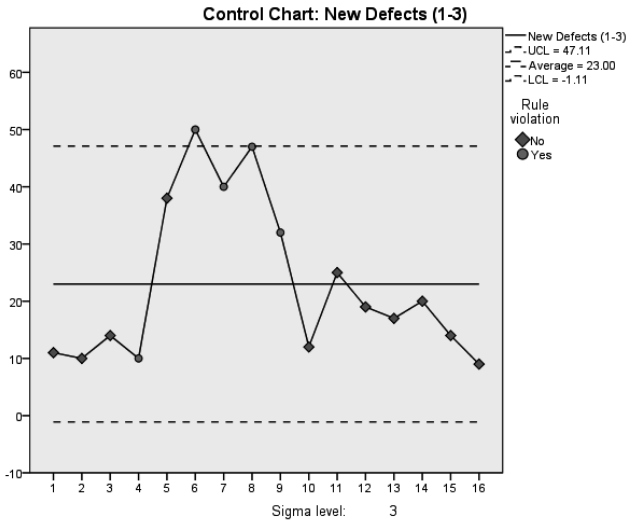


Table 1: Rule violations

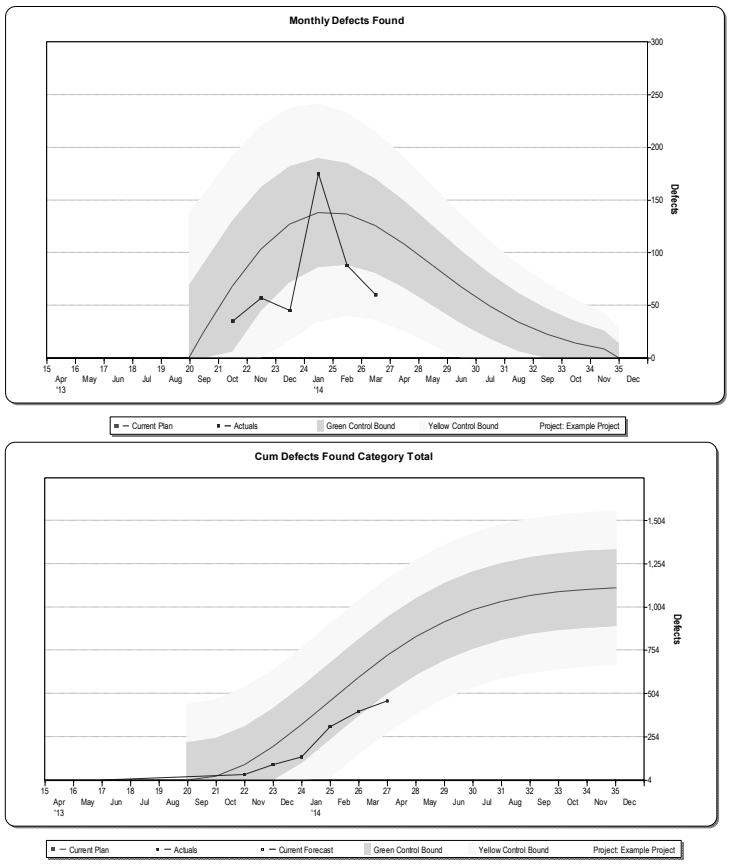
Point #	Violations for Points
4	4 points out of the last 5 below -1 sigma
6	Greater than +3 sigma
6	2 points out of the last 3 above +2 sigma
7	2 points out of the last 3 above +2 sigma
8	4 points out of the last 5 above +1 sigma
9	4 points out of the last 5 above +1 sigma

What can we surmise from this? These violations are not unusual. As mentioned previously, defect metrics commonly follow a Rayleigh distribution. In Figure B, actual defects detected monthly are overlain on a defect forecast based on the current project plan (a parametric SLIM Control forecast based on historical defect rates and project type, size, staff, and duration). We can see the peak detected as a set of rule violations falls in line with the peak of the Rayleigh curve.

Defect estimation is outside the scope of this article although an example is described in the next section.

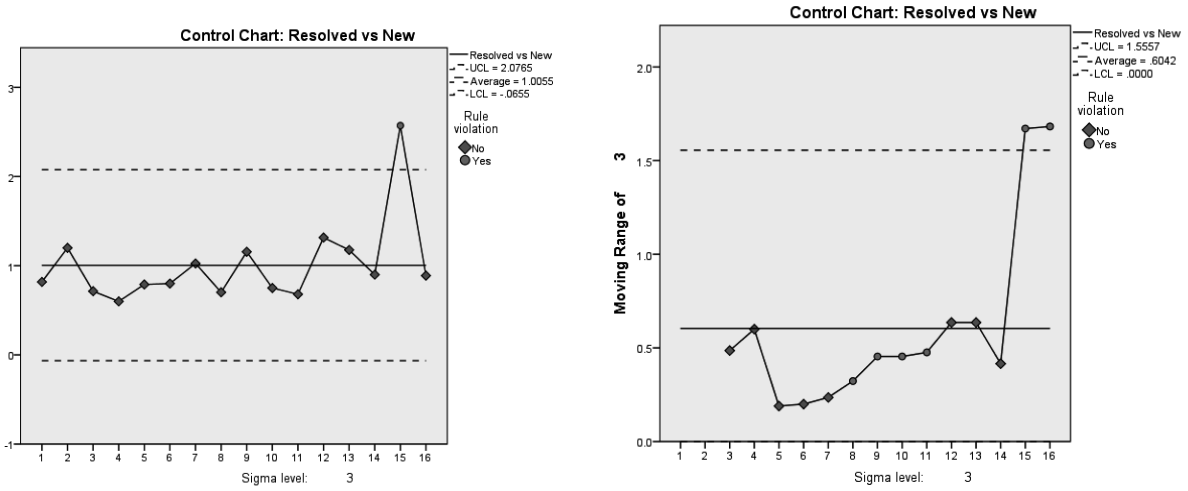
One important question is whether the drop in defects in the last few weeks is a signal that a turning point on the project has been reached, as the Rayleigh curve suggests. Control charts can be used to help verify that the signal is real and not random noise by looking to see if any of the rule violations in Table 1 were met. As will be seen in Figure C, we have evidence that the defect peak has been reached.

Figure B: Rayleigh Example, Defects Detected and Cumulative Defects Detected



Figures C and D are Individual and Moving Range charts for the ratio of defects discovered to defects resolved. This ratio measures the balance between defect detection in testing and defect repair. Values in the individuals chart greater than 1 indicate more defects were resolved than were detected during that week. Both charts show rule violations. Point 15 on the individuals chart provides evidence that the balance has shifted. This is what we would expect to happen if the project is truly on the downslope of the Rayleigh Curve, the violation corresponds to the peak in the Rayleigh curve.

Figure C and D: Individual and Moving Range Chart

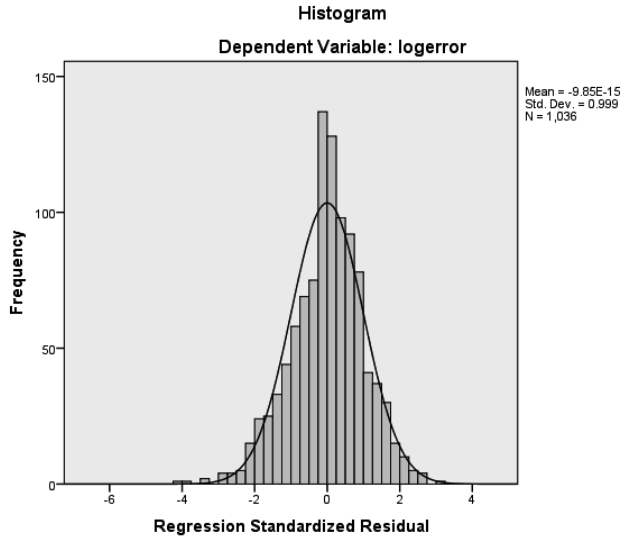


5 Defect Prediction

To make good use of a defect signal, a defect estimate is required.

The plan in Figure B was based on parametric estimating. It is possible to create very useful estimates of defects based on only a few key metrics. For example, I created a regression analysis to predict defects based on over 2000 recently completed software projects from the QSM database. This resulted in an adjusted R square of .537 using only the input variables the log of peak staff, the log of ESLOC, and the log of production rate (ESLOC per calendar month). The output variable is log of defects (Why logs? For the explanation, see[7]). The standardized residuals are plotted on a histogram in Figure E. As can be seen, the residuals have a normal distribution with mean close to zero. The model is not skewed.

Figure E: Standardized Residuals



Large projects have multiple testing phases. Such models, with multiple control charts, can be used throughout. For example, with one Fortune 500 client, I found that merely using the number of prerelease defects was an excellent predictor of their go live release defects (R Square of over 0.7).

6 Summary

Control charts can be used to determine whether apparent changes in defect rates are significant. Especially, has the peak of the defect detection Rayleigh curve been reached? Once the peak of the Rayleigh curve has been reached, the curve can be forecast into the future to predict the software quality at any given point.

One use for this knowledge is to create and improve forecasts of Program completion, or software quality at key Program milestones (which would be points along a graph such as Figure B).

Shewhart gave us this final thought regarding updating forecasts:

> "…since we can make operationally verifiable predictions only in terms of future observations, it follows that with the acquisition of new data, not only may the magnitudes involved in any prediction change, but also our grounds for belief in it."[8]

References

1. *Statistical Method from the Viewpoint of Quality Control*, Walter A. Shewhart. Dover, 1986 edition, p.9.
2. *Five Core Metrics: The Intelligence Behind Successful Software Management*, Lawrence H. Putnam and Ware Myers. Dorset House, 2003, Chapter 13.
3. *Why CMMI Maturity Level 5?*, Michael Comps. Crosstalk, Jan-Feb, 2012, pp 15-18.
4. *Do Not Get Out of Control: Achieving Real-time Quality and Performance*, Craig Hale and Mike Rowe. Crosstalk, Jan-Feb 2012, pp. 4-8.
5. Understanding Statistical Process *Control*, Donald J. Wheeler. SPC Press, 2010.
6. *Implementing Six Sigma: Smarter Solutions Using Statistical Methods*, Second Edition, Forrest W. Breyfogle III. John Wiley & Sons, 2003.
7. *The IFPUG Guide to IT and Software Measurement: A Comprehensive International Guide*, IFPUG, ed. CRC Press, 2012. Chapter 17, Paul Below, pp. 319-333.
8. *Statistical Method from the Viewpoint of Quality Control*, Walter A. Shewhart. Dover, 1986 edition, p.104.

Don’t Let Documentation Be a Buzzkill

Victoria Palmiotto and Kristin Ito

victoria.palmiotto@mindbodyonline.com and kristin.ito@mindbodyonline.com

Abstract

Whether you love it, or love to avoid it, documentation is necessary in the worlds of software development and quality assurance. Crafting content for both internal and external audiences to record the work you’ve done is essential for your organization and end users to fully realize the quality of your product. Oftentimes, a non-technical task such as writing is approached as a tedious chore; details may be overlooked as you go through the motions.

By blending some basic writing fundamentals into your documentation process, you can:

- Create quality documentation that reflects the quality of your software
- Ensure your documentation is valuable, understandable, and easy-to-navigate for people outside of your team
- Streamline documentation processes
- Develop documentation that’s usable and consistent
- Promote accountability and collaboration among your team

Biographies

Victoria is a content editor on MINDBODY’s Product Development Team, where she works at the intersection of technical and nontechnical teams. To translate tech-talk into valuable end-user content, she relies on the powers of collaboration and concision.

Kristin is a content editor at MINDBODY, where she writes about new features in their apps and products. She’s interested in style guides, diversifying content types, and developing processes to maintain content quality. Kristin was previously an editor at Expedia.

1 Introduction

We get it. Documentation takes time that you just don't have. Plus, there's an inherent conflict between the agile environment and documentation, and it's natural to wonder: Is agile documentation an oxymoron?

We're here to argue that documentation in an agile environment is not only possible, but can work really well when it's agile, too. Quality documentation has several characteristics—like its ability to change along with the software—that we'll go over in the following pages and that will help you create documentation that works within an agile context and reflects the quality of your software. After all, "your documentation is an advertisement for the quality of your code" (Raymond, 2000), and we'll show you how you can best advertise the value of your product through quality documentation.

2 What we talk about when we talk about documentation

Because there are several types of software documentation, and because people think of different things when referring to documentation (Kiss, 2011), it will be useful to lay out the types of documentation we are *not* talking about. We're not talking about technical documentation of source code or of algorithms, interfaces, and APIs. Nor are we referring to any type of requirements documentation, which is goal-oriented and created at an early stage in development; or to architecture design documentation, which also tends to be inceptive. We're not talking about testing or maintenance documents either, which do play a role in software quality, but are not the focus of this paper. All of this documentation is what Ian Sommerville defines as process documentation, or a record of the ideas and plans put forth along with the steps taken to implement them (Sommerville, 2001).

What we *are* talking about is product documentation, and more specifically under the umbrella of product documentation, what is referred to as the description document or functional description document. While this is sometimes conflated with or even considered a type of user documentation—which is often generated from the description document—we are talking about the *raw materials*, the written communication that is needed to understand how to use your software.

Product documentation serves different purposes for different audiences. While danger lurks in a one-size-fits-all approach, there is potential for efficiency and usefulness with proper documentation planning and creation. Quality software documentation can act as:

- **A single source of truth**
  This is especially important in fast-moving environments that are susceptible to tribal knowledge traps. In iterative agile development, multiple sprints with small releases may be required to produce a full product or feature. Software changes and improves as product and business objectives evolve. Documentation serves to clearly define where the product stands.

- **A vehicle for knowledge transfer and training**
  On some development teams, team members and roles change as fast as the code does. Up-to-date documentation is a quick and reliable way to get everyone on the same page at any stage of a project. New hires can familiarize themselves with the software, and team-swappers can align themselves with the goals and objectives of their new assignment.

- **Success criteria to define project completion**
  In a scrum/sprint environment, projects are constant, and the end of one sprint can often bleed into the start of another. Clear lines are sometimes not drawn to delineate when a project is complete. Imagine running a half marathon, and instead of a line to cross at the end of the race, there's a general finish area that spans a half mile or so. When do you stop running? Documentation can act as your finish line. Once you've submitted your documentation for the sprint, you can consider your project complete and gear up for the next sprint.

- **Support material for users**
  Despite how intuitive your software may be, users new and old will likely need help at some point—whether it be starting up, adjusting to a new feature, or troubleshooting on the fly. Your documentation is the basis (or sometimes the prevailing source) of how-to and what-to-do-when support material. In describing the software's functions, you're explaining how it works and how to use it, which is helpful for anyone who did not build it. What's more, if the feature you're working on overlaps with another team's feature sets, this explanation of how it works and how to use it holds internal value.

- **The announcement about what you've made and why it's great**
  With the fast-paced nature of software development, it's easy to become transfixed on the goal of project completion—but what happens once the code is perfected and the tests are complete? While finishing projects and producing new products and features is satisfying within your organization, these accomplishments hold little value if no one knows about or uses what you've created. Imagine your finger-painted childhood masterpieces never getting hung on the fridge—it's almost heartbreaking. Documentation tells people what you've made. Whether it's modified and delivered straight to end users, or funneled through a content or marketing team to package and deliver, documentation is the starting point for getting the word out about your work.

3 What is quality documentation?

The argument can be made that quality is subjective; standards may vary from one company to the next, and from one person to the next. So it's no surprise that quite a few studies have been conducted in an effort to more narrowly define quality, to peg down what it really means in terms of software documentation. For example, in *The Value of Software Documentation Quality*, the authors' survey found that "the most important quality attributes with regard to documentation quality are accuracy, clarity, consistency, readability, structuredness, and understandability" (Dautovic, Plosch, Saft, 2014). Taking these attributes into account, as well as what we've garnered from our own experience, we've compiled a list of what we believe defines quality documentation.

4 Quality documentation considers your audience

Quality software requires and deserves quality documentation, and one way to ensure this is to always consider your audience. Who you are writing for determines what needs to be included and with what level of detail. Audience will also determine the structure of your documentation, as well as language, style, formality, and other writing elements.

Considering your audience means thinking about your content. Andrew Forward found through a survey on software documentation, that "content is the most important factor. All document tasks (creation, maintenance, verification, validation) should always keep the target audience in mind" (Forward, 2002). Thinking about what your audience needs to know, what they don't need to know, and what order they'll want to read it in, will allow you to cut unnecessary content, structure it with purpose, and create documentation that is more efficient and effective.

Considering your audience also means thinking about *why* you're creating documentation. As a developer or software quality tester, you might be relatively removed from the end user in your quest to test or in your compulsory focus on the deliverables. Documentation can provide an impetus to get team members thinking about the purpose of documentation. *Who will be reading this, and why?*

Another important question to consider is: *What do I want them to understand?* If the point of product documentation is to communicate how to use your software, then the real value here is *knowledge transfer*. In other words, your job is to create documentation that your audience can understand and learn from. Furthermore, considering your audience means not making assumptions about the level of

knowledge your readers have regarding the software. Knowing your audience can prevent this from happening.

> *"I've never met a human being who would want to read 17,000 pages of documentation, and if there was, I'd kill him to get him out of the gene pool."*
> *Joseph Costello*

Listed below are a few common audiences. You may need to consider some or all of them, depending on the size and scope of your team.

- **Internal - Developers/QA**
  Much of the documentation shared between development teams will likely be process documentation; however, for this, as well as for any product documentation, you should consider that while your audience may understand the technical aspects of the project, they may be unfamiliar with other aspects. You'll need to address the distinctive elements of your particular environment, the history and status of the project, or other specifications unique to your team. For this audience, technical language and simplification will be lesser considerations, while context and regard for future developers will be larger concerns.

- **Internal - Technical support, IT**
  This audience has a strong level of technical knowledge and experience with the software; however, you'll need to consider that their knowledge is more support-focused and is not necessarily up-to-date. The main question you'll want to think about is *What will they need to know in order to assist in troubleshooting issues?* In terms of content, consider that your readers will be dealing with different settings and exceptions—this information should be included, but in a way that they can find quickly while communicating with the user. If you're documenting a change or a new feature in your software, you'll want to be specific about what has changed. Providing a concise before-and-after summarization, for example, can help this audience quickly find the answers they need to support users.

- **Internal - Editors, writers, marketing, other non-technical teams**
  For the non-technical audiences that will be reading your documentation, the most important part of the equation will be the why. Putting the why up front will allow readers to decide how much of your documentation they need for their purposes. Additionally, the further your readers are from software creation, the more important your documentation becomes: "Documentation becomes a better option for you the greater the distance, either physical or temporal, between the individuals who are communicating" (Ambler, 2001-14). For example, if a writer is working with your documentation to create a user's guide, then your documentation is likely the only source they have in terms of functionality. Without the ability to test the software themselves, how else are they supposed to know how your software works? You'll need to think about what the writer will need to know so that they can best explain it to the users.

- **Internal - Stakeholders**
  Stakeholder participation in the agile process is a contentious topic itself, but in terms of documentation, you can make sure that even if a stakeholder takes a quick look at it, they'll be able to understand the value of the project.

- **External - Users**
  If you're working alone or your team does not have the bandwidth for editors or writers, you may be responsible for creating user documentation, such as manuals or support articles. In this case, you'll need to consider your end user at all times, and make no assumptions in terms of technical knowledge. This doesn't mean "dumbing down" your content; it just means you need to more

carefully consider what the user needs to know, and what is the most clear and concise way to inform them.

Audience is a consideration that should become intrinsic. Once the creator figures out who they are writing for, and why, they should keep their audience, or audiences, in mind during the entire process.

5 Quality documentation is a part of the project

Oftentimes, documentation is wrongly treated as an independent supplement to a project, rather than an integrated component. It can be pushed to the back burner, forgotten about until the last minute, and then given a haphazard effort, which yields ineffective and inadequate documentation. To avoid this, make sure "documentation becomes part of the development process, not a separate activity" (De Lancey, 2012). Thus, at your kick-off meeting or project starting point, make sure documentation is acknowledged.

As you're establishing goals, defining requirements, and divvying tasks, include documentation as part of each. "Treat documentation like any other requirement: it should be estimated, prioritized, and put on your work item stack along with all other work items that you must address" (Ambler, 2001-14). As with any work item, documentation requires time, attention, and accountability.

- **Time**
  As you're refining a project plan and estimating efforts, be sure to include time allowances for documentation.

> *"This is how you do it: You sit down at the keyboard and you put one word after another until it's done. It's that easy, and that hard."*
> *Neil Gaiman*

- **Attention**
  At the beginning of your sprint, decide who will contribute to the documentation and what their contributions will be. Distribute assignments and establish checkpoints. "Documentation should take on a collaborative nature. It should not be written by one person to perfection, and then shared with others. It should instead be shared often during draft to gain input" (Moreira, 2013).

- **Accountability**
  Who has the final approval of the documentation? What other teams or team members will the documentation be distributed to? Who will be the point person for any questions that arise? Where will the documentation live? The answers to these questions should be added to your project requirements checklist, and should be defined when the project starts.

6 Quality documentation is quality

Hold the quality of your documentation to the same standards as the quality of your software; it should be easy-to-use, consistent, aesthetically pleasing, straightforward, error-free, and reliable. Think of the final document as your product and the content as your code.

6.1 Easy-to-use

Your documentation will be viewed by a number of audiences (which you've already carefully considered). As such, you'll want to make sure that no matter who is reading, they're able to navigate the document with ease and extract the information that's valuable to them as efficiently as possible.

- **What's in a name?**
  The title of your document and the file name should be valuable to a reader. They should know what they're getting into before they double-click. Develop a consistent naming convention that identifies the file as product documentation, and identifies the specific product. Use this convention from sprint to sprint and across all development teams.

- **Create a functional first page.**
  Every page of your document should serve a purpose and provide value, including the first page. One way to eliminate unnecessary space between the reader and the information they need is to avoid redundant, title-only cover pages. Instead, approach the first page as an executive summary. Answer some questions for the reader right off the bat by addressing:
    - What the new product or feature is and a synopsis of what it does
    - Who worked on it, and who the point of contact is
    - Project start and release dates
    - The "why" and reasons behind the project
    - Where you can find this product or feature (e.g., if it's a new feature within the software, provide a menu path)
    - The dates the documentation was created and updated

- **Headers, footers, and page numbers, oh my!**
  Make sure everyone is on the same page—literally—when contributing to or using your documentation. In your header, include the name of the project to provide an easy way to identify what the document is, whether someone starts on page one or dives in somewhere in the middle. In your footer, include page numbers, which help tremendously with individual navigation and team collaboration.

- **Always include a table of contents.**
  Different people will access the document for different reasons. Whether it's three or 30 pages long, nobody wants to scroll through each page to find the specific section they need. "Finding useful content in documentation can be so challenging that people might not try to do so" (Forward, A; Lethbridge, T; Singer, J. 2003). Break up your work into clear sections, and map those sections with a simple table of contents.

> *"Never use tricky or irrelevant headlines. People read too fast to figure out what you are trying to say."*
> *David Ogilvy*

- **Establish a heading hierarchy for your sections and subsections.**
  Google Docs and Microsoft Word make this easy with their default style options. With a style set, you'll also be able to whip up and update that table of contents with a few clicks. Make the differences between headings and subheadings distinct.

  Assume that not everyone will start on page one, so make sure your headers are descriptive enough to allow someone to pick up from anywhere. If you have multiple sections with the same subsections (e.g., a "Screenshots" subsection), name each subsection with reference to its parent section (e.g., "Screenshots: Home Page").

6.2 Consistent and aesthetically pleasing

Readers of your documentation should have a consistent experience. Thus, within an organization, product documentation should be consistent from page to page, sprint to sprint, and team to team. More so, documentation should be regarded as a product itself; the final file should be polished and presentable. Templates and style guides are effective tools in achieving uniform, refined documents.

- **Use a template.**
  Develop a template and utilize it for every piece of documentation. A good template will:
  - Define and order the always-necessary sections and subsections
  - Have an established style set
  - Have customizable headers and footers
  - Be distributed for use by anyone who creates documentation
  - Reduce creation time of new documentation

- **Exercise font awareness.**
  With hundreds of fonts to choose from, keep the following in mind: Your font should not change from one paragraph or page to the next. If you use different fonts to distinguish between headers, sub-headers, and body text, limit yourself to no more than three fonts per document.

  Choose something tried and true. Font should not distract readers or make any implications about your software.
  - Arial, Calibri, Times New Roman, and Cambria are all safe bets.
  - Fonts like Comic Sans, Mistral, or Freestyle Script will detract from the readability and credibility of your documentation.

- **Strive for scannability: white space, short paragraphs, bullets, and bold.**
  Avoid walls of text, and try to keep paragraphs to a five-sentence maximum. The white space between paragraphs and around text and images helps readers to scan and move through your document more quickly. Bulleted lists are an easy way to organize information and increase scannability.

  Use **bold** strategically and sparingly. Bolding is an easy way to draw attention to important details. Be wary of ALL CAPS, *italics*, and underlining. These text modifications can sometimes make content harder to read or create confusion; use them only when emphasis is necessary. Keep in mind, too, that many will assume an underlined word or phrase is a hyperlink.

- **Show and tell with images and screenshots.**
  Images and screenshots allow readers to see what you're talking about, and visually familiarize themselves with the product or feature.

> *"Of all of our inventions for mass communication, pictures still speak the most universally understood language."*
> *Walt Disney*

Find a balance: Your documentation should not be a series of screenshots, nor completely void of them. However many you include, make sure your graphic elements are useful. "Screenshots should include some sort of annotation. Adding an arrow, a circle, or number sequences can make end user documentation completely dummy proof, and save end users from having to figure out what to do" (DeVore, 2014).

Snagit, Skitch, and Snipping Tool are all easy-to-use screen capture tools that make grabbing and incorporating screenshots a breeze.

- **Develop a style guide.**
  Is it 9:00 a.m. or 9 AM? When is it okay to use **bold**? Are headings Title Case, or Sentence case? What about the Oxford comma? Should page names be in quotations? These considerations may seem insignificant, but if they are never addressed and defined, your content style can change from page to page or author to author, which lends to sloppiness and confusion. “Like design style guides, which lay out guidelines for colors, fonts, and other design elements, content or editorial style guides are a set of guidelines for making decisions about spelling, grammar, structure, and style” (Griffis, 2013).

  Don’t have a style guide? Resources like the *Yahoo! Style Guide*, the *Chicago Manual of Style*, and the *Associated Press Stylebook* are comprehensive and reliable starting points. As you create documentation, build an in-house style guide to set standards for words, phrases, and nuances that are unique to your company’s products, voice, and language.

6.3 Straightforward

Your documentation should be concise and easy to understand. Just as a two-click workflow is favorable over a six-click in your software, shorter sentences and smaller words are favorable in documentation.

- **Be concise.**

> *“The most valuable of all talents is that of never using two words when one will do.” Thomas Jefferson*

- **Keep it simple.**
  Avoid jargon or internal-only words and phrases. Even if it’s not, imagine that your documentation needs to be read and understood by someone outside of your organization. Spell out any acronyms or initialisms the first time they are referenced.

- **Provide ample background information and context.**
  Don’t assume the reader knows everything that you know—they don’t. If you’re explaining what’s new, be sure to acknowledge what’s old. This consideration will lean largely on your audience identification.

6.4 Error-free and reliable

Call in a second set of eyes to check for writing quality and content accuracy. Refine your documentation as you would refine your software. “Write a little bit, show it to someone, get feedback, act on that feedback, and then iterate” (Ambler, 2001-14). When you’ve completed the documentation, ask a colleague for a copy edit. A typo is a surefire route to discredit your work and create confusion.

> *“Put it before them briefly so they will read it, clearly so they will appreciate it, picturesquely so they will remember it, and above all, accurately so they will be guided by its light.”*
> *Joseph Pulitzer*

- **Check for accuracy against your teammates and the software itself.**
  Make sure the buttons, links, and pages you write about match what’s in the software.

- **Include any known future plans.**

This can be especially helpful for content and marketing teams to map their support and announcement plans for your products. For example, you might be releasing an update to your software that allows clients to purchase products with credit cards on their phones. Next month, you're expanding that capability to include gift card and PayPal payments. Instead of bugging end users with two too-similar product update notifications, your communication team(s) may opt for one combined announcement.

- **Work collaboratively and cross-departmentally.**
  If you're working on a feature that overlaps with another team's features, be sure to consult someone from that team.

- **Identify a subject matter expert for each project.**
  This point person should field all questions or requests for more information regarding the product.

7 Quality documentation is shared

Once you've created quality documentation, don't let this resource go to waste. Too often, documentation remains with the owner or in a folder on someone's desktop, rendering all of the creator's hard work futile. Quality documentation should be shared, so that your company and your users can benefit from learning about the software. Here are a few things to consider when sharing documentation:

- **Housing your documentation**
  Accessibility is vital. Whether a project manager, content editor, or stakeholder needs to review your documentation, it needs to live in a place that is easy to get to. Depending on what your company uses for other records and written communication, you might consider a wiki, Sharepoint, or any number of platforms available. The important thing is that everyone who might need to access the documentation is able to, and that those who aren't meant to access it, can't. You'll want to consider rights-based access to your software documentation and should be in conversation with your security team. In addition, a designated person or team should be in charge of uploading and updating the documentation, and should let others know when this documentation is updated and ready to be accessed.

- **Open communication and collaboration**
  Not only should the documentation be shared so that other content can be created from it, but the *process* of sharing information should be open and collaborative. For example, on many teams the software quality testers are part of development teams and are the ones responsible for creating QA documentation. Rather than just handing off this documentation to the technical writers or whoever is going to create the user documentation, the software testers and the writers should have an ongoing dialogue. Whether this is in the form of in-person meetings for Q&A, emailing back and forth, or using the comments feature on a Google or Word doc, this conversation between testers and writers ensures that the user documentation is accurate in terms of functionality: "The software quality practitioner must take an active role in the review and evaluation of the user documentation. If the software cannot be used properly, it matters little if it is a quality product" (Horsch, 2003).

> *"The single biggest problem in communication is the illusion that it has taken place."*
> *George Bernard Shaw*

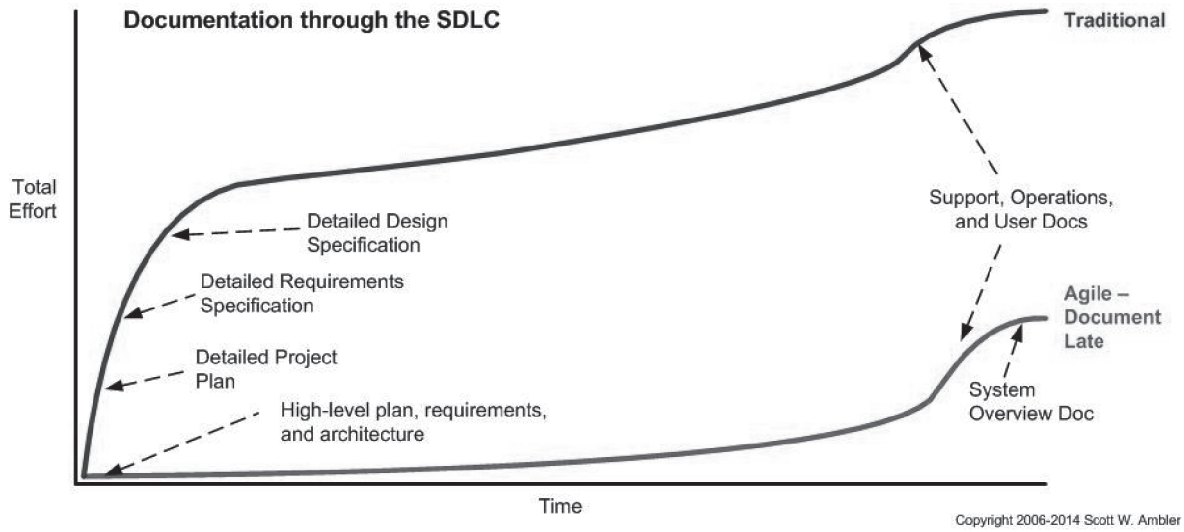
- **Osmotic communication**
  Completed documentation needs to be shared, but your documentation can also be improved by having those who communicate with your external audience, or the end users, nearby during the software grooming and creation processes. For example, most companies have writers or

marketing team members in different buildings for practical reasons. They often have to be on the phone or have jobs that would be distracting for developers or testers. However, if these communicators can be in the same room as developers, at least some of the time, the information that they receive has more context, and they are more likely to understand the reasons behind the changes in your products. This has been referred to as osmotic communication, the idea that physically sitting in a room with developers has a positive effect on the transfer of information: "When people are working close together, both physically and temporally, there exists an opportunity for what Cockburn [Alistair] calls osmotic communication: indirect information transfer through overhearing conversations or simply noticing things happening around you" (Ambler, 2001-14). This way, not only will any user documentation be more accurate and well-informed, but your own documentation can improve from being around people whose job it is to consider the needs of the user.

8 Quality documentation is dynamic

How many times have you thought, "I'll just leave the documentation until after our sprint is over, and then I won't have to change it as things change in the software"? It's true—you don't want to have to double up on your workload, especially at the beginning of the sprint when software plans are constantly changing. To avoid this, here are a couple of points to keep in mind:

- **"Document late"**
  We don't really like this term, because it implies that you should squeeze your documentation in at the end of the sprint, which can result in rushed, lower quality work. However, "document late" just means that you should document at a point in the sprint when you have the core specifications set and are past the stage of abstract ideas and hypotheticals. In other words, rather than the traditional method that begins with a surge of detailed documentation, it's more efficient to "document stable concepts, not speculative ideas" (Ambler, 2001-14).



- **Keep it "alive"**
  "The best documents are written iterative, not all at once" (Ambler, 2001-14). Once you've started documenting, it's important to be vigilant about recording changes. Whether this means taking an extra minute after your morning scrum meeting to jot down a change, or noting any updates at the end of each day, "If the documentation is not maintained in lock step with the code, it quickly becomes inaccurate" (Johnson, 2013).

In the future, we'll all likely be using documentation software that automatically updates when you update code. Craeg Strong, technical lead for the FBI, discusses this possibility: "If you had, for example, documentation that's generated from source code or maybe some reports that get automatically generated, it would be great if by refactoring the design and refactoring the software, you're simultaneously updating the documentation" (Philipp-Edmonds, 2014). Currently, there is a lot of research on ontologies as a base for documentation, and there are even authoring tools in development, such as I-Doc, which automate the process of generating software documentation, and accommodate agile maintenance (Johnson, 2013).

But until these tools are readily available, as a creator of documentation, you're responsible for keeping it "alive."

9 Quality documentation is fun!

No, but really. Writing documentation is creative in a different way than development or testing, and it allows you to take a step back and think about what you've been working so hard on. It's also an opportunity to get to know other team members and learn what they do with your documentation. Through understanding what they need to do their jobs, empathy is gained, and that's always a good thing.

It's also exciting to know that your documentation can be used in a myriad of different ways. Your documentation can evolve into online tutorials, blog posts, and other public-facing content that you can take ownership of. While it's validating to know that you've created quality software, it's just as validating to know that you've enabled people to learn how to use it.

References

Ambler, Scott. 2001-2014. Best Practices for Agile/Lean Documentation. http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm

Basaraner, Cagdas. 10 Software Documentation Best Practices. http://java.dzone.com/articles/10-software-documentation-best

Dautovic, A; Plosch, R; and Saft, M. 2014. The Value of Software Documentation Quality. Quality Software (QSIC), 2014 14th International Conference on Quality Software.

De Lancey, Tom. 2012. Emergent Documentation: One way that Agile is very different from Waterfall. https://www.linkedin.com/grp/post/43421-218531537

DeVore, Jonathan. 2014. 10 Examples of Great End User Documentation. http://blog.screensteps.com/10-examples-of-great-end-user-documentation

Forward, Andrew. 2002. Software Documentation – Building and Maintaining Artefacts of Communication. https://www.site.uottawa.ca/~tcl/gradtheses/aforward/aforward_thesis.html#_Toc26336969

Forward, A; Lethbridge, T; Singer, J. 2003. How software engineers use documentation: The state of the practice. http://www.researchgate.net/publication/3247967_How_software_engineers_use_documentation_The_state_of_the_practice

Griffis, Gigi. 2013. How to Make Style Guides That People Will Use. https://blog.gathercontent.com/how-to-make-a-style-guide-that-people-will-actually-use

Horch, John W. 2003. *Practical Guide to Software Quality Management*, pp. 220.

Johnson, W. Lewis. Dynamic (Re)Generation of Software Documentation. http://www.researchgate.net/publication/2817866_Dynamic_(Re)Generation_of_Software_Documentation

Kiss, Fabian. 2011. Documentation in the agile software development process. http://www.slideshare.net/headrevision/documentation-in-the-agile-software-development-process

Moreira, Mario. 2013. Right-sizing Documentation in an Agile World. http://cmforagile.blogspot.nl/2013/06/right-sizing-documentation-in-agile.html

Philipp-Edmonds, Cameron. Best Practices for Lean Documentation: An Interview with Craeg Strong, http://www.stickyminds.com/interview/best-practices-lean-documentation-interview-craeg-strong

Raymond, Eric Steven. 2000. Software Release Practice HOWTO. http://en.tldp.org/HOWTO/Software-Release-Practice-HOWTO/documentation.html

Sommerville, Ian. 2001. Software documentation. http://www.literateprogramming.com/documentation.pdf

Vision to Value

Creating Successful Projects Using Leadership

Todd C. Williams
todd.williams@ecaminc.com
President, eCameron, Inc.

Abstract:

Value: Rather than scope, schedule, and budget, value is the lynch-pin of project success. Although scope, schedule, and budget are key factors in project success, there is no guarantee that meeting these constraints will result in a positive outcome. Instead constantly tracking the value of the project and making adjustments to the triple constraints to attain sufficient value is critical. Arguably this is the project manager's most critical deliverable in the project. It requires significant insight to the project's customer and a thorough understanding of their needs over their wants. Project managers have to be leaders (leading subordinates, leaders, and customers) and be able to assign priorities based on a global view.

Biography

A strong comprehensive strategic foundation coupled with operational excellence allows companies to build the capabilities to thrive. Todd Williams' goal is to improve how companies implement their strategic plans. Utilizing twenty-five years of experience he helps companies turn their vision into value.

His team at eCameron, whose mantra is "Strategy, People, Process, and then Technology," specializes in rescuing troubled projects and helping organizations drive business value from their strategy.

As an expert witness, and author of Rescue the Problem Project, A Complete Guide to Identifying, Preventing, and Recovering from Project Failure, he defines a project audit and recovery process for rescuing red projects that focuses on root cause correction and prevention. He also writes for The CEO Magazine, American Management Association, his own Back From Red blog, and contributes to numerous other publications, including Fortune/CNN Money, CIO Update, ZDNet, Enterprising CIO, and IT Business Edge.

He is also an internationally acclaimed speaker doing over 40 presentations a year throughout the United States, Canada, and United Kingdom, and Europe.

Copyright Todd C. Williams 2015

1 Introduction

Project Success rates are dismally low. In no industry is it worse than in Information Technology. Estimates for project success rates range from 25% to 40%[i,ii,iii]. To understand this problem one has to understand what makes a project successful. Experience indicates that most companies believe that it is completing the project within the scope, schedule, and budget defined at the onset of the project. I disagree.

2 A Tale of Failure

Let's explore a project that most of us are aware of and see if this definition applies. The project is the deployment of the Hubble Space Telescope—the history being in the public domain. The concept of a space telescope was conceived in in 1923 by the scientist Hermann Oberth; however, it took until the 1970s for technology to catch up and allow it to be considered feasible. In 1977 the project was funded with $200,000,000 and a projected launch date in 1983. The spacecraft, however, did not enter final assembly until 1985. Launch finally occurred in 1990 with delays coming from scope changes[iv] and the Shuttle Challenger tragedy. Heralded as a huge achievement everyone waited for telescope stabilization and the transmission of first spectacular images. Unfortunately, the images were anything but spectacular. NASA was faced with an orbiting telescope originally priced at $400MM and had ballooned to $2.5 Billion, was seven years late, with a "spherical aberration" in a mal-manufactured mirror. In other words, the mirror was ground incorrectly and could not focus on distant images. To any astronomer a poorly manufactured mirror is the cardinal sin. NASA had missed the basics in Astronomy 101. The mirror is the primary component of any telescope and to have it built wrong was simple inconceivable.

Scientists eventually developed a solution and in 1993 a shuttle visit delivered and installed a high-tech set of contact lenses that allowed the Hubble Space Telescope to deliver the stunning images that most of us have become so familiar with.

But the story does not end there. In 2003, shuttle tragedy struck again and the loss of the Columbia forced NASA to restrict all flights to orbits that could take safe harbor in the International Space Station. The orbit of the Hubble, however, is not coincident with the space station and the final Hubble servicing mission was canceled. The reaction was immediate and resounding. Hundreds of millions of people complained forcing Congress and NASA to develop an alternate solution over deorbiting the telescope. In 2009, the final servicing mission was flown on Shuttle Atlantis, with the Shuttle Endeavor in full ready to be launched if troubles on Atlantis arose[v].

The question to the readers who believe in the posited definition of success is, "How can a project that was over 6 times its original cost, was ten years late (doubling the timeline), and did not function as designed, get to be so successful as to merit the risk to human life to prolong the service of its product?"

It had missed the commonly used measures of success—scope, schedule, and budget—and was a successful for one reason only. To the millions of laypeople of the United States (who could lobby Congress) and hundreds of millions of laypeople around the world the Hubble Space Telescope had value—value that was measured in pictures and discoveries that fueled imagination, hope, and dreams. It is the intangible attribute of value that makes a project successful.

3 The Components of Vision Implementation

How, then do we deliver successful projects? The answer is to trace out the origins of the project to the vision, the strategy to achieve it, and the goals your project is striving to meet that will make the vision a reality. By meeting those goals, value is delivered and success is achieved.

The required change in our companies is that the project team must understand the vision and the strategic goals. The team must see how those goals change as the business environment changes and adjust its deliverables to match. It must understand and select people with the right attitudes and skills that help make those goals a reality. Only then, once the team knows where it is headed and have selected the proper people to get it there, can it define the processes to follow.

Process, often considered the foundation of project management, can be a project's bane. Too little process and the project will spiral out of control, too much and it will flail under the stifling weight of bureaucracy. The project's domain and its people's capabilities determine the level of process required. The higher the chance to loss of life or money and the lower the accountability of the people, the larger the role that process will play.

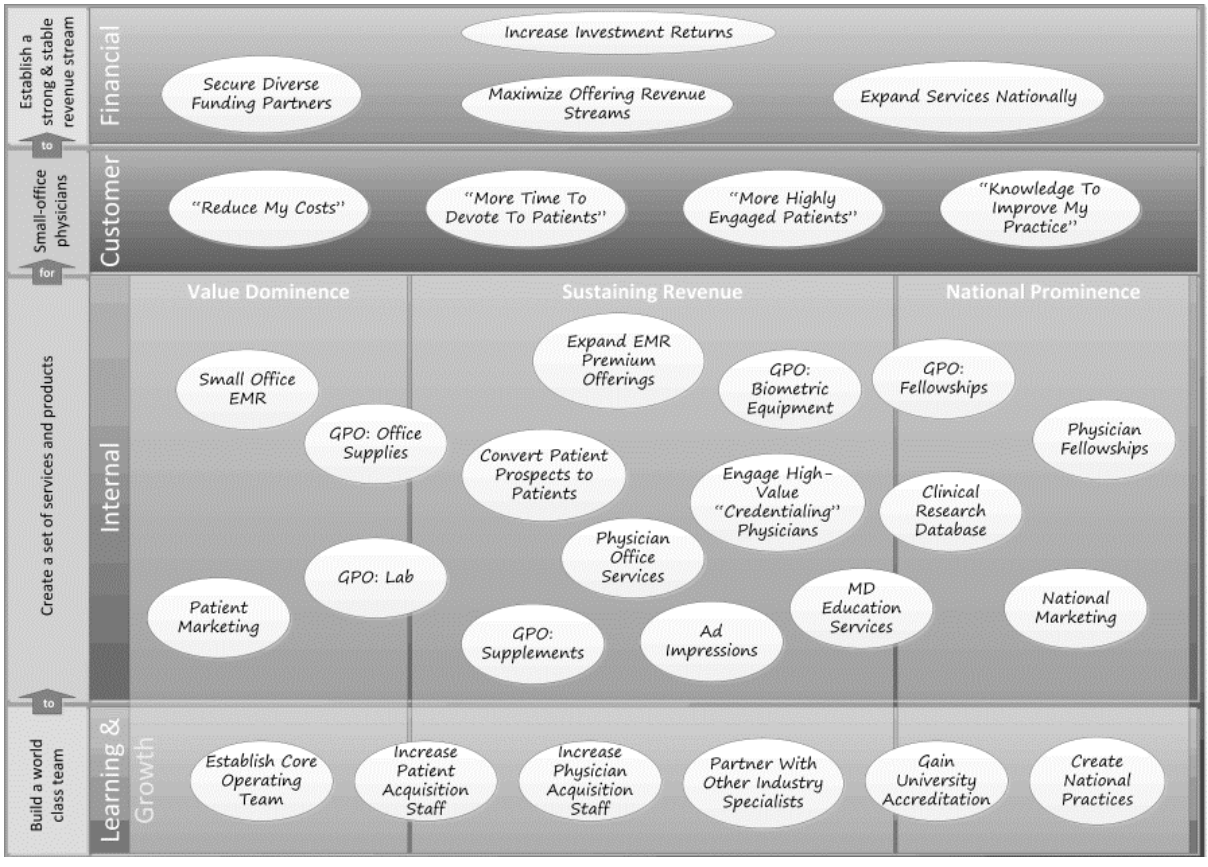
Finally we can address technology. Its role is to improve efficiency. Technology applied to ineffective people using poor process only makes your project get into trouble faster, albeit very efficiently. Today's culture often believes technology is the answer, but it is only a means to an end. As in the introductory example, neither sophisticated testing technology nor the reams of NASA procedures that could have saved the Hubble Space Telescope from its improperly manufactured mirror; it was human haste that pushed the production cycle and bypassed proper testing.

3.1 Aligning the Organization

Once the key infrastructure of strategy, people, process, and technology are in place communication trumps all. The challenge is to adopt a format usable throughout the organization. It must keep people focused on the strategic goals, flexible enough to accommodate changes in the business environment, and understandable throughout the breadth of business acumen of the people in the organization. It must relay four distinct areas of focus—the company's financial needs, the target customer's problem statement, the initiatives the company must instantiate to meet those goals, and how the company must grow to be able to execute those initiatives. There are many tools that help with this process, but I will use the balance scorecard defined by Norton and Kaplan to illustrate the concept[vi].

3.1.1 Financial

For any company or organization to survive there must be financial goals. Whether the organization's goals are to make a profit or provide services, without financial goals it will fail. These three to four goals must be clear and concise. Filtered through the customer's needs, these goals must ultimately be supported by the organization's projects.



3.1.2 Customer

Most visible and tangible to the organization is the customer's needs. These are often described in terms a customer would use. As shown in the example, they are often in quotes to underscore this point. As with the financial goals, these should be limited to a reasonable number of objectives. Optimally this consists of three to five goals. Every component of every project must be easily mapped to one or more of these items.

This is a crucial concept that every project manager must understand. If a component of a project fails to align to one of these goals that component should be removed from the project as it is diverting resources from achieving strategic goals. It is incumbent on the project manager (in conjunction with the project sponsor) to ensure this rule is followed. Failing to achieve this is the primary source of scope creep resulting in late delivery and budget overruns. Worse is that often these additional features are not what the customer needs or wants. The result is that project delivers less value.

3.1.3 Internal

The bulk of the strategy map is the internal needs section. This often consists of dozens of initiatives that must be completed to achieve the corporate goals. Each bubble could be a small as an individual project or as large as a multi-year program. They are, however, all decomposed into projects that provide some new capability to the organization. These are not operational activities (often represented in an activity map). Strategy, as Michael Porter points out in his seminal work published by Harvard Business Review, *What is Strategy?,* "attempts to achieve sustainable competitive advantage by preserving what is distinctive about a company"[vii]. Improving operational effectiveness is not strategic, since every company has to do this to stay competitive.

3.1.4 Learn and Grow

At the strategy map's foundation is the description of how the company must grow and what the people must learn. This section of the strategy map may contain initiatives to buy other companies, build new buildings, hire new expertise, train employees, or the like.

3.1.5 Telling the Story

Upon completing the Strategy Map, anyone in the company should be able to describe the goals of the company and how he or she is supporting it. In fact, many people often use the left hand column to generate the statement as to the type of work people should be doing that can be read from the bottom up. In the example map that statement is: "Build a world class team to create a set of services and products for integrative medicine physicians to establish a strong and stable revenue stream"

3.1.6 Implementation

As can be seen in the figure above, this can be a huge number of initiatives. Any company attempting to do all of these at the same time will fail. This is for two reasons—initiative precedence and spreading the team too thin.

Any organization must walk before it can run. Key infrastructural changes must be in place to support the higher level functions. For this reason, when generating a strategy map, there is an order of implementation implied in its design. Implementation starts in the lower left hand corner and proceeds to the upper right corner. This not only applies to the lower two level of the map, but also the customer and financial elements. The implication is the prioritization in meeting customer needs as some are fulfilled sooner than others. The same is true for financial elements.

For instance, if one develops a strategy map for a department or division of a company, basic processes going in the lower left while more strategic go to the right. If this is done, say, for an IT organization, implementing email or establishing reproducible transaction processing will need to be implemented and perfected long before it will be able to develop new applications for customer's mobile devices.

3.2 Projects Deliver Capabilities

Each of the projects that comprise these initiatives must deliver capabilities that provide value to the end-user. Time is the value's enemy. Value is lost by delivering the wrong functionality or delivering it at too high a cost. The longer it takes to deliver a capability costs the organization extra money in deployment and in lost opportunity costs. Lost opportunity, which is very difficult to measure, often

defeats a strategy as a competitor comes to market sooner with a solution to meet the customer’s needs.

Additionally, prolonging the time-to-customer removes critical early customer input on whether you are meeting the customer’s needs. Missing this early valuable input destroys an organization’s ability to make mid-course corrections.

At this point that we can point back to the initial premise of how a strategy is realized—through the prioritized implementation of people, process, and technology.

- The map defines the strategy with its foundation in people.
- People instantiate the initiatives and projects with minimal process in a staged manner that delivers value to the customer.
- Technology only provides efficiencies to rapidly enhance the capabilities and improve the customer experience.

4 Value, the Project Manager's Deliverable

A project manager's job is to deliver value. Achieving the original schedule, budget, and features are meaningless if the project fails to meet its value objectives. As with all simple statements, this is much easier said than accomplished. Project managers must assemble adaptable teams that use flexible, lean methodologies focused on this same core value. Arrogantly selling the latest technology or tool is narcissistic. Focus must be on value. Be vigilant at ensuring the required information is always available for the customer to reassess the project's value and for the project team to reevaluate their proposal.

4.1 Value Equations Won't Work

The first task is determining the customer's initial value objective. Most projects start with the premise that they provide considerable value. However, numerous assumptions are made in justifying that value. Many of these ill-founded hopes fail to survive the test of time, being proven false in the early stages of the project. As we all know, time begets change. Realities adjust as the customer learns about the product's possibilities, business models morph, and challenges arise in building the product. The project must transform to meet these needs and the project manager must lead this shifting vision.

At times, it is more than a gradual shift. Elucidation of major difficulties, discovery of poorly understood requirements, and loss of business segments, are a few reasons to step back and reconsider the project's original premise. The most radical change a project manager can propose is terminating the project. Once the projected value falls dangerously close to zero, the value proposition is invalid and the only sensible solution is ceasing the project. This is not failure. It is leadership at its finest.

4.2 Facilitating Increased Value

Value is the benefit reduced by the cost. Costs are generally quantifiable; however, benefits are often intangible. Goodwill, trust, esthetics, and usability are but a few attributes that can add significant value to a product. In addition to developing the project's cost, project managers must help the customer enumerate all of the benefits.

For the customer to define value, project managers must supply the information on how the product will or could function. There are no constraints to the original concept, added costs, thrown away work, or extensions to the delivery date are all part of the make-up of the product. The task is to objectively deliver the complete story and let the customer decide the next step.

This does not mean the project manager does everything the customer asks. The project manager must work with the customer's team and help them create their vision, understand their issues, and guide them toward a solution that delivers a value-laden product in the shortest possible time at the least cost.

4.3 Grooming the Team

This does not come from sitting at your desk working with spreadsheets. It comes from understanding the business' needs, the state of the deliverable, the team's capabilities, and the challenges of selling change.

Developing the customer's confidence and trust is the first step. A cohesive, agile, dynamic team is the primary ingredient in doing so. Integrate your teams with the customer, educate them on the customer's business, and immerse them in the customer's pain. This creates a responsive and customer-focused team. It reduces the tendency of teams to build products with the latest low-value gadgets.

Maintaining customer confidence is more difficult. This requires a culture and methodology that is adaptive. Too many times, draconian processes manage projects—methodologies that strive for operational excellence as opposed to product excellence. Granted, some customers (i.e., healthcare or military), require a thorough paper trail for every functional outcome, no matter how low its probability of occurrence. These are far from a majority of projects. Even in these projects, it is worth questioning the validity of the overhead and proposing new solutions.

4.4 Averting Failure

Delivering a project's features and functions on time and within budget is incommensurate with a successful project and a happy customer. Dozens of environmental factors affect a project's value. Successful project managers carefully watch these factors and lead the customer through the process of discovery, defining appropriate changes that maximize their project's value. Losing sight of the project's value will inevitably result in failure.

5 Vision to Value

The project manager must first understand the vision. These include the lofty goals of the progenitor, which are rarely thought out in any logical manner as to how they could be implemented and delivered. In many cases, the idea's applicability in the real world also lacks a touch of reality.

Turning vision into value takes equal parts of leadership and management. Understanding which parts to use is an art. Corporate leaders set the vision and define the organization's culture. If we do not imbue the qualities that build trust within our company and with our customers, growth will be elusive. Nowhere is this more evident than when your company's capabilities need to change and you start a corporate wide initiative. In these projects everyone must be aligned, communication and process is critical, yet you need to be innovative and push the envelope.

Nearly 30 years of dealing with companies large and small yield a few lessons on the balance that must be achieved in reaching for that new goal.

5.1 Managing Process and Attaining Compliance

Management is the easier task. We direct and we remain accountable. However, it limits our company's (not to mention our own) growth. Management is about applying processes, attaining compliance, and measuring performance against goals. Some key points to remember whether you are running a company or a project are:

Hire Expertise. Bringing in a person to fill a void is wrong. It is better to do without than to endorse mediocrity. Always hire the right people.

Apply Process. Process compliance is at the core of management. Properly applying process provides customers with consistency, never bureaucracy. Process stifles creativity. This is good; look what innovative accounting did over the last couple of decades.

Caution with Technology. Technology provides consistency and efficiency. Never apply it without first having the proper people and processes in place; otherwise, technology can create a bigger mess quicker and far more efficiently.

Manage the Goal. Define and manage scope, document decisions, and give the users what they need—question what they want.

Mind the Constraints. Scope, schedule, and budget, pick two and only two. The project manager will tell you the third. Trying to edict all three is the definition of a failure waiting to happen.

Learn to Compromise. You will never reach perfection. You and your customer are going to have to compromise. It is incumbent upon you to train your team in the science and art of negotiation.

5.2 Building Innovative Cultures

Businesses that excel are innovative. So are their projects. Projects, by definition (a temporary endeavor to create a unique project or service) have to be innovative. This requires building a culture around leadership and self-direction.

Maintain Objectivity. Too many project managers are overly passionate about their projects. Rather than rooting for their project like high-school cheerleaders, they need to maintain objectivity. They must be passionately dispassionate and determine what to amplify and what to discard.

Foster Teams. Teams find answers. Learn from them. Talk to them. Work with them. Sponsor and support them. Communicate with them. However, never do the work for them. Leaders let others lead and support them when they stumble. Let people make mistakes so they can learn and grow. Management means **you** know how; leadership means **they** know how.

Forget Blame. A culture of blame is the fastest way to destroy morale, teamwork, and trust. It is an infectious disease that creates finger-pointing and secrets. Do not search for blame. Once you have found it, it will only give you fleeting pleasure. There is still a problem to fix.

Heed Denial. Before any problem can be addressed, you have to first admit it exists. An open culture devoid of blame, which asks for help, and is open to new ideas, will avoid nearly every catastrophic failure. It is an integral part of any innovative culture.

Focus on Data. Relying on data and avoiding analysis paralysis is the foundation of good decisions. Numbers are truthful little bastards, squeeze them hard enough and they will tell you the truth. They cannot lie; it is integral to their job.

5.3 Turn Vision into Value

New strategies mean new business capabilities which beget projects. That is how companies grow and survive. Time to market relies on your company's ability to run projects efficiently and deliver results in the shortest possible time allowing you to achieve your desired ROI. These eleven traits help build a lean culture focused on speed of implementation. They will continue to be your focus; the challenge being establishing them throughout your organization. As the leader you must continually adjust and apply the right mixture of innovation and process. The right balancing of leadership and management will grow your people, your company, and your profits.

6 Projects Need Leadership, Not Management

It starts with leadership. Too many project managers spend their time trying to manage their customer. Trying to use directives (management's primary tool) on a customer is infinitely more difficult than leading them through the process of discovery and letting them see the faults in their plans. This is no more evident than when the role of a project manager is seen without its leadership component.

"Project management is easy. We have been managing people for hundreds of years. Just take any manager, give them a project, and tell them to get it done." Experienced project managers will accurately predict the end of this story—there is a disproportionate chance this project will fail. Rather than "manager" being the operative noun, a leader is required to deliver project value on time and within budget. To distinguish the project manager further—functional managers need only manage subordinates, while successful project managers lead extended project teams that include their managers, customers, and stakeholders. This fundamental difference drastically increases the project manager's scope of responsibility, since the project team includes an entire flock of stakeholders.

6.1 The Project Manager’s Purview

Functional managers are primarily responsible for their direct reports—the classical organization chart. On regular occasions, they coordinate with their peers or a boss, but their focus is on their staff. Project managers, on the other hand, must align a much larger array of people. Besides their project team and peers, they have an entire organization chart above them consisting of the project's stakeholders. In actuality, this loosely knit collection could be a significantly larger population than their well-organized subordinate project team.

Where I currently live, outside Portland, Oregon, USA, there has been a twenty-year long effort to build a new bridge across the Columbia River. The two, currently steel, structures, one constructed in 1917 and the other in 1958, were built long before seismic design and construction techniques entered into the mainstream. They are arguably the weakest link in the 1400-mile long freeway and only draw-bridges in the interstate highway that traverses Washington, Oregon, and California. There is a legitimate concern about their ability to sustain a reasonably sized tremblor. Without a doubt, it will be a massive project building this multimodal, dual-decked, mile-long bridge. It will take thousands of designers, managers, and construction workers. However, consider the stakeholders involved. They include: the federal transportation agencies, the US military (a reserve airbase is nearby), the Coast Guard, two state and two city governments, two transit agencies, light-rail proponents and opponents, bicyclists, pedestrians, local toll-paying citizenry, state tax payers (many hundreds of miles away wondering why they need to pay anything), boaters, businesses, truckers, commuters, environmentalists, and Native Americans, just to name a few. All have special interests; all can muck up the best project plan. Few of them know anything about project management; none of them care about the woes of the project manager. Without a doubt, the project manager will need to navigate more obstacles from the stakeholders than from the team actually building the bridge.

6.2 Training Superiors and Stakeholders

Your projects may be much smaller, but they still need project managers that can lead without authority and can train leaders and stakeholders who are ignorant of their project management deficiencies. Never expect executives and sponsors to know what project managers need to properly execute their jobs. It is paramount that project managers use these people as tools to get the project completed, which means training them on their jobs. Project managers must unapologetically assign them tasks just like everyone else on the project. They work for the project manager. The sooner everyone realizes this, the better the project will run.

To underscore the point, think back on the last few sponsors you saw assigned to projects. Did they want to be in that role? Had they done the job before? Did they ask for reports on progress or did they request assignments to help the project? Too often, projects inherit project sponsors as an afterthought—assigned under duress. Sponsors need the project manager's help in delineating what is required of them to make the project successful. This includes clarifying and constraining the project's scope, acquiring subject matter experts, finding the extra money when it is obvious that the project is bigger than anyone thought.

The executives? They should be mentoring project managers, helping with costs, and cutting through the politics. If they are not doing this, the project manager must teach them to do so. Most likely, the project team members understand their project roles; I doubt the leaders and stakeholders do.

6.3 Delegating Up

Of course, a project managers' job is to run the project; however, if they are confounded by a problem, it is better to ask for guidance than to flail and fail.

A few years ago, I was called in to fix a project that was going to be 200% over budget and schedule. Yes, 200%. That means three times the cost and three times as long. Sad, but true. The product would benefit two departments; only one was funding it. My investigation showed that nearly all of the problems were "above" the project in the management hierarchy. The leadership was dysfunctional. A Vice President for the non-funding department requested that one of the project's team members blind-copy her on all emails and communications regarding scope. The VP would then use this information to have her team bias the requirements in her department's favor. Upon discovering this, I bundled up the evidence and trudged into her boss' office—an executive three layers above me in the organization and second-in-command for the multi-billion dollar company. I made my case in a logical and dispassionate manner asking him to stop the covert action. By the time I returned to my desk

(three blocks from the executive's office) the reverberations had hit the project team, with a memo reprimanding the use of the blind copy feature. He took care of the situation as I left his office. He wanted to help, he was unaware of the problem, and when he became aware, helped me regain command and control over my project by removing the meddling manager. Less than a month later, I had to invoke the assistance of another executive, this time the VP of Information Technology, asking him to stop the drone of negativism from one manager regarding my recovered project's team. The offender apologized for hindering our progress. Executives want to help; they simply need us to tell them what to do.

6.4 Success is Contagious

Stepping up and being a leader, helps you, your peers, and the entire organization. Leadership begets success and success is contagious. Peers mimic the victories. Your actions will improve the company's culture and the change sticks because everyone benefits. The value is evident in what you do and in your project's deliverable. Change that people understand (both logically and emotionally) does not meet the same apathetic demise of other change efforts. It only requires that you focus on six directives:

Leading your team mates

- What do you need from me?
- What's in your way?
- How can I help?

Leading your leaders:

- I need you to help me by...
- I need mentoring on ...
- I need you to clarify...

7 Leadership And Attitude

Leading leaders is not easy. Some say it is like pushing string. I understand the metaphor, but it makes me think the people saying it are stuck looking at the problem wrong. The problem is the predisposition to the inevitability of the issue—there is no reason to look for a solution because it is out of our control. Worse than that, we are so defeated that we rarely ask the question "Why are we trying to push that string?"

7.1 The C-Suite's Offhand Request

This is never more evident than when orders come down from senior executives. Someone in the executive team mentions to his or her direct reports that a new widget would be nice and, within minutes, the entire organization is realigned to complete the new widget to please Ms. or Mr. C-Suite. Middle management overlooks the alternative of asking some simple validating questions to confirm priorities. Rather than probing and explaining the impact of implementing the new request, all other tasks become subordinate and everyone's schedules are re-planned to the new precedence.

7.2 Leading Your Leaders

The source of this problem lies at the feet of the executives. They are the ones that set the tone and create a culture that is reactionary to their requests and stifles the subordinate's ability to question and set priorities. In these cultures, middle managers spend an inordinate amount of time guessing how to please their bosses at the price of getting the company's work done.

At this point, many would resign to throwing their hands up in despair, surrendering to the notion that management is all fouled up. My approach is to "push a little string." Executives do not have an exclusive right to leadership. Leadership is a trait that all of us should be honing—regardless of our organization's culture. All these situations need is a little upward leading. Here are some rules to follow:

Be passionately dispassionate. Objectivity is paramount. Passion is what everyone says they want, but when you are solving a problem, where emotions flare, stick to the facts. Make sure the pros and cons are objectively laid out in a logical manner to make a decision.

Explain the problem. As so elegantly said by NASA's Mr. Wayne Hale, "remember that your leaders are not very smart." Assuming your leaders know the detail, or even the subject, of the issue you are addressing is a fatal mistake. You know every intimate detail of what you and your team are working on; your leaders do not, nor should they. They need the problem explained in concise, high-level, decision-making terms so they can give informed direction.

Tell your leaders how to solve the problem. Always have two or three viable solutions to the problems you escalate. Their job is to make decisions rather than figuring out all the workable solutions. They hired you to come up with the options.

Ask your leaders for clarification and mentoring. If you and your team are having trouble establishing a set of practical solutions, ask for guidance. Although your leaders are often far from the technical aspects of your job, they once were doing what you are now, maybe with a typewriter, but they were there. They have a wealth of experience. Remember the adage, "Old age and treachery will out maneuver youth and skill."

7.3 The Right Problem Gets The Right Solution

The challenge is not pushing string. The challenge is looking past the obstacles and biases and creating new methods to address them. What may seem impossible is only that way since we framed it incorrectly. Change the frame of reference and we can create new innovative solutions. In the case in point, deliberately taking steps to adopt the traits of a good leader, rather than sticking with those of a follower, will calm the seemingly knee-jerk reactions of lame leadership.

8 Conclusion

As shown in the opening example on the Hubble space Telescope, value is the measurement of a project's success. Tools help ensure a project strives toward value; however, leadership, focus on strategic goals, and communication are the attributes that makes it happen. Tracking scope, schedule, and budget is insufficient. Two challenges exist, often impeding progress delivering value are:

1. Project managers driving to employ those traits while in the headlong battle to get their project done.
2. Executives maintaining a current, concise, and coherent strategy and empowering project managers to implement it.

Leadership throughout the organization is paramount in creating success. Focusing on the high impact areas such as leading up, leading the customer to what they need rather then what they want, politely saying no, ensuring your project accounts for change management, and focusing on value are critical to move from the current dismal success rates to the norm of success.

---

[i] *CHAOS Summary 2009 Report*, The Standish Group International, Incorporated, White Paper, April, 2009.
[ii] Failure Rate, Statistics over IT projects failure rate. IT Cortex, http://www.it-cortex.com/Stat_Failure_Rate.htm (accessed 6/7/2015).
[iii] *Analytical Perspectives, Budget Of The United States Government, Fiscal Year 2009, Government Printing Office*, http://www.gpoaccess.gov/USbudget/fy09/pdf/spec.pdf, page 169 (accessed 11/18/2010, PDF available upon request).
[iv] *A Brief History of the Hubble Space Telescope*, NASA, http://history.nasa.gov/hubble/ (accessed 6/7/2015).
[v] *Hubble Space Telescope*, Wikipedia, http://en.wikipedia.org/wiki/Hubble_Space_Telescope, (accessed 6/7/2015).
[vi] Robert Kaplan, David Norton, 2004, *Strategy Maps*, Boston, MA, Harvard Business Review Press.
[vii] Michael Porter, 1996, *What is Strategy?*, Harvard Business Review https://hbr.org/1996/11/what-is-strategy, (accessed 6/7/2015)

Agile Portfolio Management: A Journey from T-Suite to C-Suite

Gokul Suryadevara

gsuryadevara@monsooncommerce.com

Abstract

The first principle behind the Agile Manifesto says: "***Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.***" Many questions come up as we explore the agile world:

- How does an organization know which software feature has the highest value?
- How can they develop an early understanding of the value of the software?
- How do they know how to divide the software into increments and what increment should be tackled first?

This paper focuses on a simple process to manage the portfolio that engages stakeholders, de-centralizes decision making, helps with gathering facts to make informed decisions, provides lean metrics, and aligns business closely with the development teams. If you are thinking about developing your own agile portfolio management processes, this paper might act as a starting point.

Biography

Gokul is a Software Project Manager and Product Owner at Monsoon Commerce Inc., a dynamic e-commerce solutions provider, headquartered in Portland, Oregon. Gokul started his career as a Software Developer to develop a Digital Library, a project funded by the NSF. He spent a few years in the financial industry working for Fiserv and Fidelity Investments as a Software Engineer. He then moved to Biotronik GmbH, a global medical device manufacturing company, where he managed the design, development and rollout of large projects like Manufacturing Execution System, Customer Relationship Management and Business Intelligence.

Gokul has an MS in Computer Science from Oregon State University and is currently pursuing MBA in Executive Leadership. Gokul is a certified SCRUM professional, Scrum Master and Product Owner.

© Gokul Suryadevara, 2015

1. Introduction

Organizations often work on multiple projects at the same time. It is important to understand the relationships between these projects, as well as how the delivery of the projects impacts the internal teams and customers. Executive managers are hungry for "***the completion date.***" They want to know when a project/feature can be completed so that they can start socializing it with the customers. Their view is that if there is no completion date, and then there is no revenue.

This paper is focused on the project portfolio - building a portfolio with heavy engagement from the bottom of the pyramid (the team) to the top of the pyramid (executive management – the C-Suite) - in an organization that has adopted agile practices. This paper will emphasize the various phases of the portfolio management and communicating to the C-Suite to gain support and funding.

The main goal behind the portfolio development is to follow the response to change from the agile manifesto: "***Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage***"

2. Agile Organizational Structure

The agile organizational structure consists of 3 layers:

- **C-Suite**: The Corporation's senior executives. The C-Suite is considered the influential group of individuals at a company. Being a member of this group comes with high-stakes decision-making, a more demanding workload, and high compensation.
- **M-Suite**: Mid-level managers. The M-Suite is considered the most experienced managers who guide the teams to reach their goals. They use their experience to tackle strategic business problems and coach the teams and team members to give their best.
- **T-Suite**: Development team that consists of Developers and Testers. The T-Suite is considered the ***most important team*** at a company. Their continuous attention to technical excellence and good design help in improving customer satisfaction.

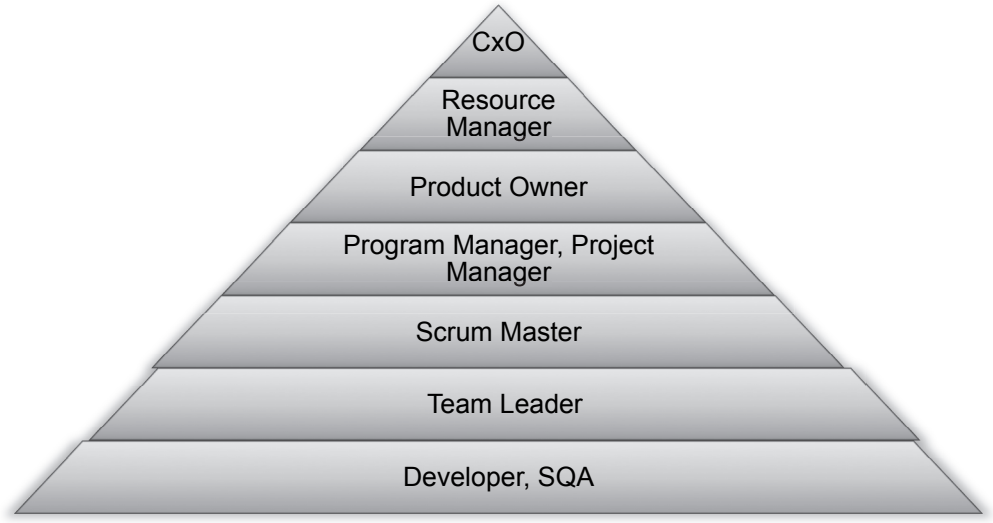


Figure 1: Agile organization structure

3. Team's Perspectives

Organizations seldom involve the teams until the idea takes a shape. The development team, however, wants to engage from the beginning (i.e. the product idea phase).

Why is it necessary to engage the development team so early?
The agile manifesto says: "*Business people and developers must work together daily throughout the project.*" In addition:

- Engaging the team *early* and *often* is key to success.
- Allowing the team members to *continuously think about architectures and designs* from the idea phase itself, could positively impact the projects that are work-in-progress.
- The team may feel *insecure* that C-Suite and M-Suite are making the decisions without their input.
- The principle of "*staying close to the customer and being the customer's voice*" not only applies to the product owner, but also to the team. This can be a motivational factor for a better team performance.
- Team wants to continuously deliver working software, even though the business itself prefers longer delivery cycles.

4. Executive Management's Perspectives

While the team thinks they should be engaged early and often, executive management wants to know the size of the idea and when can it be implemented, in a matter of hours, if not days. Again, their goal is to know when they can take the feature set to the customers. While interacting with the executive managers, they have many questions. For example:

- When can the idea can be implemented and delivered?
- What is the size, when compared to the previous features/projects that were completed?
- When can we sell the feature?
- What effort will it take to complete the feature quickly?
- What if we hire more team members, can the team deliver sooner than later?

So, there is a big gap between the perspectives of the two groups and the gap could be bridged with a portfolio management process – a process where the stakeholders are engaged from idea to implementation.

5. Agile Portfolio Management Process

The Portfolio Management (PM) process connects strategy with execution, shows dependencies, and helps the stakeholders make informed decisions on the product delivery that has customer value.

The steps of the process are outlined below.

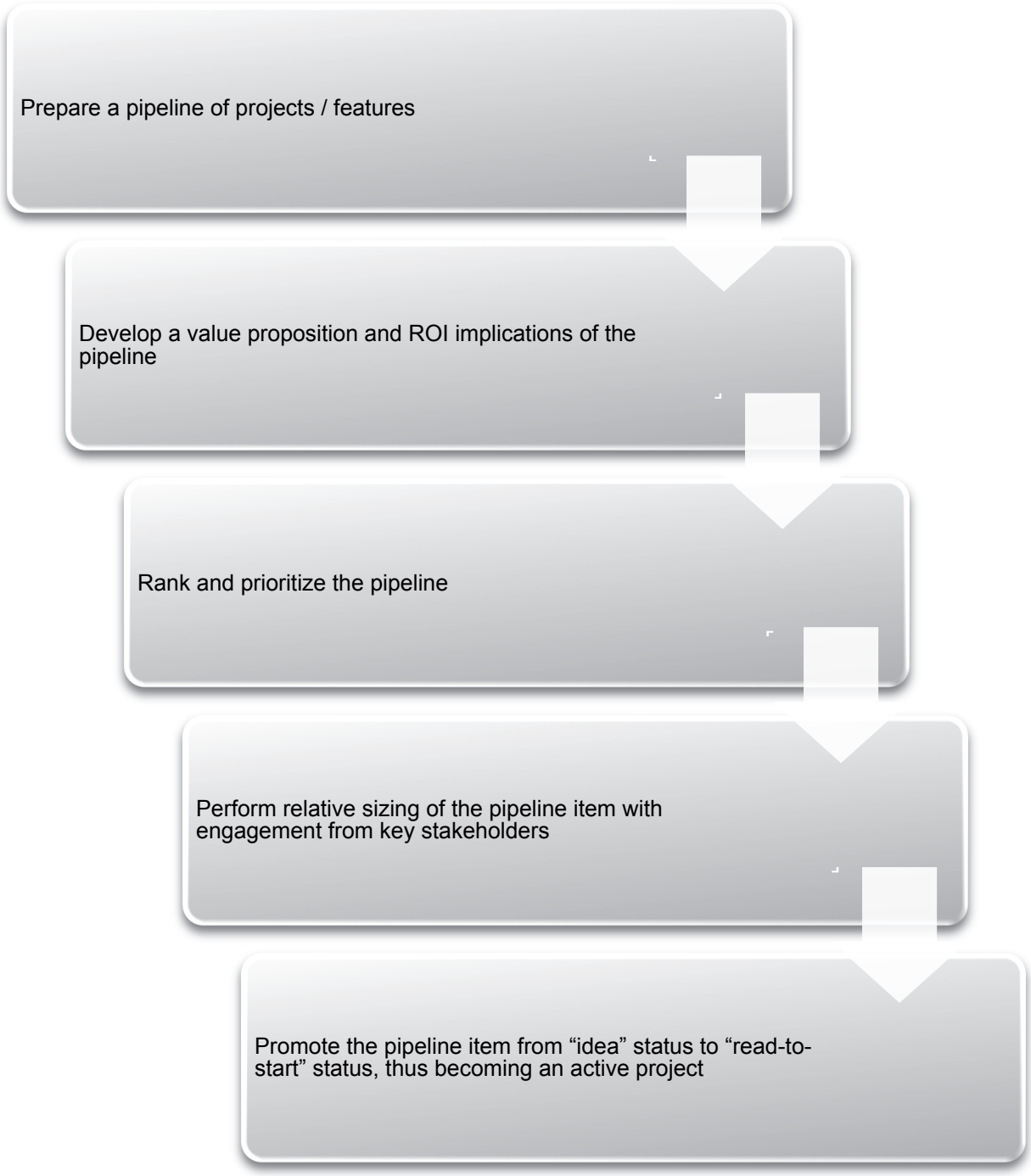


Figure 2: Agile Lifecycle Process Steps

The inputs, outputs, and stakeholders required in each step of the process, are discussed in the table below.

Step	Description	Input	Output	Participants
Preparing a pipeline of projects	List of project candidates	Project request that consists of: ▪ Idea ▪ Problem statement ▪ Objective	▪ Categorize the request in project portfolio ▪ Assign owner/domain SME ▪ Socialize the concept/idea ▪ Research the idea with cross-functional team members informally	Idea generator or the product owner
Develop value proposition	Prepare a business case	▪ Gather high level scope ▪ Cross-functional team review of solution recommendation ▪ Detail out the business impact & benefit	A short precise version of business case	Project Manager and representatives from cross functional teams
Ranking and prioritization of pipeline	Get the key stakeholders together to rank and prioritize based on the information gathered for each pipeline item in the previous step	• List of un-ranked portfolio backlog/pipeline items	Ranked portfolio backlog	Project Manager and representatives from cross functional teams

Perform relative sizing	Compare with the previously completed projects in terms of the duration, complexity, and team members required	• Rough order of magnitude (ROM)/initial estimation schedule/budget /resources	Size in terms of hours or story points or whatever metric that the teams feel comfortable to align with executive management	Project Manager and representatives from cross functional teams
Promote the pipeline item	Project is defined. Ask the executive team to release funds to start the project.	▪ Define the project stakeholders and team ▪ High level design brief, and research ▪ Entity diagrams ▪ Revised schedule, budget, and resource estimates ▪ Risk Assessment and Mitigation ▪ Establish Success Criteria for Beta	Project kickoff meeting	Project Manager and representatives from cross functional teams

Figure 3: Agile Portfolio Lifecycle

6. Portfolio Management Metrics

You should develop a simple information system to track the projects and their statuses. The information system could contain the following key :

1. Project Idea → Short name of the idea/project
2. Project Status → Status is the step a project belongs to. Refer to Figure 1: Agile Portfolio Lifecycle)

3. Deliver Team → The team(s) that could deliver this idea
4. Priority → High, Medium, Low
5. Product(s) impacted → Capture the product(s) this feature could impact
6. Project Category → The categories depend on the type of business that the organization is operating under. Figure 5 shows some of the categories represented in a software product development organization.
7. Sizing metrics → capture story points, hours, budget, complexity etc.

Some of the metrics below represent the data captured in the information system.

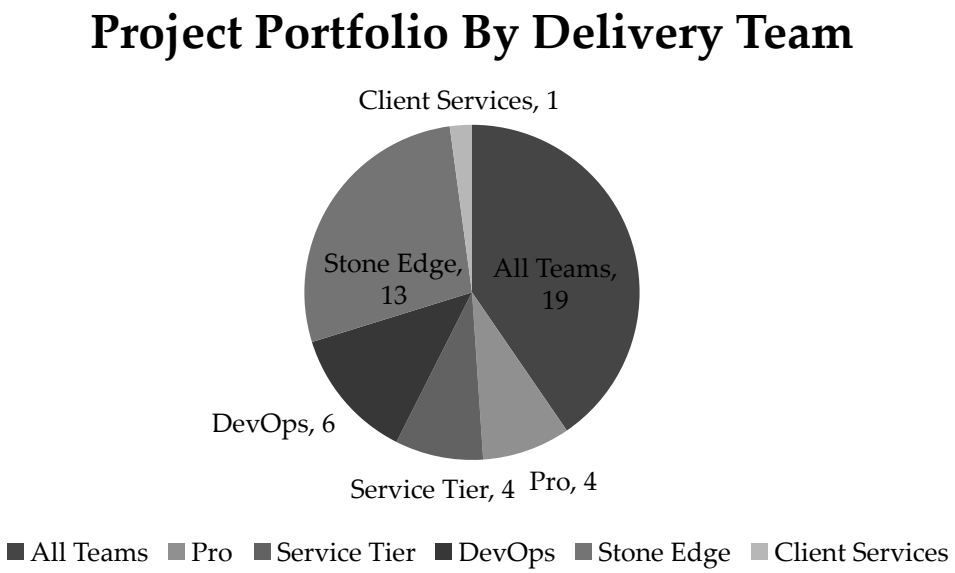


Figure 42: Number of Projects by Delivery Team

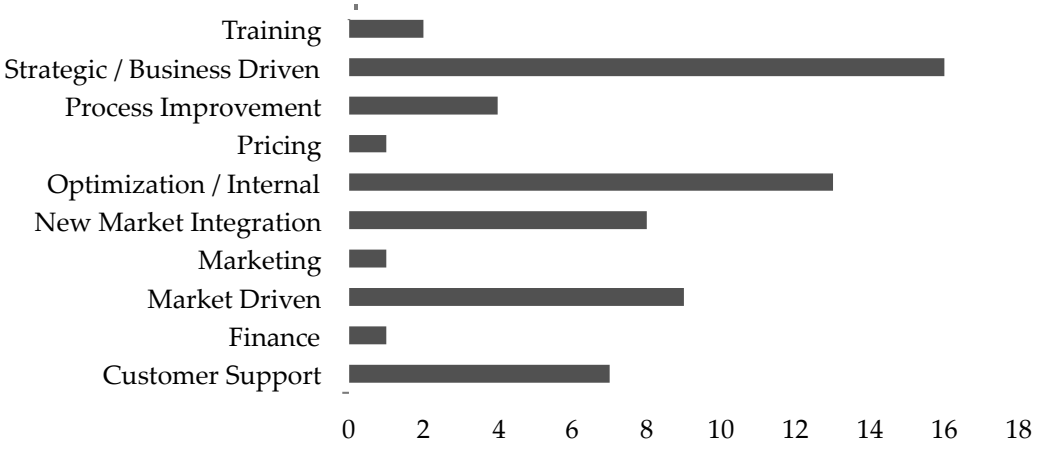
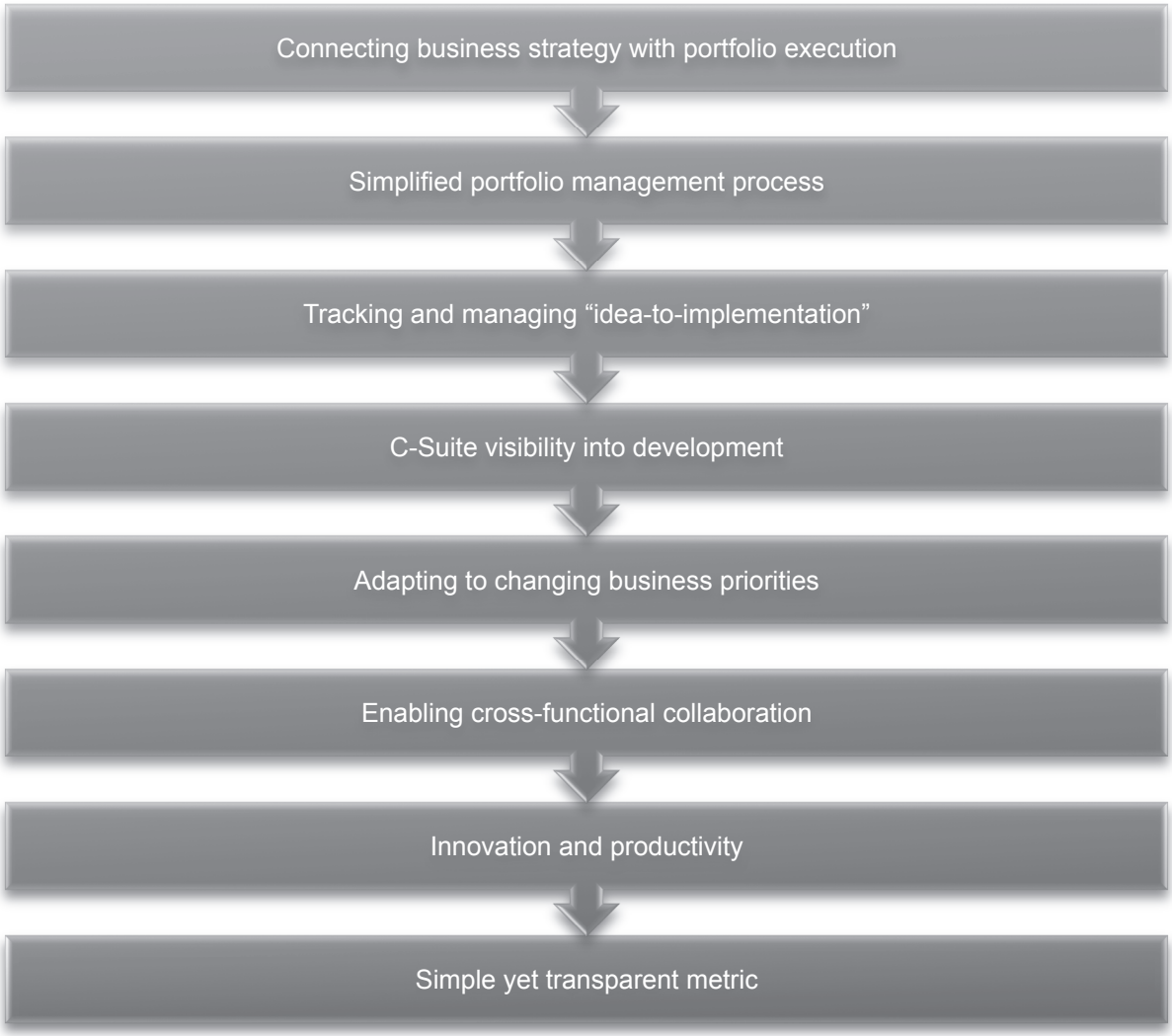


Figure 5: Number of Projects By Category

Figure 6: Portfolio Kanban Board

7. Summary

Portfolio management is one of the important functional vehicles for organizational transformation. It is the bridge between the present and the future, and guides the organizations to understand whether executing an idea is of any value. Project, program, or portfolio success is no longer limited to the scope, schedule, resources, or cost. Rather, it is multi-dimensional and we need to, as practitioners, embrace this reality and look to finding and implementing ways to make this success simpler and easier to reach. Recap of the journey is mentioned below.



References

Agile Manifesto. (2001). *Principles behind the Agile Manifesto*. Retrieved from Agile Manifesto: http://www.agilemanifesto.org/principles.html

Connor, C. (n.d.). *Rally Portfolio Manager: There is more than one way to look at it… - See more at: https://www.rallydev.com/blog/product/rally-portfolio-manager-there-more-one-way-look-it#sthash.fdOgqQRD.dpuf*. Retrieved from Rallydev.com: https://www.rallydev.com/blog/product/rally-portfolio-manager-there-more-one-way-look-it

Investopedia. (n.d.). *C-Suite*. Retrieved from www.investopedia.com: http://www.investopedia.com/terms/c/c-suite.asp

Krebs, J. (2007, July 30). *Managing an Agile Project Portfolio.* Retrieved from Scrum Alliance: https://www.scrumalliance.org/community/articles/2007/july/managing-an-agile-project-portfolio

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise.* Boston, MA: Addison-Wesley.

SYSTEMS THINKING TESTER

Russell J. Smith
savtech138@gmail.com
https://www.linkedin.com/in/rsmith85

ABSTRACT:

With the implementation of Systems Thinking as a mindset, and pulling from Systems Thinking principles, I believe that testers can be more valuable to their organization in at-least 2 ways. 1- They can apply Systems Thinking to their own existence in the system they live and work in. 2- They can educate and coach the teams they work with and improve the system as a whole. Because of the unique position we are in, and the number of individuals and teams we interface with, we are at an advantage to spreading ideas just as bees spread pollen between flowers and thus having influence on the environment in a positive and meaningful way. Because Systems Thinking can be applied so vastly I have chosen to narrow the focus of this paper to using Systems Thinking laws as a Tester in a non-management role, though some areas may be relevant or touch base on test management or organizational management and situations where you are interfacing with management. This paper will be guided by the Systems Thinking laws as described by Peter Senge and how they are at play during your everyday life as a Tester. It is aimed at helping you recognize the situations in which non-Systems Thinking is at play and how to apply Systems Thinking during those situations. Anyone reading this should be able to extract the concepts and apply them to what they do regardless of whether they are a Tester or Test Manager.

BIOGRAPHY:

Russell Smith is a Software Test Professional and Quality Advocate who started his career in 2006 doing Technical Support and Black-box testing for a high-tech GPS producer in Torrance, CA. Over the course of his career he has held positions anywhere from "Junior QA Engineer" to "Software Test Architect". He has worked on dozens of products and systems deployments and has contributed to projects that span multiple teams with dozens of people involved across multiple countries. Most recently in his career he has been focusing on building high performing software testing teams while at the same time staying hands-on as an individual contributor. He does not believe in "Ivory Tower" management and loves to "get his hands dirty". Through his career he has never lost his focus on the customer and that what he does means nothing unless his customers are satisfied. This combined with his technical skills and leadership abilities makes him a world-class software test professional and customer-centric software quality advocate. Russell now lives in Portland, OR and works as a Test Manager at Janrain, Inc., a leading provider of customer identity management and social login technology that helps businesses manage and understand their customer profiles and profile data.

1 Introduction to Systems Thinking

Systems Thinking stems from General Systems Theory and was popularized by Ludwig von Bertalanffy in the 1940's [1]. There have been many others who have advanced and greatly educated the world on systems thinking such as Deming, Senge, Weinberg and Ackoff. In the most basic sense, Systems Thinking is really just a set of tools, concepts and ideas that can be applied to dealing with messy complex systems. Note that it is not a magic wand or something that solves all of your problems for you. But with a better understanding and discipline of Systems Thinking you will be well equipped to navigate and drive the system in a more aware, intelligent and genuine way in order to achieve a desired outcome and know how you got there. As you dive deeper you may also find yourself crossing into System Dynamics, which shares concepts with Systems Thinking such as feedback loops and time delays [2], but tends to be a bit more math-heavy and uses computer aided simulations. Once you begin to adopt a systems thinking mindset you start to see the connectedness of the parts and can act more intelligently about your decisions and actions when dealing with problems of complex systems. You start to see that focusing on individual parts without understanding how the parts connect [7] and impact the greater system is rather short sighted and naive. The absence of Systems Thinking is to focus on individual parts through pure analysis and with a lack of consideration for the system the parts belong to. Without Systems Thinking people would naively assume that a solution is "the best solution" or "the right solution" without having genuinely framed the problem space they are solving in and considering alternatives. It's not to say that the decision made is right or wrong, but that there are always alternatives that you can model and tools that you can use from Systems Thinking to test your decisions and measure them. As an example we can take the transportation system. Serendipitously as I was writing this paper I stumbled upon a report for just this: http://www.cts.umn.edu/Publications/ResearchReports/reportdetail.html?id=530. In this report the author uses systems thinking to 1. identify the problem space 2. understand the relationship that the system has to other systems 3. suggests alternate recommendations for sustainable growth based on the previous two points. As cited from the report, "This study is an effort to better understand the linkages between land use, community development, and transportation in the Twin Cities metropolitan area. It is designed to investigate how transportation-related alternatives might be used in the Twin Cities region to accommodate growth and the demand for travel while holding down the costs of transportation and maximizing the benefits." This, I think, is a perfect example of applied Systems Thinking. In the context of a Systems Thinking Tester I would expect this paragraph to read something like, "This study is an effort to better understand the linkages between the Database, API and User Interface in Project X. It is designed to investigate how testing alternatives can be applied to maximize test value, effectiveness and defect discovery or prevention while holding down cost". You can keep going up a level and applying the same concepts such as the reason for having a Testing/Quality initiative or a particular team within your organization. The beauty of Systems Thinking is that the concepts and tools can literally be applied at any level you want to apply them to. Anywhere from devising a test strategy, to building a product, building a team, running a company and so on.

2 Systems Thinking Principles Applied to Software Testing

2.1 Senge's Laws of Systems Thinking - Applied Scenarios

I found the easiest way for me to guide this paper was to simply use Peter Senge's 11 Laws of Systems Thinking [3]. Reading below you will see each law's title and a replacement text that changes his words to my own, which are focused on how they relate and are relevant in the context of software testing. It is written such that common scenarios will be described where the systems thinking laws are at play and how to treat them for a chance at a better outcome. If taken and practiced, I believe these laws provide a great entry into a deeper journey and provide the foundation for testers to push from when trying to elevate their effectiveness and existence in working and living in complex systems.

2.1.1 Today's problems come from yesterday's solutions

This law may be self-explanatory, but at the expense of seeming redundant I'll try to define it before giving an example. My understanding of this is that it is the ability to create new problems with every problem solved. Inevitably when you produce a solution you introduce something into the system or change something about the system that will become a problem for something else in the system. Without understanding the trade-offs you may end up with more problems than it's worth having for your one solution.

As a tester you are responsible for coming up with solutions all the time. It's not just developers creating solutions and problems. Automation, Test Approaches, Test Tools, etc. These are all solutions to problems you face, and in-turn your company faces them too. Each individual's problem in a system is the system's problem. If you propose a solution to do automated regression then you should be thinking through the required training and maintenance that comes along with it. You should also consider how accurate the automation will be at the thing that it's doing. Perhaps an unintended consequence is that your automation returns a false positive, in which case you pass the build and a customer ends up with an undesirable experience. So now you have the added complexity that **the solution that checks your solution is correct also needs a solution to check that it's correct**. Turtles all the way down as they say. So maybe this is not the least fallible solution after all. Going up another level we can ask why the automation was even asked for. Did we have an issue with regressions which provided evidence to support the cost of adding an automated way of regression checking? What are we doing about why we have regressions instead of focusing on reacting to a symptom? Asking these types of questions is kind of like doing a premortem. This is an exercise I advocate that all teams and individuals do to question their assumptions and claims. So you see- because your solution was automation yesterday, today you deal with maintenance cost, specific skill-set to build/own it, possible brittleness which takes time away from finding bugs because you are fixing/extending/refactoring/optimizing/training for something else. You will ultimately have worse quality in the end because your team is not finding any new bugs or you delay schedules. I'm not saying automated regression checking is a bad thing but it's not always the right thing, and good testers should realize this and be able to have conversations with their teams about the trade-offs and total cost of ownership for the decisions that are made.

2.1.2 The harder you push, the harder the system pushes back

In Systems Thinking terms this is referred to as compensating feedback. Initially it may be difficult to see this in your life as a Tester. You may not be able to think of scenarios where this occurs in your day to day tasks, such as coming up with test scenarios, setting up those scenarios or executing them. However- I have come up with a few that I think might hit home.

1) You frequently send out long reports about how bad a process is or the quality of an area of the product

Result: Depending on your approach will determine if anyone listens to you or not. If you complain in a negative or impolite manner, then you may be ignored even if you are right. Quality does not improve and the situation does not change because you continue to berate the product or teams in a way that is off putting. Even if you are kind and thoughtful about it but your message is so long that it takes your teams 20 minutes to read them, they won't get read it. Maybe at first they tried to, but the more you sent these long messages the less people read them until they reached the point where they stopped completely. Eventually you have an email rule created just for you that dumps your messages into a folder labeled "Fireside Reading". You have to understand your audience and be aware of yourself when you are trying to send your message. If you send a long message and don't hear back at first, then you should go face-to-face with some of the recipients and try to collect feedback so you can adjust your approach next time.

2) You want a developer to see his/her bad ways, but they refuse to

Result: This is similar to the above point. If you are not careful to understand the developer's personality or feelings, then you can create a situation of hostility where things only get worse. You push them the wrong way for better quality builds, and they in turn feel pressured or even angered which could lead to them delivering even worse builds. Or perhaps they deliver the same build, but when you have questions or need help setting up a test scenario, they will be less inclined to be responsive or help you. All of which will end up creating a poor experience for you and likely a poor performance for the team which could result in poor experience for your customer.

3) You are worried about the quality of a product and the lack of time to test it adequately, so you decide to work late a few night to make up for it.

Result: I can relate to this example quite possibly more than any other I listed. There have been many times when I wanted to play superhero and save the schedule by putting in extra long hours myself. Huge mistake. Usually what this led to was me being late to work the next day and then trying to cram more work in shorter hours. That didn't work out, so I made up for it when I got home. But then I had to cram in my dinner and time with my family. Tried to cram in more make-up time with work before bed which shortened my sleep cycle. Made me late for work again and then the planned testing I wanted to do slipped as well. A perfect example of this principle at work I think. What you should do in this case is make your case for more time to test if you really feel it is important.

These are just a few of the examples I can come up with. Now if you just think about this from your own perspective, you will come up with more like this. Take the same approach to thinking about your own examples of this, and try to think where else you see this happen. For example I can think of some examples where I see teams suffering from this such as:

1) We want more features added so we add more stories to our sprints
2) We want less defects in production so we add more testers to the team
3) We want faster delivery of features so we hire more developers
4) We want less distractions so we create an intake process

2.1.3 Behavior grows better before it grows worse

Here we have the case of a short term solution that causes a bigger problem down the road. Initially you see improvements in the project or product, but later in time, and made possible through compensating feedback [4] (as mentioned in 2.1.2), you find yourself in a bigger mess. In addition to short term solutions you may be treating the same symptoms for a long period of time while missing the underlying systemic cause for those symptoms. Homeopathy may be a good example to showcase how a system can get better before it gets worse, where you are treating an illness with a placebo and, although the patient may trick themselves into feeling better, they are actually allowing the real disease to fester and grow into a worse affliction. Eventually the patient is now at a further stage of their illness which may not be as easily treatable as if they had gone with a more scientifically proven or medically proven treatment option.

An example that may be common for Testers is when you have an insurmountable amount of Testing to do with a schedule that seems too short to accomplish the Testing task. Your test manager or you yourself may decide to "cut" certain tests based on what you perceive as risk factors. You may even consider and ultimately decide to cut entire contexts from your testing strategy. Rather than raise concerns about the schedule and the reason that there is not enough time to do all planned testing, you proceed forward with the "slim down, schedule safe" plan. Great Success! You made your release date and shipped the product. Mission complete and on-time to meet the customer deadline. -Fast forward a few weeks- Bug reports and escalations start to roll in. P1's and long support queues build. The customer and the delivery team that were once so happy are now realizing the mistakes they have made, but each

still has a chance to make it worse or begin to make it better. Deciding to quick fix these issues may actually lead to another dilemma which is addressed in the next law. As you might be able to tell- all of the laws are connected as well. Instead what the team might have done to avoid this situation is explore alternatives. They could have cut the release commitment down to a reasonable level. They could have asked for more time. They could have added more Testers. They could have done a better job at their risk factor analysis. Of course each of these options holds the potential to have its own problems, but without playing out the alternate scenarios ahead of time, you are setting yourself up for unexpected and surprising failure. Rather than feeling a nervous anticipation, you are blind-sided by reality meeting your ill-preparedness. This example is actually a small time delay in comparison to architectural design decisions (or lack thereof) of a system that are found in time to not be sustainable. Again- the laws are at play on multiple levels depending on where you look or who you ask.

2.1.4 The easy way out leads back in

I recently saw this in action while talking with a friend at his company. He was part of a team that was having many distractions due to escalations, bug fixes and customer requests coming into their Engineering department. Because the developers and testers felt distracted by these things, causing them to complete their new feature work slower, the management team decided to institute a "Maintenance Team". This team was tasked with handling all incoming issues that required a code change. Any other issue could be handled by their support team. The maintenance team was there to help alleviate the distractions from the core Engineer team. Can you guess what happened? Of course. The distractions were moved up. Actually they weren't even fully moved up because the maintenance team still needed to ask the core team questions, since the issues came in from many different areas. So distractions did not go down, feature development went down because of the developers moved off core work items and placed onto the maintenance team, and the team tasked with handling distractions now had all the distractions! Wow! A tactical quick fix led to the same issues they were trying to solve, just at a different level. From a Tester perspective we have probably made this same kind of decision though. For example I can think of several scenarios where I have either done it myself or seen someone do it. Examples:

1) Test Case Creation

Detail: I've interviewed a lot of Testers in my career thus far. One of the questions I used to ask was something I stole from James Bach. James was lecturing at a college where he showed the class a flow diagram with 2 options. If the input was greater than a number, it would be true, and if it was less than a number, it would be false. He then asks the class how many test cases would be required to test this. When asking candidates this same problem, about 90% of them said, "3 test cases". Some even said "3 test cases at a minimum". It was always fun to see how many more they could come up with before they ever asked a single question about the problem. They would throw out answers like 3, 4, 5, or even 100 test cases. Typically though it was 3 test cases which involved the 2 paths for true and false plus one random number like a negative number or a really big number. When asked where they came up with the third number they would just say something like, "I just usually try random numbers when I am testing input". So- Let me get this straight. You just throw numbers at anything? You don't try to think about it or how it gets the input? Or the fact that all I did was draw a diagram on a whiteboard which is probably a very leaky abstraction. But it's a good point about how even testers apply easy-way thinking to problems which really deserve genuine and rigorous thinking that shows intelligence and care for the problems they are solving.

2) Follow the Spec

Detail: As a junior tester I was given a set of instructions and mechanically followed them to completion. If I needed to print something on 10 different types of envelopes from 10 different printers and measure the printed indicia with a special ruler that measures the line spacing and placement of elements on the envelope, then that is what I did. I checked a box in a spreadsheet for each one and moved to the next thing like a machine. That is how I was initially taught to be a tester. Have a spreadsheet, run through it, check the boxes and log anything that was outside of

the expected result. Eventually I was allowed to write my own spreadsheets for areas of the product that needed to be tested. I read the requirements document, took lines from it and pasted them into a spreadsheet and prepended the word “Verify” in front of them. I might have added a bit of formatting to make it look nice and readable, to show which contexts the tests were to be run in, and perhaps my iteration or time estimates. Run through that, check the boxes, log the results- boom, I’m done. “Wow, Testing is kind of easy”, I thought. So I applied this mechanically over quite a few projects before I started reading about Software Testing from people like Cem Kaner, James Bach and Michael Bolton. Once I realized how much more there was to testing it became less easy, more challenging and more fun. But the point here is that I think I might have discovered a lot of issues or been more effective as a tester if I had been given more than just a hammer to do my job. There is a saying that if all you have is a hammer, everything looks like nails. I wound up using the same mental framework for each job and thus probably was not as effective as if I knew that some nails are different. Some nails require a pin hammer or some come in strips and go into a semi-automatic nail gun. Whoa, wait what is a nail gun? You mean I can fire these nails faster if the job allows for it? But wait- how to decide if the job allows for it? And if it allows for it are there different kinds?- These are the types of questions I was not being trained to ask and did not know to ask. But the easy way out rarely requires you to do much thinking and most definitely not Systems Thinking.

2.1.5 The cure can be worse than the disease

When I first witnessed a large layoff at a company I worked at, I was in a bit of shock. Admittedly I was relieved at the same time because I was not one of the people who got the axe, but still shocked and a bit unnerved. I was relatively new (had not even been there a full year), and so I had not heard much about any turmoil or upcoming layoff. I think people were still trying to let me be a happy new employee, full of cheer and spark. It ate at me for weeks after the event. It always pulled at me, begged me to understand why and to know what businesses looked at to make such a drastic decision. After a while I started to have hallway conversations or after work drinks where it would come up. I was prying. What I eventually found out was that we just had a bad few quarters and needed to adjust some figures to make our bottom line look better for investors. Cold. But that’s the world we live in. It’s illogical and driven by money for most people and especially in the business world. However- what unfolded after that event I don’t think was expected. Slowly but surely attrition kicked in. Folks started leaving for “greener grass”, and I mean really good people too. These were people that I respected and looked up to, wanted to learn from. I’m not sure what did it but even I made the decision to jump ship as they say. Eventually the company lost a lot of good people that I don’t think they anticipated losing and wound up having to spend a lot of money in slowing down projects and re-hiring and re-training people. This is a prime example of how the cure to fix their bottom line numbers disease wound up costing them even more money in the long run, and with no guarantee the sales and bookings would improve to avoid the same situation before they laid off people.

As a Tester we can also see this in our daily lives, and if we can see it we can try to avoid it. Admittedly I had trouble seeing how in my day-to-day tasks as a Tester how I was falling victim to this law in action. I was applying “cures” to situations where I needed Test Plans, Test Cases, Test Setup, Test Execution, Test Design, etc., where it led me to being in a worse place in the long run and thus did not solve the systemic issue I was dealing with. Then I re-read what I just wrote and it hit me like a ton of bricks. Every one of those things I do to try and improve the information I generate and the quality of that information so that in turn the quality of the product we create or the expectation of quality as defined by those that matter in our system match or align with the information we generate. Putting on our Systems Thinking cap, we start to ask questions like “What if one of those things I did led me to generate information that was not of high quality and led developers and managers to re-plan and re-work solutions which were done based on faulty information because of my actions?” I just cost us time and money and possibly my job. So what this tells me is that you really must understand the context of the situation, the relationship of your artifacts to the problem and how in a sustainable way you can produce your artifacts such that it is affordable to provide them and they are of high quality and reliability. Additionally you can look beyond your day-to-day tasks and look at your team to spot this law in play. For example if you are on a team and

that team decides to implement a new process in order to reduce distractions or improve efficiency- I urge you to ask how they measure the success of such a change. What does success look like for this change, and how can it not make things worse in the future? Without knowing this, you are setting yourself and your team up to be treating only symptoms and not the systemic issues that cause them. A Systems Thinker tries to solve the systemic problem, and asks why they are doing something before they ask how they will do it.

2.1.6 Faster is slower

Have you ever been in one of those startup agile environments where everyone is like, “Go fast!”, “Go faster!”, “WE GO LIGHTSPEED HERE!”? How did that work for you? How did it work for them? Personally I feel those are the places that burn people out and create a revolving door of engineers and thus wind up having to move frantic, not necessarily fast. Perhaps cultures like this should try adopting a new way with a slogan such as: “We move methodically!” or even “We move as fast as we must to do things intelligently!” would be sufficient for me. I guess it doesn’t roll off the tongue as easily though. So instead of growing at an optimal rate, they opt for trying to grow at the fastest possible growth rate as selfishly desired by their upper management and investors. This leads to churn and usually leads to creating perceived bottlenecks in other parts of the system who were not properly tuned or equipped to be part of such a fast moving system. An analogy to describe this law would be something like putting a turbo engine into a stock mini-van. What do you think will happen? Well- none of the other parts are meant to handle the excess of air pumped into the engine cylinders and now you blew up your mini-van trying to break land speed records getting your kids to soccer practice. You might have been going faster for a while, but now you are dead still on the side of the road.

Now as applied to the role of a Tester, this should be brutally familiar and obvious to any Tester who has been in the industry for even just a few years. When was the last time you rushed a job, and how did it turn out? Did you get away with it, or did it come back to bite you and the business you work for? My guess would be the latter and not the former. Testing is not something that should be rushed. If you rush it, you will make mistakes. If you are catching yourself rushing any part of your testing process, you should stop and ask why. What led you to mode of rushing? I don’t mean to conflate “rush” with “rapid”. I believe that rapid software testing [6] can be done effectively, carefully and meaningfully as demonstrated by the likes of James Bach and his RST/CDT practitioners. I’m talking about the careless kind of rushing where you are cutting corners in your testing or buying into snake oil tools that promise to save you time and maximize your coverage without you actually vetting out such claims. In another aspect of your Tester life, you may be witness to developers and managers doing the same thing. Quick fixes, code-smells, hacks, etc. You need to gather these people into a room when you see this stuff, and ask why it’s being done and if the quick fix scenario has been played out past the snapshot of time the decision was made in. This is what Systems Thinkers do.

2.1.7 Cause and effect are not always closely related in time and space

An “RCA” document is a good candidate to showcase this concept. For those who don’t know what “RCA” stands for, it means “Root Cause Analysis” and is usually carried out like your favorite murder mystery show. In the episode you have someone (Ops) who discovers a crime (production failure) has been committed, and then a Pathologist (some Dev Manager maybe) sets out to find the cause of death (production defect) and understand the cause in order to determine if it was natural or foul play. Now the difference here is maybe that in our software business the RCA takes either natural causes or foul play and attempts to suggest a mitigation solution, so that such an event does not occur again. Many root cause analysis documents are prime examples of how we fail at understanding that cause and effect are not always closely related in time and space. They are usually shallow analysis of the situation with convenient conclusions meant to shut people up or appease them which then shuts them up. What makes it even more interesting is that the person writing the RCA likely has some reason to not be too hard on themselves for self-preservation and self-interest reasons.

2.1.8 Small changes can produce big results -- but the areas of highest leverage are often the least obvious

You may be familiar with something called the Butterfly Effect. For those who are not- it is basically the idea that the flap of a butterfly's wings in a far off place can be the cause for a thunderstorm hundreds of miles away. You might also be familiar with how crimes are solved from watching your favorite crime tv show where the big break in the case is led to by some minute detail in where most common observers who examine the crime scene or analyze the evidence would have never made the connection. These examples help me explain this law by showing you that a small detail or small change can result in a big finding or big effect. Not only can they produce big results, they are not obvious to just any observer.

As a Tester you may be surprised by how much impact you can have on quality and scope of work or the process of work being done by looking for not just small but also less obvious avenues and ideas. From personal experience there was at least once in my Testing career where I literally stopped an entire team of Engineers from embarking on an expensive journey to build something by asking a simple question that none of them could answer and no one ever thought to ask which was, “Why are we building this?”. Stunned that their Tester asked them this, yet unable to give a substantial answer, the question was raised to management, and it was found that the project had started through improper channels and should not have been started not only for the lack of approval, but for the lack of its value add and reason to exist. So you see, you don’t always have to take obvious or traditional approaches to Testing problems. Sometimes the problem is not really a problem even if you ask the right questions. Other times it may be that you can buy or rent a tool as opposed to building it, recruit developers or the front-desk person to do some testing, try out interns for a summer, etc. Don’t ever limit yourself to obvious/traditional and look for outside-the-box ways to achieve big results.

2.1.9 You can have your cake and eat it too -- but not all at once

At first it might not be clear how to see the relevance of this statement, but there are some examples we can apply this to that are likely familiar to you as a Tester. Remember- This is about “either/or” and “black/white” thinking and realizing that most of the statements or decisions made in these situations are based on thinking about them in a moment of time versus what is possible through moments of time. Think about all of the instances in your career as a tester or in your current role where you have been faced with dilemmas like:

1) You cannot increase test coverage without increasing your schedule.
2) You cannot do “good” testing without documenting your testing.
3) You cannot build automation while upkeeping your manual testing.
4) You cannot increase your quality without increasing your cost.

These are just a few examples where I ask you to exchange “cannot” with “can” and work through the steps of how you can get there eventually. There are many more examples from different points of view of the different people on your team where, even as a Tester, you can help them get past this black and white, static thinking style. As a Tester who is equipped with an understanding of this principle, you can help educate others on your team as well as yourself to not sacrifice one solution for another.

2.1.10 Dividing an elephant in half does not produce two small elephants

Think about testing a web application that has an address book for example and how you approach it. It’s a hosted service with an interface and is accessible through the a browser on a user’s device. Do you create a test plan for the database, service API, and GUI separately? Do you then create a test plan for the device contexts separately? Do you also create a plan for the performance and security of each layer discretely? Perhaps you do and that’s OK. But if you failed to account for and plan for when they all live and harmonize together, then you have failed to think about the problem in a holistic way and will likely miss a large part of delivering a successful product that meets the expectations of the users. The successful software product is not each component separately. You should not have been only concerned with the individual parts of the system in your testing and planning without considering the full

end-to-end when all parts are working together. End users don't really care if you are using Postgres or MySQL. They care that they can enter in contacts to their address book in your application. The culmination of each layer and technology creates an address book web application and not one single part alone is the address book. Looking at this whole system is the process of synthesis. Synthesis is the opposite of analysis and is used in systems thinking to understand and discuss systems problems from the point of view of the bigger picture. The quality of the whole system is an address book.

2.1.11 There is no blame

When it comes to blame, we as testers tend to catch it for things like production defects and missed deadlines. Most organizations I have worked with have had at least one or many peers, managers, and executives, that have made comments about how their QA team doesn't know how to test effectively with such defects arising in production. They say or ask things like, "How could you miss such an obvious issue?" or "We have poor quality because of how incompetent our QA team is". Another classic is that teams are behind schedule because of how slow the QA team is. It's frustrating and concerning because it's toxic and counterproductive. Not to mention they are confusing testing software with assuring quality. What is worse is that they don't stop to think if they too had a hand in the creation of such a perceived quality issue or bottle neck. I'm fairly certain that no one walks into their Tester jobs and says to themselves that they are going to purposely miss important areas of testing. I have also never worked with any Tester who committed to intentionally working slower so that we would miss a deadline. What I have seen though are testers who don't speak up when they see too much work being approved and too much work with a fuzzy definition of done being started on. It's not that they should blame the project lead or engineering manager for not defining this though. We must not look towards individual people or groups for poor quality or unhappy surprises. As stated by Dr. Deming, "~94% of all issues belong to the system" (Out of the Crisis, page 315).

3 Conclusion

The most important thing about Systems Thinking for me has been the reification of my existing thoughts and approach to problems. That and the vocabulary it has provided me. Before I picked up my first book (The Art of Systems Thinking) and moved onto my second book (Intro To Systems Thinking), I had many thoughts about exactly what these books explained. They gave me examples to show my managers and peers. Watching videos of Russ Ackoff led me to Senge, which led me to Deming. My view of the world has become much more mature, and I am more aware of my place and potential impact on the organizations I work for. I am the butterfly, and there are effects when my wings flap. If I can better understand these effects, then I can better influence the system in the ways that I hope to see it become someday. This is the whole point for me in understanding Systems Thinking better. I want to do good and be aware of my actions. I want to work with people who also want to do good but are willing to do some rigorous thinking about what that entails, not just have good intentions. After years of floating from person to person, team to team, I realized that I could bring people together and appreciated my role as a Tester even more. Ever since I realized that I could influence multiple teams, I began to experiment with planting seeds in each of them to see what happened. I began to rally people from different teams that otherwise might have never spoken to each other. Once I saw that there was positive change coming from my ability to spread messages and join people together, I realized that I had some influence of the system. I just never realized it was an actual discipline until late in my career. I hope that testers who are early in their career will find this and begin sooner than I did. I also hope testers late in their career find this and have their thoughts reified as well.

References and Sources

[1] https://en.wikipedia.org/wiki/Systems_thinking

[2] https://en.wikipedia.org/wiki/System_dynamics

[3] https://en.wikipedia.org/wiki/The_Fifth_Discipline

[4] http://www.bretlsimmons.com/2010-03/behavior-grows-better-before-it-grows-worse/

[5] http://www.bretlsimmons.com/2010-04/the-cure-is-worse-than-the-disease/

[6] http://www.satisfice.com/info_rst.shtml - James Bach

Publication: The Art of Systems Thinking - Joseph O'Conner, 1997

Publication: Introduction To Systems Thinking - Gerald Weinberg, 1975

[7] Russ Ackoff - Youtube Video Ref. (https://www.youtube.com/watch?v=OqEeIG8aPPk)

Edward Deming - Youtube Video Ref. (https://www.youtube.com/watch?v=tsF-8u-V4j4)

http://www.4grantwriters.com/Peter_Senge_The_Fifth_Discipline_1_1_.pdf

Is The Role Of Test Manager At Crossroads?

Sreeram Gopalakrishnan

Sreeram.Gopalakrishnan@Cognizant.com

Abstract

When Agile grew in prominence in the last decade and half, one of the big questions it posed was about the relevance of the role of a test manager in Agile. And today, the question stands extended beyond Agile. With organizations looking to drive down Quality Assurance (QA) costs, complemented by advancements in technology and organizations' QA maturity, the role of the traditional test manager is surely at crossroads.

What does it mean for the test manager? It means that the test manager's role will evolve and spawn newer dimensions that hitherto had been outside the traditional scope or had remained secondary. The test manager will not just manage the test phase deliverables and the associated costs, quality, schedules and risks, but will play an expansive role in the project's or program's overall scheme of things.

There are at least three distinct newer dimensions of a test manager's role that are becoming apparent. The first is that of a technology evangelist, wherein the test manager drives adoption of tools, technologies and innovations. Instead of just managing the adoption process, the test manager will have hands-on involvement in all phases from conceptualization, through to design, pilot and institutionalization.

The second is where the test manager plays an integration specialist, integrating and influencing the multiple disparate Information Technology (IT) functions that matter for total quality. The areas, to name a few, include infrastructure production readiness, code release management, application production performance, software development life cycle (SDLC) governance, quality management office (QMO), and even merger and acquisition (M&A) synergism.

The third is that of a value manager, that ensures that all quality initiatives, operational or strategic, deliver the value for their investment. This will require the test manager to adopt proven value frameworks such as the balanced scorecard (BSC) or earned value management (EVM) and institutionalize them within the project or program.

The paper will elaborate the role dimensions described above, and the best practices and frameworks test managers can leverage to successfully transition into their redefined role expectations.

Biography

Sreeram is as an Associate Director in Cognizant's Quality Engineering & Assurance practice in Canada. His specialty is large-scale and complex QA engagements, wherein he brings extensive experience in managing interdependencies, risks, integration and people. Sreeram has led significant QA transformations for his customers in the areas of virtualization, regression, automation and performance testing. An avid QA practitioner, Sreeram has spoken at marquee conferences such as STARCANADA, PNSQC and IQNITE. He holds a Master's degree in Business Administration, and is PMP-, CMST- and ISTQB- certified.

1 Introduction: The Changing World Of Quality Assurance

A lot has changed in the world of software testing over the past two decades. Testing has evolved from being an adjunct phase to development in the 1990s, to being recognized as a critical SDLC phase in the early 2000s, going on to become a strategic enabler for Information Technology (IT) in the last 10 years. In the journey, testing itself has morphed into QA. Over the next decade or so, QA is poised to evolve into a diffused engineering phase that not just fulfills the cost and speed priorities of IT and businesses, but also ensures customer experience, which in turn has become the ultimate yardstick for quality.

The evolution of testing is summarized in Table-1 below.

Table-1: Evolution of Testing

Time Period	Key Characteristics of Testing
1990s	**Adjunct to development** - Afterthought - Left to the less skilled, lower paid - Few tools and little automation - No management visibility / involvement
Early 2000s	**Important phase in SDLC** - Key to good quality software - Planned and monitored as a phase - Explosion of tools, but no tool strategy - Management involvement rudimentary
2005-2015	**Strategic to IT organization** - Own budget, clear leadership and ownership - High visibility to executive management - Integrated test activities, tool selection
2015+	**Diffused engineering phase driving business assurance** - Focus on customer experience - Automation of all wait-times - End-to-end integrated tool sets and delivery platforms enabling continuous delivery

Technology, business and people factors have all played a part in this transformation, as evident from the specific examples quoted below:

- In keeping with the demands of today's businesses, IT organizations demand faster, cheaper, and higher quality testing
- The complexities of the IT landscape have increased tremendously with social, cloud and mobile technologies. It means QA will have to test devices, infrastructure, and end-user experience, and not merely software applications.
- Speed to market is a prime concern for majority of the businesses. IT needs to have capabilities to deploy changes in production, even several times a day. The software delivery has evolved in tandem from Waterfall to Iterative to those that support continuous delivery.
- Focus on functionality has given way to customer experience. With social media and mobile devices penetrating all walks of life, end-user experience has become the ultimate measure of quality.

- Focus on defect detection has shifted to defect prevention and is progressing towards defect prediction through the use of analytics. Testing techniques are changing from black-box to more white-box.

2 What Do The Changes Mean To The Test Manager

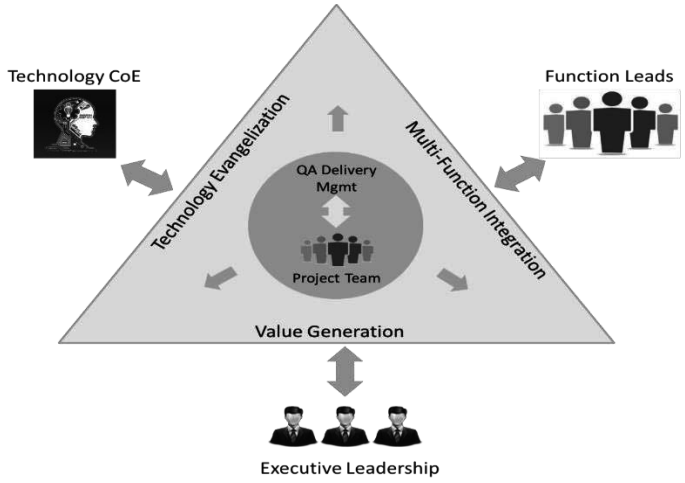
Test management is the practice of organizing and controlling the process and artifacts required for the testing effort. Test management allows teams to plan, develop, execute, and assess all testing activities within the overall software development effort. This includes coordinating efforts of all those involved in the testing effort, tracking dependencies and relationships among test assets and, most importantly, defining, measuring, and tracking quality goals.

Historically, the role of a test manager was centered on managing the testing phase within the SDLC. In this role, the test manager planned and managed the budget, schedules, resources, risks and communication associated with performing quality verification activities for a software project. The role of test manager vis-à-vis that of the project or program manager was structured in a way that confined the test manager mostly to the test phase. Integration, technology management and even quality governance were viewed as functions secondary or far removed from the test manager's role.

With the advent of Agile methodologies, the role of the traditional test manager came under scrutiny. Agile was founded on team traits like self-organizing, role blurring and skill diversification. Development and testing were tightly integrated within the Agile sprints. Hence, managing testing as a separate phase ceased to be a need. Instead, test management got diffused into the project team. On the other hand, the test manager's role took on newer responsibilities such as educating and coaching testers to instill an Agile culture based on technology and tool skills.

The changes to the test manager role did not remain limited to Agile. The technology, business and people changes that had been sweeping IT brought in newer challenges for the test manager to handle. Traditional test management no longer proved sufficient to meet the demands of the changing IT landscape of speed to market, ultimate end-user experience, all this while driving costs down continuously.

Three significant areas of change are as illustrated in Picture-1 and described in detail in the sections that follow.



The Expanding Role of Test Manager

Picture-1

2.1 Technology Evangelization

Technological complexities combined with speed to market needs have limited the efficiencies that can be achieved through mere process improvements. Meeting business objectives requires the test manager to possess superior technical skills to be able to comprehend the complexities and heterogeneity of the landscape, and excellent knowledge of the market-leading tools and solutions. More often than not, the solution for technological complexities and speed to market needs lies in the use of integrated tool sets and intelligent test platforms. The test manager should be technically inclined, lest the tool strategy remains non-operationalized.

2.2 Multi-Function Integration

From just managing test deliverables and test phases, to influencing and integrating several factors outside the QA realm to meet the cost, time and quality objectives of the project, it signals a significant shift in the scope and approach of test management. While the scope of testing itself has expanded beyond functional testing to other areas such as performance, security, infrastructure, usability and so on, it is quite evident that failures and risks are significantly higher at the interfaces of these test areas. Managing the disparate components in an integrated and seamless fashion is vital to the success of the program.

2.3 Value Management

Notwithstanding the institutionalization of the above two role changes, the test manager's role wouldn't be complete without incorporating value management. Continuous and proactive planning, tracking and communication of value to the key stakeholders in IT, business and end-user communities, collectively referred to as value management, is a vital role of the test manager in today's context.

3 Technology Evangelization

Unless the test manager is technically inclined, the type of solutions tried out to solve problems will mostly be limited to traditional document- and process-based methods. In reality, the complexities in the IT landscape demand technology-based solutions.

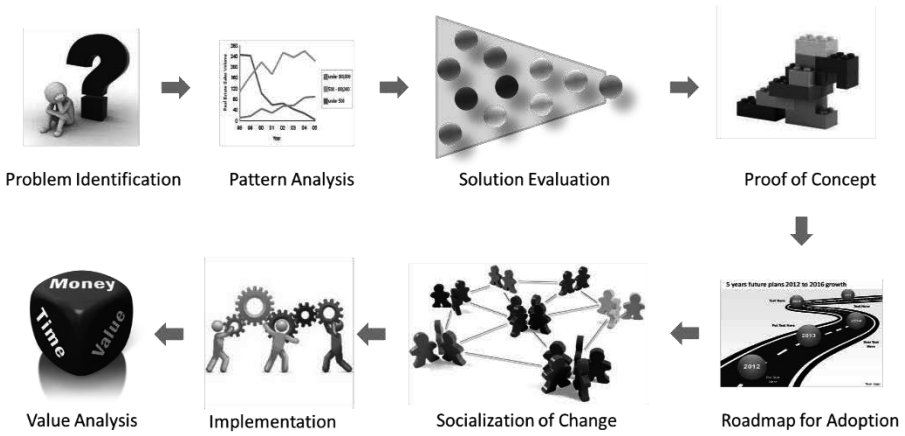
A case in point is the adoption of white-box testing. To engineer quality across the SDLC, testing early or parallel to development is a major strategy, referred to as shift-left. It requires the test team to be well versed with white-box techniques such as static code reviews, dynamic code profiling and component-level testing, where the test items are typically developer objects such as stored procedures, Java classes, web services, etc. This marks a significant deviation from the current black-box (requirement-based) test approaches, which require a system to be available in its entirety before QA can test. White-box testing approaches come with their own tool needs and different sets of test management requirements and challenges. Planning the test activities to be ingrained with development requires the test manager to be well versed with the development methodologies for various type of technologies.

Another example is the advocacy of integrated tool sets and platforms. As much as the benefits they bring, the risk involved in tool decisions is high too. The test manager has to be technically adept and well-aware of the market leading tools, in order to make the right tool choices.

From the above examples, it is quite evident that the days of document- and process-based test management are past. The test manager will have to play the role of a technology evangelist. As the technology adoption mostly occurs when a project or a program is in flight, the test manager's hands-on involvement is vital to its success.

As a technology evangelist, the test manager will have the responsibility for the end-to-end food chain of technology adoption from conceptualization to implementation. It will require an organizational framework to be defined that integrates the processes listed below, and further illustrated in Picture-2.

- Identification of pain areas and recognizing patterns of issues
- Platform to solicit solutions
- Evaluation of alternatives and shortlisting of solutions/tools
- Conducting proof of concept (PoC) / proof of value (PoV)
- Defining roadmap for wider adoption
- Socialization and change management
- Metrics and key performance indicators (KPIs) to track adoption and value



The Technology Evangelization Food Chain

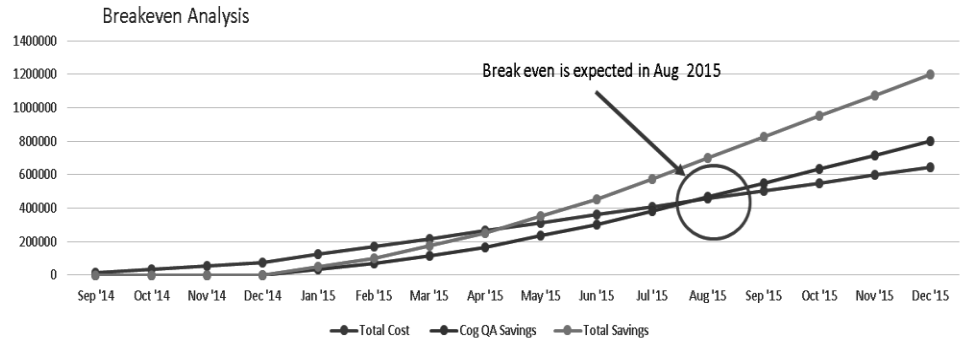
Picture-2

3.1 A Case Study For Technology Evangelization

Table-2 below documents a case study of a technology adoption, that of Service Virtualization (SV), for an enterprise customer that illustrates the use of the evangelization framework.

Table-2: Technology Evangelization Framework

Framework Component	Details
Problem Identification	• Test environment conflicts resulting from sharing 30 out of 95 applications across multiple environment landscapes. • Wait times of 2-6 weeks for projects to get test environments contributing to project delays • Higher environment maintenance with eight integrated test landscapes • Poor environment stability with approximately 16% of all defects being environment defects • High failure rate of end-to-end automated test flows due to environment issues. Late defect detection in SDLC with over 80% of all defects being caught in testing phase or later.

Pattern Analysis	Environment issues formed the single biggest category of issues impacting the faster, cheaper and better objectives of QA for three years in succession (2012, 2013 and 2014).
Solution Evaluation	Two major solutions were considered: • Service virtualization • Cloud environment provisioning (on-demand) Service virtualization addressed all of the problems listed above. Cloud environment provisioning addressed the first three, whereas the last three required different solutions such as environment monitoring and white-box testing. Further, cost and security concerns did not favor the cloud solution. Service virtualization was selected as the solution to move forward with. Two service virtualization tools were evaluated. Based on protocol support, license cost, and skilled resource availability one tool was selected over the other. A business case was created highlighting the costs and benefits. Return on investment (ROI) analysis was done as a part of the business case. 
Proof of Concept	With trial licenses and support from the tool vendor, a proof of concept was performed. A few services and interfaces were virtualized, and the objectives of reduced environment wait-times, better environment stability and early defect detection were verified.
Roadmap for Adoption	• Q4 2014 → Tool purchase and installation, analysis and scoping, tool training, team ramp up • Q1 2015 → Quick wins. 50 services based on Q1, Q2 project release priorities. Services to be used only by System Integration Test (SIT) and Regression Test teams. • Q2 2015 → 50 services based on Q2, Q3 project release priorities. Services to be socialized with Development teams for use in Unit Testing and System

<table>
<tr><td></td><td>Testing.
• Q3, Q4 2015 → 75 services based on SIT, Regression Test, Unit Test and System Test priorities. Service Virtualization leverage in Performance Testing and Automation Integration planned.</td></tr>
<tr><td>Socialization and Change Management</td><td>Socialization of the concept of service virtualization across the organization was planned through multiple forums such as:

• Trade shows
• Town halls
• Lunch and learn sessions

This was to be followed by more formal one-to-one interaction sessions with each Development team, to demonstrate the services already virtualized, and to plan the future scope in their respective areas.</td></tr>
<tr><td>Implementation</td><td>As detailed out in the roadmap section.</td></tr>
<tr><td>Value Analysis</td><td>Value Analysis starts with defining an appropriate set of metrics that can measure the objectives defined. The following metrics were defined for this initiative.

<table>
<tr><th>Measure</th><th>Metric/KPI</th><th>Formula</th><th>Target</th></tr>
<tr><td rowspan="3">SV Utilization</td><td>SV Transaction Count</td><td>Transaction count of each virtual service</td><td></td></tr>
<tr><td>% Utilization of services</td><td># of virtual services used/Total # of available virtual services - Per Period
# of virtual services used/Total # of available virtual services - Per Quarter</td><td>30% - Period basis
70% - Quarterly basis</td></tr>
<tr><td>% Projects leveraging SV</td><td># of projects used SV/# of projects delivered in a Quarter</td><td>Q2: 5%
Q3: 10%
Q4: 25%</td></tr>
<tr><td>Early Defect Detection</td><td>Early Defect Detection</td><td>FUT DRE of the projects - FUT DRE 2014 baseline</td><td>Q1 and Q2 2015 - FUT DRE Baseline
Q3 and Q4 2015 - Start the Comparison and baseline the Target</td></tr>
<tr><td>SV Delivery Productivity</td><td>SV Delivery Productivity</td><td>Count of virtual services delivered person Period</td><td>2.4 Services per Person Period
(Q1: 50 Services
Q2: 50 Services
Q3: 50 Services
Q1: 25 Services)</td></tr>
<tr><td>Applications Provisioned through SV</td><td>Applications Provisioned through SV</td><td>Count of applications provisioned through SV</td><td></td></tr>
<tr><td>Cost Savings</td><td>Cost Savings</td><td>Average 3% of QA Cost of applicable projects</td><td>$800K for 2015</td></tr>
</table>

The metrics are tracked and reported periodically at operational and strategic levels. (Strategic level reporting is described in detail in the Value Management section).</td></tr>
</table>

Measure	Metric/KPI	Current Data	Target
SV Utilization	SV Transaction Count	**Cumulative: 3929 Last 30 days: 1989**	
	% Utilization of services	**Period 4: 29% (12 VS out of total 41)** **Quarter 1: 51% (21 VS out of total 41)**	30% - Period basis 70% - Quarterly basis
	% Projects leveraging SV	**Q1 (Jan – Mar): 10% (3 out of 27)** **Q2 (Apr): 12% (3 out of 26)**	Q2: 5% Q3: 10% Q4: 25%
Early Defect Detection	Early Defect Detection	Q1 and Q2 2015 – FUT DRE Baseline Q3 and Q4 – Start the comparison	Q1 and Q2 2015 - FUT DRE Baseline Q3 and Q4 2015 - Start the Comparison and baseline the Target
SV Delivery Productivity	SV Delivery Productivity	**Q1 2015: 41 – 82%** **Q2 2015: 17/50 as of Apr**	2.4 Services per Person Period (Q1: 50 Services Q2: 50 Services Q3: 50 Services Q1: 25 Services)
Applications Provisioned through SV	Applications Provisioned through SV	**13 Applications provisioned till April** **$ 289K cost avoidance**	
Cost Savings	Cost Savings	**Q1 2015: $ 225K**	$800K for 2015

4 Multi-Function Integration

Taking quality beyond the testing phase and infusing it across the SDLC requires a test manager to not just interact, but integrate and influence several vital IT functions outside QA.

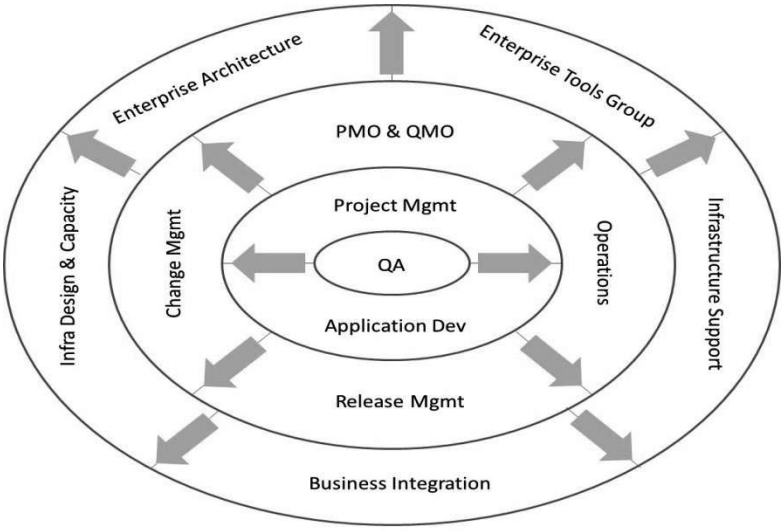
From years of experience in test management, it should be quite evident that the maximum risk for quality assurance is at the points of integration of multiple disciplines. A test manager cannot afford this vital area to become a blind spot. As an organization matures in its QA, failures in production are not as much due to lack of process and rigor within a specific discipline as due to the lack of adequate handshake and communication across the disciplines.

Quoted below are a few examples to drive home the point.

A last minute code change is approved for deployment just based on a quick functional test, whereas the performance and regression impacts are overlooked due to time and cost constraints. A database or operating system patch is moved into production through the business-as-usual (BAU) channel, without adequate assessment of the impacts to application functionality or performance. The infrastructure design and build for a new application is approved for production, without considering the lower environment build requirements.

4.1 The Key Integration Areas

The major areas of integrations are shown in Picture-3 below, and further detailed out in Table-3, along with some of the typical interactions that occur between QA and this group. The expanded role of the test manager will change the characteristic of the relationship from "interact" to "integrate and influence".



Picture-3

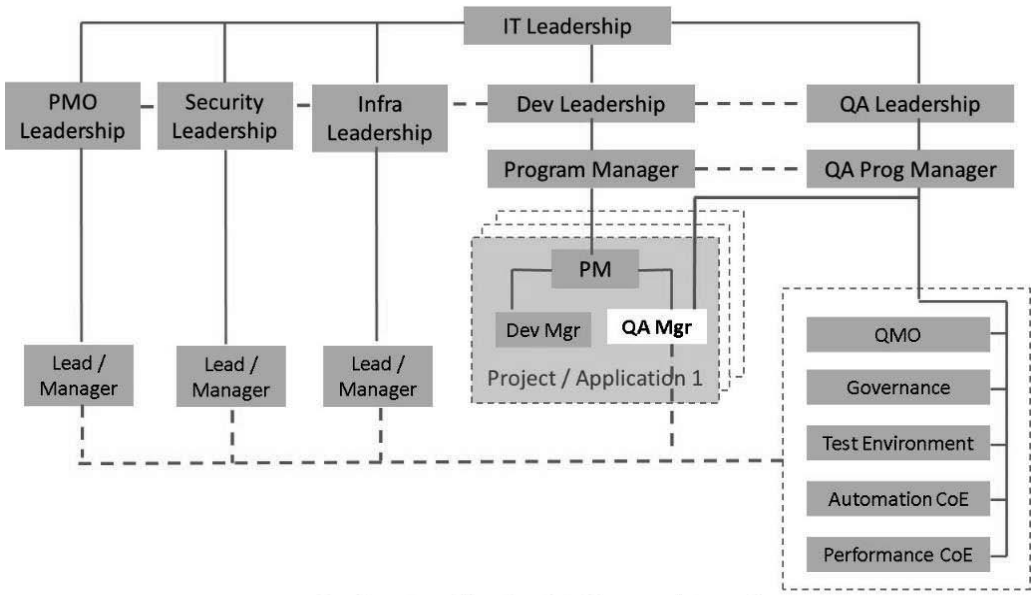
Table-3: Areas of Integration

Area of Integration	Key Interactions
Project / Program Management	Risk and issue management, scope control, status reporting, dependency tracking and management.
PMO and QMO	Continuous process improvements, metrics and KPI definitions, baselining and tracking, defining and maintaining estimation guidelines,
Application Development	Early involvement in requirements and design, white-box or component level test enablement, tracking and sharing Development vendor performance KPI.
Release Management	Define release calendar, control release scope creep, and govern code deployment to production and lower environments.
Enterprise Architecture	Review and maintain solution blueprints, non-functional requirement review and maintenance, access to application and server inventory in production and lower environments.
Enterprise Tools Group	Introduce new tool ideas, perform PoC or PoV, garner support for tool decisions, and propagate tool adoption.
Change Control / Management	Define / update Change Request (CR) process, enforce CR governance.
Production Support / Operations	Production ticket analytics, production incident root cause analysis, pro-active production monitoring of applications.
Infrastructure Design and Capacity Planning	Lower environment build and maintenance, right-scaling of performance test environments, planning production capacity based on performance baselines.
Infrastructure Support	Establish or leverage proactive production monitoring of servers.
Business Integration	Drive early business involvement in QA test phases, to involve business in data strategy, production incident root cause/pattern analysis.

4.2 Right Organizational Structure As An Integration Enabler

One of the key factors aiding the integration and influencing aspects is the organizational structure. The organization structure should be designed for efficiency, effectiveness and integration. While the test manager should be sufficiently empowered to influence and integrate the functions within and outside QA, the organizational structure should complement this by clearly defining the escalation routes to support the test manager.

Picture-4 depicts an example where the functions within and outside QA are organized for optimal efficiency, effectiveness and integration.



Picture-4

In the organizational structure shown above, the QA Manager has counterparts from each tower (PMO, Security, Infrastructure, Development and QA Shared Services) that he or she can work with to integrate and influence the functions to achieve the program objectives.

There is a clearly defined escalation mechanism, from the QA Manager to the QA Program Manager, and further up to the QA Leadership, to handle issues that call for upper management involvement.

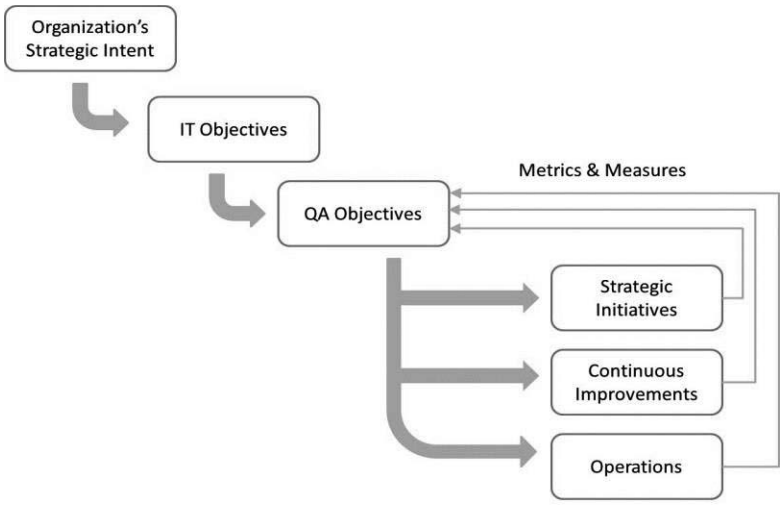
5 Value Management

While the first two extended roles, that of technology evangelization and multi-function integration, focus on quality and speed, the third dimension, that of value management, focuses on the balance between quality, speed and cost, assuring maximum value and alignment with the organization's objectives.

In this role, the test manager has to manage the value realization from QA's operational and strategic initiatives. In today's world, where the cost of quality is under scrutiny, the importance of this role cannot be overemphasized.

5.1 Balanced Scorecard Technique

Value management can be performed through the use of the balanced scorecard technique, whereby the organization's strategic objectives are translated into tactical and operational objectives. The technological initiatives and operations must achieve or align with strategic objectives, fully or partially. The strategic alignment is critical to ensure executive support for the initiatives and operations. This hierarchical organization is demonstrated in Picture-5.



Balance Scorecard Technique of Value Alignment

Picture-5

5.2 A Case Study For Value Management

Tables 4 and 5 below form a case study that illustrates the use of the balanced scorecard technique as applied to an enterprise customer's QA organization.

Table-4: Balanced Scorecard Aligning QA Objectives to Org Strategy

Organization's Strategic Intent	IT Objectives	QA Objectives (SMART)
Build culture, talent and employee engagement	New vision, structure, operating model and governance Improve collaboration and skill sets	Increase customer satisfaction (CSAT) by 20% over the course of the year through improved service levels, better business and IT alignment of QA
Be the best in food business	Deliver key business enabling solutions	Extend quality across the SDLC to enable early (pre-test) defect detection for 90% of the projects and achieve pre-User Acceptance Test (UAT) defect removal efficiency (DRE) of 92% for all projects
Gain efficiencies	Cost effective technology delivery and support	Reduce QA cost to 20% of overall project cost, from the current 25%, through multiple efficiency levers (automation, regression, onsite-offshore ratio) and innovations.
Focus on growth	Deliver enhanced colleague and customer experience	Reduce major production incidents in the area of customer facing applications such as eCommerce, Loyalty and Supply Chain.
Focus on operational	Resilient technology, stable and	Contribute to faster IT delivery by

excellence	secure	achieving 10% reduction on the identified QA dependencies/schedules such as data refreshes, environment provisioning and web performance testing.

Once the QA objectives are aligned with the organizational and IT objectives, the next step is to align the strategic initiatives, continuous process improvements and operations to the QA objectives as demonstrated below. Each initiative will have associated metrics or KPI's to track its progress against the target values defined.

Table-5: Balanced Scorecard Mapping Metrics/KPI's To QA Objectives

QA Objectives	Initiatives	Metrics / KPI's	Target Value
Increase Customer Satisfaction by 20% over the course of the year through improved service levels, better business and IT alignment of QA	Track customer escalations to closure and track the resulting QA improvement actions to completion	Customer escalations tracked to closure with an action/improvement plan	95% of all escalations received
	Conduct Periodic Operations reviews with Key Internal Customers and achieve improvement in CSAT scores	CSAT score	20% Increase on Q1 Baseline to be achieved by Q4
Extend quality across the SDLC to enable early (pre-Test) defect detection for 90% of the projects and achieve pre-UAT DRE of 92% for all projects	Integrate QA into a Business Requirement Document (BRD) and Functional Design (FD) review process for 90% of all 2015 projects	Projects where QA reviewed and signed off on BRDs and FDs	90% of all 2015 projects
	Drive increased adherence to project change request processes	Number of (applicable) projects where the defined CR process was followed	90% of all 2015 projects
	Achieve pre-UAT DRE of 92%	Pre-UAT DRE	92% for all 2015 projects
Reduce QA cost to 20% of overall project cost, from the current 24%, through multiple efficiency levers (automation, regression, onsite-offshore ratio) and innovations.	Move to 20:80 onshore/offshore mix by end of the year	On : Off ratio by portfolio	Q2: 25-75 Q3: 22-78 Q4: 20-80
	Increase Test Execution productivity by 5% for projects	Test Execution Productivity for projects	43 Test Case Points (TCP) by Q4
Reduce major production incidents in the area of customer facing applications such as eCommerce, Loyalty and Stores	Establish Regression Bus for all major and minor releases	Regression Bus Implementation progress	Q1: Minor releases Q2: Major releases
	Move from Quality Assurance to Engineering by	Code profiling / code	Q3

	implementing code profiling & coverage Tools	coverage tool PoC	
Contribute to faster IT delivery by achieving 10% reduction on the identified QA dependencies/schedules such as data refreshes, environment provisioning and web performance testing.	Shift left automation to SIT on at least 10 projects	Automation leveraged in SIT	2 projects: Q2 4 projects: Q3 4 projects: Q4
	Increase regression automation coverage	% of regression gold suite automated	75% by Q4

6 Conclusion

As described in the above sections of this paper, the role of the test manager is unmistakably at a crossroad. Just as testing has evolved over time, the role of test manager too is going through interesting and very essential transformations. Three distinct dimensions of the expanding role are that of technology evangelization, multi-function integration and value management.

The transformations are very essential for quality assurance to keep up with the value demands on QA from technology and business change agents.

The paper discusses a number of best practices and frameworks, mostly based on the author's successful prior experience with them. However, they are by not meant to be prescriptive. Each program is different, and there can be several alternate solutions possible for the challenges.

The best practices and frameworks discussed here would qualify as excellent materials for detailed analysis and study, and can thus contribute to the expansion of the body of knowledge in the area of QA/test management.

7 List Of Acronyms

Table-6: List of Acronyms

Acronym	Expansion
BRD	Business Requirements Document
BSC	Balanced Scorecard
BAU	Business As Usual
CR	Change Request
CSAT	Customer Satisfaction
DRE	Defect Removal Efficiency
EVM	Earned Value Management
FD	Functional Design
IT	Information Technology
KPI	Key Performance Indicator
M&A	Merger & Acquisition
PM	Project Manager
PMO	Project Management Office
PoC	Proof Of Concept
PoV	Proof Of Value
QA	Quality Assurance
QMO	Quality Management Office
ROI	Return On Investment
SDLC	Software Development Life Cycle
SIT	System Integration Test
SMART	Specific, Measurable, Achievable, Realistic and Timely
SV	Service Virtualization
TCP	Test Case Point
UAT	User Acceptance Test

References

Books:

Black, Rex, and Dorothy, Graham. 2012. *Foundations of Software Testing: ISTQB Certification*.

Black, Rex. 2009. *Managing the Testing Process*.

Web Sites:

Linders, Ben. 2015. “How Agile Has Changed Test Management”. Entry posted May 11, 2015. http://www.infoq.com/articles/agile-changed-test-management

Davis, Chip. 2006. “Test Management Best Practices,” Entry posted Nov 7, 2006, http://www.ibm.com/developerworks/rational/library/06/1107_davis.

Rothman, Johanna. 2010. “The Role of Test Manager in an Agile Organization”. Entry posted Feb 9, 2010, http://www.stickyminds.com/article/role-test-manager-agile-organization

How manual testers can break into automation without programming skills

Jim Trentadue

Enterprise Account Manager at Ranorex: jtrentadue@ranorex.com

Abstract

Test automation is seen as the way to make testers more efficient and productive for doing their job. It's also known that you can expand your test coverage greatly by automating nearly all permutations of a given scenario. For large project teams and large releases, automating your testing is the only way project schedules can be met.

Test automation solutions come with a misperception though. A number of solutions require you to have a programming and development background or skillset. But this is not a skillset most manual testers possess nor want to. Often there is a divide between the manual testers and automated testers as these are now two different jobs.

So what kind of bridge can be built across this gap of the two different roles? How can the automated testers get more useful information from the manual testers? How can the manual testers be more productive with what they can contribute to the automation effort? It starts with understanding the automation structure and framework.

Whether they are writing the actual test automation test case or not, manual testers can be productive immediately by understanding a few key points: Where was the Test Automation industry before and where is it headed? What are the available frameworks in the industry? What are the key categories that a manual tester can be effective immediately with test automation?

This paper will focus on three main categories of work for test automation. These are Preparation, Execution, and Analysis. Within each of these three categories, there are three key points in each that cover the full category, which will be explored in depth.

Biography

Jim Trentadue has over 16 years of experience as a coordinator/manager in the software testing field. He has filled various roles in testing over his career, focusing on test execution, automation, management, environment management, standards deployment, and test tool implementation. In the area of offshore testing, Jim has worked with multiple large firms on developing and coordinating cohesive relationships. Jim has presented at numerous industry conferences including the SQE STAR East, STAR West and Better Software Conference East conferences, Software Test Professional's STP Conference, IBM Rational Software Development Conference, IIST's SQTM conference, QAI's Quest conference and QAI chapter meetings. Jim has been invited as a guest speaker at the University of South Florida's software testing class, mentoring students on the testing industry and trends for establishing future job searches and continued training.

1. Introduction

This paper has been developed with the premise that there may be elements of test automation existing in your environment. Test automation largely has been considered to be a development activity without a clearly understanding of where a manual tester can contribute. This manual tester may have technical skills, but not a development or programming skillset. But as you'll see within this paper, there are many ways where a manual tester can add value to the test automation process. For example, there are challenges from the test automation engineer's perspective to get better requirements or test documents from the functional manual tester and there are challenges from the functional manual tester to have the automated test case meet the same criteria as a manual test case.

Some other key assumptions is that the majority of test analysts are still performing a manual testing role. But with the points below, they would participate in the automation effort without being fully dedicated to the test automation team.

Based on over 16 years of Testing/QA experience in the industry and also working for a test automation vendor, this paper is based off of my recommendations from industry experience. If a manual tester was starting an automation initiative, many of the same principles below would serve as requirements for selecting a solution.

2. Categories for test automation

Below you will find an outline for the three different categories for manual testers to get into test automation: Preparation, Execution and Analysis. Within each of these categories have specific sections where manual testers can provide immediate contributions.

1. Preparation
   1.1. Planning test automation activities into your testing process
   1.2. Creating data sheets or SQL statements for data driving your tests
   1.3. Editing existing manual test cases for automation
2. Execution
   2.1. Learning how objects are recognized within test automation
   2.2. Working with the object repository and organizing this
   2.3. Building test automation actions from the ground up
3. Analysis
   3.1. Adding error handling in your testing flow
   3.2. Knowing how to step into test cases and debug
   3.3. Reading report results and interpreting this

2.1 Preparation:

2.1.1 Planning test automation activities into your test process

QA organizations have defined testing processes, without the incorporation of test automation. Methodologies stem from an Agile, Waterfall or other SDLC methodologies. Largely, this serves the need for the manual testers. In most cases, there is not an integration of the test automation activities into the manual process. To illustrate how these activities could be mapped, a simple V-Model is used, regardless of an Agile or Waterfall SDLC methodology.

2.1.1.1 Project Initiation ⟹ Test Strategy

It's recommended that QA release leads own the inclusion of test automation activities within each of their releases. It's unrealistic to think that test automation activities can occur for longer periods of time without impact to project schedules. If you list out the test automation activities for creating or executing automation test cases within the Release / Master Test Plan or Test Strategy document, your Project Manager or Product Owner will have visibility on planned automation within the given release.

2.1.1.2 Analysis ⟹ Test Scenarios

As manual test analysts are drafting their test scenarios or test objectives during the analysis phase of their project, they should divide the scenarios between manual and automated. Everyone will know their objectives and can plan for good coverage on each.

2.1.1.3 Design ⟹ Test Cases

Typically, manual test cases are not written for certain error-handling conditions that automation requires. Considering this during test case development will prepare the manual test cases for automation.

2.1.1.4 Develop ⟹ Test Scripts

Record your automation test cases with the same principles as you would write your manual test cases. Just ensure that you are going slow enough to recognize all objects you are interacting with.

2.1.1.5 Testing ⟹ Test Results

Unlike manual test case execution, automated test execution may require replaying the test several times and making several tweaks and modifications to make test run as expected.

2.1.1.6 Deploy ⟹ Test Summary

Similar to the Test Scenarios document, list out what was executed with automated test cases vs. manual test cases. This information is listed in the Test Summary document prior to deployment.

2.1.2 Creating data sheets or SQL statement for data-driving your automation

In many cases, the test automation associate is not the subject matter expert (SME) in the application under test (AUT). The functional test automation analyst will know the UI, as well as the underlying database tables and fields. It would be in the best interest from all team members on the testing team to utilize the knowledge the functional test analyst has for using dynamic data within the automated tests.

It's also important for the functional test analyst to know a typical process that a test automation analyst would follow. Knowing this helps provide a common understanding of how the test automation test case is formed and when the dynamic data element is needed. This is a standard three-step process:

2.1.2.1 Create a variable on what you want to data-drive

There are values where the automation test case will either select or input data to the respective field. This data is static or hard-coded unless driven from a source. The test automation associate will create a variable instead of using the original value so static data can be substituted out for dynamic data ongoing.

2.1.2.2 Link to your data source where the data resides

Test data should reside in a central, secure location and the test automation analysts can link data from various formats: Relational Databases (RDBMS), Excel spreadsheets or CSV files. The functional analyst should select what's easiest for them, but try to use the same data source for all of their data-driven tests.

This will reside on different data sheets or tables, but the connector the test automation associate uses will be the same.

2.1.2.3 Map your data source columns to the defined variables

Once the variables have been defined for the fields and the data source, connecting the data source fields to the variable fields should be straight-forward and essential. This must be done to have the correct mapping on data elements.

2.1.3 Editing existing manual test cases for automation

Most QA organizations have hundreds, if not thousands of manual test cases that they would like to be automated. But the chances are that many of these have not been written in a format suitable for automation. The functional test analyst should not worry too much about the overall test case as there are just two areas that should be focused on for editing: Test Steps and Test Data, but not Test Results.

2.1.3.1 Test Steps

The test steps are the items that map directly to either the recorded steps, or to the created steps from the test automation analyst usually through code. Every step must be accounted for; no implication that the automation will know what step to do next. Also, the wording used in a common manual test case is not needed for the automation test cases. If a keyword-driven automation framework is used, limiting the wording is preferred. For example, a manual test case may have a step phrased like this:

- *Click on the City field and input the following state: 'Florida'*

The automation test case could have the same functional step, phrased as the following:

- *Action – Event – Object or Repository Item*
- **Mouse – Click – CityEditBox**
- **KeySequence (Input) - Florida**

2.1.3.2 Test Data

As mentioned in the section above, data-driving the test is a best practice. Just as hard-coding data within development code is to be avoided unless absolutely mandatory, hard-coding data in tests should follow the same practice. Of course there are times that having static data elements may be beneficial to the test, but overall, this would be better to have an external data source for maintenance. This way, the responsibility is shared between team members. The test automation analyst is responsible for maintaining the harness and the functional test analyst is responsible for maintaining the data.

2.1.3.3 Test Results

The reason why this is not listed as a step required for edits and maintenance is that you can rely on the test automation solution to provide all of the test results automatically for you from the execution run. Not only will this be a more reliable source of information, but also will provide much more. It's a best practice to separate your results from your planning information. You will usually have red-flagged or a failed test result if a test step or validation does not match, green-flagged if everything proceeded as expected.

2.2 Execution:

2.2.1 Learning how objects are recognized in test automation

This is a key area of understanding for a functional test analyst, how objects and controls are identified by the test automation associate or the test automation solution. There may be a new term learned called the '*Accessibility*' layer. What this is referring to are those attributes and properties that are made

accessible from the development team. Some examples of this are Object Name, Object Index, Inner Text, Caption, Label, etc. The test automation solution latches on to one of these properties to be able to accurately identify this test step with repeated success.

Again, this is such a key attribute to be able to know how the test automation analyst is working or how the test automation solution is working to replay your tests with success each and every time.

2.2.2 Working with the object repository and organizing this

You may have an application that has a number of forms, windows or screens. Then on each or some of these forms, there may be 30+ controls or objects contained within here. When the automation solution is interacting with each of these, the object repository itself could get very large. The object repository is the collection of objects that the test automation solution has either clicked, pressed, inputted data or done any other event on that specific object.

With respect to the section above, there are attributes that the test automation solution has been in contact with and that information is stored in the repository. This information is centralized so you would not have to record or write another test to be able to work with that object. What you may see though is a series of controls that may or may not have good object names or other properties for automation. For example, if you have a series of radio buttons for each state in the United States, the name of the object might be RadioButton1 through RadioButton50.

What's recommended is that following the initial recording or coding for that object, make sure to edit the name of the object to make it more meaningful if it is not already. Most solutions have an object spy tool that can aid with this. You don't want to have to go back to the object repository to figure out object names for a test you did weeks or months ago.

2.2.3 Building test automation actions from the ground up

When starting off with an automation effort just like anything else, planning is key. It would be good to have *'an idea'* of how you wanted this structured, but it should not be mandatory. For example, you may create the following structure for your automated test cases:

Login TC – Administration TC – Work Order TC – Checkout TC

What if there are steps you want to add to the Administration TC that impact the Work Order TC and ultimately the Checkout TC? Does that mean you have to re-record or re-code your effort? You shouldn't.

What is recommended is to concentrate on a quality recording, end-to-end. Make sure the steps flow clean with the proper objects clicked on. If done through code, the test automation analyst should build this in such a way that there is flexibility to add new test logic so the functional test analyst can modify this as needed without starting fresh. If you wouldn't recreate a brand new test suite or case for some inserts of logic for a manual test case, then you shouldn't have to with automated test case.

2.3 Analysis:

2.3.1 Adding error-handling in your testing flow

Test automation error-handling is different than manual testing in many regards. For example, some automation conditions that may have to be dealt with are:

- What if an unexpected window appears in the middle of the automated test execution run?
- What if objects are moved, added, removed or changed within my screen or application?
- What if objects are delayed in appearing in the automated test run?
- What if additional steps need to be added to your test automation recording?

These questions should be on the forefront of automation preparation and error-handling for the functional test analyst. Knowing how the automation will and should react to these situations will ensure these tests are robust for future test execution runs.

2.3.2 Knowing how to step into test cases and debug

Test automation test cases are similar to a manual test case in documenting the step-by-step procedures. Most of the test cases have code generated in the background. One key advantage of automation is the ability to step into a test case to see where an error has occurred or what application behavior is present. The following is a three-step process for enabling this detection:

2.3.2.1 Set your breakpoint

Evaluate your test case and put a stopping point on the specific step you would like. What this does is run the test up until the point and does not proceed further until instructed to do so.

2.3.2.2 Choose the 'Step' for the breakpoint

You want to select this after careful analysis because this is dependent on the state you want to see your application in. Knowing this step is very important in driving the subsequent step where the application displays an unexpected behavior from what you thought.

2.3.2.3 Execute the test case to the breakpoint

Once the breakpoint is hit, you have a few different options for further analysis:

- *Step Over*: stepping over the breakpoint to execute the next set of commands
- *Step Into*: stepping into the current function to see if that runs as expected
- *Step Out*: stepping out of the function entirely

2.3.3 Reading report results and interpreting this

Test Automation reports can be very informative and produce much more information than a manual test case would have noted. The fact that this information is produced automatically makes it great to review. Such report information is now available for your review within the automated test case execution run:

- Execution Time
- Machine Name
- Operating System
- Screen Dimensions
- Language
- Duration
- Total Errors
- Total Warnings
- Global Parameter Values
- Ability to jump right to the failed step
- Total Number of Iterations

3. Case Study

Titled: *Automation through the Back Door (By Support Manual Testing)* [1]

Background: To improve the rate of test automation in the organization, modifications were made to the test automation framework to support manual testing.

Technical Solution: Develop a framework that is based off of keyword-driven testing called command-driven testing.

What is Command-Driven Testing?

- Uses keywords that are simple commands (SELECT, BUTTON)
- Interpreter scripts are the same for all products
- Using the advantages that come from Data-Driven testing, navigation was placed into a DRIVER-File; data in a DATA-File
- These two files built together would form a command script
- The script-runner reads sequentially the commands in the DRIVER-File. DATA-Codes are substituted with data from the DATA-File

With Command-Driven testing:

- Testers don't necessarily need to learn tool scripting
- The separation in navigation and data sections makes the command scripts flexible and reusable
- DRIVER-Files can be easily ported to different applications
- DRIVER-Files need not be changed on migrating to another tool
- The test tool is needed only to prepare the templates and to run the tests

Prerequisites:

- You cannot start if you don't know the application and the test cases that are to be automated
- You need a working engine that can interpret all the commands you are going to need for your test cases
- You must have registered all the GUI elements that will be used in test execution in the proprietary mapping of the deployed capture/replay tools in order to normalize the names of the GUI controls

Process Steps:

1. Record the test case with the capture functionality of the test automation solution
2. Translate generated script to command-scripts
3. Developing planned TC's from template and build test suites

Case Study Key Points:

- Due to limited testing resources for test automation, significant effort still had to be spent on regression testing. However, the command-driven framework was adopted for ***manual testing*** as well as automated testing
- This approach helped limit the number of times a tester would simultaneously work on the same template, which was a current weakness
- Defect reporting became much easier. It avoided the testing team having to repeat the same steps to recreate the defect
- Continuous reviews were done to assess what features were available and what was needed to support manual testing

[1] *Experiences of Test Automation – Graham & Fewster*

- Implementation included a feature that supports the execution of partially automated tests, that could aid with tedious test preparation tasks
- Manual tests focused on customer-specific conditions now instead

4. Conclusion

Test Automation has different requirements than those for a traditional manual testing role. Understanding object recognition and repositories, error-handling techniques, automation frameworks such as data-driven testing, as well as debugging and reporting are key elements in test automation that are not so prevalent with manual testing.

Knowing some of these aspects going into the automation effort, allows a manual tester to jump right in and aid in the record / playback. The industry trend has been driving more towards scriptless automation so manual testers can make the transition easier. Knowing the application under test (AUT), is the largest hurdle and most manual testers know this already.

All manual testers should get involved in some aspect of automation. Not only does it increase your knowledge of testing tools in the market, but also it increases your market worth as a tester!

Brewing Analytics Quality For Cloud Performance

Li Chen[1], Pooja Jain[2], Kingsum Chow[1], Emad Guirguis[1], and Tony Wu[1]

{li.chen, kingsum.chow, emad.guirguis, tony.wu}@intel.com, pxj131230@utdallas.edu

Abstract

Cloud computing has become increasingly popular. Many options of cloud deployments are available. Testing cloud performance would enable us to choose a cloud deployment based on the requirements. In this paper, we present an innovative process, implemented in software, to allow us to assess the quality of the cloud performance data. The process combines performance data from multiple machines, spanning across user experience data, workload performance metrics, and readily available system performance data. Furthermore, we discuss the major challenges of bringing raw data into tidy data formats in order to enable subsequent analysis, and describe how our process has several layers of assessment to validate the quality of the data processing procedure. We present a case study to demonstrate the effectiveness of our proposed process, and conclude our paper with several future research directions worth investigating.

Biography

Li Chen is a data scientist at Intel. She received her PhD degree in Applied Mathematics and Statistics from the Johns Hopkins University in 2015. She specializes in applying analytics to system performance data.

Kingsum Chow is a principal engineer at Intel. He received his PhD in Computer Science and Engineering from the University of Washington in 1996. He specializes in performance, modeling and analysis of software applications, and leads analysis of cloud application performance. In his spare time, he volunteers to coach multiple robotics teams to bring the joy of learning Science, Technology, Engineering and Mathematics to the students.

Pooja Jain is pursuing her MS degree in Computer Science and Engineering at University of Texas, Dallas. She is an intern at Intel. She specializes in characterizing user behavior for cloud applications.

Emad Guirguis is a software performance engineer at Intel. He received his MS degree in Computer Science from Texas State University-San Marcos in 2012. He specializes in optimizing software applications in the cloud.

Tony Wu is a system architect and an engineering manager at Intel. He received his PhD degree in Electrical Engineering from the University of Minnesota-Twin Cities in 2005. He specializes in optimizing software applications.

---

[1] System Technologies and Optimization, Software and Services Group, Intel Corporation
[2] Computer Science and Engineering, University of Texas at Dallas, TX

1 Introduction

According to the definition of Cloud Computing by the National Institute of Standards and Technology (NIST), “Cloud computing is a model of enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.” There are five essential characteristics for the cloud computing model: On-demand self-service, resource pooling, rapid elasticity, measured service, and broad network access. These characteristics are met through three types of service models for cloud computing: Software-as-a-service (SaaS), Platform-as-a-service (PaaS), Infrastructure-as-a-service (IaaS), and four types of deployment models: public cloud, private cloud, community cloud, and hybrid cloud [1]. Such easy to deploy properties are propelling a transition into cloud computing.

Enterprises used to run their applications on dedicated servers and hardly ever on a shared or cloud environment. Once it was apparent that even enterprises were looking to go along the cloud path and spend big in the process, testing the application performance in the cloud has become essential. Java started being featured in some hosting/cloud solution. The interaction of Java garbage collections across applications brings an additional challenge in assessing cloud performance. Cloud application usage is very variable, and there are several ways to measure performance; thus optimization can be done in many directions, where improvement in one factor may negatively impact another. End user satisfaction and quality of service establish the guideline for performance optimization. The complexity of the Java Cloud solution, including several services such as IaaS, PaaS and SaaS, adds to the tediousness of measuring the performance. The main challenges lie in two areas: there are complicated interactions between multiple applications running on the same machine, and different loads may be applied to those applications. Early work on delivering quality in software performance and scalability testing [2] needs to be extended to cloud computing.

In this paper, we describe a process, implemented in software, to assess the quality of cloud performance data. This process combines performance data from multiple machines, spanning across user experience data, workload performance metrics, and readily available system performance data. These performance statistics have been readily used to classifying Enterprise applications [3]. The data collection and processing procedure across the system is able to generate concrete data for posterior analytics. Principled statistical analysis on cloud performance data will serve as valuable tool to assess cloud performance.

User experience data, such as throughput and response time, can be obtained from typical load driver systems used in software testing. One typical load driver, Faban [4], is a driver development framework used in SPECjEnterprise [5], SPECvirt [6] and SPECsip [7] benchmarks, and can control a number of load generation nodes. Such a framework defines operations, transactions and associated statistics collection and reporting. Faban is one of the tools that can be used for cloud computing workloads.

System Activity Report (SAR) [8] on Linux systems can save system performance data such as CPU activity, memory/paging, device load, network, etc. It writes the contents of selected cumulative activity counters in the operating system. The accounting system, based on the values in the count and interval parameters, writes information the specified number of times spaced at the specified intervals in seconds. Performance Counters for Linux (Perf) [9], a Linux profiling tool for performance counters, can add additional detailed performance counters for software processes and microarchitectures.

One challenge of dealing with performance data sets is bringing them into a tidy data format [10] for analysis. Before discussing the issues of bringing performance datasets into tidy data format, we first provide the motivation of why such an action is desired. Cloud workload is much more complex than enterprise workload. Manually examining the cloud performance data may not be the optimal way. In addition, workload modeling requires the dataset usable for statistical modelling purposes. Dynamic analysis on the time series cloud performance data can provide new insights for cloud performance optimization. Techniques such as model selection, non-stationary time series analysis, and stochastic processes can be adopted to analyze the cloud workload. Furthermore one could also borrow the techniques such as spectral clustering [11], sparse representation classification [12] [13], vertex

nomination [14] [15] and graph matching [16] [17] in the field of random graph inference to model the user behavior graph of the cloud workload. All these inference frameworks and methodologies would benefit from the tidy data format.

There is no standard format in which the data is collected. Therefore, parsing the data into a simple and coherent format is as essential as collecting and analyzing the data. Data munging gets tedious when dealing with data sets collected from different sources. Merging these files based upon time series is one of the major challenges. Elaborating on this, various data sets might have different formats of time in which the data is collected, namely the a) world clock time and b) epoch (UNIX time). , which might further differ in the following criterion: i) time zones, ii) units of measurements (seconds or milliseconds), iii) the time interval at which the data is collected and iv) the frequency at which data samples are collected. The data set is too huge to manually check consistency of results produced by the software components.

To overcome these challenges, we first add a set of basic queries and scripts to check the correctness of the data and data format. As data can vary a lot, we then add a set of performance models to check the quality of the performance assessment. By using ensembles of performance models, we minimize potential errors in the software performance assessment tool chains.

Our paper has the following three key contributions: firstly, we define the taxonomy of challenges in identifying performance bottlenecks in the cloud; secondly, we develop a software tool that applies analytics to test the application performance in the cloud; and lastly, we establish a methodology to evaluate and improve the quality of the analytics used for cloud performance assessment. We organize our paper as follows: In Section 2, we discuss the motivation for proposing our software for cloud performance assessment and the major challenge behind it. In Section 3, we present in detail our software chain in order to assess cloud performance. In Section 4, we demonstrate in a case study of how our software is used. Section 5 summaries our paper and discusses future direction for this line of work.

2 Challenges

Analyzing cloud performance data is one way to assess cloud performance. While we expect insights from advanced statistical analysis or machine learning techniques, the first hurdle is to bring collected performance data into the tidy data format [10] before analytics are applied. As different performance tools generate data in different formats, parsing the data into a simple and coherent format is just as essential as collecting and analyzing the data. Moreover, validating the quality of the data processing procedure is necessary; otherwise all the subsequent work becomes in vain. In this section, we first start with a primitive and idealized scenario for quality assessment for data processing, then define the challenges in identifying performance bottlenecks in the cloud, and present our insights on dealing with and working around these challenges.

2.1 An Idealized Scenario: A Tale of Two Engineers

Here, we explore an idealized scenario for the purpose of illustrating how to assess the quality of processing raw performance data. Suppose two engineers each use a different program and write separate scripts to process the raw data. The first engineer exports her processed data into the traditional n-by-m matrix formats, where the rows denote the number of samples (depending on the situation, the samples can be time stamps for example), and the columns denote the dimension of cloud performance metrics. The second engineer also exports her processed data into the traditional tidy data format. Although the two engineers participate in the same process – transforming the same raw data into a traditional data matrix, the resulted processed data could be different due to error in coding. The most straightforward validation on data quality is to compare the two processed datasets entry-by-entry.

2.2 Back to Reality: Way More Challenges Ahead

In reality, way more challenges lie ahead to transform the data and validate the quality of processing. The term “data munging” refers to the mapping from raw data into another convenient format useful for further purposes. In our case, this format will conveniently enable data visualization, data aggregation, statistical analysis and predictive modeling.

Merging the several files, which are collected from multiple sources, based upon time series is not trivial. In practice, cloud performance data comes in high volumes, variable velocity, and different raw formats. Hence, manually checking the consistency of the datasets can be exhausting.

Even after multiple datasets are merged with consistent alignment of the time samples, the quality of munging still needs more assessment. The issues often arise when the data transformation method is incorrect, causing inconsistency between the raw dataset and processed & merged datasets. Again, manually checking consistency here is a difficult task, especially during the age of big data.

3 Assessing Analytics Quality for Cloud Performance

To overcome these challenges, we developed an approach consisting multiple layers of quality assessments. Our goal is to assess the quality of the processed data for further analysis. In a nutshell, we first add a set of basic queries and scripts to check the correctness. As data can vary a lot, we then add a set of performance models to check the quality of the performance assessment. By using ensembles of performance models, we minimize potential errors in the software performance assessment tool chains. **Error! Reference source not found.**depicts the flow of our proposed software. In the following subsections, we describe in detail how our software works.

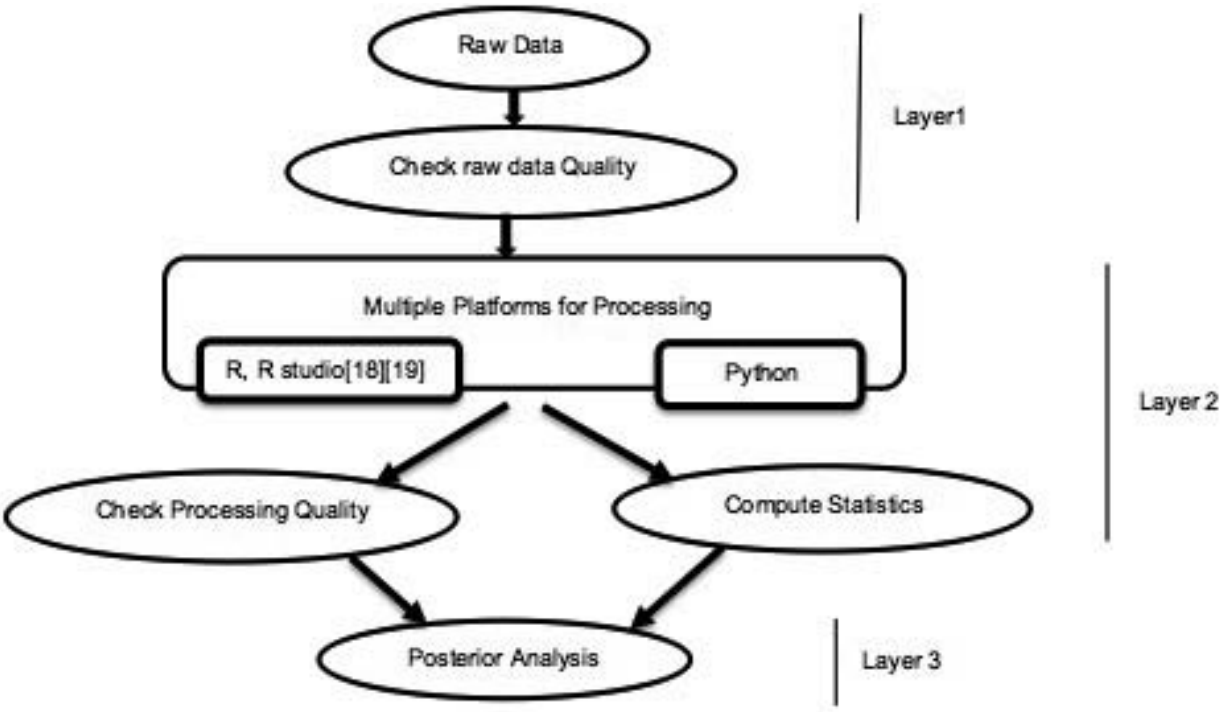


Figure 1. A quality model for Cloud performance analytics

3.1 Layer 1: Raw Data Quality Assessment

We obtain raw performance data from multiple sources such as SAR, Faban or other cloud testing sources. The very first step is to check the quality of the raw data; otherwise error propagates through the entire processing flow. We note that there are no universal rule for checking the quality of the raw data, as the rule checking strongly depends on the data source. For example, using the Faban driver, one rule to check the quality of the raw data is to examine a combination consisting of a particular time stamp and a particular machine name. This rule check is easily implemented using packages in R [18] or R studio 0, which reshape the data with counter values fitted in the data table.

3.2 Layer 2: Processing Quality Assessment

After verifying the quality of the raw data, the next step is to process the raw data into tidy data formats, which are convenient for further performance analysis. Our software model enables multiple programming languages, including Python and R [18], to process the raw data. Furthermore, our software model incorporates the freedom of using various packages even within a particular language. We define this step of our software as “multiple platforms for data processing”.

The key design reason that the raw data are processed via different channels is that this gives an extra layer of quality assessment. When the tidy data formats are generated using various package tools in different processing languages, our software can check the consistency of the processed datasets across multiple platforms. We believe this is a necessary step to guarantee data processing quality.

After the consistency of the datasets across multiple platforms is confirmed, we introduce another quality assessment, which is to compute statistics. Stated generally here, this step is also domain-specific. Our software enables such consistency checks by implementing rules based on domain expertise. For instance, the clock-per-instruction (CPI) cannot be lower than 0.25, or the CPU utilization cannot be over 100%. We note that these rules are not the gold standards, because real cloud performance data may have violated some rules, but the data itself is correctly measured.

3.3 Layer 3: Posterior Analysis Assessment

The last layer of quality assessment is posterior analysis. After the raw data is formatted into a tidy format, which can be input for statistical software tools such as R, posterior analysis delivers the final layer of quality assurance in our software. We test the posterior consistency by applying principled data analytics methodologies and detect anomalies that exist within the datasets. Although the anomalies may exist due to the intrinsic and unavoidable noise in the dataset, some anomalies directly reflecting the poor quality of the datasets can still be identified using basic statistical methods. Particularly, the statistical tools such as outlier detection, clustering and density fit can pinpoint the “impossibles” in the data. This will alert the engineers to have a second look at the data processing procedure. We note that the step of posterior analysis is important for assuring the quality of the data processing procedure. However, practitioners will not rely solely on this step, since most mistakes should be captured in the earlier two layers of quality assurance.

3.4 Test Cases

Although not presented in depicting our approach, we consider several test cases to validate the data processing quality. The key reason for including test cases is that, with prior knowledge of the test cases, we expect to see certain outputs after data processing. When such outputs are missing, we know the data processing procedure contains errors, and does not pass our quality assessment.

4 Case Study

In this section, we present a case study of processing cloud performance data using our approach, and demonstrate that our approach, with several layers of quality assessment, is capable of capturing mistakes during data processing.

4.1 A Cloud Workload

For our demonstration we use a Software-as-a-Service (SaaS) workload composed of dozens of Java applications serving requests in a group of domains. The details of these applications are beyond the scope of this paper. As illustrated in Figure 2, the workload is driven by five groups of users simulated on the driver. Each user group simulates a particular type of users, sending a sequence of requests to the service. Upon receiving the response to a request, each virtual user waits for a period of time, called the think time, before sending the subsequent request. Different number of virtual users are assigned to each user group. In an experiment, they are gradually increased to a maximum level, based on the design of the experiments.

On the server side simulating the cloud computing, there are 54 Java Virtual Machine (JVM) applications running as the SaaS workload, responsible for addressing different sets of requests. For instance, consider it as an employee portal offering various services for management of wages/vacation tasks etc, and each service handled by set of JVM instances. These Java applications are configured with different parameters. They have different run-time behavior from each other. They also interfere with each other in the cloud. There is also a database process providing data support to these applications.

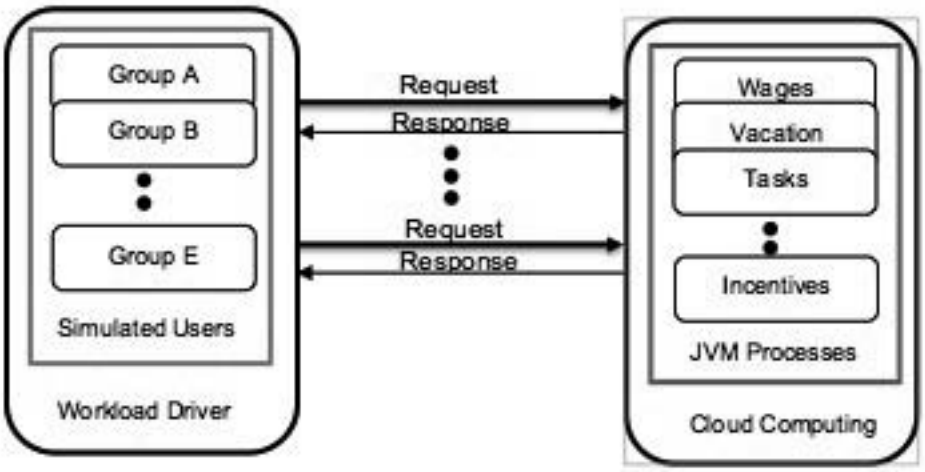


Figure 1. A Cloud workload is composed of a driver of multiple user groups sending requests to and receiving responses from the services in the cloud.

4.2 Performance Data Collection

We demonstrate our experiment using two machines, one as a web server and the other as a client server. The client server hosts an application driver that generates the workload. As described in the previous section, it drives the process of sending in the requests to the server through virtual users. In parallel it also increases the load on the server by ramping up the number of virtual users interacting with

the server. There are 45 application servers running over 54 Java virtual machines, a combination of which are responsible in processing each request.

To learn the performance of the cloud workload, we collect data on both the machines, or say from every source applicable. Therefore we collect data from every domain that reflects performance of the workload, such as on server, or system level, which is determined by operating system. This data usually helps us determine the bottlenecks of the system. That is, it gives us an idea about how much the system resources (CPU, input/output, memory, and network) are being utilized over time. These application specific data points usually describe the behavior of the application, given the load. In our case we collect java garbage collection (GC) logs of each of the 45 application servers to see their activity and interaction among themselves. On the driver simulating virtual users, we collect workload specific data, which emphasizes the user experience. The driver records data like the number of users, response time, failed transactions etc.

As the sources of the data collected vary as described above, the datasets vary from each other in terms of frequency of data points, metrics collected, units of measurements, etc. For example, if we consider frequency, the SAR data is collected on a server every 10 seconds, so that it doesn't impact the system performance, whereas the workload data on a client is collected every 5 seconds, as the performance on client server is not critical. To some extent, the frequency at which the data is collected via these tools is still controllable, but when it comes to the application specific data, in our case the GC logs, the data points are only recorded when a garbage collection takes places. The kinds of metrics collected by each tool is different, but getting all of this data together in a tidy format for analysis may help us generate an insight about which attributes impact the other or how they are correlated.

4.3 Performance Data Processing

We describe how we process the data collected from SAR, GC and client server. The original garbage collection data is in log files. The log files are human-readable: for example, each line may state the time when garbage collection occurs, the time elapsed since start of the workload, GC type (either GC or Full GC), memory before and after GC, heap size, and GC pause time. Performance engineers may manually examine the log files line by line to identify performance issues. The purpose of developing our tool is to enable an automated process for cloud performance analytics. Additionally, our tool provides the first necessary step to apply multivariate time series analysis to the performance data. Our tool uses Python scripts to parse the log files line by line. The processed GC dataset has time stamps in epoch milliseconds in each row, and attributes in each column. The time stamps correspond to when GC activity happens. The attributes are memory before GC, memory after GC, heap size, GC type (GC or Full GC) and GC pause time. The resulting data set is thus in tidy data format, and simple statistical analysis implemented in R can be done on this dataset. Furthermore, transforming the GC log files into tidy data formats provides a common ground for merging the SAR data and client server data, and this gives a better perspective on the cloud workload. The role of our tool is to assure the quality of the data is sound.

The second source of data is collected from the System Analysis Report. Recall that SAR records performance every 10 seconds in this workload. Therefore, the file is structured such that the time stamps (in epoch seconds) are in each row with increment of 10 second, and the features are in each column. The features are mainly describing the aspects of CPU, memory, disk I/O and network. Some examples of features include CPU percentage utilization, paging statistics, packets received statistics and so on.

Particularly for this workload, we also have performance data collected from the user experience perspective. As discussed before, this data is collected by a specific client driver, which is a software tool to simulate user groups, where each group has specific user behaviors. The original data format is in terms of unique pairs of time stamps and performance counters, i.e., (T1, P1), where T1 is the recorded time stamp and P1 is the recorded performance counter at T1; one can think of it as edge lists. The data format is different from the GC logs and the SAR data, and provides challenges to directly merge this data with GC and SAR. Moreover multivariate time series analysis cannot directly be done on the paired data, and therefore our tool plays the role of reshaping the data set to enable subsequent merging and analysis. Here we transform the unique pairs into standard data format such that the rows are time

stamps (in unit y:m:d:h:m:s) and the columns are performance counters for the user behaviors. Some examples of performance counters are server times, web pages received per second, kilobytes received per second and so on. Note that the time increment is 5 seconds, as the client server records the data every 5 seconds. Data processing on the client data is done using R using specific packages. To guarantee the quality of processing this dataset, our software implements two independent methods to transform the client data to tidy data format, and compares the consistency of the transformed datasets entry-by-entry. This step provides an extra layer to assure the quality of data processing.

4.4 Merging of the Performance Datasets

The step of data processing described in Section 4.3 transforms the SAR, GC and client server datasets into three tidy data formats respectively. These three datasets describe the workload from three aspects. Statistical analysis can be done directly and separately on these datasets. However our software aims at providing a tool to bring together all three datasets, so that statistical analysis can be done on the entire unified dataset, and explore the correlation among SAR, GC and client server. There are several challenges for merging these datasets. Firstly, the units of time in these three datasets are different: one in world clock, one in epoch seconds, and one in milliseconds. Secondly, the mean of the time stamps vary across three datasets. The time stamps for GC dataset denote the occurrence of GC activities, and thus the increments between the time stamps are not constant, and but are in fact random. On the other hand, the increments between the time stamps for SAR data are 10 seconds, as the system-level performance counters are recorded and reported every 10 seconds. The increments between the time stamps for the client data are 5 seconds, as the client driver records the response time and user experience related performance metrics every 5 seconds. Our tool overcomes the challenges of time stamp unit and interval discrepancy by converting all time units into epoch milliseconds and interlacing the epoch times of all three datasets. Therefore, our tool is the first tool to enable cloud workload characterization from system, applications and client perspective. This provides a convenient platform for subsequent statistical analysis for cloud workload performance.

4.5 Performance Data Profiling and Analytics

After the step of data merging, we have one coherent data set containing multivariate time series to describe the cloud workload. The rows in the dataset are the time stamps in epoch milliseconds and the columns in the dataset are performance metrics for operating system, user experience as well as garbage collection statistics. We have discussed the advantage of utilizing such a merged dataset. At the same time, some issues arise that are worth our attention. First of all, if the original SAR, GC and Client data matrices have dimensions k by l, m by n, and x by y, then the resulted data matrix has dimension a by b, where a ≤ k + m +x, and b = l+n+y. This merged data matrix has more time samples and a higher number of attribute dimensions. Secondly, the merged data matrix has a notable number of missing values, which are created artificially when merging and interlacing the time stamps across three datasets.

Although one could examine the merged data matrix entry by entry, it is more economical and practical to appeal to data profiling on the big data matrix to assess its quality, especially for the reason that all three datasets were examined entry-by-entry respectively as discussed in Section 5.2. Our tool implements the calculations of mean, median, minimum, maximum, range, the number of missing values, and percentiles to profile the datasets. These computations of the profiling statistics are implemented in R and formulated as below. For each performance metric X, let X1, X2, …, XT denote the time series of X during the cloud workload. The software profiles the performance metric by calculating:

$$mean(X) = \frac{1}{T}\sum_{i=1}^{T} X_i.$$

$$median(X) = midpoint(X_1', X_2', \dots, X_T'), where\ X_1' \le \cdots \le X_T' and \{X_1', X_2', \dots, X_T'\} = \{X_1, X_2, \dots, X_T\}.$$

$minimum = min_i\{X_1, X_2, \dots, X_T\}.$

$maximum = max_i\{X_1, X_2, \dots, X_T\}.$

$range(X) = maximum - minimum.$

$Number\ of\ missing\ values\ for\ X = \sum_{i=1}^{T} indicator\{X_i\ is\ missing\}.$

$Percentile\ p_i = 100(i - 0.5) for\ (X_1', X_2', \dots, X_T'), where\ X_1' \leq \cdots \leq X_T'\ and\ \{X_1', X_2', \dots, X_T'\} = \{X_1, X_2, \dots, X_T\}.$

Computing these profiling statistics for every metric gives us a concrete overview on the quality of the merged data. For example, the number of missing values cannot be greater than or equal to number of time stamps; otherwise this attribute is merely noise. Another example could be that the clock-per-instruction (CPI) cannot be lower than 0.25, and one could use the minimum and percentile to verify. Or the CPU utilization cannot be over 100%, which can be checked using these profiling statistics. Again, we note that these rules are not the gold standards, because real cloud performance data may have violated some rules, but the data itself is correctly measured. However, the profiling statistics of the performance metrics alert the practitioner if too many inconsistent or counterintuitive phenomena occur.

4.6 Assess Performance Analytics Quality

Discrepancies in data processing may arise and it is difficult to spot errors, especially when the data is high dimensional. Recall in the tale of two engineers, where they write scripts respectively and independently to process the data, and then compare their results entry by entry to ensure quality. Even though this is an idealized scenario, this particular process gives an extra check on the quality of data processing.

Our tool indeed embraces this piece of philosophy of two engineers working on the same data processing problems in parallel and detecting differences along the way. Our software implements two independent scripts in parallel, using different packages in R or Python, to process the same cloud performance data. Then the software compares the processed performance datasets from two parallel channels. The comparison is done not only entry-by-entry, but also ensures the attribute names and orders are consistent.

Additionally we added a third-party involvement feature to our software. This feature enables the software user to upload his/her own data processing scripts, and compares the processed dataset from their scripts with the processed datasets using the two already implemented and parallel processing channels. This feature provides feedback on the quality of the user's processing script, and double checks the quality of the data processing scheme.

5 Summary and Discussion

Cloud computing has transformed the IT industry. Analysis on cloud performance has an immerse impact on the cloud computing environment, and sensible analysis can lead to better and more efficient cloud computing environment. However, cloud performance data comes in very raw form, which means they are not readily for further data analytics. In this paper, we propose a software that transforms the raw data into conventional data formats, and explain the software's several layers of quality assessment to ensure the data processing quality. We present a case study of cloud performance data, and demonstrate the effectiveness of our software at data processing and particularly the quality of data processing. We have established a methodology to evaluate and improve the quality of the analytics used for cloud performance assessment.

Acknowledgement

The authors would like to thank the referees Dave Patterson and Emily Ren for their insightful comments and suggestions.

References

[1] Mell, Peter ,and Tim Grance. "The NIST definition of cloudcomputing." *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg*. (2011).

[2] Khun Ban, Robert Scott, Huijun Yan and Kingsum Chow, "Delivering Quality in Software Performance and Scalability Testing", *Pacific Northwest Software Quality Conference*, October 2011, Portland, OR. http://www.uploads.pnsqc.org/2011/papers/T-40_Ban_etal_paper.pdf

[3] Zhidong Yu, Hongjie Wu, Da Teng, Zhengzhu Xu, Ethan Huang, Peng-fei Chuang, Tony Wu and Kingsum Chow, "Classifying Enterprise Applications using Computer System Performance Statistics", *The 2nd ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2011)*, Shanghai, China (July 11-12, 2011)

[4] https://java.net/projects/faban/

[5] http://www.spec.org/jEnterprise2010/

[6] http://www.spec.org/virt_sc2013/

[7] http://www.spec.org/sipinf2011/

[8] https://en.wikipedia.org/wiki/Sar_(Unix)

[9] https://perf.wiki.kernel.org/index.php/Main_Page

[10] Wickham, Hadley, "Tidy Data", under review. 2014. http://courses.had.co.nz.s3-website-us-east-1.amazonaws.com/12-rice-bdsi/slides/07-tidy-data.pdf

[11] Ng, Andrew Y., Michael I. Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm." *Advances in neural information processing systems* 2 (2002): 849-856.

[12] Chen, Li, Cencheng Shen, Vogelstein, Joshua, and Priebe, Carey. "Robust Vertex Classification."*IEEE Transactions on Pattern Analysis and Machine Intelligence, accepted for publication 2015.*

[13] Shen, Cencheng, Li Chen, and Carey E. Priebe. "Sparse Representation Classification Beyond L1 Minimization and the Subspace Assumption." *arXiv preprint arXiv:1502.01368* (2015). Under review in *the Journal of the American Statistical Association*.

[14] Fishkind, Donniell E., Vince Lyzinski, Henry Pao, Li Chen, and Carey E. Priebe. "Vertex Nomination Schemes for Membership Prediction." arXiv preprint arXiv: 1312.2638. *Annuals of Applied Statistics* (2015).

[15] Chen, Li. "Pattern Recognition on Random Graphs." *PhD Thesis, Johns Hopkins University* 2015.

[16] Conte, Donatello, Pasquale Foggia, Carlo Sansone, and Mario Vento. "Thirty years of graph matching in pattern recognition." *International journal of pattern recognition and artificial intelligence* 18, no. 03 (2004): 265-298.

[17] Lyzinski, Vince, Daniel L. Sussman, Donniell E. Fishkind, Henry Pao, Li Chen, Joshua T. Vogelstein, Youngser Park, and Carey E. Priebe. "Spectral clustering for divide-and-conquer graph matching." arXiv preprint arXiv: 1310.1297. *Parallel Computing* (2015).

[18] R Core Team (2015). R: A language and environment for statistical computing. *R Foundation for Statistical Computing*, Vienna, Austria. URL http://www.R-project.org/.

Studio, R. "R Studio: integrated development environment for R." (2012).

Performance Test Modeling with "ANALYTICS"

Jeevakarthik Kandhasamy

Performance test Lead Consultant – Capgemini Financial Services USA

jeevakarthik@gmail.com

Abstract

Websites and web/mobile applications have become the imperative sales enablers for most organizations. The "optimized user experience" of a website becomes critical for organizations to outperform their competitors. This demands performance testing to be proactive and shift dynamics from measuring responsiveness to improving user experience.

User Experience (UX) index of a software product can be measured by the ease of use, providing navigational comfort and quick responsiveness to end user – In a nutshell – better performance of an application results in better user experience – but unfortunately even after a huge spend critical performance issues are detected / observed in production. Some of the critical factors for these include

- Performance testing objective remains to be meeting response time SLA's
- Ambiguity in load test modeling
- Not emulating actual user behavior during load tests
- Lack of simulation of devices used by end users

Web Analytics tools like Google Analytics, Yahoo analytics etc have come a long way and simplified the process of analyzing, collecting and reporting user behavior which serves as a input in creating a comprehensive performance test strategy and test bed which included loads of compulsive features.

Biography

Jeevakarthik Kandhasamy – certified, test engineering Leader & Architect with experience in Functional & Non-Functional Testing. With over 12 years of experience in Performance testing, consulting, strategic partnering & program management, executing transformation, change management programs in testing space – Jeeva has served financial services majors in India, China, Europe and North America. His interests include championing organizational wide activities like IP lead test assets creations, Frameworks, assessments, Innovation drives & testing road shows.

He currently based in Chicago and leads a testing engagement for a leading global insurance customer

Copyright <Jeevakarthik Kandhasamy> <06-14-2015>

1 Introduction

A Recent Cisco survey reports that 78% of the North American shoppers use internet, social media or mobile devices for research and purchase of products and services – Along with the brand value, quality and other marketing campaigns most firms rely on their web applications to yield revenue – DISGRUNTLED user experience may turn out to be a perfect recipe for IT organizations failure. Some of the risks associated include

- ➔ Increased spend on IT
- ➔ Failure to meet sales goals and business objective
- ➔ Decreasing revenue
- ➔ Denting organizations brand value

The think tanks in most online shops have realized the importance of improved UX Index and have stressed the need and importance of non-functional testing and specifically performance testing. This involves a lot of money for

- ➔ Setting up infrastructure for performance testing
- ➔ Tools and licenses
- ➔ Resource cost

In spite of the organizations spending tremendous money for performance testing – most of the Performance issues are detected in production – The key reasons of failure are

- ➔ Performance testing objective remains to be meeting response time SLA's
- ➔ Ambiguity in load test modeling
- ➔ Not emulating actual user behavior during load tests
- ➔ Lack of simulation of devices used by end users

All these clearly indicate that lack of analytical data on the actual user behavior in production. The approach of determining user and system behavior with server logs becomes tedious and uncertain.

Performance testing strategized without understanding the actual user behavior results in finding out issues in production.

2 Solutioning with Analytics

Web Analytics tools like Google Analytics and Yahoo! Web Analytics have come a long way and simplified the process of analyzing, collecting and reporting user behavior which serves as an input in creating a comprehensive performance test strategy and test bed (load test model). Some of the compulsive features include:

- ➔ Providing a holistic view of business
- ➔ Decision making
- ➔ Custom and integrated reporting
- ➔ User conversion rate
- ➔ Bounce rate and reasons for users exiting the website
- ➔ For applications yet to be developed experimental sites are available with analytical data based on trends from competitors and industry standards.

➔ Understand visitor behaviors like

- How visitors use your website
- How they arrived on your website
- Geographical locations of users
- Devices used by users accessing the website

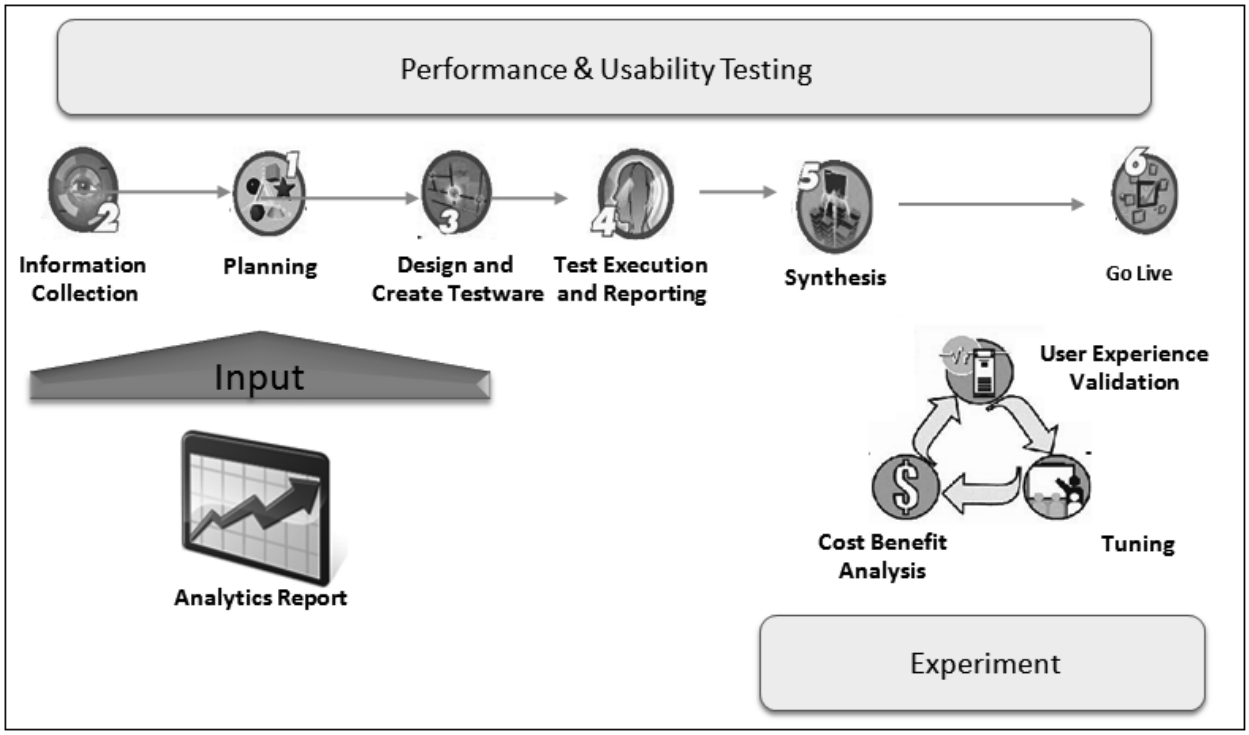
Web Analytics simplifies the process by providing the non functional tester inputs on:

- Who the end users are,
- What they do
- Entry and exit points of the application.
- What they like and don't like in the application.

It's fair to say that web analytics tools can help determine the performance testing objective, help creating the performance testing strategy, load test modeling and executing load tests simulating actual user behavior.

3 Modeling a Performance Test using Google Analytics

The section below explains how a performance test could be designed and executed using Google Analytics. I have provided snippets and screen shots from Google Analytics. The flow diagram below indicates the end to end performance test process.

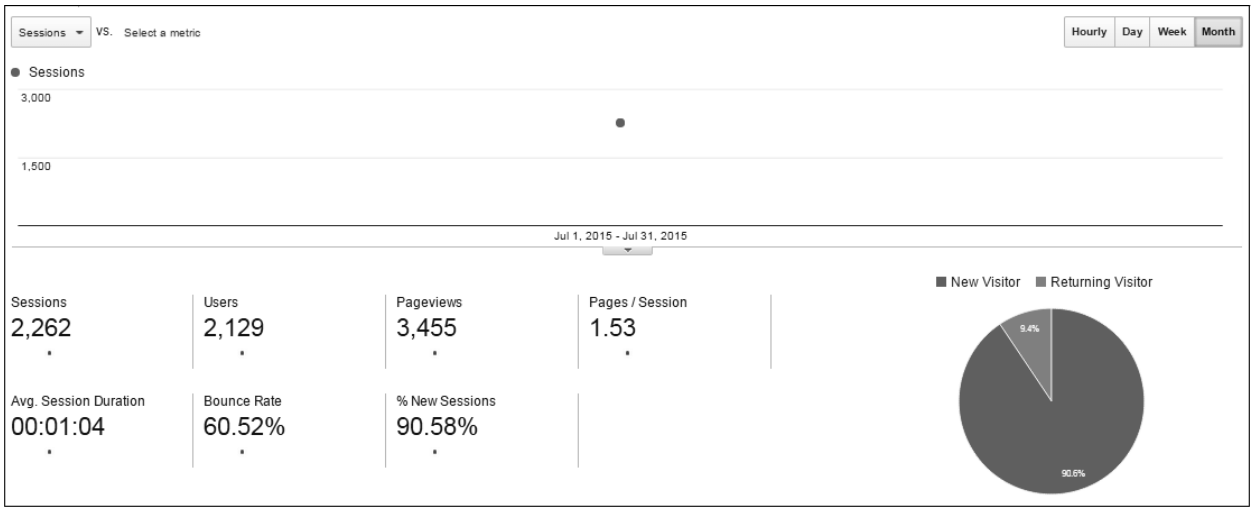


Performance test planning will include designing, executing and reporting a performance test. Some of the key factors for designing of a performance test will depend on:

- ➔ Type of the performance test to determine the virtual user load and the duration of the test – e.g. load, stress, endurance, spike etc.
- ➔ User flow or traversal flow
- ➔ Platform – Client and server systems
- ➔ Geographical distribution of the users

Determining the type of test

For a portal that is already existing in production – user, session, content metrics are available on hourly, daily, weekly, monthly and yearly basis. This behavior analysis should help a performance engineer to correlate and identify the test load, duration and objective of the test. In the section below we will walk through a few scenarios for modeling a performance test, and the advantages of using Analytics tool.



User profile

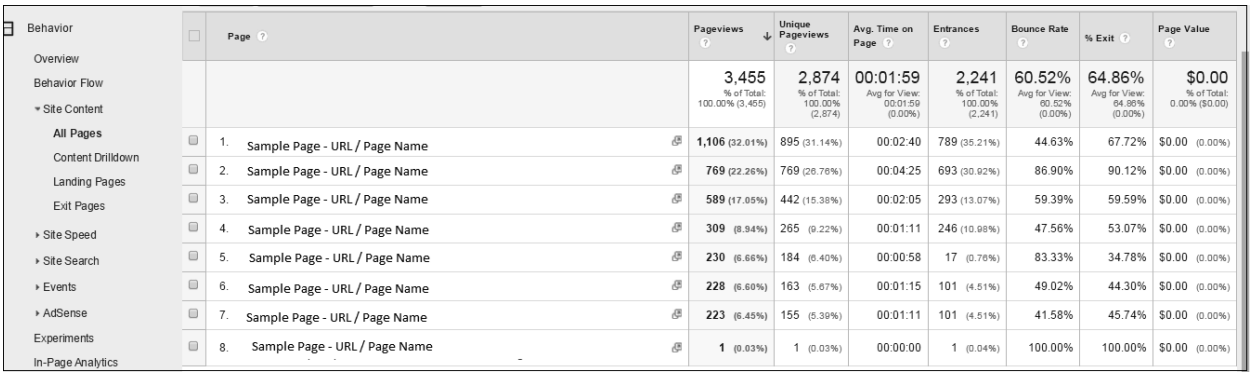
One of the main flaws in performance test design is not mimicking the user profile – but with analytics various factors is factored in – type of user, demographic, geographical location and the most important factor being - % of new users vs % of returning users which will depict the actual user behavior.

Traversal flow

While identifying the performance test flows it's usually based on the Key Performance Indicators and ensuring they are covered as part of the flows. User flow analytics provides users with the following information:

- ➔ User landing page – that includes total number of sessions – active and inactive along with the drop-offs
- ➔ First Interactions, second and subsequent interactions

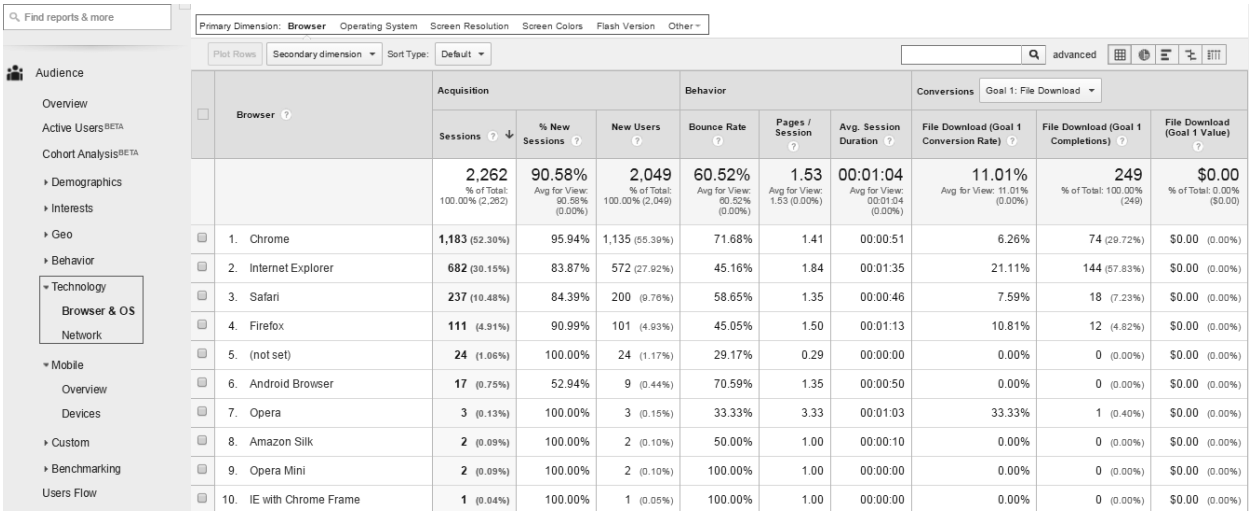
This will help us to identify the user flow, interacting internal and external systems and services.



Page	Pageviews	Unique Pageviews	Avg. Time on Page	Entrances	Bounce Rate	% Exit	Page Value
	3,455 % of Total: 100.00% (3,455)	2,874 % of Total: 100.00% (2,874)	00:01:59 Avg for View: 00:01:59 (0.00%)	2,241 % of Total: 100.00% (2,241)	60.52% Avg for View: 60.52% (0.00%)	64.86% Avg for View: 64.86% (0.00%)	$0.00 % of Total: 0.00% ($0.00)
1. Sample Page - URL / Page Name	1,106 (32.01%)	895 (31.14%)	00:02:40	789 (35.21%)	44.63%	67.72%	$0.00 (0.00%)
2. Sample Page - URL / Page Name	769 (22.26%)	769 (26.76%)	00:04:25	693 (30.92%)	86.90%	90.12%	$0.00 (0.00%)
3. Sample Page - URL / Page Name	589 (17.05%)	442 (15.38%)	00:02:05	293 (13.07%)	59.39%	59.59%	$0.00 (0.00%)
4. Sample Page - URL / Page Name	309 (8.94%)	265 (9.22%)	00:01:11	246 (10.98%)	47.56%	53.07%	$0.00 (0.00%)
5. Sample Page - URL / Page Name	230 (6.66%)	184 (6.40%)	00:00:58	17 (0.76%)	83.33%	34.78%	$0.00 (0.00%)
6. Sample Page - URL / Page Name	228 (6.60%)	163 (5.67%)	00:01:15	101 (4.51%)	49.02%	44.30%	$0.00 (0.00%)
7. Sample Page - URL / Page Name	223 (6.45%)	155 (5.39%)	00:01:11	101 (4.51%)	41.58%	45.74%	$0.00 (0.00%)
8. Sample Page - URL / Page Name	1 (0.03%)	1 (0.03%)	00:00:00	1 (0.04%)	100.00%	100.00%	$0.00 (0.00%)

Technology, platform and devices

Application performance in most situations are based on server performance, but with modern technologies – performance of a particular application will depend on the system that is used by the end user. Some of the factors influencing this would include: Browser, Operating Systems, Screen resolution, Flash version and Script control of the user:



Browser	Acquisition			Behavior			Conversions Goal 1: File Download		
	Sessions	% New Sessions	New Users	Bounce Rate	Pages / Session	Avg. Session Duration	File Download (Goal 1 Conversion Rate)	File Download (Goal 1 Completions)	File Download (Goal 1 Value)
	2,262 % of Total: 100.00% (2,262)	90.58% Avg for View: 90.58% (0.00%)	2,049 % of Total: 100.00% (2,049)	60.52% Avg for View: 60.52% (0.00%)	1.53 Avg for View: 1.53 (0.00%)	00:01:04 Avg for View: 00:01:04 (0.00%)	11.01% Avg for View: 11.01% (0.00%)	249 % of Total: 100.00% (249)	$0.00 % of Total: 0.00% ($0.00)
1. Chrome	1,183 (52.30%)	95.94%	1,135 (55.39%)	71.68%	1.41	00:00:51	6.26%	74 (29.72%)	$0.00 (0.00%)
2. Internet Explorer	682 (30.15%)	83.87%	572 (27.92%)	45.16%	1.84	00:01:35	21.11%	144 (57.83%)	$0.00 (0.00%)
3. Safari	237 (10.48%)	84.39%	200 (9.76%)	58.65%	1.35	00:00:46	7.59%	18 (7.23%)	$0.00 (0.00%)
4. Firefox	111 (4.91%)	90.99%	101 (4.93%)	45.05%	1.50	00:01:13	10.81%	12 (4.82%)	$0.00 (0.00%)
5. (not set)	24 (1.06%)	100.00%	24 (1.17%)	29.17%	0.29	00:00:00	0.00%	0 (0.00%)	$0.00 (0.00%)
6. Android Browser	17 (0.75%)	52.94%	9 (0.44%)	70.59%	1.35	00:00:50	0.00%	0 (0.00%)	$0.00 (0.00%)
7. Opera	3 (0.13%)	100.00%	3 (0.15%)	33.33%	3.33	00:01:03	33.33%	1 (0.40%)	$0.00 (0.00%)
8. Amazon Silk	2 (0.09%)	100.00%	2 (0.10%)	50.00%	1.00	00:00:10	0.00%	0 (0.00%)	$0.00 (0.00%)
9. Opera Mini	2 (0.09%)	100.00%	2 (0.10%)	100.00%	1.00	00:00:00	0.00%	0 (0.00%)	$0.00 (0.00%)
10. IE with Chrome Frame	1 (0.04%)	100.00%	1 (0.05%)	100.00%	1.00	00:00:00	0.00%	0 (0.00%)	$0.00 (0.00%)

With the increase in use of mobile devices – most applications today are mobile compatible. Analytics tool provides information on the devices (Browser, OS), network speed, carriers etc.

Geographical distribution

With the business model today being global, performance testing should also follow and be distributed. This could be achieved by understanding the user geographies and simulating it using latency and network simulations during a performance test.

Along with these benefits, analytics dashboards can be used for monitoring production environment real time.

4 Case Study

This section contains a case study describing the need and benefits of using web analytics tools for performance testing for leading auto insurance provider in North America insuring 18 million vehicles across USA.

4.1 Situation

Clients Auto insurance application was the application under test. This was Java / J2EE based system. Based on the Non-functional requirements the objective of performance testing was to ensure the response times of critical Key Performance Indicators (KPI) were less than 7 seconds. The downsized test bed was to ramp up 700 concurrent users and create 2500 auto policies with different driver / vehicle combination in the performance test environment.

4.2 Issues Observed

The client was spending close to 1.5 million dollars annually for performance testing that includes the cost of resources, infrastructure, tools etc. The following issues were reported by business and production team

- ➔ Response times for page navigation took more than 11 seconds
- ➔ Multiple search failures reported between noon – 3 PM EST
- ➔ 2 production outages
- ➔ Declining online sales conversion

4.3 Assessment observations

A detailed assessment was executed to understand the reasons for failure by using Google Analytics in both production and Performance test environment.

Some of the key observations related to load test modeling failures are listed below

- ➔ The test simulated 700 users creating 2500 policy in test environment – but the actual situation was 2000 + users accessing the system and only 700 users navigate and proceed till purchase – **USER AND VOLUME SIMULATION** issues.
- ➔ Close to 65% of the agents were using browsers IE 7 and below which did not support the caching and other UI based performance accelerators – **CLIENT & BROWSER** related issues.
- ➔ The client data centre was located in Georgia while more than 40% of the users were from north east and west coast locations – the load test did not reciprocate the **NETWORK LATENCY** similar to production.
- ➔ Search initiated from home page took more time while search from other pages were fine – **SITUATIONAL & USER BEHAVIOUR** issues.
- ➔ The peak volume reached increased by more than 30% during promotional offers – failure to execute **SPIKE test** due to promotional / un-expected volume.

4.4 Assessment observations

- ➔ Non-functional requirements and load test model, test scenario design was updated based on the actual user behavior from Analytics report – the user pattern was not available from server logs.

- ➔ Additional plug-in Recommendations provided to agents for browsers
- ➔ User pattern was modified to include 120 MS and 245 MS network latency respectively for east and west coast users.
- ➔ Multiple performance issues related to code and configuration were observed during performance tests which resulted in the following performance tuning recommendations
    - o DB indexing
    - o Client side caching
    - o Search optimization
    - o Load balancing
    - o Network performance

4.5 Results

Following are the positive results by using web analytics for performance testing:

- ➔ Realistic simulation of user behavior from production
- ➔ Detected 82% of performance issues during performance testing phase – reduced risk of identifying performance issued in performance
- ➔ Improved user experience – obtained based on customers and agent survey.

Observed trend of improved sales by 6% – 8% on a half early basis.

5 Conclusion

Using web analytics tools adds value to the performance testing process. Some of key benefits include:

- ➔ Efficient load modeling
- ➔ Improving user experience
- ➔ Improve sales and user conversion
- ➔ Decision making
- ➔ Support and adapt based on evolving changes in production
- ➔ Cost effective solutions – availability of open sourced and licensed tools & reports

In tough economic situations, organizations today are forced to deliver high performing applications to retain existing customer base and attract new customers.

The goal of Performance testing needs to shift from measuring responsiveness to improving the customer experience and web analytics tools can help achieve these objectives and help organizations succeed.

References

http://en.wikipedia.org/wiki/Web_analytics_tools

www.google.com/analytics

http://www.adobe.com/solutions/digital-analytics.html

Moving up the Product Security Maturity Model

Authors: Joshua Cajetan Rebelo and Patrick McEnany

joshua.cajetan.rebelo@intel.com patrick.s.mcenany@intel.com

Abstract

The world has become a global village, and it is being ruled and prominently controlled by technology and electronics in particular. This results to consistent increase in the availability of personal, corporate, and financial information in cyberspace. This creates enormous opportunities for cyber attackers to access the data and misuse it through hacking tools and tutorials.

One such recent example is the intellectual property theft in the Xbox One gaming console and Xbox Live. The hacking of Target and Home Depot networks lead to the leakage of sensitive data such as email-ids and credit card details. Another example is the data breach of 4.2 million individuals in the US Government Office of Personnel Management (OPM). These incidents clearly emphasize the necessity to deliver a comprehensive secure product. For organizations, the goal must be to adopt a better strategy and protect the data and resources in a more proactive manner.

Organizations span the spectrum when it comes to the maturity around creating secure products. Some organizations have a well-defined process that ensures the delivery of highly secure products whereas some organizations want to improve the security maturity model but lack management support. Some organizations are not even aware of product security and they are not sure from where to start.

This paper defines a multi-layered security approach that can be applied to any platform, product, and programming language. The multi-layered security helps the product teams that having little knowledge of product security to uncover the low hanging security defects. As the team gains expertise they become Evangelist and Champions of secure software development.

Since the threat types and attack vectors are evolving at a rapid pace, creating a security maturity model for the product that can provide up-to-date protection and realign its capability to handle the latest security challenges are vital.

Biography

Joshua Cajetan Rebelo is a Senior Technical Lead and a Product Security Champion at Intel Security Group with 11 years of experience in the security domain. He ensures that the products are under continuous development to incorporate the highest Security Software Development Lifecycle (SDL) standards and adhere to the Product Security Maturity Model (PSMM) for pro-active in findings vulnerabilities. He has vast experience in conducting security code reviews and audits, evaluating software architecture for potential risk, and recommending design changes to address security concerns. He also brings innovation with new ideas and has three patents issued to his credit.

Patrick McEnany is a Senior Manager and a Lead Product Security Champion for the Intel Security Group (ISecG) with 18 years of experience in the software product security domain. He is an extended core team member of the ISecG Product Security Group and directly manages the product security programs across 17 core Intel Security Enterprise Products. Many of his primary responsibilities include adoption and enforcement of the Product Security Maturity Model (PSMM) and Product Security Incident Response Team (PSIRT) program. He is continuously working with products teams to mentor volunteers designated as Product Security Champions at the product team level.

Intended Audience

The target audience for this paper includes program and project managers, developers, quality assurance engineers, and IT professional who are familiar with the software security literature. This paper is also meant for people who are passionate about security and want to integrate security into their standard software development lifecycle (SDL) processes and to enhance the product security maturity model to stay secured.

1. Introduction

Many organizations do not plan product security until the end of the overall development process. Instead of planning and executing it for all phases of the release, they run the external security audits at the end. Following the traditional Waterfall method at the end of a development release will always uncover security issues such as incorrect input validation, SQL injection, XSS attacks, weak SSL design, or vulnerable third-party libraries with known vulnerabilities. Finding the security issues at the end of the cycle always hits the release schedule. If the security issues are left unresolved, it increases the risk for the product, affects the brand recognition in the industry, and financial loss for organization, and the customer loyalty. Critical security flaws often require architectural design changes that in turn increase your development costs by refactoring the components involved.

Some organizations think that product security can be addressed through a check-list. This will actually not always work out because attacks are evolving and becoming more sophisticated as time progresses.

Some organizations might follow a reactive approach by responding to vulnerabilities only when they are reported. This is the most dangerous model, because the “Bad Guys” will conceal any successful attacks against your products as long as possible to exploit the weakness for years before the issue is fully resolved.

This paper will help product teams to achieve the following product security goals using the maturity model.

- Allocate resources, set aside budget and create unique titles / roles within the team.
- Identify the security engineers who play a vital role in addressing the security issues of the product.
- Identify the job responsibilities of the security engineer.
- Help product teams to fit themselves into the maturity model and to get started with product security.
- Product teams who are already addressing security in bits and pieces will follow the guidelines and implement the security maturity model explained below to identify risks/security flaws earlier in the SDL in an organized manner.
- Addressing product security throughout the product development lifecycle right from the concept phase to architecture, planning, design, and implementation phase.
- Achieve higher benchmarks of product security.

2. Understanding the Security Posture

To understand the security posture of the product, the Security Engineer and the Product Architect should come up with the security requirements.

2.1 What is Security Requirement?

Security Requirements describe functional and non-functional requirements that need to be satisfied to achieve the security attributes of a system.

When the security requirements are clearly defined at the onset of the project, it allows the development teams to trade-offs cost of baking security into a product. This will define the scope to build a strong defense in depth for the product.

At the onset of the project, the Definition of Done (DoD) criteria should make the goals of the security requirement **S.M.A.R.T**

Specific	Target a specific area for improvement
Measurable	Quantifiable or at least suggest an indicator of progress
Attainable	Requirements should be achievable by developing the required abilities and skills
Realistic	Requirements should be realistically achievable
Timely	Requirements should be grounded within a time frame

DoD is a minimum set of tasks that need to be defined and executed to mark the status of security requirement as completed. Depending on the DoD criteria, the security requirement can be classified as one time activity or a continuous activity.

One Time Activity: A requirement that is implemented once can be verified and marked as completed according to the DoD. Example: A strong password.

Continuous Activity: A requirement that has a DoD but is a continuous activity that needs to be executed for every build drop or at every release milestones. This type of requirement is important because vulnerabilities are introduced as and when new codes are checked in. For example, running vulnerability scans and performing code static analysis on every build that is delivered to QA for testing.

2.1.1 Examples of Security Requirements

The following are some of the most applicable security requirements that can be classified as one time or continuous activity. The details of these security requirements are explained in the maturity model below. Security requirements can vary based on the user and product requirements.

Sl. No	Security Requirement	One Time	Continuous
1	Resource/Headcount Identification & Budgeting	✓	
2	Mandatory Trainings/Certification	✓	✓
3	Product Hardening		✓
4	Secure Code Standards		✓
5	Code Review		✓
6	Vulnerability Assessment	✓	✓
7	Static Analysis		✓
8	Open Source / Third Party Library Review	✓	✓
9	Compliance	✓	✓
10	Secure Storage of sensitive data	✓	✓
11	Common Vulnerability Scoring System (CVSS)		✓
12	External reported security flaws		✓
13	Security Bulletins and KnowledgeBase articles	✓	
14	Secure Functional Requirements		✓
15	Cryptographic Support Requirement		✓
16	Least privilege implementation	✓	✓
17	Threat Modeling	✓	✓
18	Secure Architecture Review	✓	✓
19	Product Fuzzing		✓
20	Reverse-Engineering		✓
21	Writing Exploits		✓
22	Security Test Plan	✓	✓

In the above table, some of the requirements are tagged as both one time and continuous activities. For example, the Security Test Plan. This is because as the product matures and the product team moves up the security maturity model, the requirements need to be re-visited, reviewed and the DoD criteria needs to be updated.

The finalized security requirements need to be reviewed with the product team by asking the following questions:

- Are we doing the right things to address the product security?
- Have the right set of tools and people been identified to accomplish these security requirements according to the DoD?
- Are these security goals S.M.A.R.T?
- Will the DoD of the requirement help to move up the product security maturity model by setting higher benchmarks for security?
- Is the product team confident that the product will become more secured with the identified security requirements?
- After executing all security requirements, is the product team confident that there will be a drastic reduction of security issues in the product?
- Is the approach reactive or proactive?
- Will the security requirements help to identify the risks found earlier in the SDL?
- Will the security requirements help the product to be one step ahead of the “Bad Guys”?
- Do the security requirements protect the product against the latest attacks?
- You can find good security requirement checklists in the web. For example, CERN security checklist for developers.

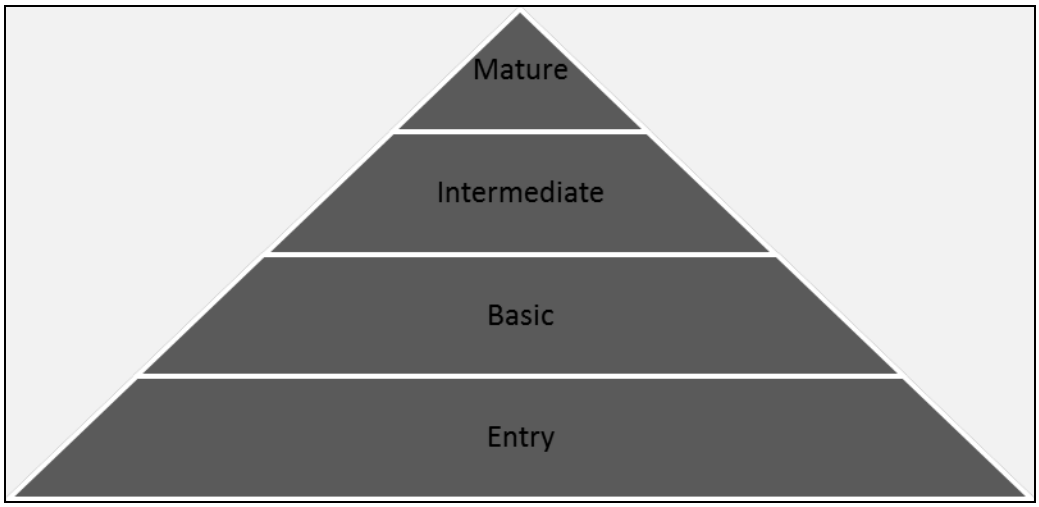
2.1.2 Responsibilities of the Security Engineer

The Security Engineer is the one who is a single point of contact for all security-related issues. When it comes to product security, the Security Engineer is the eyes and ears for the team.

- Work with the Product Architect to identify security requirements.
- Prioritize, plan, and execute all security requirements.
- Convert the identified security issues or flaws to user stories and update the project backlog.
- Prioritize all security user stories in project backlog and work closely with the product team to complete them.
- Conduct periodic security reviews.
- Assess and confirm the reported security issues and analyze the risk involved by doing the CVSS scoring.
- Create Security Bulletins for all reported security issues that have a CVSS score 4.0 and later.
- Create KnowledgeBase articles for any security issue having CVSS score lesser than 4.0.
- Review and update the Threat model for missing attack vectors.
- Be the eyes, ears, and advocate of security by becoming an influencing Security Evangelist within the team and the business unit.
- Participate in all security-related meetings and discussions.
- Address all security-related escalations.

3. Product Security Maturity Model

The product security maturity model level can be classified into four groups.



The approach followed in this paper is to start with the requirements that are easy to execute and that helps to get started with security. As you gain experience and become an expert in security it gets into your DNA. Then you move up the product security maturity model.

This paper focuses on achieving goals at the lower levels, then move up to the higher levels of maturity model to achieve tougher goals. This progressive approach will help the team as a whole to mature in product security. If the team works at all levels simultaneously, it becomes difficult to rate and position the team in terms of the maturity level of their product security.

Complex requirements can be broken down into two parts namely basic and complex parts, and it can be addressed across the maturity levels. The basic part can take care of implementing the basic security at the lower levels. As expertise increases the complex part of the requirement can be implemented at the higher levels. For example, simple fuzzing and basic SSL support can be implemented at lower levels and the complex part is implemented at higher levels.

Depending on the nature of the requirement, it needs to be executed by:

- The Security Engineer alone (For example: doing a third-party library audit, performing vulnerability assessment, etc.)
- An individual engineer (For example: secure coding practices by each development engineer)
- All members of the product team (For example: code review for security)

To assess the progress of each requirement, the Security Engineer must conduct regular review meetings with the product team throughout the development lifecycle. The meeting objective should be to track the progress made on each requirement by rating them as:

- None
- Initial
- Basic
- Acceptable
- Mature

For the detailed parameters to measure these levels, see, **Appendix**.

Rating security requirement with these levels will help to achieve:

- Comfort level of the product team in understanding and executing the requirements.
- Overall progress made on each requirement.
- Positioning the product into the product security maturity model.
- Create room for improvement and optimization of the requirements.

3.1 Security Levels

3.1.1 Entry Level

At the entry level, the following tasks need to be executed to lay a strong foundation for the product security.

- Separate resources are allocated to address product security. For example
  - Budget approval for Security Engineer headcount
  - Budget allocated for security training for creating security awareness
  - Budget allocated to procure security tools

- The Security Engineer is identified for each product team. The mid and junior security engineers who will assist the Security Engineer can also be identified.

- Mandatory Security Trainings / Certificate: Arrange for an in-house or online training on security. Depending on the job responsibilities, mandatory training / certification needs to be assigned to the team including the Security Engineer to improve their skills and knowledge on security.

  The following are some of the trainings and certifications:

  - OWASP Top 10 vulnerabilities
  - WASC Threat Classification
  - Secure coding practices
  - Secure coding in Java / C / C++ and other languages
  - Practice sessions on WebGoat tool
  - Ethical Hacking (CEH)
  - Metasploit Certification

- The Security Engineer conducts a brainstorming session with the product team to identify all possible attack vectors. Some of the points to focus in this meeting are:
  - Enumerate all possible entry points to the system.
  - Enumerate all possible threats for the system and its intended deployment.
  - Enumerate all possible threat agents who would show interest in the system.
  - Enumerate the objective/goals of each of these threat agents.
  - Enumerate the typical attack methods of the threat agents.
  - Intersect attack methods against the entry points to get the set of attack surfaces.
  - List all existing security controls (mitigations) for each of the attack surfaces.
  - Filter the threat agents who have no attack surfaces exposed to their typical methods.
  - Filter all attack surfaces for which there is sufficient existing protection.

  This exercise will help the team to focus on security, and the Security Engineer can use this data while creating security requirements. This exercise needs to be done at all levels of the maturity model.

- The Security Engineer, the Product Architect, and the product team altogether need to harvest the security requirements for attack surfaces for which there is no sufficient mitigation.
- Segregate all harvested security requirements and integrate them into the product security maturity model which is explained in detail below.

3.1.2 Basic Level

The foundation is set after making it through the entry level. The next level in the maturity model is the basic level. At this level, the product team will have to work on the following set of security requirements which will help to uncover most of the low hanging security defects with little knowledge of security. At this level it's all about learning security and getting hands-on experience.

3.1.2.1 Product Hardening

Product hardening is a process of securing a system by reducing its surface of vulnerability by deploying and operating products in a secure manner. Reducing available attack surface typically involves the following:

- Run processes and services with the least privileges and not with the system or the root privileges.
- Protect the backup files with password.
- Authenticate all communications and data transactions between the product modules.
- Limit the data access of all tenant users to their own tenant scope in a Multi-Tenant mode.
- Review the default configuration and make sure that only the whitelisted ports and services are allowed to run.
- Uninstall the unwanted applications from the base operating system which is used for product installation.
- Review the third-party modules configuration and disable the defaults.

3.1.2.2 Secure Coding Standards

This requirement suggests following secure coding practices and coding guidelines. The mandatory trainings completed at the Entry Level will help the developers when implementing key takeaways from the security training. These best practices will take care of most of the input validation and other vulnerabilities at the development phase.

3.1.2.3 Code Review

All code needs to be reviewed for security. Teams can start doing peer code review then move on to a centralized tool which will allow multiple team members to participate in security code reviews. This in turn will help in early security defect detection.

3.1.2.4 Vulnerability Assessment

Vulnerability scanners need to be evaluated and procured. These scanners will help to uncover the basic security defects. Running these scanners does not need any expertise but clearing off the noise and false positives from the full report needs some basic understanding of security.

Burp Suite is a good web vulnerability scanner to identify the top 10 OWASP vulnerabilities. Some of the other top web vulnerability scanners are Acunetix, Rapid 7, Retina, Cenzic, McAfee Vulnerability scanner and OWASP ZAP proxy.

3.1.2.5 Static Analysis

There are many tools that you can use for static analysis. The tool selection depends on the coding language used in the product.

- Coverity is a good static analysis tool which can be integrated into the build system to run static analysis on every build. There are two components of coverity:
  - Server component – Runs analysis against every build delivered to QA.
  - Client component – Used by development teams to check if the code is written securely before committing the code to the configuration management system.
- FindBugs is another static analysis tool which can be used to identify security issues in Java.
- Checkmarx is another code analysis tool which supports multiple coding languages.

3.1.2.6 Open Source / Third-Party Library Review

Recent vulnerabilities such as Heartbleed, Freak Attack, and Logjam have increased the importance of conducting third-party library review. All third-party libraries that are used by the product must go through the security review process of known vulnerabilities. This review process calls for a proactive exercise of upgrading/updating all legacy libraries with new and safer versions of the libraries.

For example, upgrading the OpenSSL, Apache, and tomcat libraries which are of interest to the “Bad Guys”.

OWASP Dependency Checker is a good tool to find known vulnerabilities in third-party libraries.

3.1.2.7 Compliance

EU/PII Data protection, FFIEC, HIPAA, ISO 2700x, PCI, NCUA, FIPS compliance will help to embed security into the product from compliance point of view. But the maturity model should never be compliance driven. It should be one of the important requirements which need to be achieved in the product security model.

3.1.2.8 Secure Storage of Sensitive Data

All sensitive data collected or generated by the product module must be stored securely. Following are some of the pointers to secure the sensitive data.

- No hardcoding of credentials.
- Credentials stored in DB or system files should be encrypted.
- Data logged into log files should not contain sensitive information in clear text, it should be masked.
- Passwords or passphrases used to protect sensitive information should be non-static and unpredictable. It should be randomly generated and recycled periodically.

3.1.2.9 CVSS Scoring

CVSS scoring needs to be validated for all security flaws reported on the product using the CVSS calculator. CVSS scoring also needs to be done for security flaws reported from the field to reassess the risk because this score can either be inflated or kept low by the researchers or hackers. If a vulnerability is reported on any third-party library that is used by the product, the reported CVSS score needs to be re-validated. The risk introduced by these vulnerable libraries can either go up or come down depending on the implementation, impact, exposure, and distribution.

The following guidelines can be followed to address the scope of fixing the security flaws:

Risk Type	Score	Required Action
Critical	8.5 > 10.0	Must be fixed immediately.
High	7.0 > 8.4	Must be fixed within a week.
Medium	4.0 >6.9	Must be fixed within a month.
Low	0.0 >3.9	Must be fixed in the next major version.

3.1.2.10 External Reported Security Flaws

The Security Engineer should respond to security flaws reported from external sources by validating the CVSS score of the reported vulnerabilities as described in the CVSS scoring section and perform risk reassessment. This assessment will either increase or decrease the risk based on the following.

- **Not Vulnerable** — Risk or Impact is low. [Risk Score - None]
- **Vulnerable** — High Risk due to usage of vulnerable function / code. [Risk Score - Critical, High]

- **Vulnerable, but Not Exploitable** — Medium Risk since no usage of vulnerable code/function but flagged by vulnerability scanners due to version match. [Risk Score – Medium]
- **Vulnerable, but Low Risk** — Low Risk due to less impact, exposure and distribution. [Risk Score - Low]

Risk reassessment can create an opportunity to identify new entry points and generate new security requirement or update the existing requirements.

In some cases, responding to complex security flaws requires some level of expertise and understanding of the internal implementation of the product.

3.1.2.11 Privacy Impact

In some cases, products deployed in customer environment collect customer-related information and send it back to its manufacturer. This personally identifiable information (PII) needs to be safeguarded from the time it leaves the customers environment. Once it reaches the backend servers in the manufacturer network it needs to have certain safeguards mechanism in place, so that we know who is accessing the data. This data needs to be anonymized to the point that it's impossible to track back to the user or system on which it was generated. This end to end protection model needs to be analyzed that if the security controls were ever circumvented by the attacker what would be the most that they could gain from this data.

3.1.2.12 Security Test Plan

All the above security requirement will have to go into the product Security Test Plan which needs to be reviewed by the product architect and team and signed off.

To assess this level, the Security Engineer needs to do a risk assessment of the above mentioned requirements and complete the following tasks:

- Prioritize all the above mentioned requirements based on the assessment.
- Identify the "Minimum Shipment" requirement and executed them on priority.
- Fulfill the DoD criteria to mark the requirements as completed.
- Redo the CVSS scoring for all identified security issues by considering parameters like environment, impact, exposure, and distribution. This will either increase or decrease the risk.
- Address all critical risk flaws identified at CVSS scoring immediately followed by High, medium and low risk.
- Perform risk analysis for security flaws which demand 'Design Change' or 'Architectural Change'.
- All 'Design Change' or 'Architectural Change' high risk items need to be converted into User Stories and implemented on priority.
- Replace all third-party vulnerable libraries with safer versions.
- Prioritize the fix for vulnerabilities reported by vulnerability scanner.
- Address all high risk security flaws found in the static analysis.
- Uninstall the unwanted services and software from the system.
- Make sure that all entry points have authentication, authorization and identity management.
- Document the false positives of vulnerability scans, static analysis, and 3rd party library reviews.
- All security defects found at code review need to be immediately analyzed and addressed before code check-ins.
- All sensitive data should be stored securely.
- Mitigations or mitigation plan should be in place for identified security flaws.
- Define security controls that will build a strong defense in depth for identified attack vectors.
- Respond quickly, efficiently and accurately to all security flaws reported in the field.
- Outcome of reviews / finding / learning / are converted into enforced policies, to be followed by the product team members.
- Perform risk analysis of all identified security limitation and document them.
- Updated Security Test Plan with new findings.

At this stage, the Security Engineer and the product team get a fair idea about security and different types of attacks to which the product is exposed to when input validation and input data sanitization is not taken into consideration.

The vulnerability scanners and the static analysis tools will help to identify most of the input validation flaws. It is at this stage the product will be able to handle most of the input validation attacks such as, XSS, SQL Injection, XML injection, CSRF attack, Malicious File upload attack, Client-Side Validation Flaw, Buffer Overflow, or Cookie poisoning.

3.1.3 Intermediate Level

With the Security Engineer advocating secure coding practices and the development team following the security patterns during the development cycle, the product team is now ready and moves to the Intermediate level of the product security maturity model.

At this Level the Security Engineer and the product team should start working on the following security requirements.

3.1.3.1 Secure Functional Requirements

Every functional requirement created during the product development should have a security component in it to address the security concerns of the feature. The Security Engineer has to be a part of every design and planning meetings to advocate product security and to identify security design flaws.

3.1.3.2 Cryptographic Support Requirement

In this review the following areas need to be addressed

- Use strong ciphers for SSL communication and avoid weak ciphers like null & export ciphers.
- Use SHA2 for certificate generation.
- Implement certificate revocation module.
- Implement certificate verification module.
- At minimum, keys size should be >= 2048 bit.
- No hardcoded keys should be used in the encryption module
- Recycle Crypto keys periodically.
- On failure, SSL module should fail securely and should not fall back to plain text.

3.1.3.3 Least Privilege Implementation

Least privilege implementation helps to create roles and permission to implement the least and the required amount of privileges to get all functionalities are working as expected.

Data classification is implemented in this requirement so that the data collected or generated by the system is classified and put into buckets that have different levels of privileges assigned to them.

3.1.3.4 Threat Modeling

Threat Modeling is an organized procedure to optimize the security of the product by identifying security objectives and vulnerabilities, and then defining countermeasures to prevent or mitigate the effects of threats.

Threat modeling requires the creation of Data Flow Diagrams (DFD) by the Security Engineer and reviewed by the Product Architects. It is good to have threat modeling done at the Intermediate product security maturity level because at this stage the security engineer will have a clear idea of security and internals of product implementation. This will help in creation of accurate DFD’s and effective Threat Model.

3.1.3.5 Secure Architecture Review

Product architecture needs to go through regular cycles of security review. With the ever-evolving attack patterns, the existing defense can fall weak. Hence all the existing input validation controls need to be reviewed and fixed to handle the new attack vectors.

The product needs to be dissected into different layers and new security controls need to be integrated at each layers if not present. This will ensure multiple levels of defense system in the product. Hence if the attacker circumvents one line of defense, he still needs to break the next level of the defense.

The possible layers are

- **Client side** - Here mostly the security control will be in the form of JavaScript to restrict malicious inputs.
- **Server side** - Depending on the technology and design, there can be multiple layers and security controls you need to place at each of these layers. For example, the 'parameterized SQL query' takes care of SQL Injection and escaping functions which will take care of XSS and SQL Injection.
- **Database side** - Security Controls need to be placed to validate the type and size of data before storing in the database. It will also validate the privileges of the user performing the operation.

3.1.3.6 Security Test Plan

All the above security requirements are subjected to risk analysis, prioritization and ranking, and need to be updated into the Security Test Plan.

All issues that are uncovered through above mentioned security requirements need to get converted into User Stories or security backlog. This security backlog needs to be cleared to move to the mature level of product security maturity model. The Security Engineer needs to assess this level as he did in the Basic level by rating the executed security requirements.

3.1.4 Mature Level

At the mature level of the maturity model the Security Engineer and the product team has a good understanding of product security and the product is clean from all low hanging security issues. It also can handle most of the attacks such as OWASP Top 10, DoS attacks, SSL attacks etc. With in-depth understanding of product security, the security engineer now starts executing the following security requirements.

3.1.4.1 Product Fuzzing

Fuzzing is a negative testing which involves throwing all possible types of data at the identified entry points to find security flaws.

Fuzzing can be done in two ways:

- **Dump fuzzing**: The fuzzing tool does the job of throwing random data at the entry points of the product and it is done without understanding the structure of the data the product consumes.
- **Smart Fuzzing**: This type of fuzzing needs in-depth understanding of the data format consumed by the services/processes. For example, Peach Fuzzer requires expertise in PIT file creation. When using the PIT files, the user should be able to control the fuzzing at any layer of the product. The other good fuzzing tool is Codenomicon fuzzer.

3.1.4.2 Reverse-Engineering

This requirement will help to determine if attackers can reverse engineer the product to bypass licenses validation, crack password, establishing rogue entry into the system, or retrieve sensitive data by circumventing the existing security controls and counter measures.

3.1.4.3 Writing Exploit

Executing this requirement requires a lot of expertise and deep understanding of the system and years of experience in writing exploits and experience of tools like Metasploit. It will need the Security Engineer to identify a vulnerability which is exploitable, then execute the malicious payload on the victim system to remotely control it.

4. Conclusion

This product security maturity model helps organizations to proceed with the four levels of maturity management and defines a roadmap from the current state of product security to its desired state. At each level of the maturity model, the team will be able to demonstrate measurable results by rating the requirements. The defined proactive approach will help to define and achieve milestones focused on reducing the product risk and helps to identify the risks earlier in the SDL.

This maturity model will build multiple defense layers in the product and make it extremely difficult for the Bad Guys to break through. The effectiveness of this model can be seen at every security level because the team addresses security earlier in the development stage. Over the period of time, the security defect rate can reach to near-zero bug.

Appendix

Security requirement rating parameters examples

	Resources Budgeting	**Static Analysis**
1 - None	Not properly resourced	Use no Static analysis tools or use compiler flags.
2 - Initial	Security engineer headcount is approved	Static analysis done before the code commit
3 - Basic	Security Engineer is identified and he is briefed on his job responsibilities	Static analysis runs automatically with builds
4 - Acceptable	Security Engineer actively works on his job responsibilities	Product is analyzed frequently; defect rate is decreasing; reported findings are triaged
5 - Mature	Have a seasoned Product Security Architect (PSA) dedicated for each product	Fixed the defects; Enable the required security checks to reduce noise; Real defect rate is near to zero (0)
	Mandatory Security Trainings / Certification	**Fuzz Testing**
1 - None	Have little or no product security training	Fuzz testing not performed
2 - Initial	Have identified and use free product security courses	Free tools are used
3 - Basic	Mandatory security courses identified; Assigned training completed	Security Engineer / Team is trained on fuzzing tools (E.g. Peach and Codenomicon)
4 - Acceptable	Major topics are delivered and internal processes are covered	All products are fuzzed frequently; new custom scripts are created;
5 - Mature	Keep program current and have internal Subject Matter Expert (SME)	Controlled fuzzing gives ability to fuzz at different layers of the product.
	Security Tools	**Secure Coding Standards**
1 - None	Have little or no security tools are identified	No secured coding standards
2 - Initial	Have identified and used free security tools	Aware of standards but occasional adherence
3 - Basic	Procured the required licensed security tools and apply them appropriately.	Adopted appropriate standards
4 - Acceptable	Tools are integrated into the build system and to run on regular basis	Following adopted standards
5 - Mature	Custom security tools developed to close the identified gaps	Standards are integrated into manual code reviews and static analysis
	Product Hardening	**Open Source / Third Party Libraries**
1 – None	Product running with the default configuration, unwanted software's and services.	Selected by any engineer; used with no constraints
2 – Initial	Product is reviewed for open ports, default configuration, unwanted software's etc.	Manually maintain lists of used binaries
3 – Basic	Product hardening exercise is completed and the security issues are addressed	Run inventory tools (BlackDuck) and tools like dependency checker
4 - Acceptable	Every new component that is added to the system is reviewed.	Fully maintaining all documented Third party libraries; Replace vulnerable Libraries with safer versions.

5 – Mature	Any change to the system can open an entry point and it must be reviewed and approved by the Security Engineer and Product Security Architect (PSA)	All Open Source / Third Party Libraries integrated and used securely.
	Vulnerability Scans / Penetration Testing	**Code Review**
1 – None	Escalations from customers	Ad-hoc; Without security patterns in mind
2 – Initial	Vulnerability scanners / tools are evaluated, identified, and used by Security Engineer	Aware of standards, occasional adherence
3 – Basic	Vulnerability scans are performed regularly and defects are analyzed	Conducted on risky new code by multiple engineers
4 - Acceptable	Pentesting expert available; defects are addressed; findings are used to improve early defect finding	Conducted on all potentially risky code using a shared tool
5 – Mature	Every release is pentested by an external pentesters; defects are fixed before the product launch	Conducted regularly using a code sharing collaboration tool (e.g. SmartBear Collaborator)

References

Examples of the big data breaches:

- http://www.forbes.com/sites/moneybuilder/2015/01/13/the-big-data-breaches-of-2014/
- http://www.networkworld.com/article/2864382/microsoft-subnet/hackers-released-xbox-one-sdk-claimed-unreleased-games-may-be-leaked.html
- https://www.opm.gov/cybersecurity/

Security Literature:

- https://tysonmax20042003.wordpress.com/tag/types-of-exploits/

Definition of Done:

- http://scrumguides.org/scrum-guide.html#artifact-transparency-done

Defense in depth:

- https://en.wikipedia.org/wiki/Defense_in_depth_(computing)

Common Vulnerability Scoring System:

- https://www.first.org/cvss/calculator/3.0
- https://nvd.nist.gov/cvss.cfm?calculator&version=2

Examples of Security Requirements:

- https://security.web.cern.ch/security/recommendations/en/checklist_for_coders.shtml

Training / Certification / Tools:

- https://www.owasp.org
- http://projects.webappsec.org/f/WASC-TC-v2_0.pdf
- http://www.eccouncil.org/Certification/certified-ethical-hacker
- https://www.offensive-security.com/metasploit-unleashed-training/metasploit-unleashed-mastering-the-framework/

Examples of Threat Agents and Methods:

- https://ics-cert.us-cert.gov/content/cyber-threat-source-descriptions
- http://www.lovemytool.com/files/vulnerabilities-threats-and-attacks-chapter-one-7.pdf

Integration testing among enterprise security products

Jeyasekar Marimuthu

Jeyasekar.m@gmail.com

Abstract

An ecosystem that consists of millions of managed nodes or internet of things (IoT) that communicate with each other will require diverse integration testing before the system is shipped to customers. The products and nodes that interact with each other are truly heterogeneous. They have few things in common to communicate between them.

Writing integration tests between products of heterogeneous nature needs a modularized approach that can isolate technologies, platforms the products run on. The runtime objects will have many uncommon properties between each other. An integration test framework should be able to handle the uncommon characteristics and validate the integration tests.

This paper highlights the challenges faced in such environments and some of the integration techniques developed to address those challenges. One of the testing techniques is to have developed an integration interface that hides the underlying native tests run on individual system and run the integration tests seamlessly across the enterprise. Some of the tests are SDK based and some are REST based. The paper also reviews some of the tools and techniques used in complex environments. Adapting to development process like Behavior Driven Development (BDD) saves significant amount of time to setup complex environments.

Biography

Jeyasekar Marimuthu is a lead software developer at Intel Security, Hillsboro with comprehensive experience in architecting and developing solutions, project execution, process improvement, cross product development, and client relations. His strengths are: understanding business requirements, evaluating risks, providing architecture and designs, and strategizing profitable execution. A skilled developer and technical leader, Jeyasekar communicates with management, vendors, team members, and staff at all skill levels. He brings innovation with ideas and drives improvements in development and processes. He has 15 years of development experience in enterprise and security platforms using Java, J2EE, database technologies.

.

1 Introduction

An enterprise that wants to protect its assets (work stations, servers and each device attached to the network) from security threats will have to depend on a number of interconnected security solutions across the eco-system.

In such environments, a large enterprise will normally consist of millions of nodes or assets that exchange data are unified through a common data exchange framework. The node data will be processed by each point product and will have to exchange with other products. The verification tests and technologies used to write the tests need not be same between one to other.

Before each point product is readied for integration tests, it is tested with a number of feature verification tests. When there is a need to exchange data among two or more products, the system as a whole needs to be tested with integration tests that can ensure integrity of data.

A key concern when writing integration tests is that each point product deals with a different test framework. The underlying technology between products is not the same. The datatypes used among products to exchange info are heterogeneous in nature. The runtime objects are complex. This paper explains how to address incompatibilities between products and how to perform integration tests seamlessly across products.

Let us look at how this might work.

2 Background

A security ecosystem within IOT (The Internet of Things) is the network of a wide array of physical objects embedded with electronics, software, sensors that offers connectivity to achieve greater value and service. An Ecosystem consists of one or more security products managing data from many (“millions”) of nodes across the network exchanging data between them. A large volume of data is sent to enterprise products through data channels. An enterprise might deploy a number of security products together. Each one of them will have its own kind of tests developed by its respective engineering team. Each security service should be testable under its own test framework. Tests that qualify each point product to production-ready are FVTs (Functional Verification Vests).

The next phase of testing is to integrate all of point products and to perform an integration testing. When the products are unified and deployed in a common operating environment, each set of tests written for system ‘A’ need not be compatible with system ‘B’ due to various reasons. The reasons could be due to less common qualities between systems, or the scripting language used by system ‘A’ may be different from system ‘B’, or setting up an environment with all products may be a challenging through automation.

Any compatibility issue mentioned above will challenge writing and executing system tests. There is a need to address the incompatibilities between products and a need to develop an interface that can execute FVTs and system tests seamlessly. Without such an interface, it will be next to impossible to convert the test object of one system into another using a different scripting language.

3 Enterprise Ecosystem Architecture

Each point product may host one or more security services. A node in an enterprise network can leverage service from one or more security products. The data exchanged between subsystems or point products needs validation before a point product sends a request to use services from other point products. The

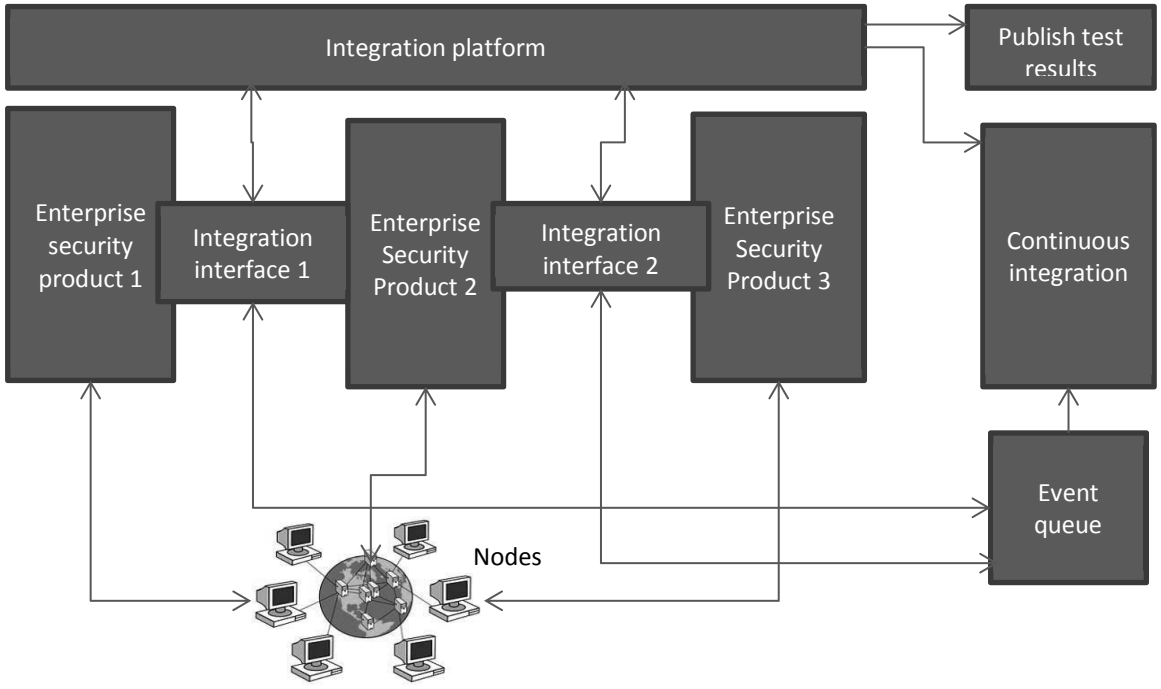
following sequence of automation ensures the testability of a point product in standalone mode.

1. Ensure each service or function point has sufficient unit tests.
2. BVTs (Build Verification Tests) are present.

3. Whenever there are build changes BVTs are run by a continuous integration server like Teamcity.

The following diagram explains how an integration test platform should be. The integration framework consists of the following components.

1. Nodes or assets that are linked to the enterprise network. Each network may have a number of subnets.
2. One or more security products that consume the data collected from loads of assets or nodes.
3. An interface that bridges one product with other. Depending on the number of interconnections, the number of interfaces will vary.
4. An integration platform that contains platform independent abstract definitions for various interfaces.
5. An event queue that aggregates all the runtime test results and queues them as events.
6. A continuous integration server that consumes the events from event queue and runs the integration tests.
7. A management console that publishes the integration test results.



4 Integration Tests In Security Ecosystems

Integration testing (sometimes called integration and testing) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing. The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. Carrying out integration tests on an enterprise ecosystem that contains two or more point products and exchanges data between them must consider the following system design.

1. Each subsystem or point product is unique in nature
2. Unit tests of each subsystem might not have been written in the same language in which the system tests are created.
3. Understand the platform specific tests and use them for the system automation framework.
4. Data that come in from various products in asynchronous fashion.
5. Each subsystem deals with a number of client machines.
6. Each client can subscribe to one or more services across the network.

Integration between subsystems normally happens in the following way.

1. Feature Verification Tests are run on each subsystem before the results are sent for system tests.
2. Integration tests are written. Each system test deals with source and target objects.
3. The system deals with a definite size of data for tests.
4. An automated or manual system to review the results and to correct if there are issues.
5. Integration tests are executed on need basis.

5 Integration Automation Tests and Challenges

System automation tests written for one security system may not be runnable on another system due to their heterogeneous nature. Simulation of a system and mocking the behavior is key to automating system tests. Upon successful mocking, the test system should make sure the tests are run seamlessly from system to system by dealing with the technology crossovers.

Some of the key challenges are listed here.

1. Integration tests are written and each integration system deals with logic that converts the original call into a native call that the targeted system can understand. This consumes a lot of time to complete all integration tests. A chunk of redundant code exists in each integration test.
2. Tight coupling between the source and target systems.
3. Handling varying data size from time to time and lack of performance loops in the automation cycle.
4. Executing integration tests on need basis. They are not continuously built within build room.
5. No coordination among point products builds. The FVT and integration cycles are independent in nature.
6. Adapting to tests that are written using unfamiliar scripting language and the time to learn the targeted platform and to convert tests for the same.
7. A significant learning curve to mock objects for targeted systems.
8. Challenges in setting up the environment by automation scripts.

6 How to fix?

This model simplifies integration automation testing.

1. We do not intend to learn and write tests for each subsystem.
2. We do not intend to change the underlying language that the system tests are written in.
3. The system tests should make use of the existing unit tests from each subsystem.
4. The integration system should be horizontally scalable as it needs to deal with more security services.
5. Results obtained from subsystems should be unifiable and should produce one collective report
6. Tests should be integrated into a continuous integration build system.

The solution to this scenario is as follows.

Write an adapter for each subsystem that can receive instructions from the system automation framework and can convert them into native subsystem calls.

1. Use Behavior Driven Development (BDD) for setting up environments. This reduces the learning curve for automation engineers. It allows setting up multiple test environments with multiple configurations. It simplifies traceability between stories and test scenarios.
2. Use BDD for feature injection.
3. Define request and response objects that follow adapter patterns and work bidirectional.
4. Create mock objects according to the generic object model and let the adapter convert them into the targeted system object in runtime. For example, Source system A has a number of Python tests and Target B has a number of Java tests. Writing an adapter that accepts python calls and invoke a java tests with appropriate arguments saves the integration tester's effort significantly. When the adapter is prototyped, it makes anyone job further easy.
5. Achieve seamless integration into any number of subsystem by just writing an adapter rather writing a number of native redundant tests.
6. Feed the results into the performance test system. This paper does not focus on accomplishing performance tests.
7. Integrate tests into continuous integration loops using build servers like Teamcity. The Teamcity builds will do the following.
    a. Run FVTs on each subsystem nightly
    b. Create runtime environments using BDD (say using Gherkin)
    c. Deploy builds on the virtual environment and trigger system tests
    d. The Adapters in the integration framework invoke the appropriate native tests as part of the Teamcity build.
    e. Publish results from the build system.

7 Advantages

1. No expertise needed to configure environments: When BDD is used, the framework does not demand every tester to write the underlying scripts.
2. Seamless integration: When an interface is written to integrate any source and destination system, the integration tester is going to focus only on writing in the language that the system is built with. He/she can just plug and play the tests for the target platform.
3. Cost saving: When there is time saving, it reflects in costs as well.
4. Scalable: As there could be more tests added dynamically using integration framework, addition of tests prove to be scalable.
5. Focus remains only on writing integration tests. Configuring and learning underlying technologies are not required.

8 Conclusion

Developing an integration framework with one or more interfaces can seamlessly deal with multiple security point products. When the complexity of environments grow, it is recommended to go with tools that use Behavior Driven Development. It can simplify the task of writing many lines of code. Also, it allows creating multiple virtual environments with diverse settings as the products require.

The defined model works very well for enterprise setups that deal with few million nodes. When the network is expanded with more IOT devices, the heterogeneous nature of systems and the size of data will expand unimaginably. In order to address such needs, the integration framework will need to be revisited according to growing needs.

References

- https://en.wikipedia.org/wiki/Internet_of_Things

"Internet of Things." *Wikipedia*. Wikimedia Foundation, 03 Aug. 2015.

- https://en.wikipedia.org/wiki/Integration_testing

"Integration Testing." *Wikipedia*. Wikimedia Foundation, 26 May. 2015.

- https://pythonhosted.org/behave/philosophy.html

"Behavior Driven Development." *Behave 1.2.5 Documentation*. 07 Aug. 2015.

Web Application Security – What You Need to Know

Bhushan B. Gupta
Gupta Consulting, LLC
bhushan.gupta@comcast.net

Abstract

There have been some significant web security breaches in “Corporate America” Sony, Target, and Home Depot, to name a few. Such breaches not only impact corporations financially, they also tarnish the brand image. The customers loyal to the corporations lose their confidence in their private data security and take their business to safer pastures, thereby financially impacting the business. Both corporate America and Government agencies are working towards controlling cyber security threats.

This paper is focused on raising awareness about Web application security. It starts with an exposure to the fundamentals of Web security (Vulnerability, Threat, Risk, Exposure, and Controls), discusses control types, and touches on the principles of the Zachman Architecture and The Open Group Architecture Framework. It then dwells into the “Top 10” OWASP (Open Web Application Security Project) most critical Web application threats including, SQL Injection, XSS (Cross Site Scripting), and CSRF (Cross Site Request Forgery). Furthermore, it provides some approaches to mitigate risks to make a Web application more secure.

The paper is a self-study of Web Security and is intended to provide the necessary information to raise audience awareness.

Biography

Bhushan Gupta has 30 years of experience in software engineering, 20 of which have been in the software industry. For the past several years, Bhushan has been involved with agile processes, quality methods and metrics, and general process improvements and now with the cyber security. Bhushan has been a presenter and a reviewer for PNSQC for several years and has also presented at other conferences. As a change agent, Bhushan volunteers his time and energy for organizations that promote software quality.

Bhushan Gupta has a MS degree in Computer Science from New Mexico Institute of Mining and Technology, Socorro, New Mexico, 1985.

Copyright Bhushan Gupta, October, 2015

1 Introduction

The data breach at Target Corporation, the largest retail hack in US history, exposed 70 million records with the customer information and some 40 million stolen records with credit card information [Reuters, 2014]. Another security breach labeled as worst security breach of the year 2014 [NetworkWorld, 2014], was at Sony Pictures Entertainment. The attackers obtained access to documents, emails, and movies yet to be released. This attack also caused some political backlash since the allegations seems to have suggested that the attack originated from North Korea. The other notable attacks that took place in 2014 were Home Depot and JPMorgan Chase. There have been fewer security breaches at the Federal Govt. level but were of similar magnitude.

Clearly, cyber space has been invaded by the “Bad Guys” and our security and privacy has been threatened. The Web security of big corporations, and the Federal Govt. has been compromised, and the consumer has suffered identity theft as well as loss of credit card information. Web security is becoming increasingly important and to protect ourselves, we must have a sound understanding of security and how breaches are manifested.

2 Basic Elements of Security

A secure system should provide Availability, Integrity, and Confidentiality of critical assets; also known as the Security Triad. Availability refers to reliable and timely access to the data. Integrity is the data protection against unauthorized modification. Finally, confidentiality is assuring that the data is not disclosed inappropriately as it moves around in its environment and finally stored in its destined source. The five basic elements of security are, Vulnerability, Threat, Risk, Exposure, and Control [Harris, 2013]. Each of these are briefly described in this section.

2.1 Vulnerability

Vulnerability is the extent of unauthorized exposure of your valuable assets in combination with the countermeasures that are in place. A vulnerability may be a process in the system, a table that contains the system data, a database table, or a web service running on your server. A high degree of exposure may not be a serious problem if you have countermeasures in place. A simple analogy would be, keeping your house door wide open but guarding it.

2.2 Threat

Threat is the potential of exploiting the vulnerability by an agent and could be external or internal. An intended activity or mistake by someone working with the data is also a threat.

2.3 Risk

Risk is likelihood of exploiting the system vulnerabilities and causing negative impact on business. The risk can be made up of multiple factors such as an inadequately configured firewall, an undocumented critical manual system process, or an untrained system administrator.

2.4 Exposure

An instant of exploiting a vulnerability and thus causing a negative business impact.

2.5 Control

A control is a mechanism to reduce the potential risk. Control could be preventive, detective, corrective, deterrent, and compensating. An example of a compensating control is buying an insurance policy against the damage to your house.

2.6 Security Standards

In the early days when there were no standards for security, British Standard 7799 (BS7799) became a de facto standard but no standard body was enforcing it. The standard laid out how security should cover a wide variety of topics. The BS7799 was expanded by International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) which established ISO/IEC 2700X where X took a digit representing a necessary component of the security systems. For example, ISO/IEC 27000 provides Overview and Vocabulary and ISO/IEC 27001 provides standard for security requirements.

3 Access Control

While all the elements listed in section 2 are important, the exposure is actual incidence in which the security is compromised using an unauthorized access. The following discussion highlights the necessary elements of access control.

3.1 Identification

Identification is the process of uniquely recognizing an entity before letting it use the system. In most cases it is a unique string of characters such as a name, an email address, or an account number. In the simplest scenario the security involves verifying the string with the access control list (ACL). The methods to assure that the string is not programmatically generated are now emerging, enter a system displayed text, where the consumer has to perform an additional task to prove the physical existence.

3.2 Authentication

Authentication is the process of assuring that a user has successfully proven to be a legitimate entity to utilize the system. Authentication is based on three aspects – something specific you know, something specific you have, and a characteristic unique to you. Something specific you know is normally a password the user has setup. The password is sometimes reinforced with a set of questions and answers, which also is a form of something you know. Something you have can be a string of characters randomly generated by a hardware device such as a pin or a physical object, a card that you swipe to get access. A physical characteristic unique to you is a biometric aspect such as your figure print or cornea. A strong authentication should include at least two of these three factors.

3.3 Authorization

Not every user needs access to all system resources, programs, processes, or data. The access to these resources should be controlled by an access criteria based upon the security policy. These criteria may vary from “No Access” as a default to “Need to Know”.

3.4 Audit

A sound security system must include an audit log of failed and successful access attempts. Repeated unsuccessful access attempts are of particular interest to avoid a potential security breach. The successful attempts are of critical importance if a breach has occurred to perform a detective analysis.

4 Web Application Security Threat Scenarios

A Web application is a software program that runs in a browser on the client machine [Wikipedia, 2013]. It is a convenient way to provide access to numerous clients without installing it on client systems. Since these programs run on the client systems the server side does not control the processes and the sessions they run under and can be a victim of injection, inadequate authentication and session management, improper exposure of data and functions, and week security configuration. On the server side there may be improper data and object exposure as a web application serves the data to the client.

Figure 1 shows the main components of a Web application environment. The engagement and the security aspects are described below for each component.

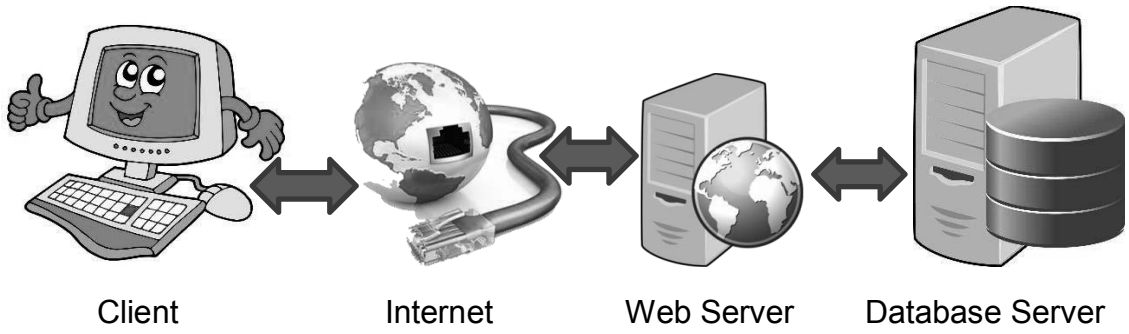


Figure 1: Web Application Environment

Client: The client system provides the run time environment for the Web application and is, therefore, responsible for managing the execution process which prominently includes, process and memory management. An application should not cause memory overflow and write data into unassigned memory. The potential security threats that arise from the scenario are remnants of code or data on the client system that can compromise the security of the client.

Internet and Web Sever: An internet networking stack has seven layers; Application, Presentation, Session, Transport, Network, Data Link, and Physical [Beal, 2015]. The session layer controls application to application communication, the communication between the client and the server, and the data transfer. The Web server manages the session ID for the multiple applications that are communicating with it. An inadequate session management can provide opportunity for an attacker to hijack a session and compromise the network security.

The transport layer establishes the protocol between the client and the server to assure the data is received in its entirety and its integrity is maintained. A security compromise at this level can cause data manipulation providing the attacker access to unauthorized data.

Software Practices: It is also meaningful to view the risk scenarios from the perspective these risk are originated. Inadequate identification and unauthorized access to processes, sessions, and data is a scenario that is internal to development. Adding to this list is also a lack of consideration of security configuration, use of known vulnerable software components, and improper exposure of sensitive data.

The Bad Guy: The attacker is always thinking about beating the system. The practice of injection, phishing, spoofing, social engineering, and fuzzing are here to stay and new once will be added to the list as the ‘Bad Guy” is always on the lookout.

5 OWASP Security Risks

The Open Web Application Security Project (OWASP) [OWASP, 2103] foundation is an independent non-profit entity mostly supported by volunteers. The OWASP foundation encourages creating application security programs that are compatible with the organization goals, culture, and technology. The OWASP Top 10 security risks were first released in 2003. This list was based on prevalence. The next major follow up release was in 2010 and the items were also based on risk and not just on prevalence. The 2013 version consolidated some items while it reprioritized others. This section describes major security risks as prioritized by OWASP in its 2013 release.

5.1 A1 – Injection

Injection is an attack technique which is common, easy to exploit, and can result in a serious security breach. In an injection an attacker normally inserts malicious data into client queries, OS commands, XML Parser, program arguments etc. [OWASP, 2013] which are executed by the server side. The most wide spread injection is SQL injection, which comes from an untrusted source such as a Web form. The malicious data entered in the Web form is used to construct a dynamic query that results into security breach. An attacker can exploit this technique to:

- Get access to data thereby compromising data confidentiality
- Modify data resulting into loss of data integrity
- Changing data authorization if stored in the database.

In its simplest form, the user can make the database throw an error by simply adding a single quote to the URL. The following example is taken form Web Application Injection Vulnerabilities [Couture, 2013] with a slight change to input. In the scenario, a Web form is requesting a name and the password from which it will create a SQL query which should like:

SELECT id FROM users WHERE username = 'Foo' AND password = 'QWERTY'

The attacker submits Username as 'OR 1=1 --/ and Password = anything. The server side will create a query by concatenating the values which looks like:

SELECT id FROM users WHERE username = 'OR 1=1 --/ AND password = 'anything'

Since the input field in this case, is not cleansed of escape characters, the double dash is interpreted by the parser that everything to right is a comment, and thus dropped. The parsed query that gets sent to the DB is: SELECT id FROM users WHERE username = '' OR 1=1

Which is interpreted as "Return all user ID's where the username is a null value, or 1=1" (which it always does). The string will always be true and thus dump all the stored user IDs.

The attacker can generate inputs like these by using automated tools like sqlmap [Moon, 2012] and can obtain a variety of information by embedding simple commands into an URL.

5.2 A2 - Broken Authentication and Session management

An article titled Survive the Deep End: PHP Security [Brady, 2014] has provided a detailed account of how risk can be attributed to insufficient transport layer security. A communication between a client browser and a Web server is maintained by a unique session using a session cookie on the client browser and the details with the Web server. It is critical that the privacy of both the client and the Web server, as well as the integrity of any data exchange, is maintained. The transport layer in the OSI stack is responsible to achieve this goal and its security is critical.

The negotiation about the encryption key between the two parties, client and the Web server, is carried out at the time of initiating a secure connection. An attacker can get in the middle and pretend to be the server thereby obtaining the encryption key. This is called the "Man in the Middle Attack" (MitM) attack. It is therefore essential that the initial negotiations happen with a trusted server that has an authentication ID. In addition, the session ID should be protected to avoid session hijacking. Brady recommends that the transport layer should be sufficiently secure between the client and the Web Server and the Web Server and a third party server and have proposed programmatic ways to achieve it. OWASP has cited some simple examples such as passing raw (un-encrypted) data in a URL, not properly logging out from a public client system, and improper hashing of database user password that can lead to this problem.

5.3 A3 – Cross Site Scripting (XSS)

Same-origin-policy is a concept of trust that states that if contents form one site are granted permission to access resources on the system then any content from that site will share those same permissions [Wikipedia, 2015]. The cross site scripting occurs when an attacker injects client side script into a web page. Due to the same origin policy, the script is considered to be from the trusted code as the Web server has already been granted permissions. If the attacker can get access to client session ID the effect may range from simple nuisance to a serious breach

There are two types of XSS attacks, non-persistent and persistent. A non-persistent attack occurs when the data provided by the client in http is immediately used by the Web server without sanitizing and an output is immediately generated. This leads to stealing of the client data. A persistent attack occurs when the non-sanitized data is stored in the database or any other locations such as blogs, forums and gets executed when other clients use the page. Here is a brief scenario of how XSS attack can take place:

User Gdguy always logs to site www.bgc.org using his login name and password and thus is a trusted user of the site. An attacker, Bdguy determines that the site www.bgc.org is vulnerable to XSS and creates a URL embedded with a JavaScript, http://www.gbc.org?q=cheaptickets<script%20src**=”http://attaker.org/steallogin.js”**>. The attacker sends this URL in an email about cheap tickets to the Gdguy. Gdguy does not pay attention to the URL and clicks on it. The JavaScript gets executed in the Gdguy's browser and steals login credentials form the session cookie. With the stolen login credentials Bdguy now logs in as a trusted user and has control of Gdguy. Using an email spam the Bdguy can take control of www.bgc.org.

Detailed examples of session hijacking for both types of XSS vulnerabilities are provided on Wikipedia [Wikipedia 2015]. For a more comprehensive discussion on XSS refer to Survive the Deep End: PHP Security [Brady, 2014].

5.4 A8 – Cross Site Request Forgery (CSRF or XSRF)

In this attack an attacker lures a victim to click on an image and send unauthorized commands to the web server that the web server trusts once the session cookie has been established. Thus, the attacker exploits the trust of the web server in the client browser. An HTML image element is crafted that attracts victim's attention and include the malicious state change request. The following example is taken from the OWASP Top 10.

<img src=http://example.com/app/transferfunds?amount=1500&destinationAccount=attackersAcct#
width=”0” height=”0” />

If the victim is already authenticated to example.com and visits any of the attacker's sites the forged request will already include the session info, authorizing the attacker's request. CSRF has not been widely reported but has been used for malicious bank activities.

5.5 A5 – Security Misconfiguration

Security misconfiguration is any part of the web server environment that is not adequately hardened with respect to security. This could be an update to the operating system or any open source or third party software, user accounts and passwords, and databases. The application developers should use and rely on a secure framework and keep it up to date. Hunt [Hunt, 2010] has discussed .Net framework scenarios specific to keeping it up to date with special aspects that include actions to take if a security flaw has been discovered in a framework, customizing error messages to not expose any system information, turn the debugging off, and encrypt sensitive configuration data.

6 Approach to Risk Mitigation

The exhaustive methods to protect yourself by building a security perimeter will not be cost effective. An organization should diligently address security based upon its goals and strategize its approach to risk mitigation to best achieve these goals. The following are some guidelines to make your environment less prone to risk.

6.1 Protect Your Most Valuable Assets First

As a basic principle, prioritize all your assets by value based upon your liability. When you are managing your customer data that might have legal consequences if compromised, it should be ranked higher on your priority list compared to the list of employees in your organization. Security breaches that involve customer data get significant public attention. The least vulnerable scenario is when there is no sensitive data to exploit for an attacker, in which case securing a Web application may not even be necessary. In the early days of the internet, the Web servers only served static pages with well thought out contents and there was no or only minimal security risk.

6.2 Information Classification

Information classification is an effective way to protect your environment internally. An organization can develop its own classification scheme or adopt one from the existing schemes depending upon it is requirements. Harris [Harris, 2013] has discussed effective access control models and methods to administer access control. The proper access control methods should be implemented so that business sensitive information is exposed only to those it is intended for.

6.3 Sound Development Processes

To deal with security today, we are using duct tape, band aids, and ointments that come in the form of firewalls, Intrusion Detection Systems (IDS), anti-virus software, and vulnerability scanners along with the security patches to the operating system. Just like achieving product quality is an integrated development activity and not an afterthought, so should be security. This concept can be best enforced if the Software Development Life Cycle (SDLC) proactively includes security activities and deliverables for each phase. Some early lifecycle activities should include; classification of sensitive data in the requirements phase, recognition of vulnerabilities and development of strategy in the design phase and security specific code reviews and testing in the implementation phase. The scope of security should be well understood and documented throughout the lifecycle to build a secure Web application. Harris [Harris, 2013] has provided a list of security activities and deliverables in each SDLC phase.

Although a Web application executes on the client system, it primarily serves the needs of the Web server by capturing critical data and providing the requested data back to the client. A secure Web server environment is essential for keeping both the server and the client trouble free. On the client side, since a Web application runs on the client system, the development methods should include practices that do not leave the client system vulnerable to attacks. Although the operating system can provide support for activities such as access control, the application itself must use robust development methods to protect the client system. The following discussion highlights some development practices that will result in a more secure operational environment for both the Web server and the client system.

6.3.1 Web Server Environment

A Web server environment is quite complex and includes numerous variables such as an operating system, web server software such as Apache, application development tools, databases, 3rd party support such as credit card support system, etc. Each of these variables requires dedicated attention for a successful operation. Listed below are some guidelines that will make your environment more secure:

Environment Upgrade/configuration – The web server environment should be up to date at all times. Any security updates to the operating system, development environment upgrades, as well as database updates should be up to par with the latest release. By default, all operating system and 3rd party software services may be turned on and configurations set; they should be reviewed and their status should be adjusted to match security requirements. The development environment tools may also need scrutiny as well. The network protocols are gateways for intrusion and only the necessary protocols should be enabled.

Authentication and Access Control – Making sure that only an authorized client is accessing the Web server is very critical. Section 3 describes the main elements of access control - identification and authentication being related to client's identity. Once the user provides her/his credentials they should be transported securely over the network using HTTPS or some other secure mechanism. In case of failed login attempts using UserID and Password as credentials, the error message should not reveal which component of the credentials (userID or Password) was incorrect. Only a limited number of failed attempts should be allowed in an applications that deal with sensitive information. There have been recent trends where additional authentication is performed via security questions. To avoid data corruption due to spam generated by an attacker the websites now use Captcha standard where a client enters a slightly distorted text after logging in.

Access control is very important even if you have a fool-proof authentication mechanism. Access control should enforce a mechanism that has been designed based upon business requirements. Both vertical segregation (where users at the different level have access to different data set) and horizontal segregation (where users at the same level have access to different data set) should be considered when designing the access control lists. This is critical when the access for administering the system is being considered. In addition, the permissions (setting a flag value to True or False) should not be controlled by the HTTP request which can be manipulated by the client.

Input Data Validation – In a Web application the data can come from a client, generated by a third party application, server's own environment, as well as from a database. Failure to validate the data could lead to serious vulnerabilities. In its guide, OWASP [Frenchi, 2014] has provided a detailed discussion on data validation including the characteristics the data should be validated against and validation strategies. The data validation should, at least, include a check for data integrity – any tempering during transport form the client to browser or 3rd party to browser. The data characteristics such as data boundaries, data type (numeric, alphanumeric etc.), sign of data (positive or negative) and validation against business rules, for example a price discount cannot be 100%, should also be adequately carried out. These checks should be made at the layer the data is generated. For example, a form data should be checked at the client side before sending it to the Web server. It may be too late from the security perspective if the bad data has already been sent to the server.

The data validation strategies recommended by OWASP range from accepting only known good data, sanitization such as eliminating HTML special characters, to void SQL injection discussed in Section 5.1, and take precaution to syntactically validate the data in the context of its grammar. The OWASP guide also provides steps to assess if you are vulnerable and how to protect yourself.

6.3.2 Session Management

A session is a period of time a client is interacting with the Web server including the time the client is not active. The web server creates a session upon the first HTTP request from the client and stores it locally. This session creation takes place for every single client connecting to the Web server and this could amount to thousands of client sessions. A session consists of Session ID along with other information such as login and password. The Session ID is unique to a client and is used to identify the client each time he/she makes a HTTP request in a session until the session expires.

The two aspects of the Session ID that the developers need to be diligent about are its generation and its transport back to the client. The algorithm used for the Session ID generation should be complex enough so that it cannot be reverse-engineered by an attacker. If an attacker is able to determine the Session ID, he/she can take over all the clients currently on the server and ultimately control the entire Web server.

Off-the-shelf Session ID generation algorithms are found to be prone to attack. If a new algorithm is implemented it should at least be better than an off-the-shelf algorithm. Highly sensitive applications such as the ones used in the banking environment, generate a new Session ID for each HTTP call. Additionally, the sessions are terminated either after a fixed time period or if there is no activity from the client for certain time.

The second aspect, session ID transport, is equally important. The session ID should be encrypted as it moves between the client and the web server. Once again, a strong encryption should be used for transportation and the encryption key should be well protected.

6.3.3 Programming Language

Web applications are created using a programming language such as Javascript. Programming languages have their shortcomings which can be exploited by an attacker if she/he can find out the application programming language. The developer needs to be aware that the programing language is not displayed in error messages or any other communication that is taking place between the client and the Web server. The first step an attacker takes is to map the application that includes attempts to identify the programming language and the development environment of an application. Some programming languages such as C, leave the remnant information in the memory due to stack overflow on the client system, which can be harmful if an attacker gets hold of this information. If a programming language has such a behavior, the developers need to be aware of it and must take adequate countermeasures.

6.3.4 Use of Proper Tools

Even though Web application security is a relatively new challenge, its compromise can cause serious harm to the consumer. There are enough resources available to proactively circumvent the problem. OWASP has a vast amount of information that provides guidelines and methodologies to build more sure applications. Stuttard and Pinto [Stuttard, 2015], in their Web Application Hackers Handbook, have discussed numerous tools such as Burp Proxy, WebScarab, and Paros, as well as methodologies that can be utilized for application testing and make testing more robust.

7 Conclusion

In general, the business of information security is complex and application security is an integrated component of it. On top of it, the “bad guy” enjoys the challenge of being ahead of the creators of applications and has gathered an arsenal of sophisticated tools. The industry has been fortifying the computing environment from outside by using a variety of tools mentioned earlier. But, once the attacker penetrates the security perimeter, she/he is capable of doing serious damage as evident from recent events. We have two choices; become numb to the sequence of events or take application security seriously as the consequences are dire. The software security should be an integral part of the SDLC and built-into development practices. What has been presented in this article is the tip of the iceberg. The information technology community has to come together to win the war against the bad guy.

8 Acknowledgements

The author would like to extend his gratitude to his ex-coworker Joshua McKinney of Nike Inc., for inspiring him and building his passion for Information Systems Security. He would also like to recognize Laura Bright and Satish Yogachar of McAfee, part of Intel Security, for their invaluable review efforts.

References

1. Reuters, http://www.reuters.com/article/2014/03/13/us-target-breach-idUSBREA2C14F20140313
2. NetworkWorld, http://www.networkworld.com/article/2861023/security0/worst-security-breaches-of-the-year-2014-sony-tops-the-list.html
3. Beal, Vangie, 7 Layers of OSI Model, http://www.webopedia.com/quick_ref/OSI_Layers.asp
4. Harris, Shon, All in One CISSP, 6th Edition, McGraw Hill, 2013
5. Wikipedia, http://en.wikipedia.org/wiki/Web_application
6. Couture, Erik , http://www.sans.org/reading-room/whitepapers/application/web-application-injection-vulnerabilities-web-app-039-s-security-nemesis-34247, 2013
7. Moon, Silver, Sqlmap tutorial for beginners – hacking with sql injection, http://www.binarytides.com/**,** 2012
8. Padraic, Brady, Survive the Deep End: PHP security, http://phpsecurity.readthedocs.org/en/latest/Injection-Attacks.html, © Copyright 2014
9. Wikipedia, https://en.wikipedia.org/wiki/Cross-site_scripting, 2015
10. Stuttard, Dafydd and Pinto, Marcus, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2011
11. Troy Hunt, OWASP Top 10 for .Net developers Part 6: Security Misconfiguration, http://www.troyhunt.com/2010/12/owasp-top-10-for-net-developers-part-6.html, 2010
12. Frenchi, https://github.com/OWASP/DevGuide/blob/master/old/OWASP%20Guide%203.0.docx , 2011

Preventing Skunky Test Automation Shelfware: A Selenium-WebDriver Case Study

Alan Ark

Eid Passport

aark@eidpassport.com

Abstract

Eid Passport had a suite of Selenium tests with a bad reputation—difficult to maintain, broken all the time, and just plain unreliable. A tester would spend more than four days to get through one execution and validation pass of these automated tests. Eid Passport was ready to toss these tests into the trash. Alan Ark volunteered to take a look at the tests with an eye toward showing that Selenium-based tests can, in fact, be reliable and used in the regression test effort. Alan shares techniques he used to transform a sick, test automation codebase into a reliable workhorse. These techniques include AJAX-proofing, use of the Page Object model, and pop-up handling. The test process that used to take more than four days to turnaround now finishes in under two hours. And this is just the beginning.

Biography

A senior SDET with Eid Passport, **Alan Ark** helps the test organization cultivate and develop its automation solutions. Alan draws on more than twenty years of experience in various software quality roles when creating automated solutions to testing problems. Several of his projects have been presented at Quality Week (1999) and PNSQC (2008, 2011, 2013). These projects include validating a buggy data conversion of historical stock trading data into the Euro, championing the use of automated testing in new organizations, and creating testing frameworks from scratch. See Alan's LinkedIn profile at https://www.linkedin.com/in/arkie.

1 Introduction

Automated Graphical User Interface (GUI) testing is currently being used at most companies to help with their regression testing needs. A major challenge with relying on this method for regression testing is the maintenance. As the User Interface (UI) changes, the GUI automation must change with it, or else there is a good chance that the tests will not run correctly.

Improperly implemented GUI automation projects can turn into a maintenance nightmare. Sometimes, the burden of maintaining the automated tests can be so harrowing that they are tossed aside. The result is usually shelfware — software that sits on the shelf, not providing any real value to the engineering organization.

This paper looks at one Selenium-Webdriver based UI automation project that almost turned into shelfware. I took lessons learned from my previous experiences with web based test automation with Web Application Testing in Ruby (WATiR) and applied them here at Eid Passport.

The result is a suite of tests that are reliable and depended upon to give the development team confidence in any build of the software under test.

2 Help!

At Eid Passport, there was a full-time automation engineer who was in charge on the Selenium based test automation. He was a full time employee who was tasked with maintenance and creation of new tests that needed to be implemented. When he left the company, the automated tests were not actively maintained. The upkeep was a task that was done as time permitted by anyone who had extra cycles.

As more people started to run the regression tests, they noticed several patterns that were worrisome. The biggest worry was that the tests would appear to fail randomly. When taking a failed test and running thru it manually, it would always pass. Random tests would fail randomly during nightly runs. The problem was exacerbated by running the tests on different machines. If we are seeing random failures during regression test runs, how can the team have any confidence in the quality of the software?

Another point of concern was the amount of time that it took to get through one pass of the regression tests. Ninety tests took over 4 hours to run. The team was thinking about not using the automated GUI based regression tests anymore due to these two issues.

I volunteered to do a code analysis to see if the above issues could be addressed. I had championed the use of Page Objects in this automation suite, so I knew that any modifications to the code could be made in a confident matter. During the analysis phase, I noticed that the automation suite did not handle the AJAX parts of our application very well – which would result in the non-reproducible errors that were haunting the test suite. I felt that I could use the majority of the existing code, but tweak some of the underlying methods to make them more reliable. I also felt that with some optimizations, the test suite would finish in less time as well.

3 How Did We Get Here?

The Selenium based tests were actually version 2 of the GUI regression test suite for Eid Passport. The original version was implemented in Microsoft Coded UI and did not use a Page Object model. Using Microsoft Coded UI locked in the browser that you are using to Internet Explorer (IE). At the time, using any other browser for running the test automation would be out of the question. Microsoft has since remedied this to allow coded UI tests to be run on multiple browsers, but this was not available at the time

of original implementation. By not using a Page Object model, the original test suite was difficult to maintain as UI changes were made to the system under test.

I discussed the pros and cons with the automation engineer who was tasked with maintaining the regression tests and gave him an overview of a new design using Selenium. The biggest con to the project would be that he would have to reimplement the test suite. But moving to a new solution based on Selenium-WebDriver would give us the following advantages. The WebDriver API is now a part of the World Wide Web Consortium (W3C). By using Selenium-WebDriver, the company gained much more flexibility into the browsers that could be used. The support ecosystem for Selenium is also richer as it has gained popularity across the world. This also means that it would be easier to find new employees who could hit the ground running upon hire. By using the Selenium libraries, we had a choice in what language to use. The fact that Eid Passport is a Microsoft shop coupled with Selenium's good support for coders in C# made it a natural choice. By using Page Objects, the test suite would be much easier to maintain.

These pros readily outweighed the cons. He got the green light! Onward with the Selenium based tests.

4 Analysis of the Project

After spending some time looking at the code, I came up with a plan of attack to make the suite of tests more stable, reliable and useful for the rest of the team.

4.1 The Page Object Model

The reason to use a Page Object model is to separate the tests from the functionality on a screen. By keeping them separate, the idea is that if a page changes, or the functionality offered by a page changes, one needs only to modify the tests in one place - the definition of the page. Since the tests are consumers of a page, they do not concern themselves with the underlying implementation.

As mentioned in section three, the initial version of our automated tests did not use the page object model. When changes were made to the UI, there would be multiple tests to update. Sometimes, a test would not get the proper update and would fail accordingly. Fix that test, try again, and repeat until your fingers fall off. Contrast that prior approach with the one used in the initial Selenium based approach that uses the page object model. The tests themselves would remain untouched. Only the code that is used to represent the page that was modified gets changed. The modification should be invisible to the tests themselves. By scoping the changes only to the page object, the scope of the changes needed to match UI modifications can be kept to a minimum.

Page Objects can be used to help minimize the pain normally associated with code maintenance. If our current test automation did not use Page Objects, I probably would not have volunteered to examine the test suite.

4.2 Verify Assumptions

UI tests are fickle in that there is an inherent assumption in place that a manual tester will gloss over, but is very important to an automated test - Are you on the page that you think you are? As a manual tester of a web site, you know by looking at the browser whether or not you are at the right place. If the app is displaying the wrong page, it is easy for a manual tester to make the needed adjustments to get to the right place.

For an automated test, this usually is not so simple. The automation will be expecting that you are on the login page so that it can enter your name and password and click on the login button. But what if the test is already logged in? It will never find the login page.

One trick that I use is to verify the assumptions being used by the test.

Are you on the page that you think you are on?

Is the thing that you need to interact with being displayed?

Is it even on the page?

The answers to these questions will go a long way in predicting whether or not the tests will be reliable.

4.3 Generous Logging

One of the first things that I did to the code was to add more logging. In this day and age where disk space is cheap, it makes sense to add logging wherever possible so that any forensic analysis of test runs will contain all the information needed to be useful for troubleshooting and debugging.

Adding log messages on each transition will give more information about where on the site the test was during the run. These log messages are not super valuable when the tests are passing, but when they fail, they can give a lot of insight to what was going on, and save you time in the long run.

4.4 Unique Locators

Locators are ways to identify specific UI elements on a page. The most common way to find items are by using the HTML name, id, or class attributes. A mix of XPath and CSS selectors were used on this project, but what was troubling was the reliance on using index numbers on some selectors.

Using index numbers is not generally recommended as they are static. If changes are made to the page, the layout might also change, which will result in index numbers being regenerated. A test that is relying on a particular order of UI elements is inherently unstable and will most likely fail over time. This will lead to extra maintenance overhead as compared to tests that use a more descriptive locators.

```
// Find an element by attribute
driver.FindElement(By.Name("myName"));
driver.FindElement(By.Id("myId"));
driver.FindElement(By.ClassName("myClass"));
```

Another advantage of using good locators is that they can be immune to changes in layout. If the tests are looking for a textbox called ‘Name’ and a textbox called ‘Password’, then moving those fields will not break the tests as long as the names of those fields are not changed. The textboxes can even be moved inside a div that is displayed/hidden based on some JavaScript. As long as the names of the elements are not changed there is a good chance that changes on the UI will not affect the running of the automated tests.

4.5 Polling Sleeps

The most glaring offender in our test automation was the use of hard-coded sleeps. Our application under test relies on external jobs being run to completion before continuing. The original solution to this problem was to hard code a number to pause the tests in the hope that the jobs would finish before the test would continue. While this probably worked well on the original developer’s machine, it proved to be problematic in getting the tests to be reliable under different conditions. The solution I implemented makes use of polling sleeps.

The idea behind a polling sleep is that you would check for some expected condition. If the condition was not satisfied, the tests should sleep for a small amount of time, then recheck. This cycle would be repeated until either the condition is satisfied, or some pre-determined number of retries has been met.

Selenium has some built-in ways that this can be handled.

4.5.1 WebDriverWait

By using WebDriverWaits, there is no need to build your own wait code. You can initialize a wait object with a number of seconds to "wait" until the condition is met. One trick that I use is to ensure that the element that I would like to interact with, is actually on the page. Using WebDriverWait in this method will prevent the "No such element" exception that is rather common in automated UI testing.

```
// Create a new WebDriverWait Object with a 30 second timeout.
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

// Wait up to 30 seconds for the element with the given Id to appear.
// Throw an exception if it does not appear within 30 seconds.
IWebElement myDynamicElement = wait.Until<IWebElement>((d) =>
    {
        return d.FindElement(By.Id("myButtonId"));
    });
```

4.5.2 Expected Conditions

Selenium takes the idea of waiting for an expected condition a step further. It exposes convenience methods that will wait for common things to occur before continuing. The actual conditions may vary slightly based on the language being used for the Selenium based tests, but the C# and Java implementations of ExpectedConditions are the richest.

The C# binding includes support for such ExpectedConditions as:

```
TitleIs
TitleContains
ElementExists
ElementIsVisible
FrameToBeAvailableAndSwitchToIt
```

Using WebDriverWait in conjunction with ExpectedConditions can save time when coding, and make the resulting tests easier to read.

The references section includes a link to the C# source code for ExpectedConditions. It should be noted that not all languages support the same ExpectedConditions.

4.6 Popup Handling

Our site makes use of a number of different kinds of popup windows — JavaScript confirm messages, alert boxes, windows that surface new functionality, and popup windows that open other popup windows.

The original solution made use of hard-coded sleeps and then switching to a window expecting that it was there. The sleeps were sufficient for most of the windows to appear, but in the failure cases - when a pop-up was not seen, or an unexpected window appeared, the tests would simply stop and log a failure.

The new solution made use of polling waits in combination with comparing window handles against a list of known windows. At a minimum, we can keep track of the main window that gets launched with the automation. By getting the list of current window handles, we can compare it to the original to find out if the current window is the main window or a popup that we need to handle.

We also need to be aware (and handle) pop-up window exceptions that caused so many problems with the original implementation. Extra care should be used when interacting with apps that use pop-up windows.

Similarly, sites that use faux pop-ups like controls that are displayed or hidden based on some condition being met can be thought of in the same light. If you need to interact with an element that is not normally displayed by default, verify that your assumptions are being met before trying to interact with those elements or else you might run into the dreaded “No such element” exception, or even worse, the “Stale reference” exception.

4.7 Frame Navigation

Most of our site does not use frames, but if you are testing a site that uses frames, just keep in mind that it’s the same as any other Selenium interaction, once you have set your browser reference correctly. Just note that you need to switch the driver reference to the level where the frame resides.

```
//Find the frame, then switch to it
IWebElement mainFrame = driver.FindElement(By.Name("MainFrame"));
driver.SwitchTo().Frame(mainFrame);
```

4.8 IE Considerations

When dealing with Selenium, there were a few issues that tripped me up when dealing with the IE browser. The most glaring was that sometimes clicking on an element had no effect for the application under test. It turns out that this is a bug with the way that the IE Driver is implemented in Selenium. This bug affects multiple versions of IE. There is a workaround — hit the Enter key while focused on an element rather than sending it a click event.

Instead of

```
button.Click();
```

Use

```
button.SendKeys(Keys.Enter);
```

5 Results

The culmination of these above changes described earlier had a great impact on the automated GUI tests. What was once a finicky test suite prone to multiple random failures became a set of tests that are more reliable. Failures reported by the test are pointing to real failures within the system under test. The logs that are created by the tests can pinpoint to a problem with the software itself or a failure on a related system (and outside the scope of the testing focus for this team).

Another big win was in the amount of time to get thru one test run. Previously, it took over 4 hours to run thru a single pass of the tests, with multiple days spent tracking down the issues. As a result, the automation could be run only once a month before a release went out. A full week was blocked out for the purposes of regression testing.

Currently, run time is down to less than 2 hours. The tests are running reliably and are now a part of nightly regression testing. Any errors being reported by the tests usually are vetted within 30 minutes. Many times, the vetting is a simple blink test based on the report of tests pass/failed. No extra time is specifically blocked out for additional regression testing. It has just become a part of the process that we go thru in testing our software.

6 Conclusion

Test automation will only be used if it can be relied upon. Presented are some general ideas of how to structure your test automation in a way that can be easily maintained in the future. This paper also presented some ideas specific to using Selenium. While the code used in this paper is based on the C# bindings, the ideas can be translated to the other bindings, as well as to other GUI automation frameworks as well. In fact, many of the ideas used on this project come from my work in Ruby and Web Application Testing in Ruby (Watir).

Web sites are becoming increasingly "responsive" in their behavior, which introduces challenges when trying to validate functionality from the GUI perspective. By implementing robust test code, the test suite will be better suited to handle the irregularities that abound when dealing with communications on the web. This greater reliability will be a key factor to whether or not your automation framework will be used, or become dreaded shelfware.

Avoid shelfware and your organization will be better for it.

References

SeleniumHQ. "Selenium WebDriver." http://docs.seleniumhq.org/projects/webdriver/ (accessed June 26, 2015).

World Wide Web Consortium. "WebDriver" https://w3c.github.io/webdriver/webdriver-spec.html (accessed June 26, 2015)

Selenium. "Page Objects" https://code.google.com/p/selenium/wiki/PageObjects (accessed June 26, 2015)

GitHub. "Expected Conditions" https://github.com/SeleniumHQ/selenium/blob/master/dotnet/src/support/UI/ExpectedConditions.cs (accessed June 26, 2015)

AutoIt. "AutoIt". https://www.autoitscript.com/site/autoit/ (accessed June 26, 2015)

Watir. "Watir". http://watir.com/ (accessed June 26, 2015)

Variability vs. Repeatability – An Experience Report

Jonathan LI ON WING

jonathan.lionwing@mail.mcgill.ca
jwing@groupon.com

Abstract

Have I been approaching testing all wrong this whole time?

In my earlier years of being a test engineer, I was taught to write tests that are repeatable: given a certain set of parameters and inputs, map out the expected output – repeatable and simple. However with a developer-test ratio of 10 to 1, I was quickly falling behind. I found myself having to decide between spending time on my automation test suite and exploratory testing. The former gave great reliability regarding regression, but the latter caught most of the bugs. The former was repeatable and could run practically 24 hours a day, the latter only when I found time. How about if we implemented some of the exploratory concepts in the automated test suite?

In this paper, I will discuss about my experience in adding variability to test automation and creating an oracle to be able to determine correctness. I will demonstrate that, in my experience, adding randomness to the tests can help quicken delivery time, improve the discovery of bugs, and increase confidence, without compromising repeatability, when compared to traditional automation methods.

Biography

Jonathan Li On Wing is a Software Development Engineer in Test at Groupon in Seattle, WA. He earned his B.Sc. Cum Laude in Software Engineering from McGill University specializing in Artificial Intelligence and Requirements Analysis. He has 9 years of industry experience in various roles working at small and large companies such as Microsoft, Oracle, Expedia, and SMART Technologies.

Jonathan's interests lie in Human-Computer Interaction, Automated Testing and Software Life Cycle Processes. He believes in bringing testing upstream, and has been improving software testing processes in several companies. Jonathan sees himself as a customer advocate and enjoys challenging everything.

2015 © Groupon, Inc. All rights reserved.

1 Introduction & Motivation

The goal of software testing is to determine the correctness and completion of a given product based on its requirements. These requirements come in many forms which include requirement specification and through personal experience. Dr. Cem Kaner defines testing as an “investigation done to provide stakeholders information about quality of a product or a service." This examination is done to provide a level of confidence to stakeholders about the level of quality. However, as Edsger Dijkstra states, “testing can be used to show the presence of bugs, but never to show their absence!” So, if we are to provide a level of confidence, we ought to be careful in how we test, and how we discover bugs.

Throughout most of my academic career and for a part of my professional career, test cases would be scripted with expected inputs leading to expected outputs. They would either be run manually or automated. Initially they were mostly run manually, and as our profession started to mature, I found that we were moving towards having all our test cases automated. Additionally, as the feature code increases, manually testing has a harder time keeping up to cover all old regression testing and new functionality. Moreover, as time progressed, the developer-test ratio became exceedingly more developer heavy. Also, it was more efficient to have a machine run the tests. Unlike me, it can run multiple tests simultaneously and quicker, and seemingly it didn’t complain when asked to work than 40 hours a week.

There is no denying, that when it came to scripted tests, the machine would be more efficient, but how about exploratory tests. It has been discussed at length the benefits of exploratory tests. In my experience, scripted tests were very good at finding regression bugs or acceptance tests, but would not discover additional bugs. While exploratory testing would be pretty good at finding new bugs due to the fact there is a freedom in what we test – we adapt our tests based on our experiences and we vary our inputs and steps.

This leads me to ask: can we automate these tests? Since exploratory tests in one sense, requires human intuition, the quick answer would be no. However, can we take some aspects and use them for automation? Part of exploratory testing is learning and adapting our testing. This seemed quite hard, but definitely something I would like to tackle in the future. Another part is varying our tests. This is the motivation for the paper.

2 Methodology

In this section, we will consider the way I went about adding variability in our test framework.

2.1 Previous Methodology

Typically, we had tests similar to a framework shown in Figure 1, in one form or another. To test a service, the tests would gather the inputs and outputs from data files. The inputs would be used to call the web service and compare the response from the call, to the corresponding output.

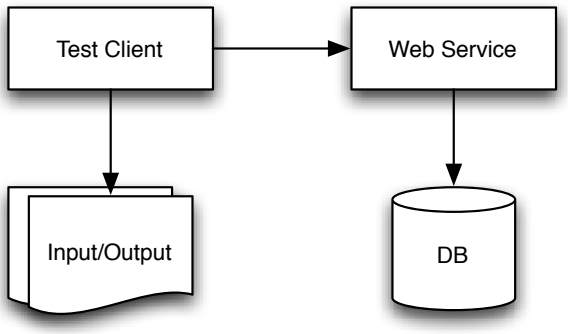


Figure 1

The problem with this methodology is its maintainability. First, the data files would need to be maintained as the data in the database could be changing. Additionally, as features get added, responses may not look exactly the same – this requires changing the data files often.

Another drawback is that these tests would always be testing the same thing over and over again. These tests, once written, found no new bugs. It did, however, grant confidence that with each new build, there were no regressions found.

2.2 Implemented Methodology

Our approach to deal with the aforementioned issues was two-fold.

2.2.1 Method 1

Let us consider a web service that performs a hotel search. A feature may be:

- As a client of the web service, I want to be able to get the details like its location and description for a specific hotel.

These features are straightforward and can be simple fetches to the database. Using the feature above, the methodology used to test it is as follows (a high level view can also be seen in Figure 2):

Algorithm 1 – Testing ability to get details of a hotel

Query database for a random hotel
Create an object model from Database response
Send hotel to HTTP Client, which will build an http request
Send the request, and create an object model for the response
Compare the two objects
(Repeat as often as required)

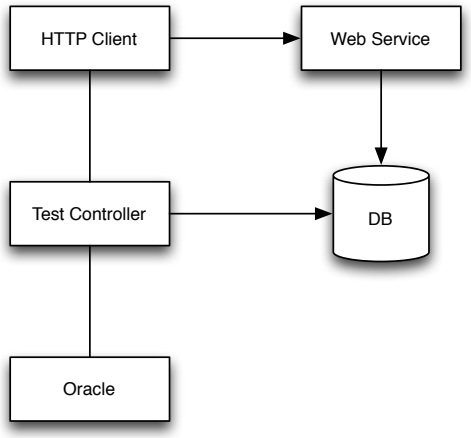


Figure 2

2.2.2 Method 2

Let us consider the same web service, but consider the following feature:

- Given a region[1], return all the active hotels in this region.

In this case, as it is not a simple fetch, we would need a way to validate a response's validity. There are three things that need to be validated:

- Are all the returned hotels in the region (see Algorithm 2);
- Are there any missing hotels (see Algorithm 3);
- Is the information, such as the hotel details, correct? (This can be tested similarly to previous test)

Algorithm 2 – Testing if all hotels returned are in the region

Query database for a random region
Send the region to the HTTP Client
Query the service for all the hotels
Oracle verifies that all hotels are inside region's polygon
(Repeat as often as required)

Algorithm 3 – Testing if there are any missing hotels

Query database for a random region
Query database for all hotels within a bounding box
Send the region to the HTTP Client
Query the service for all the hotels
Oracle verifies that, for each hotel
    Determine whether it should be inside or outside the region
    If inside, ensure it is in the response
    Otherwise, ensure it is not

We can also target our tests so that we can do more in-depth testing. Using the same story, we can come up with test cases such as:

[1] Regions are already created in the database and used production data. They are can be represented as multipolygons.

- Verify that a region that spans across the 180th meridian correctly returns all hotels
- Verify that a very large region returns all the valid hotels

In these cases, when we query the DB, we add the limitations when we search for random regions. That is, we can test with a random region that spans across the 180th meridian.

2.2.3 Logging

These tests will be quite hard to use if at any time the tests can randomly fail. Therefore, we also ensure that for each test run, we log enough information as to what failed, why, and how to reproduce it.

3 Observations

Although it was fun and interesting to build this framework, was it useful and worth it? The following are my observations.

3.1 Time

One of the main concerns is the time it takes to build this framework, and for that we will consider three parts: initial construction, adding new tests, and maintenance.

3.1.1 Initial construction

Custom frameworks always take more time to build than an out-of-the-box data-driven test framework (such as TestNG). As we already had an in-house flexible reporting engine, there was no need to build that part, and thus I would estimate took me a couple weeks longer – I was not working on this project at 100%.

3.1.2 Creating tests

Adding tests could be quite easy, and it varied between being faster and up to 3 times slower. When features could be hard to test by hand, it was simpler and faster, and if they were simple, it could take longer. Test cases generally didn't take longer than a few hours to write and run. Creating tests for more complex features were faster to write – for example, lets look at a scenario involving finding hotels within a region.

In a standard data-driven test approach, I could either create the appropriate data or use production data. Let us assume that the data is already there. I would need to find a nice region to test; usually we chose New York City. To determine which hotels are found in the region we could rely on many third party libraries available. If the code is not yet ready, I could create a response based on my investigation – which included the distance between the hotel and the city center. If the code was ready, I could compare the response with each hotel. Finally, they could be added to the data files. This process could be repeated for each test scenario, such as using a region across the 180th meridian – in other words slow, and to be frank, tedious.

In our approach, it was quite easy. Using the same example, my test case would look similar to the code seen below. Line 3, can be changed to get a random region that spans the 180th meridian if we wanted to add that test.

Pseudocode 1 - Testing if all hotels returned are in the region

```
01 testAllHotelsBelongInRegion() {
02  //Get 10 random regions
03  List<Region> regionsToTestWith = DBWrapper.getRandomRegion(10);
04  foreach (Region region in regionsToTestWith) {
05    Response response = HttpClient.getHotelsInRegion(region);
06    Oracle.assertVerifyHotelsBelongInRegion(response, region);
07  }
08 }
```

Of course, it is quick and easy to add tests when the functions like `getRandomRegion()` and `assertVerifyHotelsBelongInRegion()` are already implemented. However, the former utility function is a fairly trivial SQL query, and for the latter, there are many third-party libraries that can take care of finding if a point is inside a polygon. This works for us, but even if we did not have third-party libraries, in general I find that verifying the correctness of a response from a web service is easier to implement than the feature code. However, yes, this can make creating tests take more time.

As for simpler tests, such as testing whether we can get the details of a hotel, a function could look like the following:

Pseudocode 2 - Testing ability to get details of a hotel

```
01 testGetHotelDetails() {
02  //Get 10 random hotels
03  List<Hotel> hotels = DBWrapper.getRandomHotel(10);
04  foreach (Hotel hotel in hotels) {
05    Response response = HttpClient.getHotels(hotel);
06    Hotel hotelReturned = new Hotel(response);
07    Oracle.assertEquals(hotel, hotelReturned);
08  }
09 }
```

3.1.3 Maintenance

As our expected outputs are not hard coded, the time spent on maintenance is considerably lower. If a new field is created or modified, then we can just change the model class. For some change in functionality, we would just need to change the oracle's function. In general, this effort takes us less time.

Note that in some cases, maintaining tests may not be longer. Consider the story of getting the hotel details. Verifying that the fields are correct and that there are no new additions or missing fields can easily be done using traditional methods, by having a library generate the expected outputs and inputs. In this case, there is no difference in maintenance time.

3.2 Bug Count

As this was implemented with a new project, the way I compare how good this method performs, is by considering the bugs that would not have been caught in a traditional method. As expected, by adding variation, we were able to catch more bugs, albeit just about 10%. However, the majority of additional bugs caught, were of low priority. Of the ones that were caught, most were due to data issues – data we were not expecting – which I suppose is a good thing we were testing with random real data. An example of such issue can be seen in Figure 3, what is considered inside the polygon?

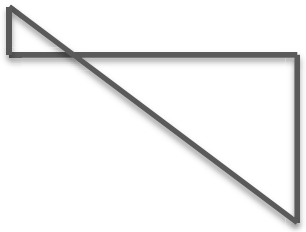


Figure 3

Another issue that was caught by the tests, although traditional test methods may have caught it, is the way distances are calculated. There are a few methods to calculate distance between two coordinates on the globe, as it was not specified which in the requirements, by calculating distances independently, we were able to figure out that we needed to define and be consistent in our distance calculations.

3.3 Confidence

Confidence is a hard metric to measure. Did we provide the right confidence level to our stakeholders? I do not know. As for the test team, we feel more confident. Using the same story example regarding finding hotels in a region, I do not imagine myself verifying the results of more than 3 to 5 regions – not to mention the chance of human error. However, by having randomly chosen 10-50 regions every night, I can be fairly confident that all the regions have been tested, even if there are new ones created.

Additionally, by varying the permutations, having tests run about 12 hours a day, I am pretty sure that is more permutations that I can test in my lifetime whether manually or by traditional automated data-driven tests.

4 Conclusion

As it is close to impossible to determine all the possible use cases a particular feature will be used, it is in my opinion that varying our automated tests was quite beneficial. Not only did they uncover more bugs, they were able to make the automation more maintainable, and it was easier to add new tests. However, even though they did uncover additional bugs, most of these bugs were of lower priority. These additional bugs that were found were mainly edge cases that did not much business value. They did, however spark conversations as to whether these bugs had other implications or whether the design was correct. In general, however, it was deemed more beneficial to know about the bugs and make a conscious decision not to fix them.

On a personal note, it was also much more interesting to add these tests as opposed to modifying and generating XML and JSON responses.

4.1 Comparison of variability versus repeatability

The following is a comparison of the pros and cons of adding variability as opposed to repeatability.

Variability	Repeatability
- Uncover more bugs - Easier to maintain - Generates more confidence - Tests more variation	- Faster to develop initially - Does not require coding skills - Predictable

4.2 Criticism

This work has generated a lot of criticism. The following are the main issues.

4.2.1 Repeatability and predictability

As described in the introduction, one of the main characteristics of a good test case is repeatability – running the tests over and over should yield the same result. How can we trust the tests if they fail one second and pass the next? In most cases, engineers will ignore the failures and just try rerun the tests.

If tests are failing randomly, then usually it means that either there is a data set that exposes a bug in the test code or feature code. It warrants investigation. In the traditional methodologies, we would not have caught the bug, therefore isn't it better to know of a bug?

Note that it is important to keep tests green. So if we ever got a data set that made tests fail, we added a blacklist such that when we got random data from the database, we would ignore those on the blacklist. At times, the blacklist would be a condition, not necessarily identifiers – since the list of ids can be long.

4.2.2 Duplicating feature work

When proposing this idea, or sharing our work, it is often thought that we are just reproducing the feature work. In fact, we are writing code to verify the correctness, which generally is simpler and quicker to write. Although, when it is simple fetches, it may be quite similar. The general concern is: is it worth it? The team can only answer this question, and it definitely will not be suitable for all teams and scenarios. Perhaps a mixture of test approaches?

4.2.3 Testing the tests

One question that arises often is how do you test the tests, and are there any concerns for false positives or false negatives? That is in fact a concern, but not as big as expected based on my experience.

There are two possibilities with the tests, the fail or pass. When a test fails, it could mean that there is a bug in the test code, feature code, or both. After some investigation, we will find out what, and it will be resolved. If a test passes, it means that either there isn't a bug found, or that both feature code and test code are expecting the same results. As they were done independently, and the tests will vary over time, the chances that both consistently passes is minute.

In my experience, it has happened once, but neither traditional data-driven tests, nor exploratory testing would have caught. It took weeks for the developer team to find and fix, and it was due to an assumption both the developers and testers made.

4.3 Future Work

To deal with the repeatability concern, it could be interesting and useful to add a quick and easy way to replay the tests. These can come from replaying based on logs, or storing a test session id with a seed.

Part of exploratory testing, and one of the reasons it is touted as one of the best testing methodologies, is its ability to adapt. I would like to see if I could use machine learning and statistical analysis to have the tests adapt itself and explore areas on its own.

References

TestNG. http://testng.org (accessed June 8, 2015).

Whittaker, James A. *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. Upper Saddle River, NJ: Addison-Wesley, 2010.

Acknowledgments

I would like to thank all those who helped me review this paper and bounce ideas off: Sean Chitwood, Lory Haack, Yumi Rinta, Robert Sabourin, Devan Samineedi, and Shruti Van Wicklen.

I would also like to thank my wife and son for being patient as I wrote this paper – and always being there.

The Journey of Testing with Stubs and Proxies in AWS

Lucy Chang

lucy_chang@intuit.com

Abstract

Intuit, a leader in small business and accountants software, is a strong AWS(Amazon Web Services) partner and has migrated several services to the AWS platform. We are also implementing a service-oriented architecture to provide better user experiences. With a service-oriented architecture, it is common that the service that one team works on depends on other services, which is referred to as depended-on component (DOC). When the DOC is not ready for integration, we can utilize stubs to simulate the interaction and continue the work. Stubs are canned responses used to replace the real DOC for test-specific purposes. We can create different permutations of test data stubs to increase code coverage. Combining this with the fault injection for resiliency testing helps to ensure the availability and robustness of our services. In addition to stubbing, we also need a proxy to forward the requests to the real DOC for end to end integration testing.

We at the Intuit QuickBooks Online team have researched different options for setting up stubbing and proxies suitable for the unique AWS operating environment. It needs to be easy to learn and easy to adopt. We implemented Wiremock server in AWS for the team and were able to increase our code coverage significantly and do resiliency automation testing which was very difficult to automate before.

This paper describes the following.

1. A basic introduction of architecture design in AWS

2. The Wiremock tool and why it was chosen

3. How to automate setting up Wiremock in AWS

Biography

Lucy Chang is a Senior Software Engineer in Quality at Intuit, currently working in the QuickBooks Online team in Mountain View, California. She has extensive experiences in web services automation testing, performance testing and mobile automation testing.

Lucy has an M.S. in Computer Information Technology from the University of Pennsylvania.

1. Introduction

As the industry embraces the service-oriented architecture, it is common practice that engineers work in a team that focuses on developing a single web service. Most likely this service will depend on other services, which are developed by other teams. Since the depended-on component (DOC) belongs to different teams, it is naturally hard to align the timeline for all the teams to complete all the work by the time engineers need to integrate [1]. Therefore, engineers are forced to wait until the DOCs are ready in order to start the integration work. This affects the release schedule and does not put the engineers' time to the best use when they are blocked and waiting for others to catch up.

What if there was a way that engineers could continue the integration work without the DOC being ready? Using Wiremock to stub the DOC to simulate its designed behavior does exactly that [2]. The application server can communicate to the Wiremock server and the Wiremock server will return stubbed responses as if it is the DOC. This is seamless to the application server and unblocks the engineers for the integration work.

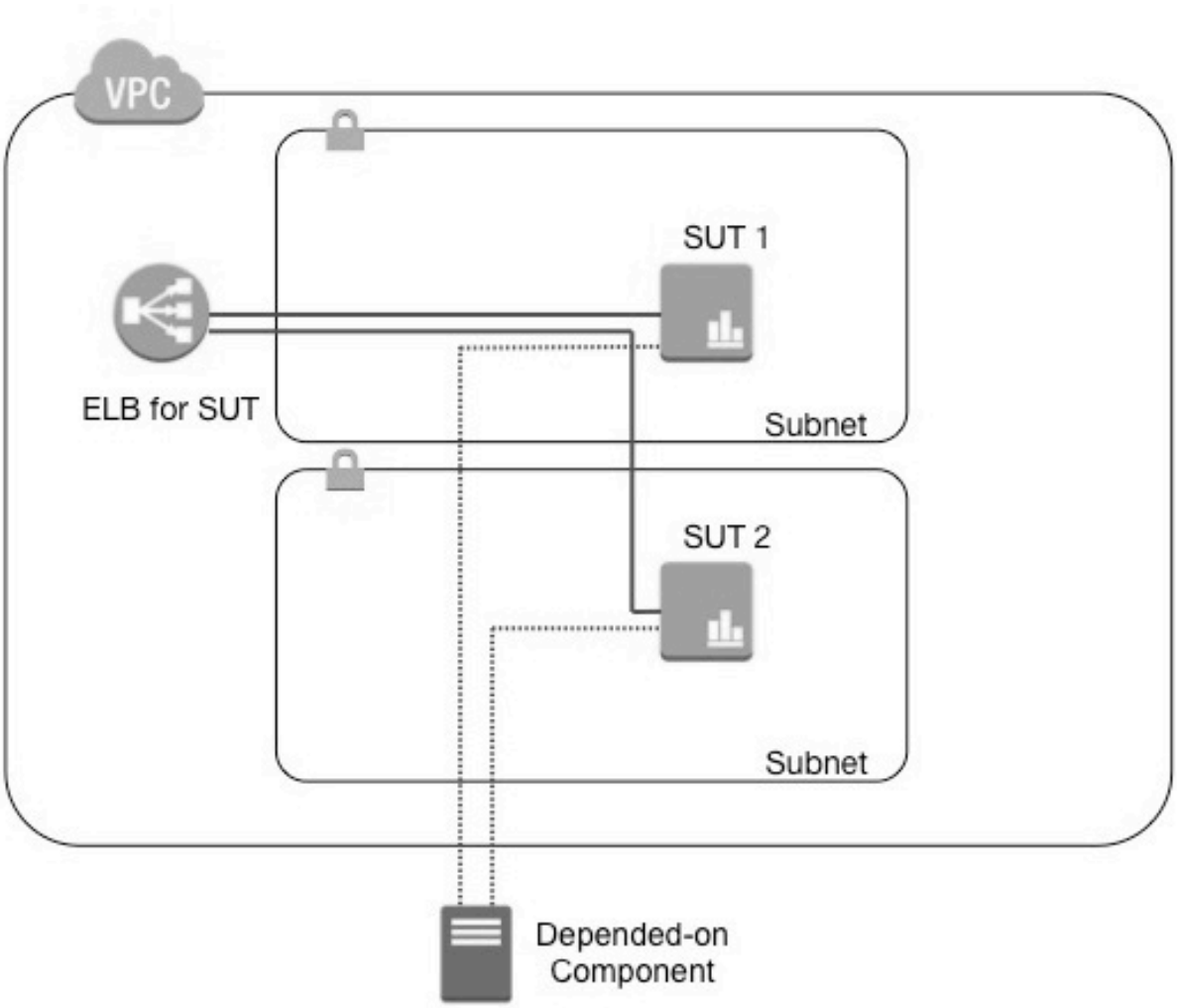
To software quality professionals, this allows for integration and end-to-end testing without relying on the DOC. With stubs, we can test different permutations of responses or negative test cases. Moreover, we can even do fault injection [3] for resiliency testing [4] using stubs in the automation. This opens a new world to the automation code coverage and allows for automation that we were not able to do before. In addition, we still need to do end to end testing with the real DOC. The Wiremock server can act as a proxy to forward the request to the DOC and relay back to the system under test (SUT). We will cover more about how it works in Wiremock section.

Another industry trend is to move the servers to cloud computing services for scalability, availability and the pay-as-you-go pricing model. One of the cloud computing services leader is Amazon Web Services (AWS). Combining these two trends, opportunities arise for testing using stubs and proxies in AWS.

2. AWS

Traditionally, it takes several months to provision new IT hardware before it actually gets into the hands of engineers. With AWS cloud computing services, users now can provision IT resources at any time. Only a few clicks in the AWS console are required to spin up load balancers, Elastic Compute Cloud (EC2) instances, or databases within few minutes. EC2 is a web service that allows users to launch the EC2 instances in the AWS web console. Users can select different capacities, different operating systems, and set up network permission and security access for each EC2 instance. Behind the scenes, an EC2 instance could be a virtual machine or a physical box. This is a huge advantage to the engineers who now have full control of the operating environment. Moreover, using cloud computing services gives the engineers the ability to automate provisioning of IT resources. We will cover more about the automation part in later sections.

Below is a simplified architecture for a system under test (SUT) in AWS. The architecture setup in AWS is built to have high availability, which directly affects the way we set up our Wiremock server. The impact will be discussed later in the paper.



In AWS, we created a Virtual Private Cloud (VPC) dedicated to our AWS account [5]. In our VPC, we can provision AWS resources according to our needs. In this example, we created two subnets located in two different availability zones. Availability zones are distinct physical locations so that when one availability zone is down due to unforeseeable events, for example an earthquake, the AWS resources in other availability zones will not be affected. This significantly increases the availability for our system under test (SUT). Inside each subnet, we launched one EC2 instance and deployed our application on it, which is our SUT. Our SUT talks to the depended-on component (DOC) that is, in this case, outside of our AWS VPC. To distribute the load to the two instances of the SUTs, we set up an Elastic Load Balancing (ELB), which resides in the VPC. The load balancer is responsible for taking the incoming traffic and distributing the requests to the SUTs. All the traffic has to go through the ELB before it is forwarded to the SUT instances.

3. Wiremock

3.1 What is Wiremock

Wiremock is a library for stubbing and proxying web services. It creates an actual HTTP server that integration tests or an application server can connect to as if it is a real web service [6]. After starting up the Wiremock server, we configured the SUT's endpoint URL to point to the Wiremock server instead of the DOC. Then we called the Wiremock API to set up the stub response with a respective request filter. A

request filter can be a specific endpoint, a specific header or a combination of both. If the request matches the request filter, then the Wiremock server will return a stubbed response as if it is from the DOC when the SUT hits the Wiremock server. For the requests that do not match the request filters, the Wiremock server proxies the request to the DOC and relays back. It is seamless to the end users calling the SUT.

3.2 Why Wiremock

3.2.1 The Requirements for Our Team

We were looking for a tool that is able to stub web services requests dynamically from the automation tests. Besides that, it was a plus to allow users to stub without modifying the system under test code except for configuration changes. This empowered the software quality engineers to stub without spending too much time to figure out or set up SUT code. Last but not least, it needed to be easy to learn and straightforward to use.

3.2.2 Why Wiremock was chosen

There are many options for stubbing and proxy tools. Wiremock met all our criteria. Below are the reasons why Wiremock stood out from the others.

1. It requires minimum changes to the SUT code. A simple endpoint URL configuration change pointing to the Wiremock server suffices.

2. It does not require users to dive into the underlying logic for the depended-on component or read the codebase for a system under test. All you need is a sample response and request filter, so that the Wiremock server can return the stub response according to the request filter.

3. It is very easy to set up. After uploading the Wiremock jar to the EC2 instance, a simple one-line command calling the jar will start up the Wiremock server.

4. It allows dynamic stubbing. Users can call the Wiremock API to set the stubs on the fly, which allows users to stub different responses for different test cases in the automation.

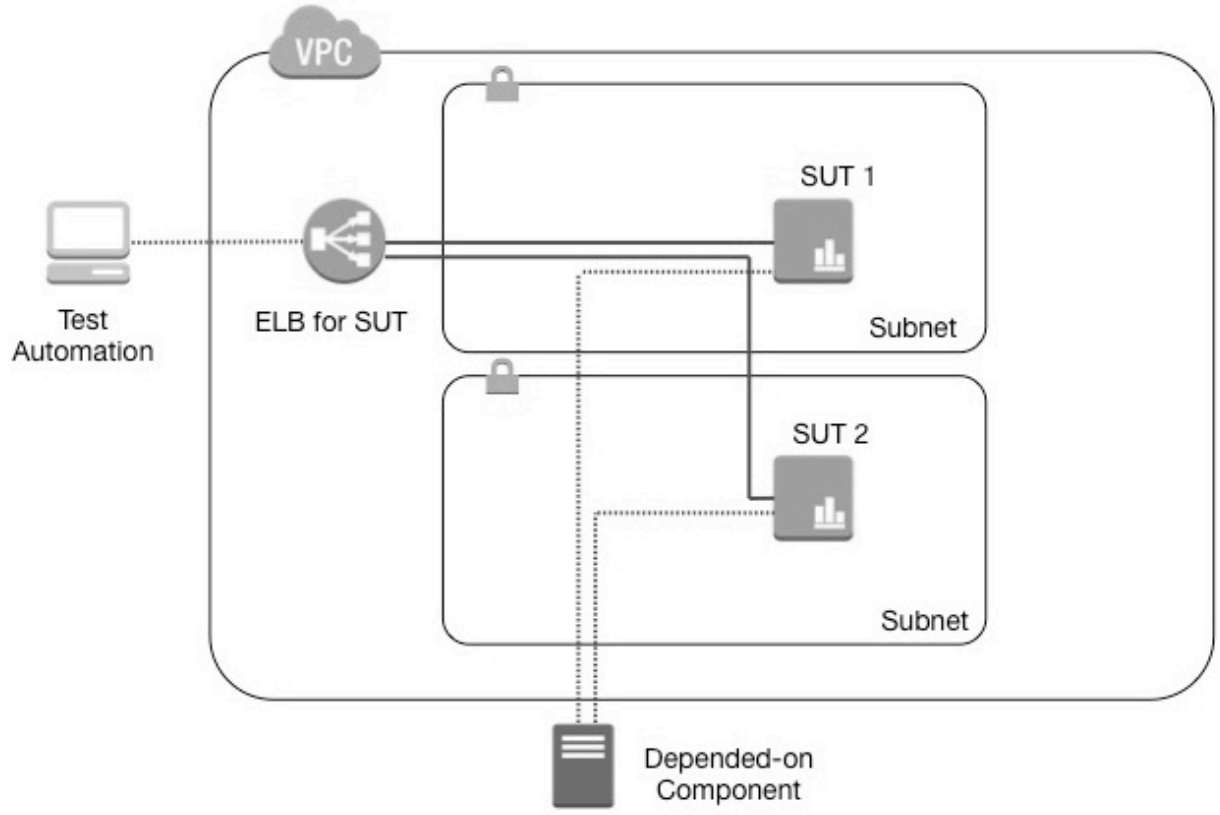
5. The Wiremock API supports fault injection. It empowers us to do resiliency testing for automation that was very difficult to simulate before.

4. Setting Up Wiremock in AWS

4.1 The Architecture in AWS

4.1.1 The Architecture and the Flow Before Setting Up Wiremock

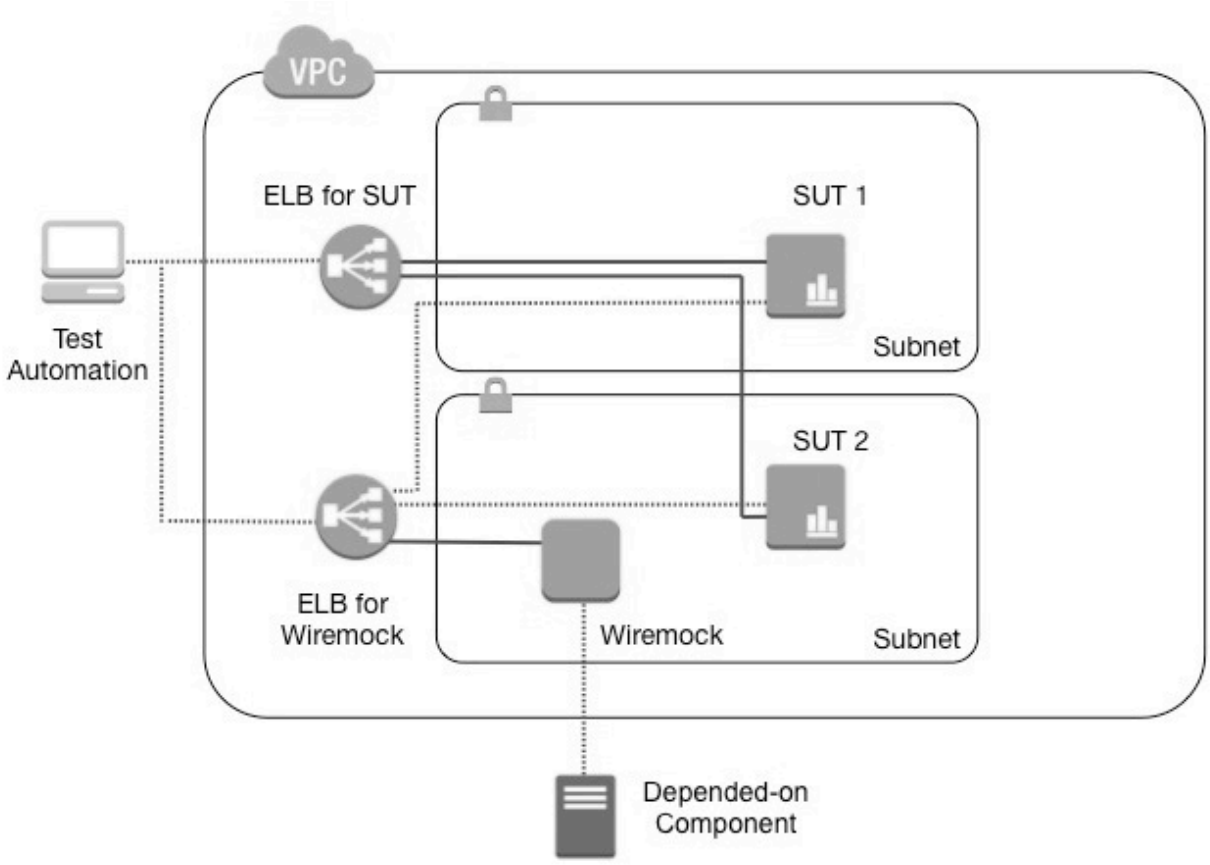
The Test Automation icon in the diagram represents the integration tests, usually written by software quality professionals. SUT 1 and 2 are the systems under test. Before using Wiremock, the automation tests called the ELB for API testing. The ELB forwarded the requests to one of the SUT instances that will in turn call the depended-on component.



4.1.2 The Architecture and the Flow After Setting Up Wiremock

We provisioned an EC2 instance in one of the subnets and deployed the Wiremock jar into the instance. We intentionally chose not to deploy the Wiremock jar into the system under test EC2 instances even though it is more cost efficient and saves all the set up effort for the new additions of the AWS resources. Imagine for a moment that we deploy the Wiremock into SUT 1. The automation calls the Wiremock API to set up stubs. After the ELB receives the Wiremock API request, it can forward the API request to SUT 1 or SUT 2. Let us say it happens to proxy the requests to SUT 1 and sets up the stubs successfully in SUT 1. When the automation hits the ELB again to run the test, the ELB forwards the requests to SUT 2. Since we set up the stubbed response for SUT 1, not SUT 2, when the automation tests hit the SUT 2, the tests will not get the stubs and the tests will fail. Due to the nature of the distributed system of AWS

has, there is no way to set up stubs and get a consistent stubbed response back if we deploy Wiremock to the SUT instance. Thus, we spun up a dedicated EC2 instance for the Wiremock server.



In addition to the dedicated EC2 instance, we provisioned a dedicated ELB for the Wiremock EC2 instance. In AWS, you cannot specify which instance to forward to for ELB. If we reuse the ELB, then the requests might be forwarded to SUT instances instead of the Wiremock instances. There are other benefits for setting up its own ELB. First, we can set up a domain name server (DNS) name for the Wiremock ELB. Some companies have the policy to delete the AWS resources periodically for security purposes. If the DNS name is set up, then we do not need to change the endpoint in our automation even if we spin up new instances and delete old ones. Secondly, the Wiremock ELB and Wiremock instance are independent from the SUT set up. We can deploy a SUT without bringing down the Wiremock instance and losing the stubs that were previously set. If we do not want the Wiremock server anymore, we just need to simply point the SUT's endpoint back to the DOC and delete the Wiremock instance and Wiremock ELB.

In this set up, the flow is: The automation test calls the Wiremock ELB, which forwards the request to the Wiremock EC2 instance to set up the stubs. Then the automation test calls the system under test ELB that proxies the request to the SUT instance. The SUT calls the Wiremock ELB. Wiremock ELB forwards the requests to the Wiremock EC2 instance. The Wiremock EC2 instance returns the stubbed response back to the Wiremock ELB and goes back to the automation test through the same route.

At the time we set up stubs, we also specify the request filter. Wiremock will only return the stubs if the request matches the filter. If it does not match, it will forward the request to the DOC and relay the response back.

4.1.3 Stub Requests

All the Wiremock code in this paper uses Rest-assured [7], a web service testing framework, to do the Wiremock API call.

This is the command line call to start the Wiremock server.

java -jar wiremock-1.53-standalone.jar --verbose --port 8080 --proxy-all=[DOC DNS Name]

This sets the Wiremock to return with status equals to 200 and the body we specified only when the endpoint equals to “/from/where”. “get(urlEqualTo(“/from/where”))” is the request filter for this stub.

```
stubFor(get(urlEqualTo(“/from/where”))
                    .willReturn(aResponse().withStatus(200)
                    .withHeader("Cache-Control", "no-cache")
                    .withHeader("Content-Type", "application/json")
                    .withBody(“Taiwan” )));
```

This calls the Wiremock API to do fault injection for the specified endpoint.

```
stubFor(get(urlEqualTo(“/some/thing”))
                    .willReturn(aResponse()
                    .withHeader("Cache-Control", "no-cache")
                    .withFault(Fault.EMPTY_RESPONSE)));
```

4.2 Automate Setting Up Wiremock in AWS

4.2.1 Automate Launching the ELB and EC2

Potentially you can manually create the ELB and EC2 and just leave them there. However, some companies have policies to regularly clean up the AWS resources for security purposes. Then it is essential to automate these otherwise time-consuming and tedious manual steps. Automating them saves a lot of time, eliminates possible human errors and allows other teams to reuse them. Another major advantage is that you can put the automation script under revision control and review different versions if things ever go wrong.

We used AWS CloudFormation to automate launching ELB and EC2. CloudFormation is a service that helps the user to model and set up AWS resources easily [8]. It is provided by AWS and is widely adopted by the AWS user community. In addition, our team was already using it for application server deployment. With those reasons, we decided to choose CloudFormation. We created a CloudFormation template for the ELB and EC2 that specifies all the parameters that are required to launch the resources, i.e. instance type, security group, IAM roles and other parameters. The templates are in JSON format. You can also use CloudFormer to create CloudFormation templates from existing AWS resources. After the templates were created, we then called the CloudFormation API with the template in the payload to launch the AWS resources. It is simple and elegant.

4.2.2 Automate Deploying Wiremock

Deploying Wiremock requires downloading the jar into the EC2 instance and starting the Wiremock server. While you can do so manually, automating this allows other teams to reuse it and you can check in the code and track different versions. At Intuit, we chose to use Chef, a widely popular infrastructure automation framework. Chef turns infrastructure into code. You can code how you build, deploy and manage your infrastructure [9]. For example, you can write a Chef web server recipe to instruct Chef to download and install apache and JDK, run shell scripts to start the server, set up ports and users on the target server, etc.

In the Chef repo, we used Berkshelf, a dependency manager for Chef, to pull all the required recipes [10]. For our Berkshelf, it pulls the Java recipe and Wiremock recipe from our Intuit repository. Java recipe installs Java on the EC2 instance. The Wiremock recipe downloads the Wiremock jar from the maven and executes the command to start the Wiremock server. We extracted several parameters into attributes that are common to the cookbook. This allows using the same cookbook to deploy Wiremock to different pre-production environments. Extracting the parameters common to different environments into attributes ensures that they are consistent when deploying to the different environments. One example will be the Wiremock jar version. We want it to be the same for all environments.

5. Conclusion

With stubs and proxies in the AWS, we are able to test different permutations of responses from depended-on components easily in the test automation. It not only significantly increases our code coverage, makes our system under test more robust, but also allows teams to develop in parallel, move faster and be agile without being blocked by a DOC. Moreover, the fault injection functionality provided by the Wiremock allows us to do resiliency automation testing with a simple API call. We are able to catch bugs before they escape into production. It is hard to achieve before. Doing stubs and resiliency testing in AWS poses some challenges, including the high availability architecture and learning curve for AWS API and AWS operating environment. But it also presents unique opportunities to empower the team, since we now have full control over our infrastructure management. We can delete the instances whenever we want and provision new instances and deploy quickly for a fresh environment. With all these combined, the code quality is significantly increased and fewer bugs are introduced to production.

References

[1] "Depended-on Component (DOC)." *DOC at XUnitPatterns.com*. Web. 28 July 2015. <http://xunitpatterns.com/DOC.html>.

[2] "Test Stub." *At XUnitPatterns.com*. Web. 28 July 2015. <http://xunitpatterns.com/Test Stub.html>.

[3] *Wikipedia*. Wikimedia Foundation. Web. 28 July 2015. <https://en.wikipedia.org/wiki/Fault_injection>.

[4] "Resiliency Testing." *The IT Law Wiki*. Web. 28 July 2015. <http://itlaw.wikia.com/wiki/Resiliency_testing>.

[5] "VPC and Subnets." *Amazon Virtual Private Cloud*. Web. 28 July 2015. <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Subnets.html>.

[6] "WireMock." *WireMock*. Web. 28 July 2015. <http://www.wiremock.org>.

[7] "Rest-assured." *GitHub*. Web. 28 July 2015. <https://code.google.com/p/rest-assured>.

[8] "AWS CloudFormation." *AWS CloudFormation*. Web. 28 July 2015. <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>.

[9] "Chef." *Chef*. Web. 28 July 2015. <https://www.chef.io/chef/>.

[10] "Berkshelf." *Berkshelf*. Web. 28 July 2015. <http://berkshelf.com/>.

Write Once, Run Everywhere: Beaker & Puppet Labs

Alice Nodelman

alice@puppetlabs.com

Abstract

Over the past two years, Puppet Labs has developed the open source Beaker acceptance testing tool. Beaker provides 'write once, run everywhere' testing. The same test can be run locally or in the cloud, it can exercise Puppet from source, Puppet from package or Puppet from enterprise installation, and it can be executed against all of Puppet Enterprise's 14 supported platforms which includes *nix, OS X, and Windows.

Beaker is written in Ruby and uses object inheritance to create a host abstraction. Test writers interact with the abstraction, while Beaker manipulates the actual platform under test. Beaker also includes a domain specific language (DSL) which wraps common testing tasks. Combined together, the host abstraction and DSL make test writing cheap and easy.

Building a new testing system is expensive and complicated. It requires the same effort as any other software development project, including designing, developing, documenting, testing and ongoing maintenance. The significant investment that Puppet Labs has put into Beaker has resulted in the creation of over two thousand Beaker acceptance tests - with more added every day by both employees and community members. Any regression discovered is converted into a Beaker test to ensure future behavior. Puppet Labs project builds are now subjected to days of testing effort without any manual intervention. Beaker is central to Puppet Labs' automation infrastructure and essential to the dependability of Puppet Labs software.

Biography

For over a decade, Alice Nodelman has specialized in automated testing of hard-to-test software. She believes that software should keep its promises. She has mocked out online marketplaces, generated giant databases, replicated web browsing, and otherwise pressed buttons that she shouldn't have. Currently a Senior Quality Engineer with Puppet Labs, she is lead developer of Beaker — an open source, multi-platform, multi-node, multi-cloud acceptance testing tool. Someday she will automate everything.

1 Introduction

Puppet Labs provides IT automation solutions to easily automate repetitive tasks and centrally manage configuration of systems. The software supports a wide variety of operating systems, from *nix-style to Windows and OS X. Puppet can be deployed on as little as one node, or throughout an entire data center. The combination of the number of supported platforms and the deployment methods makes Puppet an incredibly complicated product to test. In addition, Puppet supports a thriving open source module, or plugin, community that needs to test their contributions reliably. An automated testing system that could use computer cycles and free person-hours was a necessity.

Four main areas of functionality were identified for a Puppet acceptance testing tool:

1. Create understandable, platform agnostic tests to avoid code duplication
2. Flexibly provision test nodes locally, on the internal network, or in various cloud systems
3. Provide a variety of methods to install Puppet, either directly from source, as an open source package or from the enterprise installer
4. Be available both to employees and community members

With this problem space defined, the Beaker automated testing tool was created.

1.1 Terminology

Object inheritance is the concept that when a class of objects is defined, any subclass that is defined can inherit the definitions of one or more general classes.

Puppet is a configuration management solution that allows you to define the state of your IT infrastructure, and then automatically enforces the desired state.

Cloud Provisioning is the creation of new virtual instances on an internal or external cloud computing provider.

Software is tested on a *System Under Test* (SUT), which could be the user's own hardware, local Virtual Machines (VM), or cloud-provisioned boxes. They can also be termed *nodes*, *boxes*, *machines, instances,* or *VMs*.

The *full stack* describes all of the front and backend dependencies necessary to install and run a project – from the hardware, to libraries, databases, and external services.

Acceptance or *acceptance level* tests are those that ensure a piece of software keeps its promises or contract. These are full stack tests on 'clean' (non-development) environments that match a user's projected environment. These tests confirm that the software can be installed and interacted with, regressions are absent and functionality is verified. The software itself is treated as a black box that tests interact with in the same way a user would.

A *hypervisor* or *virtual machine monitor* (VMM) is a piece of computer software, firmware, or hardware that creates and runs virtual machines.

Provisioning is a mechanism for creating nodes or *instances* in a particular hypervisor.

An *Application Program Interface* (API) is a precise specification written by providers of a service that programmers must follow when using that service. It describes what functionality is available, how it must be used, and what formats it will accept as input or return as output.

A *domain-specific language* (DSL) is a computer language specialized to a particular application domain, in this case custom created to aid Puppet Labs automated acceptance testing.

2 Beaker Basics

Beaker is a tool that creates and communicates with a set of SUTs. Beaker provisions and configures the SUTs, installs appropriate dependencies, and then executes tests on those nodes. All communication between Beaker and its SUTs is done over SSH (a UNIX-based command interface and protocol for securely getting access to a remote computer). Test commands are processed sequentially, one per SSH channel on a dedicated SSH connection.

Beaker is written entirely in Ruby. In the Puppet Labs ecosystem, Ruby is the *lingua franca* – known internally and externally. It was a natural choice to make it as easy as possible for people to become test authors – both employees and open source community members.

2.1 Test Files

Beaker tests are Ruby scripts. A single test is captured in a single file. Here is an example test that echoes 'hello' on each SUT:

```
test_name "say hello to all my hosts!"

hellos = 0
@hosts.each do |host|
  step "i'm going to say hello to #{host}"
  on host, 'echo hello'
  hellos += 1
end

#make sure that i've been polite!
assert_equal(hellos, hosts.length, \
            "#{hellos} <> #{hosts.length}")
```

From this example, you can see the use of a *hosts* object. The *hosts* represents an array of all SUTs currently under test. By looping through these one at a time we can execute code against each SUT.

Running against two test hosts, a CentOS 6 box and an Ubuntu Lucid box, results in the following console output:

```
Begin tests/confirm.rb

say hello to all my hosts!

  * i'm going to say hello to pe-ubuntu-lucid

pe-ubuntu-lucid 15:10:02$ echo hello
hello

pe-ubuntu-lucid executed in 0.71 seconds

  * i'm going to say hello to pe-centos6

pe-centos6 15:10:03$ echo hello
hello

pe-centos6 executed in 0.18 seconds
tests/hello.rb passed in 0.90 seconds
      Test Suite: tests @ 2015-06-03 15:10:02 -0700
```

```
  - Host Configuration Summary -


        - Test Case Summary for suite 'tests' -
 Total Suite Time: 0.90 seconds
Average Test Time: 0.90 seconds
        Attempted: 1
           Passed: 1
           Failed: 0
          Errored: 0
          Skipped: 0
          Pending: 0
            Total: 1
```

Since no exceptions or assertions are generated during Beaker test execution, this test will pass. A Beaker test suite is a directory of Ruby files. Each individual Ruby file will be executed in alphanumeric order by file name.

2.2 Logging And Reporting

Beaker provides three methods for exploring and interpreting test results: terminal logging, plain text logging, and JUnit XML.

2.2.1 Terminal Logging

Beaker generates a large amount of information printed to the terminal during test execution. When the `--debug` option is enabled, Beaker reports each command executed and the resulting output from each individual SUT.

Below demonstrates a command first executed on a CentOS 7 SUT and then executed on a Windows 2008 SUT.

```
 * Replace site.pp with a new manifest

host1.net (centos7-64-1) 05:54:43$ puppet agent --configprint
node_name_value
host1.net

host1 (centos7-64-1) executed in 0.48 seconds

host2.net (windows2008r2-64-2) 05:54:44$ cmd.exe /c puppet agent --
configprint node_name_value
host2.net
```

2.2.2 Plain Text Logs

After the test run, Beaker generates three log files in `./log/#{HOST_YAML_FILE}/#{TIME}`.

```
log/example_hosts_file.yml/2015-06-25_15_23_42
├── sut.log
├── tests-run.log
└── tests-summary.txt
```

The `sut.log` file contains information about the SUTs used during the last test run.

```
$ cat sut.log
2015-06-25 15:23:42    [+]    fusion    el-6-i386    pe-centos6
2015-06-25 15:23:42    [-]    fusion    el-6-i386    pe-centos6
```

The `tests-run.log` contains the same data printed to terminal during test execution. All test execution information is contained in this single, flat file.

`tests-summary.txt` has the overall pass/fail/error/skip results of the tests that were included in the run.

```
        Test Suite: tests @ 2015-06-26 05:46:50 -0700

          - Host Configuration Summary -

                  - Test Case Summary for suite 'tests' -
           Total Suite Time: 545.70 seconds
          Average Test Time: 1.48 seconds
                  Attempted: 369
                     Passed: 42
                     Failed: 0
                    Errored: 0
                    Skipped: 327
                    Pending: 0
                      Total: 369

          - Specific Test Case Status -

Failed Tests Cases:
Errored Tests Cases:
Skipped Tests Cases:
  Test Case /var/lib/jenkins/workspace/qe_beaker_intn-sys_vpool-
pe372/agent/windows2008r2/master/centos7/pe_acceptance_tests/acceptance
/tests/aix/aix_package_provider.rb skip
  Test Case /var/lib/jenkins/workspace/qe_beaker_intn-sys_vpool-
pe372/agent/windows2008r2/master/centos7/pe_acceptance_tests/acceptance
/tests/puppetdb/puppetdb_cert_whitelist.rb skip
<snip> . . . . more skipped tests here . . . . . </snip>
```

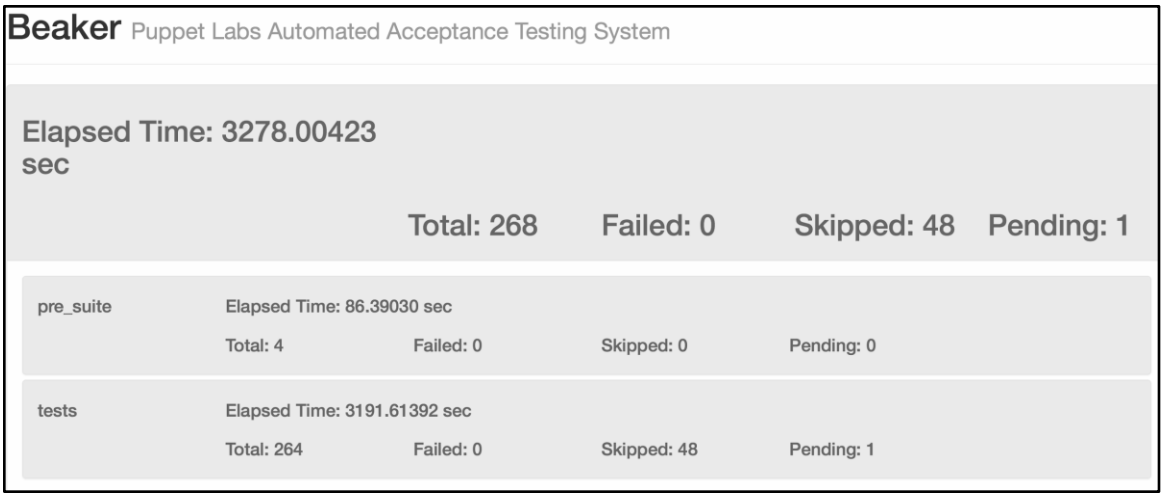
2.2.3 JUnit XML And Web Interface

Due to the difficulty of sorting and interpreting the amount of information captured in a Beaker test run all of the collected log information is converted to JUnit XML format (a popular Java unit test framework). This format can be interpreted by a number of third party tools, including a Jenkins plugin. The Beaker team has also created a web based interface using Bootstrap (a popular HTML, CSS, and JS framework for developing web projects), which is included in Beaker.

The JUnit web interface color codes test output green, orange and red for passed, skipped and failed/errored tests, respectively. The test execution log is initially collapsed, but can be expanded by clicking on a specific test.

Post testing you can view this interface by loading the file `./junit/log/#{HOST_YAML_FILE}/#{TIME}/beaker_junit.xml` in your web browser.

First look:



With `agent_disable_lockfile.rb` collapsed:

With `agent_disable_lockfile.rb` expanded:

3 The Many Platforms Problem

Puppet supports a wide variety of platforms - from Red Hat, Ubuntu, and OpenBSD to Windows, AIX, and many more. Consider doing a simple task such as creating a temporary file.

On a Unix or Linux system it would look something like:

```
$ mktemp -t tmpname.XXXXXX
```

On AIX it would be:

```
$ rndnum=${RANDOM} && touch /tmp/tmpname.${rndnum} && echo
/tmp/tmpname.${rndnum}
```

On Windows:

```
> echo C:\Windows\Temp\tmpname.%RANDOM%
```

An automated test requiring the creation of a temporary file on a SUT would be:

```
if platform =~ /unix/
  # unix only code
elsif platform =~ /aix/
  # aix only code
elsif platform =~ /windows/
  # windows only code
else
  # some fallthrough here
end
```

The code quickly becomes difficult to read and maintain. Any new platform will require the rewrite of all tests to ensure that functionality is preserved.

3.1 The Beaker Host Abstraction

Beaker wraps all of this complexity in a host abstraction. Test authors do not have to consider what platform a test is to be executed on, they only have to consider what part of the software they are attempting to exercise.

For temporary file creation, Beaker reduces the test code to:

```
host.tmpfile('tmpname')
```

Behind the scenes Beaker does the right thing per-platform. If more platforms are added Beaker can be updated without having to revise individual tests.

3.2 The Beaker Host Inheritance Architecture

To manage the many platforms problem, Beaker depends upon a host object inheritance architecture. Each Beaker host is described in a YAML-formatted host configuration file provided by the tester. A host definition is required to contain a platform string that describes a supported platform type. The platform string determines the host object that is created to handle interaction with that node. The individual host types (windows, pswindows, unix, freebsd, aix, mac) inherit the basic host operations (ssh connection,

scp, parameter storage) and then provide their own appropriate methods for platform dependent methods (temporary file creation, package installation, environment variable creation, etc).

An example `hosts.yaml` file:

```
HOSTS:
  node1:
    roles:
      - master
        - agent
        - dashboard
        - database
    platform: el-6-i386
    snapshot : clean-w-keys
    hypervisor : fusion
  node2:
    roles:
      - agent
    platform: windows-2008r2-x86_64
    snapshot : clean-w-keys
    hypervisor : fusion
```

The platform for `node1` is `el-6-i386` and the platform for `node2` is `windows-2008r2-x86_64`. Individual, platform-specific host objects are created by comparing the provided platform string to known, supported operating system types.

Host creation based upon provided platform string:

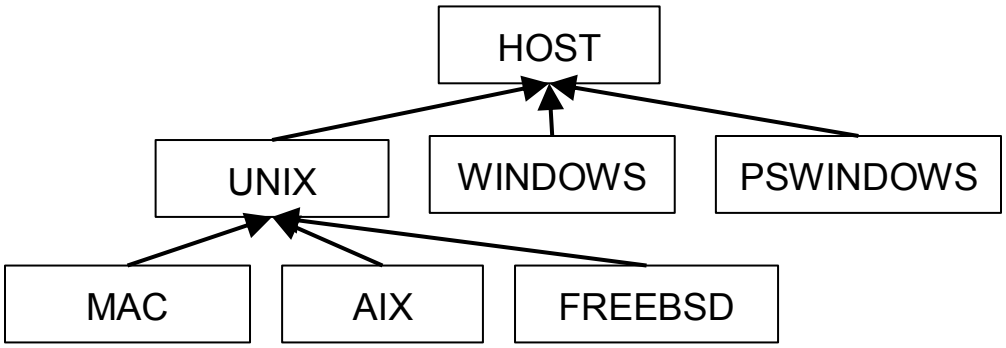
```
def self.create name, host_hash, options
  case host_hash['platform']
    when /windows/
      cygwin = host_hash['is_cygwin']
      if cygwin.nil? or cygwin == true
        Windows::Host.new name, host_hash, options
      else
        PSWindows::Host.new name, host_hash, options
      end
    when /aix/
      Aix::Host.new name, host_hash, options
    when /osx/
      Mac::Host.new name, host_hash, options
    when /freebsd/
      FreeBSD::Host.new name, host_hash, options
    else
      Unix::Host.new name, host_hash, options
  end
end
```

The node defined as `el-6-i386` will become a `Host::Unix` object, while the node defined as `windows-2008-x86_64` will become a `Host::Windows` object.

There are currently two Windows style objects: `Host::Windows` and `Host::PSWindows`. `Host::Windows` objects are assumed to run on Windows nodes that have Cygwin (a Unix-like environment and command-line interface for Microsoft Windows) installed, while `Host::PSWindows` nodes rely on native PowerShell commands.

The Beaker inheritance structure makes it simple to add new platform types. Simply inherit from the appropriate location – either from the base `Host` object or one of its children – and then override any platform-dependent code.

The Beaker Host object inheritance architecture:



3.3 Not Everything Can Be Platform Agnostic

Some tests are simply too platform specific to be shared. Maybe you are testing adding registry entries on Windows, which has no analog on non-Windows systems. For these situations, the Beaker DSL provides a custom-built method: *confine*. When included at the beginning of a test, *confine* informs Beaker in which situations an individual test can be executed. It is the exception to the 'write once, run everywhere' Beaker motto.

Confine to only Windows SUTs:

```
confine(:to, :platform => 'windows')
```

Confine to everything except Windows SUTs:

```
confine(:except, :platform => 'windows')
```

4 The Many Hypervisors Problem

Acceptance tests are executed for a variety of reasons:

- Locally, on a developer's box to ensure that their new code does not cause a regression
- Within a continuous integration pipeline
- As part of scale/performance testing
- At a customer site to diagnose local failures

Each of these scenarios has different requirements for the SUTs. Should they be local for offline testing? Should they be in an internal network for speed and control? Should they be in a cloud provider for mass provisioning? Should there be a combination of local, internal, and cloud SUTs?

Beaker solves this issue by supporting a variety of provisioning techniques from local (VMware Fusion, Vagrant), to internal (VMware vSphere, OpenStack) to cloud (Google Compute Engine, Amazon EC2). The tests are independent of the location that they are executed; the only difference is in the configuration information provided to Beaker at execution time.

4.1 The Beaker Hypervisor Abstraction

The YAML formatted host configuration file provided by the tester also contains important information about where SUTs are located, how they can be accessed, and what configuration steps are necessary for them. In the previous example the hypervisor string is *fusion* for both SUTs.

At runtime Beaker sorts each host into the correct hypervisor bucket based upon the provided hypervisor string. The Hypervisor objects contain all the necessary API calls to provision new SUTs, get them running, configured, and ready for testing. A Beaker user does not need to know the complexities of the APIs of each individual hypervisor - they are responsible for providing some basic hypervisor-specific parameters and Beaker handles the rest.

Beaker code creating Hypervisor objects based upon user provided hypervisor string:

```
        def self.create(type, hosts_to_provision, options)
          hyper_class = case type
                when /^aix$/
                  Beaker::Aixer
          when /^solaris$/
                  Beaker::Solaris
<snip> . . . . more buckets here . . . . . . . . . . . . </snip>
          when /^openstack$/
                  Beaker::OpenStack
           when /^none$/
                  Beaker::Hypervisor
           else
                  # Custom hypervisor
                  begin
             require "beaker/hypervisor/#{type}"
            rescue LoadError
              raise "Invalid hypervisor: #{type}"
                  end
                  Beaker.const_get(type.capitalize)
        end
```

4.2 Build Your Own Hypervisor

Beaker includes a generic Hypervisor object. To add a new hypervisor method simply inherit from the Hypervisor object and override a handful of methods.

A new, barebones hypervisor:

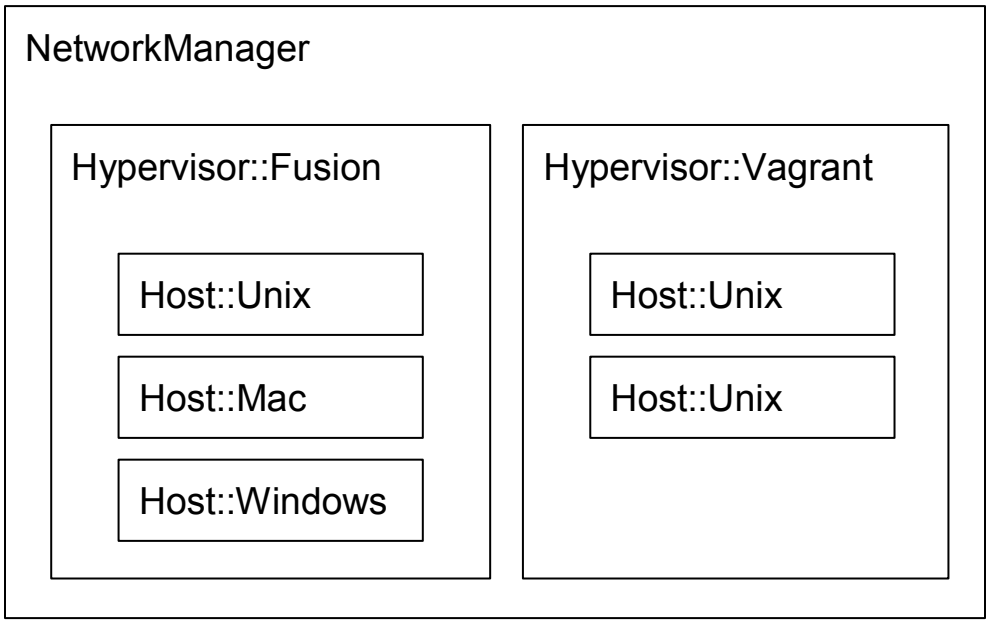
```
module Beaker
  class NewHypervisor < Beaker::Hypervisor

      def initialize(new_hosts, options)
        @options = options
        @logger = options[:logger]
        @hosts = new_hosts
      end

      def provision
        # create your hosts here, once complete should be
        # able to log in by hostname or vmname or ip
      end

      def cleanup
        # cleanup hosts here
      end
  end
end
```

The process is simple enough that from an initial set of 7 hypervisors (Vagrant, VMware Fusion, VMware vSphere, Amazon EC2, vCloud, and custom support for managing Solaris and AIX boxes) two years ago, Beaker now supports 16 – including Docker, OpenStack, and Google Compute Engine.

4.3 The NetworkManager Container

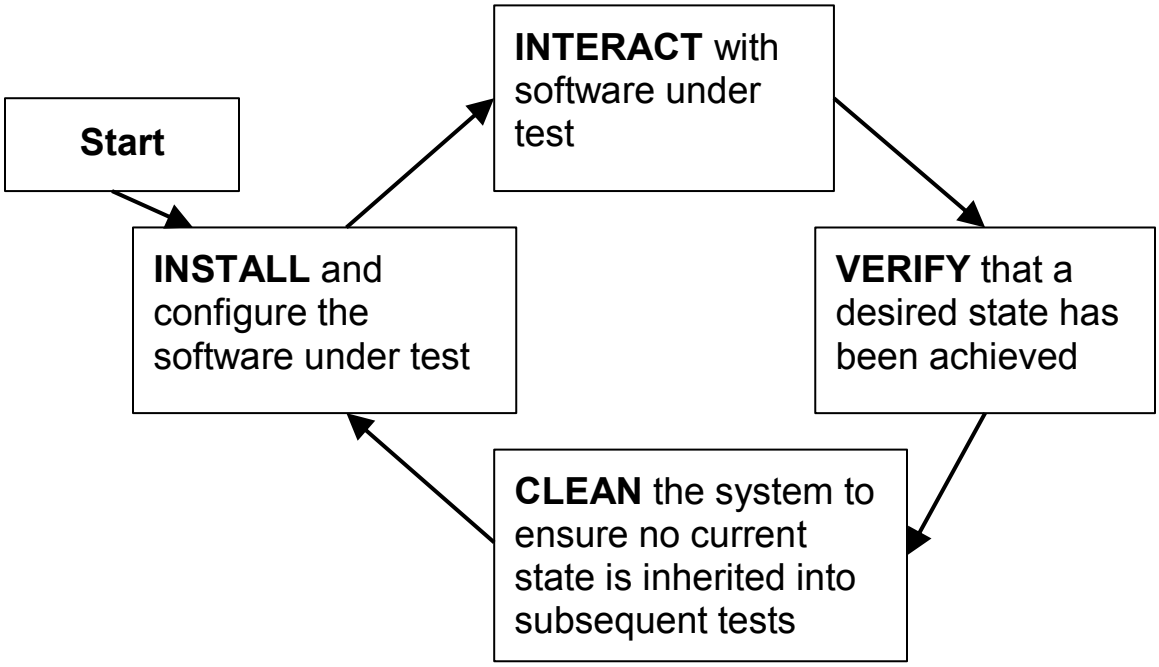
To manage sets of hosts provisioned by different hypervisors, Beaker uses a NetworkManager object which holds the sets of currently active Hypervisors. The Hypervisor objects in turn hold groups of Host Objects. The NetworkManager is responsible for triggering provisioning, configuration, validation, and cleanup on it's hypervisors - which in turn performs those tasks through the appropriate API calls to individual virtualization systems.

An example of the Beaker network management layer:



5 DSL

Acceptance testing follows a consistent pattern:



Each stage of testing contains steps and groups of related steps that can be bundled together into shared methods to accomplish useful tasks. The bundles of steps are tested and versioned in Beaker to make up the Beaker DSL (domain specific language). Instead of having to update tests upon change in an installation procedure, command line interface or API, Beaker can be patched, tested, and released with confidence – and existing tests can continue to run and produce useful information.

Consider installing Puppet Enterprise on a series of hosts. Depending on the version of PE, there are significant differences in the installation steps. Within Beaker this is captured in the Beaker DSL method `install_pe`. Many suites of Beaker tests start with a simple pre-suite:

```
test_name "Installing Puppet Enterprise" do
  install_pe
end
```

The `install_pe` method itself is over 600 lines of code and includes logic for

- determining the version of PE to install
- downloading the correct, platform-specific version of PE to each node
- running the installation scripts
- performing setup and configuration of the installed software

The Beaker DSL is made up of over 130 methods that wrap useful tasks. These include general purpose methods to copy files to and from hosts, run scripts on hosts, install packages on hosts, or run arbitrary commands on hosts. Puppet specific DSL methods include installation (FOSS or Enterprise), service restart and module installation.

The Beaker DSL is a powerful tool to make tests easier to write, easier to read, and easier to maintain.

6 Open Source

The Puppet Forge is a repository of modules that are contributed by members of the Puppet community, Puppet Labs, and other software companies. It currently hosts over 3,300 modules for download and use by Puppet users, both commercial and open source. Modules are self-contained bundles of code and data that wrap functionality – such as the puppetlabs-apache module that installs, configures, and manages Apache virtual hosts, web services, and modules.

For modules to be useful for Puppet users they must be dependable, which means they must be tested. Module testing has the same set of challenges as Puppet testing itself.

From the beginning of the Beaker project it was clear that any automated testing solution chosen by Puppet Labs would need to be available for all to use. Beaker's first 0.0.0 release was on August 20, 2013 under the Apache License, Version 2.

7 Adoption

Significant effort was involved in developing a Puppet Labs built acceptance testing tool. To make the choice to build-your-own is always risky – it takes time to design, build, test, and maintain. But, by taking on the challenge, we've been able to build a system that makes test authoring simple. People with minimal scripting experience can write meaningful, cross-platform tests. Those tests can be applied to both the current software and software in the future.

Beaker has been adopted both internally and as a community automated acceptance testing tool. Beaker ensures that Puppet Labs software is dependable and reliable – testing is integrated into the development process from initial planning to successful release. It is used for Puppet testing, both FOSS and Enterprise, module testing, and acceptance testing of additional Puppet Labs projects.

7.1 Beaker Tests, By the Numbers

A simple metric of success is the raw number of tests - it shows that people working on diverse projects are able to write tests and that test writing is simple enough that lots of individual tests have been written. Beaker tests by the numbers:

Puppet Labs Project	**Current # of Beaker Tests**
FOSS Puppet (master branch)	278
Puppet Enterprise (4.0)	441
PuppetDB	28
Hiera	6
Facter	23
25 Puppet Labs Supported Modules (includes concat/inifile/stdlib/etc)	351
56 Puppet Approved Community Modules (includes wget/python/etc)	31
Puppet Labs QA team internal test suites	917
	2075

7.2 Beaker Code, By the Numbers

Beaker has a growing community of developers. The Beaker GitHub repository is a busy place. Beaker currently has 3,535 commits by 92 contributors over 143 forks. A new minor version of Beaker is released every two weeks and can see 3,500 to 5,000 downloads.

8 Conclusions

Acceptance testing at Puppet Labs presents a challenging set of requirements. Any testing system had to provide tests that are:

- Shared across a continually expanding set of operating systems.
- Capable of being executed locally, internally or in the cloud.
- Maximize code reuse through creation of a centralized, versioned set of shared methods.
- Contributable by both employees and community members.

Such a system was not available as an out-of-the-box solution and we undertook the creation of the Beaker acceptance testing tool. With Beaker, a single test, without alteration, can be executed on all supported platforms, on SUTs hosted on a variety of hypervisors, against many versions of Puppet and be run internally by employees or externally by community members.

We've learned that the effort placed in tooling is inversely related to the effort involved in creating a single automated test. The better the tooling, the easier it is to test. Tooling reduces the cost of test writing and lead time for test inclusion in automated suites. Having a single, shared automation tool increases engineering efficiency and lets QA spend time on what they are good at – writing high quality, meaningful tests.

Creating a new tool is a difficult task – getting people to use it is even harder. The success of Beaker is in the simplicity of each individual Beaker test. The easier it is to test, the more likely you are to test and the more reliable your software will be. Make testing easy: write once, run everywhere.

RESTful Wrappers

Tony Vu

tony.vu@puppetlabs.com

Abstract

Web APIs are the backbone for our daily Internet life. When functioning smoothly, services go unnoticed as JavaScript, HTML, and CSS paint a magical world on top of them. However, when the services go down, everything comes to a grinding halt. In the fifteen years since Roy Fielding's dissertation on REST [1], there has been significant growth in avenues available for testing RESTful Web APIs. There is now a wealth of products and several competing platforms available to test your own API. However, with this wealth of options comes confusion, as the speed of growth makes today's testing tomorrow's ruinous remains of a once gloriously conceived testing solution.

Assembling and building your own RESTful wrappers for testing services gives you the opportunity to share your tool with other elements of your organization, such as other quality engineers or other development engineers not familiar with your product space. The upfront cost will be higher than implementing a commercial service to test your API, but the resulting library will eliminate the siloing effect of isolated test code and tools, unusable by other members of your organization. For the purposes of this paper, we will consider RESTful wrappers to be a kind of library, and will use the term '"library" and "tool" to mean "RESTful wrapper" unless otherwise specified.

This paper is organized into two sections:

- An argument for quality engineers to create RESTful Wrappers meant to be shared with cross-functional teams.
- An in-depth study of building such a library for RESTful services.

Biography:

Tony began his quality assurance career testing software as a manual tester; after learning how to automate repetitive tasks, he became enamored with the idea of automating all the things. He now spends his work hours as a QA Engineer automating tests at Puppet Labs, and spends his off hours trying to automate everything in his house.

1 Why library building matters

1.1 Quality beyond Testing

It is a virtual guarantee that in any job listing for a Software Quality Engineer you will find the word "test(ing)" peppered throughout the description. "Test Engineer" might be a better title for these people who work under the umbrella of "Quality"; it is commonly believed that it was Microsoft [2] who injected 'test' into a now industry standard title, Software Design Engineer in Test--or SDET--to distinguish automated test writers from manual testers. Indeed, this seems to indicate that the most sought after skill of any Quality Assurance (QA) Engineer is to produce and execute tests.

To test the veracity of this proclaimed virtual guarantee, this paper collected 25 jobs listed for a quality engineer during the month of June 2015. The listings came from a variety of sources including indeed.com, craigslist.org, and glassdoor.com. Each descriptions was analyzed for keyword frequency. In 20 out of the 25 listings analyzed, "test(ing)" was the most frequent keyword, and in 24 of the listings "test(ing)" was listed in the top three. Only one listing did not contain "test(ing)" in the top 5 most frequent keywords. This analysis--admittedly rough and course--indicates that the 2015 quality engineer market prioritizes what the American Society for Quality [3] refers to as "verification and validation".

This should not be interpreted to mean "quality = testing". In fact, Quality Engineering encompasses much more than "verification and validation"; one does not have to search long outside of software quality job postings to see examples of quality engineers engaged throughout the entire Software Development Life Cycle [4]. How the market/engineering organizations came to prioritize testing is not a focus for this paper. Rather, confirmation of testing's dominance and prevalence in current software quality practices gives this paper a platform on how to expand out from a testing heavy environment and provide value beyond validation.

This paper explores expanding that value by building common libraries targeted not only for test writers to employ, but for all engineers to utilize. Specifically, this paper examines important design considerations when building wrappers for HTTP services with a RESTful architecture. Included in this examination is how Puppet Labs evaluated those considerations in the construction of their own wrappers.

1.2 Intermediary users before the end user and test result

With the advent of Agile and the rise of User Stories, tests are often conceived with QA Engineers representing the consumer. QA Engineers are almost always first to evaluate development efforts, proxying the end user. Positive test results are a confirmation that the end user's requirements are met. However, with this emphasis on the relationship between the end user and the software, we neglect the interfaces between separate engineering teams who are often interacting within the same project. Quality testing should be concerned not just with the end user and software, but also these interfaces between disparate teams as well.

The health of these interfaces is crucial in determining the risk of issues arising from integration failures of separate software modules. Well organized engineering projects will immediately see the need for tight integration on separate but highly dependent, linked services; in these scenarios, it is unlikely that two development teams might strongly benefit from a RESTful wrapper, since the integration risk is understood and scoped early on. However, libraries can provide enormous benefits in scenarios where two teams are not highly engaged but still present dependencies that will manifest at an integration level. Despite the best planning, we cannot always foresee these interactions. By building a RESTful wrapper for interaction with other development teams, we can mitigate the cost of these unforeseen collisions. Unless there was a disaster in planning, most of these collisions are shallow in their requirements and should be easily handled with a readily available library.

A library is also an essential way to maintain cohesion as Quality organization. QA Engineers are now commonly embedded on specific engineering teams, often separated physically from other QA engineers. Developing RESTful wrappers as a Quality organization links these QA engineers back together, keeping

their methodologies in sync with each other. A common library is also essential to preventing code duplication and establishing and encouraging best practices amongst the QA engineers. The library itself cannot enforce best practices or prevent code duplication, but it does provide an avenue for these behaviors to manifest.

1.3 Assessing if a RESTful Wrapper is the right solution

The previous section of this paper lays out arguments for the development of a RESTful wrapper. However, a RESTful wrapper is not always the right solution. While this paper certainly advocates for their implementation, there are situations where their creation is not warranted.

- **smaller, undivided work streams**
    - Engineering organizations can vary in size greatly; if you have a team with several sub-teams that work out of band with each other, a library may absolutely be appropriate. However, if your organization is small and the number of disparate teams is minimal, then taking the efforts to build a library may not be appropriate. One advantage of a library or tool is that an increasing number of users using it indicates that the initial upfront cost of developing it was worth it; however, this value is severely diminished if only a minimal number of teams get involved with using it.

- **shallow complexity of requirements**
    - One major goal of your tool should be to reduce the complexity for users not intimately engaged with your team. Gauge the complexity of your software module and the needs for an interface into it; there could be insufficient complexity to warrant the building of a library. Also, consider augmenting the current tooling in place; it may be that the current tool chain can be modified to suit your needs, thus not requiring a new dependency. For example, the tool cURL [5] is installed by default on nearly every Linux distribution and thus already part of the tool chain; instead of pulling in a new dependency, you may find that wrapping cURL will be sufficient for your needs.

- **questionable longevity of project**
    - Creating a library or tool requires maintenance and long standing costs; you should assess the longevity of the software module that you are proposing to build a library for. If the software module or project is at risk, you should consider taking on more temporary measures for testing that do not require such a high, upfront technical cost. This may sound like allowing for technical debt to grow. However, sometimes taking on technical debt is reasonable until you have assurances that building out the library will be supported, both financially and by the teams involved.

It should be noted that the RESTful wrappers described in this paper are generally not considered customer facing, though they may be robust enough for customer use. Their intended target audience is inward facing. However, the RESTful wrappers described in this paper could be excellent candidates for a Free and Open Source Software (FOSS) release, meant to facilitate third party integration efforts. There are several libraries available that perform this function. Some are truly third party implementations, such as the koala ruby library [6] for Facebook. Others are more official, such as the python-sdk [7] owned by Facebook itself. Determining if a FOSS release is suitable for your library is a complex decision and outside the scope of this paper's topics, but should nevertheless be considered when creating your RESTful wrapper.

2. Building out RESTful Wrappers

2.1 Choosing your dependencies

Most languages have standard libraries that support sending and receiving HTTP traffic; in some cases, these libraries may be sufficient for your wrapper's needs. In other cases, you may have to use these standard libraries because of restrictions in your test environment that do not allow for non-standard

library inclusion. However, in most situations and languages, there are several competing HTTP client libraries that you can incorporate into your wrapper. Determining which of these competing HTTP libraries to build off of should be decided with great care. Consider the following factors:

2.1.1 Ease of use vs. complexity of implementation

Many HTTP client libraries are created simply because of poor interfaces into the standard HTTP libraries of the language itself. Some of the highest praise of these non-standard HTTP client libraries is in the simplicity of their implementation. Indeed, that allure can seem quite promising, encapsulating intricate logic away from your own RESTful wrapper and using a cleaner interface.

On the other hand, some libraries come with very powerful features beyond what the standard libraries can offer. Some of these features include the parallelization of requests or the ability to maintain web state--such as maintaining cookies or localStorage. These libraries tend to not sell themselves on their ease of implementation but on their relative strength and speciality when compared to HTTP libraries shipped as a standard component library.

2.1.2 Extensibility and community involvement

Some HTTP client libraries are extensible by design; other libraries are designed without this requirement. Extensible HTTP client libraries are not necessarily better than those designed to be a single solution, but that extensibility is certainly something to consider when building out your own RESTful wrapper. On one hand, managing an extensible HTTP client library and several extensions may actually be less appealing than an all-in-one, closed solution. On the other hand, choosing an HTTP client library that is extensible allows for a wider range of community support, as extensions can live in separate repositories without affecting the core client code.

Community involvement can be measured in several ways, not just in the number of extensions available. An HTTP client library that has many contributors is likely to have been vetted and approved by several sources. Likewise, an HTTP client library that has been downloaded several times suggests that it has been sufficient for several other entities, and may be sufficient for your own RESTful wrapper.

Ultimately, there is no single question that will illuminate which HTTP client library to use. For instance, it is this author's sneaking suspicion that some libraries are downloaded frequently simply for their ease of use, and are only kept as a necessity for dependency compatibility; once embedded into a system, removing your HTTP client library will prove difficult to remove, so choose wisely!

2.1.3 HTTP client libraries used at Puppet Labs

The Ruby programming language is the test language of choice at Puppet Labs with RubyGems [8] the standard for package and library sharing. According to The Ruby-Toolbox [9], there are 10 HTTP client libraries hosted by RubyGems with over 1 million downloads. With such a variety of HTTP client libraries available on RubyGems, engineers at Puppet Labs can choose the library that best suits their needs.

Many projects at Puppet utilize the Ruby gem *HTTParty* [10]; that gem is particularly focused on improving the interface into the standard *Net::HTTP* library that comes standard with a Ruby installation, proclaiming "(HTTParty) makes consuming restful web services dead easy." [11] However, Puppet's utilization of HTTParty is often not for a library to be shared out with other engineers. Rather, HTTParty is employed as a shallow wrapper, mostly as a vehicle to deliver highly crafted, specific traffic to some service under test.

I initially used *HTTParty* to construct my RESTful wrapper to be consistent with the most common usage at Puppet, but found that its focus on easing simplicity into the standard *Net::HTTP* library of Ruby was no longer a compelling reason for its inclusion. Additionally, the singleton class design of *HTTParty* became constrictive in how I could construct my own wrapper; it worked well as a shallow wrapper, but extending *HTTParty* to behave in more complicated patterns and usages became too complex to feasibly maintain for the lifetime of the RESTful wrapper.

After surveying other popular HTTP client gems, I chose the *Faraday* [12] gem based on its pluggable architecture and wide set of gems compatible with it. Since *Faraday* was designed to to be highly extendable and pluggable, I knew that it was unlikely I would find the same restrictions as I did with *HTTParty*. Out of the box, Faraday lacked some of the features of HTTParty, but third party support more than made up for those absent features.

2.2 Creating the initial wrapper base layer

Once you have determined the HTTP client library you wish to use for your RESTful wrapper, you should begin construction by establishing a foundational wrapper layer of RESTful methods, each method mapping directly to an exposed endpoint of the REST service. These base methods should only expose the endpoint and any optional or necessary parameters required for successful interaction. These base methods should not alter the data returned. The response object should always be true to the response from the server and contain all the information returned from the service: headers, codes, and bodies.

How these base methods manifest in your RESTful wrapper can vary greatly. For instance, you may not wish to embed the route to the endpoint at all, forcing the RESTful wrapper implementation to provide the route as an argument at calltime:

```
//strategy 1: code of an implementation
//requiring the route
client.get('/v1/books')
```

In this example pseudo code, it is up to the caller to understand the routes that the RESTful API has exposed. At the opposite end of the spectrum, we can create RESTful wrappers that completely abstract out the routes:

```
//strategy 2: code of an /implementation requiring
//no knowledge of the underlying REST API
client.get_books
```

In this example pseudo code, the caller is not required to know the routes for the resources in question.

Both implementations listed above are valid strategies. When the implementor is required to provide the routes to the RESTful API, the underlying RESTful wrapper becomes more stable. Changes to the API are handled at the implementation level of the RESTful wrapper and not with the RESTful wrapper itself. One drawback of this implementation is that updating the routes to accommodate an update in the REST API will not follow the principle of “Don’t Repeat Yourself”, or DRY. However, a change in the REST API does not necessarily mean an update to your RESTful wrapper.

When the implementor is able to abstract away the specific routes of the RESTful API, the RESTful wrapper becomes more brittle, subject to changes in the API breaking the wrapper. However, changes to the API can be handled at the RESTful wrapper level, making updates more frequent but potentially far more DRY.

If your RESTful wrapper has a simple, non-volatile REST API it communicates with, you may want to consider strategy 1. While this may push some of the responsibility to users to update their implementations of the wrapper when the underlying REST API changes, a simple, non-volatile API should not require constant maintenance, nor should it continually fluctuate in it’s implementation. This is also a suitable strategy when your RESTful wrapper behaves more like a shim[1][2][3] than a library, and doesn’t handle much complexity.

However, if your RESTful wrapper behaves more like a library than a shim, than you may wish to consider strategy 2[4]. In this strategy the RESTful wrapper becomes substantially more than simply a shim; it evaluates logic and extends the interface for the REST service.

2.3 Adding abstraction on top of your base wrapper layer

Layering abstraction almost certainly requires the implementation of strategy two. In theory, it could be accomplished with an implementation of strategy one, though your code would be littered with hardcoded paths throughout all the abstraction, since it would not have a predefined set of base methods available for use and would have to constantly hardcode the paths in.

The most evident use case for an abstracted method is to utilize multiple base wrapper methods to chain together multiple calls to the server. For example, it is common for password resets to occur in two API calls to the server: one call to generate a one-time use token, and a second call passing in that token along with a new password. With only base wrapper methods, we might be forced to write something like:

```
//example 1: Resetting a user's
//password only using base wrapper methods
client_id = client.get_uuid
token = client.get_password_reset_token(client_id)
client.update_password(token, new_password)
```

However, with an abstraction for this, we could simplify it into a one liner:

```
//example 2: Resetting a user's
//password with an abstracted method
client.reset_and_update_password(new_password)
```

Chaining methods in this manner allows for us to eliminate much boilerplate code that would otherwise have to exist if we only used base wrapper methods. However, we should be careful to implement all the calls in base wrapper methods; jumping straight into a more complex method indicates an area of weakness in the testing, where certain endpoints might not have the proper coverage, and are only incidentally covered by these more abstract methods.

Beyond chaining, abstraction allows for the RESTful wrapper to implement arbitrary logic to create methods that might otherwise not exist. For instance, imagine we are dealing with a REST API that does not support filtering query parameters. If we stuck with only the base wrapper methods, we would force the implementor of the RESTful wrapper to search the results themselves:

```
//example 3: Forcing a user to search the results of a GET request
books = client.get_all_books
favorite_book = books.search(“Huckleberry Finn”)
```

A RESTful wrapper could easily support that functionality, implementing the logic to search for the book title in an abstracted method itself:

```
//example 4: Filtering a result on the client side
favorite_book = client.get_book_by_name(“Huckleberry Finn”)
```

These abstract methods don’t have to mimic common RESTful patterns; they could filter on virtually any criteria available.

Abstracted methods proved to be the most popular feature of my RESTful wrapper; more than just simply wrapping a RESTful API in an interface, abstracted methods provided real value to users to make their work easier to get done, eliminating lines of code they would have to otherwise implement on their own. Determining what to abstract can be difficult; it is likely you will have to ask the prospective users of your wrapper what they might need. Alternatively, you could encourage users to write their own abstracted methods and contribute back to the wrapper’s code.

3 Conclusion

RESTful wrappers are an integral part of getting any adoption of a new REST service. While the wrappers are intended to be tools to make that adoption as painless as possible, the wrappers themselves must be well understood and targeted precisely. A poorly thought out wrapper will be disregarded and unused, driving down adoption of that REST API.

Quality Assurance engineers are in the unique position to craft and implement these wrappers. Instead of just proxying the end user, QA engineers can create wrappers that act as bridges between separate engineering teams, easing integration pain points and providing a much needed holistic view.

References

[1] Fielding, Roy. “Representational State Transfer(REST)”, University of California, Irvine. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (accessed July 31 2015).

[2] Wikipedia. “Software Design Engineer in Test”, https://en.wikipedia.org/wiki/Software_Design_Engineer_in_Test (accessed July 31 2015).

[3] American Society for Quality. “Body of Knowledge - Software Quality Engineer Certification - CSQE”, http://asq.org/cert/software-quality-engineer/bok (accessed July 31 2015).

[4] Hutchinson, Allen. Han, Ray. “The difference between QA, QC, and Test Engineering”, Google Testing Blog, http://googletesting.blogspot.com/2007/03/difference-between-qa-qc-and-test.html (accessed July 31 2015).

[5] cURL Library homepage. http://curl.haxx.se/ (accessed July 31 2015).

[6] Koppel, Alex. “Koala Facebook Library”, https://github.com/arsduo/koala (accessesd July 31 2015).

[7] Facebook SDK for Python. https://facebook-sdk.readthedocs.org/en/latest/ (accessed July 31 2015).

[8] RubyGems, https://rubygems.org/ (accessed July 31 2015).

[9] Olzsowka, Christoph. “The Ruby Toolbox - http clients”, The Ruby Toolbox, https://www.ruby-toolbox.com/categories/http_clients (accessed July 31 2015).

[10] Nunemaker, John. “HTTParty”, http://johnnunemaker.com/httparty/ (accessed July 31 2015).

[11] Nunemaker, John. “HTTParty”, https://github.com/jnunemaker/httparty/blob/master/httparty.gemspec (accessed July 31 2015).

[12] Faraday HTTP client library homepage. https://github.com/lostisland/faraday (accessed July 31 2015).

Acknowledgements

A special thanks to Andrea Lemhouse of Puppet Labs who helped collect and analyze quality engineering job postings.

Web Based Automation Framework for Beginners

Scott Rodgers

Scott9Rodgers@hotmail.com

Abstract

Quality Assurance teams are overwhelmed with the amount of testing needed for a successful release. The answer is automated testing but easier said than done. This paper shows how to design and build an easy framework for Automation Testing using Freeware products that beginners can use to rapidly build automation for a web based product.

Upon being hired as a developer in QA to setup automated tests on a product that had been in production for over ten years with no real automated tests the amount of work far exceeded what one person could do. Other QA team members had no experience in writing automated tests. It was important to have a way to leverage their help in getting the work done.

By using TestNG, Selenium, Java, and a basic organizational style of programming to setup a framework other team members were able to quickly create automated tests. A single team member was able to create over 50 tests within 2 days to test navigation and verify the titles of the web pages. These tests used to take an hour to manually navigate the page links and are now are done in less than two minutes.

This class will instruct others on how to setup the framework and will help QA organizations with little or no automation in place to enable beginners to write tests successfully and quickly.

Biography

Since graduating from Cal State Fullerton in June of 1991 with a BS in Computer Science, Scott Rodgers has worked in the computer industry as Technical Support rep., API Support rep., Senior Developer, Build Master, QA engineer, and Automation engineer. He has also worked for a variety of different companies from printers, banks, start-ups, and Family History dept. of the LDS church. He has set-up several different frameworks for automation using Silktest, HttpUnit, and Selenium.

Copyright Scott Rodgers 2015

1 Automation Backlog

Let's presume that you have just been hired by this company that wants to start creating several suites of automation tests and you are the only developer on the QA team. How do you get through the Backlog of tests that need to be coded or written? The Backlog is all the test or test-cases that need to be run to meet the turnaround allotted by the Product Management team. With none of your team members having very much or any coding experience, it leaves you with a dilemma: how to meet this deadline?

One way to meet the deadline, is to have the some of the team members (beginners) write these automation tests. How this possible? With the proper framework and tools that are outlined in this paper, this is possible. Within a few days they will actually be writing fully functioning tests and reducing the Backlog.

At Simplifile we have found that these select tools, TestNG, Selenium, and Java along with the basic organization style of programming have provided the means for our QA team members to come up to speed and write tests that work properly in short order.

2 Freeware Trio

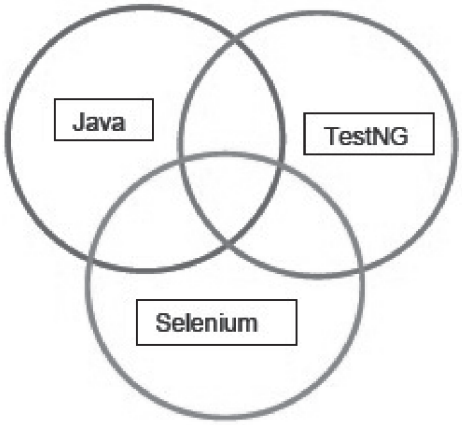
TestNG, Selenium, and **Java** (all freeware products) can work in concert together to create the website automation. Each product plays a specific role. There are many advantages to using this trio, since they work seamlessly together.

TestNG is used to run the whole suite of tests, groups of tests, or a single test. It provides the user great flexibility as to which tests or test that can be run at a given time. TestNG allows the user to create special methods that are run automatically upon start up and shut down of the test(s). TestNG provides the perfect structure to run automation tests. TestNG supports both Selenium Classes written in Java and thus Java itself.

Selenium is used to control the web browser as it accesses the website, which includes the navigating from page to page, entering data, checking for the presence of certain elements that should be contained on a page, and so many more things that should be tested for a given website to validate that it is ready for primetime.

Java is the language that the test code is written in. The tests are executed by Java and in conjunction with TestNG. It leverages the Selenium Classes to perform the necessary steps to complete the desired task being defined in the test cases for testing the website.

The trio provides the complete set of tools for beginners to write web automation that works in days, instead of months or years.



Free Ware Trio 1

This diagram shows the intersection of the trio in which they work in concert together to create the website automation. Let's look more in-depth at what makes each one a powerful part of doing web automation.

3 TestNG

TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers). [1]

At Simplifile we are using TestNG as a framework to run our web automation tests in conjunction with Selenium and Java. Here are the TestNG building blocks that we use.

3.1 Annotations

Annotations are used to denote a method for a given purpose. Annotations also support parameters that enhance how they are interrupted during the running of the test. The following Annotations are only a few of the ones available, but provide the needed functionality required to write effective tests.

3.1.1 Test

```
@Test (groups = { "Nav" })
```

Test is used to denote that a method as an actually test and not a regular method. If the Test Annotation includes the Groups parameter then that test can be executed as a part of that group. There can be single test or multiple tests for that Group. The Groups parameter are used to organize the tests in a given class or multiple classes.

3.1.2 Setup (Before…)

```
@BeforeGroups  (groups = {"LiveColumn1","LiveColumn2","LiveColumn3",”Nav”})
```

The BeforeClass, BeforeMethod, and BeforeGroup provide for different types of methods to be created for doing specific setup for tests.

3.1.2.1 The method annotated with “BeforeClass “will be invoked after the test class is built and before any test method that is defined within the Class is run. This allows those objects that need to be shared across other tests to be created once before any test are executed.

3.1.3 Teardown (After…)

```
@AfterGroups   (groups = {"LiveColumn1","LiveColumn2","LiveColumn3","Nav"})
```

The AfterClass, AfterMethod, and AfterGroup provide for different types of methods to be created for doing specific teardown for tests.

3.2 Asserts

There are many different types of Asserts to choose from when writing tests, as to which one you use will depend on what is to be tested. We have settled on using three of them, which are **assertTrue, assertEquals**, and **assertNotEqual**.

The **assertTrue** is used to verify that Web Elements exist, and for cases when something should contain a certain value. For example a text field should contain “joh” plus “nny” or “n” then the assertTrue will pass.

The **assertEquals** is used when the desired outcome is for the two objects are the same, such as the expected URL and the current URL.

The **assertNotEqual** is used when the desired outcome is for the two objects to not be the same.

How many asserts should be used in each test is based on what is defined in the test case for passing. The test should follow exactly the requirements that are defined in the test case. Since the Beginners aren’t that familiar with what makes a good automation test, this is the principle we follow at Simplifile.

3.3 Results

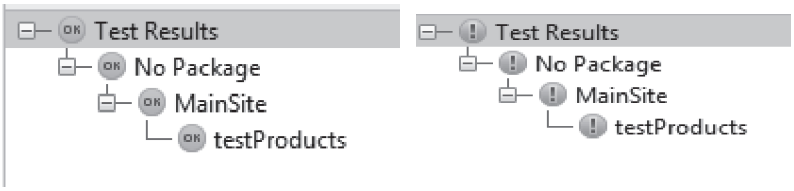
TestNG provides results for each tests run, as to whether it passed or failed. The overall result information will look like this:

```
===============================================
```

Custom suite

Total tests run: 1, Failures: 0, Skips: 0

The Test Results information regarding the actual test run will look like this, green is for passing and orange for failing:



If the test fails there should be some type of output saying what the cause was of the failure. When using asserts it is a good practice to take the time to create the message that can be a part of the assert method. As shown below:

java.lang.AssertionError: https://simplifile.com/e-recording/ vs https://simplifile.com/sf/login/

Expected :true

Actual :false

There is always the possibility that either result could be a false positive depending how the code was written. It is good practice to have someone else do a peer review of both code and results until the beginners have demonstrated that they can determine the difference between a false positive and a real pass or failure.

Using **annotations, asserts**, and the **results as reports** that TestNG provides is the one of the major benefits provided by the Trio. The **annotations** provide the controls by which groups of tests can be run and what setup methods are called to allow the tests to function properly. **Asserts** provide the way to determine if the test passes or fails. The **results** provide the means to determine if a single test, a group of tests, or the suite of tests have passed or failed. The **results** can contain lots of clues for failing tests that will assist in resolving the issue, especially if the time was taken to add messages to the **asserts** in that test.

4 Selenium

Selenium automates browsers. That's it! What you do with that power is entirely up to you. Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) also be automated as well.[2]

Selenium WebDriver; a collection of language specific bindings to drive a browser -- the way it is meant to be driven.[2]

In other words Selenium WebDriver allows the Website or WebApp to be navigated within tests that incorporate it. The navigation is done very smoothly and in a logical way. There are many methods that are object based that leverage language specific bindings that the WebDriver provides. These are the ones that we use to assist our beginners in writing tests.

4.1 FindElement

```
WebElement we = driver.findElement(By.cssSelector("#logo > a > img"));
```

The FindElement is used to locate the First WebElement that matches the descriptor provided by the By.cssSelector(<WebElement Name>). This method returns a WebElement that can be acted upon, with actions like getText or Click.

String value = we.getText;

The String “value” now contains the text from that WebElement.

we.click();

That WebElement received that click and if the onClick method was available it ran the javascript that was a part of that method.

4.2 FindElements

```
if (driver.findElements(By.cssSelector("#menuDiv > iframe")).size()>0) {
```

The FindElements is used to locate a list of WebElements that matches the descriptor provided by the By.cssSelector(<WebElement Name>). This method returns a list of WebElements that can be acted upon. It can also be used to validate if a WebElement is present or not by checking the size of the list. To determine if the WebElement is present the size will be greater than zero.

4.3 Get

driver.get(HTTPS_SIMPLIFILE_COM_SF_LOGIN);

The get is used to load up the browser with the URL passed into the method. The driver is now using that URL to perform any commands requested through the driver methods.

4.4 GetCurrentUrl

String stateUrl = driver.getCurrentUrl();

The getCurrentUrl retrieves the URL for the current page. The driver is currently using that URL to perform any commands requested through the driver methods.

4.5 SendKeys

textWebElement.sendKeys("AutoMation");

The sendKeys will act like it is typing those characters. Text Field WebElements allow for the sendKeys method to be used to fill out forms.

These are the minimal five methods of the Selenium WebDriver that provide the capability to write tests to perform the test cases in a Step by Step fashion. They are used to find WebElements for validation purposes, navigate to or retrieve the URLs, and to provide the means replicate any action a Keyboard could do. This allows the test to perform the actions of a user of the website without actually doing it manually.

5 Java

Java is a programming language that is used by many businesses. It is used because it is very versatile and has a strong structure. Through its wide acceptance there are many examples that can be found on the internet to assist in writing test. Beginners can google the desired functionality they want and quite often find the java code they need. We use java because it is used by the Development team and it is rather easy to learn. It supports all the basic organization style of programing described in the next section.

6 Basic Organization Style of Programming

There are several ways to design the tests: by using Page Objects or straight forward Step By Step. Both have advantages and disadvantages, but for our purposes the Step By Step method is what will be discussed in this paper. It is really easy to create tests with this method. The tests can be quickly written without much thought as to how to organize them, since it follows the natural progression of a test case. Test cases are written step by step order. At a later time, they can be organized by using the Groups as a part of the Test Annotation. Whereas the Page Objects method requires a lot of planning as classes will need to be created for each web pages, in-order to be written correctly. Each class that is created for each page should contain methods to match each function that can be performed on the page. It is a lot of up front work. Since we are looking to produce test more rapidly this method doesn't offer us this ability. Plus there is a high learning curve to write classes for each page.

6.1 Constants

Declaring the most common URLS as a constant provides for easier validations and reuse of those URLS. It is also useful from the standpoint of doing updates, by having to update the URL in only one location.

6.2 Global and Local Variables

Declaring WebDrivers as global variables will allow them to be used by all the tests and can be set to the Local variable in a given test. The Local variables will get cleaned up upon the finishing of the method. The global variables will get cleaned up upon the finishing of the class.

6.3 Methods (Repeatable Code)

Taking code that is copied from test to test is a candidate for becoming a new method which in turn can be called by all those tests that contain that copied code. This is very basic coding principle that makes code more robust. It sometimes takes some re-engineering to create variables that match the different

WebElements that are needed in the newly created method. The re-engineering doesn't need to happen right away. It is important to let the beginners create lots of tests at first to get their feet wet so to speak. By doing so, they will be able to see on a grander scale the reasoning for the creation of methods that consolidate code. It may take time before the beginners can recognize what needs to be a candidate of this process. It is worth the effort to assist them in this process of determining when to construct a new method that replaces the copied code.

6.4 Logic Constructs and Data Types

Logic Constructs and Data Types should be leveraged always when writing code for the simple fact that they are very powerful. The power comes in that the logic constructs provide means to traverse the data objects, perform the same section of code multiple times, perform a certain action based on a value provided, and many more.

The logic constructs could be as simple as a For Loop for an object, For Loop by count, While, Do-While, If, If-Else and many more. These examples can give great structure to the writing of the tests, which ones that will be used should be represented in the test case in the Step by Step method as it defines what needs to be verified.

The Data Types could be as simple as a Hashmap, a List, or an Array to store data for comparison or that needs to be pasted from method to method. The use of data types is a good coding practice and should be implemented whenever possible. It provides the writer of the code the means to evaluate the data being used in the test, which is invaluable when trying to determine why a test is failing.

This basic organization style of programming lends itself to beginners in assisting them with concert concepts to follow while writing automation tests. By creating a template class that contains tests in the sample in the next section our beginners have a guide to writing test within days of starting.

7 Sample

This section of the paper will further show the code that actually tests the Simplifile.com Main Page, along with an image of the page.

7.1 Main Web Site Page Layout

Black triangles are to show the links that need to be present for the test to pass and then tested to verify that they work in another test.

7.2 Test Code for If Links are Present

```
   @Test    (priority = 4,groups = { "Nav","SAMPLE" })

   public void testMainPageLinksRPresent(){

       //validate that links are present

       assertTrue(driver.findElements(By.cssSelector("#post-27 > section > div.home-banner > div > div > div > div.home-banner-collaboration.fourcol.first > div.home-banner-person > a > img")).size() > 0);

       assertTrue(driver.findElements(By.cssSelector("#post-27 > section > div.home-banner > div > div > div > div.home-banner-erecording.fourcol > div.home-banner-person > a > img")).size()>0);

      // There are more links that are being tested in this method, shown by the yellow highlighted links, but these two lines are enough to get the idea across.

   }
```

This is the best way to see if the links are present on the page that is outlined by the test case and then report the result. The actual testing for the link is done by the checking the size returned by the findElements for that WebElement Link. In order to return true, the size must be 1 or greater. There

should only be one link found by that selector, otherwise there could be duplicate links on that page. The assertTrue is used to report whether the link was found or not.

We chose to tests all the links of this page at once, instead of creating 12 additional tests that just do one link at a time. It was a savings on the number of tests that needed to be written and executed, with little or no cost to debug any issues with missing links.

7.3 Test Code for If Link Redirects Work

```
@Test    (groups = { "Nav" })

public void testProducts(){

    //validate that links go to correct location

    goToPage("#menu-item-34 > a",HTTPS_SIMPLIFILE_COM_E_RECORDING);

    goToPage("#menu-item-135365 > a",HTTPS_SIMPLIFILE_COM_COLLABORATION);

    goToPage("#menu-item-135364 > a",HTTPS_SIMPLIFILE_COM_POST_CLOSING);

}
```

Calls method goToPage() for each link and passes in the desired redirected page by URL. This method performs the same test like “If Links are Present”, but actually clicks on the link and then verifies that the new URL matches what the newly re-directed URL Then reports if all these checks passed.

```
private void goToPage(String link, String newURL) {

    assertTrue(driver.findElements(By.cssSelector(link)).size()>0);

    driver.findElement(By.cssSelector(link)).click();

    assertEquals(driver.getCurrentUrl(),newURL,driver.getCurrentUrl()+" vs. "+newURL);

    home();

}
```

Calls method home(), which will always redirect to the Home Page by URL. This method contains a special case check for a page that doesn’t follow the rest of the website format style and has no Home link in order to navigate back to the Home Page.

```
private void home(){

    if(!driver.getCurrentUrl().equals(HTTPS_SIMPLIFILE_COM_SF_LOGIN)) {

        driver.findElement(By.cssSelector("#logo > a > img")).click();

    }else if (!driver.getCurrentUrl().equals(HTTPS_SIMPLIFILE_COM)) {
```

```
        driver.get(HTTPS_SIMPLIFILE_COM);
    }
}
```

These are just two samples of test code that have been written to test links on the home page for Simplifile.com. These tests embody the Freeware Trio along with Basic Organization Style of Programming which have been outlined up to this point. Each item outline from 2 through 6 has been addressed in these two tests at least once. By using these two test as a pattern in our template class, our beginners can write their own tests very quickly.

8 Summary

The Freeware Trio are widely accepted in the Testing Community as being standard tools. They all work well together in building a highly functioning Automation Testing framework as outlined in this paper. There is no financial obligation in adopting these as a part the testing arsenal. There is a low learning curve. Through leveraging TestNG, Selenium, Java, and basic organization style of programming the most novice QA team member can use a Java Class Template as a pattern to write automation tests within days. Taking a mostly manual QA team to a new productivity level through writing automation using Freeware is now possible by using these practices that were outlined in this paper.

9 References

[1] TestNG, Cédric Beust (cedric at beust.com) “Introduction” Current version: 6.9.4, Created: April 27th, 2004, Last Modified: May 9th, 2015 http://testng.org/doc/documentation-main.html#introduction (accessed June 9,, 2015)

[2] Selenium, http://www.seleniumhq.org/ (accessed June 9,, 2015)

Obtaining True UI Automation Speed

Sam Woods

sam.woods@puppetlabs.com

Abstract

How many people get frustrated after spending hours writing tests, then even more time debugging failures before finally being able to determine a root cause or fix? I have spent a lot of time thinking about this and experimenting with ways to increase efficiency for these 2 aspects of UI automation. Through trial and error throughout my career, I have concluded that there are simple and effective ways to increase velocity of test authoring and root cause analysis of test failures, which are often overlooked. Even teams with relatively mature automation processes can be blissfully unaware of the opportunities they have to make major improvements to their automation velocity.

Besides the well known Page Object Pattern [1], we will walk through effectively using setup and cleanup functions, how to properly structure helper functions, how to model data consumed by the application, and more. Finally, techniques for increasing the speed of determining the root cause of automation failures will be demonstrated, such as: built in logging, taking screen shots on failure, useful integration with your defect tracking tool, and determining which failures are likely to be the same root cause through stack trace analysis. By including these relatively simple strategies in your own automation, you could increase your efficiency and be able to add more automated tests while spending less time debugging failures.

Biography

Sam Woods recently joined Puppet Labs as a Senior Software QA Engineer with 17 years of QA and automation experience, including 13 years at Microsoft. He is a self taught developer who has evolved to the point of helping multiple mid to large size organizations create automation strategies, build strong automation teams, create the necessary automation infrastructure to support the strategies, and then execute on them. He has presented on UI automation at the local Portland Selenium meet up and presented multiple times to audiences as large as 300 people within the companies he has worked for. He currently has the 5th highest all time ranking on the SQA Stack Exchange [2] forum. Sam gets excited about improving efficiency through both automation and best practices, as well as getting into the weeds with automation and automation infrastructure. Outside of work, Sam enjoys singing karaoke and spending lots of daddy time with his two young daughters.

1. Introduction

Are you working on, or have you ever worked on automation that just seemed un-maintainable? Automation you spent more time diagnosing and fixing problems with than you spent adding new test cases? Have you had automation that was so unreliable that you couldn't trust the results even when tests passed, or had large portions of the automated tests failing nearly every run, seemingly at random? It's enough to make even the most stoic automation engineer pull their hair out in frustration.

Many have attempted to find solutions and failed, and many ignore the problems and continue writing new tests while amassing additional technical debt. The title of this paper refers to the fact that many teams writing UI are focusing on solving the wrong problems, and for example care more about the execution speed than test authoring speed. But, don't fear! While these symptoms can be daunting, they are often simpler to find and fix than you might imagine.

I will provide solutions and examples that will help you increase the speed of writing automated tests as well as decreasing the time to determine the root cause of failures (the two areas I refer to with 'true UI automation speed')

2. The Problems

The first step is a diagnosis. What exactly is the root cause of these symptoms? The 3 most common problems that I have observed that result in the symptoms stated above are:

- Instability in the product itself
- Too many automated tests
- Unreliable automation

Typically, you will see some combination of the 3 problems listed above. I will outline these problems below, but the primary content of this paper will be directed towards the third issue, unreliable automation.

2.1 Instability in the product itself

- Constant and major changes to various portions of the user interface
- New features that required rework of old features.
- Major updates to or overhauls of existing features.

If there is a lot of instability in your product - especially if you are working in an agile environment - your team should ask the following questions of themselves:

- When we implement features, are they really "done" done?
- Are your customers actually happy with or benefiting from these changes? Or perhaps they are just as frustrated as the QA staff that things keep changing out from under them?

If you answered yes (and really feel like that's an honest answer) to both of these questions, I would challenge you to gather some actual data on the above points and review it.

2.2 Too many automated tests

- A large automation code base with many hundreds or thousands of tests.
- Automated tests written years ago that aren't updated unless they break.
- No clear idea of what kind of test coverage you get by executing the tests.
- Lots of duplicated or largely equivalent tests.

Often times QA teams will horde tests for no apparent reason. Perhaps out of nostalgia, or even fear? If you are suffering from an overwhelming pile of unmaintained automated tests, I would pose the following questions:

- What value are you getting out of your automated tests?
    - Is executing these tests allowing you to skip manual testing for entire features?
    - Are they an immediate and reliable indicator of the quality of your application?
    - Is every test validating a unique aspect of the application or feature?
    - Can you easily determine the answers to the above questions?
- How long is the feedback loop?
    - Is running 5000 tests (or insert your own number of tests here) even manageable when there are failures to investigate?
    - Can you provide valuable feedback on a build within an hour (or even 5-10 minutes) of when new code is checked in and deployed? (Don't forget to include how long it takes you to triage failed test results)

If the answer to any of the above questions was no, then you need to face your fear of cutting out the crap, wade in and start shoveling. Take a rational approach to trimming down the volume of tests. Which tests are virtually equivalent and can be consolidated into a single test or just removed (see section 4.3 about using code coverage results as one avenue to determine this)? Which tests cover the highest priority or most used customer features? Which test failures would be the highest severity and have the most impact to customers? These tests should be prioritized, and the rest either de-prioritized, consolidated, or removed.

Do, however, take caution in your diagnosis. I have seen teams with a couple hundred tests come to the same conclusion as teams with a few thousand tests, and while you may have some unnecessary or inefficient tests, the primary issue may just be…

2.3 Unreliable automation

- Test results that are unclear, incorrect, or not reproducible.
- Tests that fail when they should pass, or vice versa.
- Flaky tests that return seemingly random results.
- Tests that are brittle and break due to even minor changes in the application.
- Tests that pass when the site is running fast, but not when it takes a couple seconds longer to respond.

I will be detailing ways to increase the speed of automated test authoring and troubleshooting, while simultaneously improving reliability and reducing the overall cost of maintenance for the remainder of this paper. All examples in this document are pseudo-code and are not in any particular language.

3. Solutions

We'll start with some strategies to improve the velocity of test case development, and overall quality and maintainability of your automation:

3.1 The Page Object Pattern

Many people believe the page object pattern is synonymous with Selenium WebDriver's [3] implementation of it, and I think that is a mistake. I personally dislike the PageFactory, and would much rather see lazy initialization of the web elements allowing you to simply create a page class and create instances of all of the element objects right in the constructor. The concept of the page object pattern is

simple, and it is extremely important when it comes to increasing efficiency of writing and maintainability of automated tests. It is the minimum bar for increasing maintainability of automation.

Don't over-complicate it. The basic idea of the Page Object Pattern is that you want to separate your test cases from the definitions of your pages. That's it. You can use whatever implementation you're comfortable with, as long as you're doing that.

Bad Example:

```
public void Test1(){
   FirstNameTextBox = new WebElement(…);
   FirstNameTextBox.SendKeys("John"); // Hard coding name values
   LastNameTextBox = new WebElement(…);
   LastNameTextBox.SendKeys("Doe"); // Hard coding name values
}
```

Good Example:

```
public class MyPage {
   FirstNameTextBox = new WebElement(...)
   LastNameTextBox = new WebElement(…);

   // A new helper function to enter name, avoiding hard coding values
   public void EnterName(String FirstName, String LastName){
      FirstNameTextBox.SendKeys(FirstName)
      LastNameTextBox.SendKeys(LastName)
   }
}

// Somewhere in your test class:
// You will need to initialize the page class somewhere,
// usually in a constructor, or setup method.
public void Test1(){
   myPage.EnterName("John", "Doe")
}
```

This is critical to any UI automation. Without this abstraction in the example above, if anything at all changes about entering your name on the page, fixing your automation would entail finding all instances where you are referencing those elements and updating them - assuming you are even using the same names or locators for those elements in the first place. You may have them defined multiple different times using different criteria and different names, making them all but impossible to find without executing your automation, and then triaging failures that you fix one at a time. By contrast, if you use the good example, if something about the form to enter a name changes, you update the one method in the page class and fix all of your tests at once.

3.2 Modeling User Defined Data

This is an important one. In most modern web applications there are complex data constructs at play - maybe a user profile, shopping cart, array of messages, or any other number of logically grouped data sets. If anything like this exists in your web application it needs to be modeled in your automation, just how developers model it to keep it maintainable within the application itself. Modeling it is usually not a difficult task, you look at how and where the data shows up in your web site, and you create a simple class structure to make it easy to enter and validate the data. I'll use a relatively simple one for an example:

```
public class User {
   string FirstName
   string LastName
   string Address1
   string Address2
   string State
   int ZipCode
   string PhoneNumber
   string UserName
   string Password

   // Note the use of optional parameters with default values in the
   // constructor, most languages support this.  If you're unfamiliar
   // with it, look it up for your language of choice.
   public User(firstName = "testFirst", lastName = "testLast"…){
      FirstName = firstName
      LastName = lastName
      … // Continue assigning the class properties to the method argument values.
   }
}
```

The above is very simple, but it amazes me how many times I see hard coded values all over the place in automation. The above data structure now ties in easily with and simplifies your page object helper functions as well. Imagine a single helper function called “SignUp” that takes a User object as a parameter. If a new field for Gender is added to the signup page, or a phone number removed, or any other change at all, it will be very simple for you to modify that one SignUp function and one User class and all of your existing automation will again function normally. Now let’s say you add various payment methods to your site, and you want to be able to tie them to that user. Easy, create a PaymentMethod class that can be either a credit card, or Paypal, or whatever else you may accept for payment and make it a property of the User class. Maybe most Users won’t have payment methods on file? No problem, make it null by default and have your helper functions ignore it if it is null.

3.3 Helper Functions

Helper functions are a simple concept, but can be tricky to get right. Basically, helper functions fall into two categories. They are functions to either simulate user behavior, or validate expected results. I have seen people create helper functions called “EnterTextIntoNameField”, or “ClickSubmit” where all they are doing inside the helper function is a single call to a webdriver method or property. This may be overkill, especially if these are all part of a form with a dozen other fields on it. It may also be perfectly reasonable in other circumstances, if say you wanted to include some additional logging, or needed to do some tricky waiting for certain parts of the page to load prior to entering text, or a number of other things that can complicate a relatively simple thing like entering text or clicking a button.

Common areas of confusion:

- How much should you put into a single helper function? Can you nest smaller helper functions into one larger one?
- What should the scope of a helper function be?
- Can a helper function span multiple pages?
- Should you have separate functions for entering data and submitting data?
    - What if you need to test a specific field in a form, but want to automate entering data into all of the other fields?
    - What if you are purposely entering bad data to validate error messages?
- Should the helper function throw an exception if it doesn’t complete successfully, or return a Boolean value to assert on?

- Should you log what you're attempting to do, as well as the result?
- What data, and in what format do you pass in to a helper function?
- Should you return the page object of the page the helper function transitions to?

Many of these are up to the preference of the developer, but this is what I have adopted as best practices, and have found to work well:

- Helper functions can be as narrow or deep as they need to be. If it makes sense to break a registration page up into "EnterCredentials", "EnterProflileInfo", "EnterBillingData", etc, that's fine, but also include a "Register" method that encapsulates all of those to make it easy from a test case, especially if most of the time you won't need that granularity except one off tests.
- Parameters to helper functions should typically be a single object, whether that is a string for a method such as SelectEmailByTitle(String title) or an instance of a custom class for a method such as Register(User user).
- Helper functions can definitely span multiple pages, especially in the case of multi-page forms, or navigation methods where there is no direct way to navigate to the other page.
- When entering and submitting data, have a single method to both enter and submit the data.
    - Include an optional Boolean parameter "submit" and have it default to true. If you are writing field validation tests for the form, you can now fill out most of the form, then enter specific data for the one field you care about before submitting.
    - Include a Boolean parameter "success" that defaults to true, so you can validate errors if necessary by asserting on the result of the method. If you don't expect an error, validate in the helper method that you arrived on the correct following page. If an error is expected, create a separate method to validate the error message.
- I am a fan of throwing exceptions in helper methods and having them bubble up, unless the helper method is specifically a validation helper method. Here are some reasons why:
    - Many test runners differentiate between an exception (something broke before your test even got to the point of validating what it was intended to validate) and an assertion failure (The thing the test was trying to validate was broken). This is a very useful piece of information when it comes to determining the root cause of a test failure. With this information, you will know definitively if the test itself failed, or if some step in the setup steps failed.
    - In the case of a validation helper method such as "ValidateMissingInfoMessage", you should return a Boolean value signaling success or failure, which could then be called directly from an automated test case where you assert on its response.
- My preference is having a static class with a list of page instances rather than returning the page object instance in helper functions. This is entirely preference. Your automation will be successful regardless of which approach you choose. I like the readability of a test case a bit better using a static class, but if you like having the type of page returned by the method to make it simpler to determine where you are supposed to end up, that works too.

Here is one example of what I would consider a good helper function, making use of the User class we saw in the "Modeling User Defined Data" section:

```
public class RegistrationPage{
    // All elements defined with locators
    //(CSS or XPATH definitions of the element)

    // Use our custom Users class with a default user already set up
    public RegisterNewUser(User user = Users.defaultUser,
```

```
        Boolean submit = true, Boolean success = true){
        log.info("Registering new user " +
        user.firstname + " " + user.lastname)
        try {
           if (user.firstname != null){
              firstNameTextBox.text = user.firstname
           }
           if (user.lastname != null){
              lastNameTextBox.text = user.lastname
           }
           // etc...

           If (submit){
              log.debug("Submitting registration.")
              registerButton.click()
           }
        }
        catch (exception ex) {
           // Whatever you need to do to log the caught exception details.
           log.error("User registration failed!", ex)
        }
        log.debug("Completed registration")
        if (success){
           // Make sure we're actually on the welcome page
           welcomePage.verify()
        }
    }

}
```

3.4 Effective use of Setup and Cleanup functions.

Setup and cleanup functions can save you from copying a lot of boilerplate code for similar test cases. They can also be used to decrease execution time and increase maintainability. Finally, it also makes it very clear if there is a failure in a shared setup or cleanup function that the actual functionality the test is asserting is not the culprit. This makes it simple to group common failures together into likely buckets of the same root cause, which can turn investigating 30 test failures into investigating only a handful of unique root causes. More on this later.

Because setup and cleanup functions are critical and need to be as reliable and execute as fast as possible, it is not always necessary to use the UI at all for them. Let's say you have a test where you are associating 2 user accounts in some way, maybe making one user a child of another user. You shouldn't care about going through the UI to create two new users prior to associating them, they just need to exist so you can test the critical functionality of associating the two users in the UI. This can be accomplished by making HTTP requests directly to the web server. If your team is already using a tool for performance testing, you likely already have most of what you need to construct these requests. If not, and you are starting from scratch, there is a little more to it, but it is worth the effort to invest in automating the web requests directly. It is even possible to send an HTTP request to authenticate, use Selenium (or whatever automation tool you are using) to set cookie values for authentication and then navigate directly to a page on your site, already logged in – without even going through the UI at all!

Whether you make use of HTTP requests in the setup and cleanup functions or not, here are the best practices I have adopted in terms of how and when to use project, class and test level setup and cleanup functions.

- Project level setup and cleanup

    - The project setup method should setup anything that will be used in all tests. For example, it may setup and start a logger, or create a Selenium driver instance. It could also initialize any shared objects such as default users, or create instances of all of your page classes, etc.
        - One note about Selenium driver instances – if you want to execute your tests in parallel and are using a Selenium Grid, or cloud based Selenium Grid such as Sauce Labs [4], you may have to quit and create a new driver instance for each test, in which case those should live in the test case setup and cleanup.
    - Project cleanup similarly can be used to quit selenium, stop loggers and summarize logs, destroy objects in memory, etc.
    - In the case where the test execution framework does not have a project level setup and cleanup, I commonly have a base test class that I inherit from and then execute the setup and cleanup from the super class. Different test runners handle inheritance differently, so refer to your documentation or experiment a bit to ensure it is working the way you intend it to.
- Class level setup and cleanup
    - Group test cases together by feature or functionality into buckets of similar tests. For example, you may have a class of tests that simulate users making purchases in a shopping cart and because every test needs a user to log in as. You can create 1 user in the class setup and use that user in all of your tests.
    - The cleanup may delete the user, or purge the purchases from the database.
- Test setup and cleanup
    - Often, you should have a test method in a base test class that logs into the application and does any other boilerplate setup that may be necessary.
    - The test setup is also where you can include common steps for the particular class of tests. In the example of a shopping cart, your test setup could log in and navigate to a product search page prior to every test beginning.
    - Avoid using the test cleanup function to get back into a good state for the next test case to run, there are too many potential problems with that approach. You are nearly always better off starting over for every test with a fresh new driver instance.

I will not provide a full example for this topic, simply because it would require showing the skeletons of many multiple related classes, but I will speak a bit about why this is important to do well and share a small example with basic usage. Imagine an e-commerce site that allows you to search for and then purchase products. A few classifications of tests for that site could be:

- Login and registration
- Product search
- Product details
- Comments and ratings
- Purchases

Each one of these classes of tests have very different pre-requisites. Some may not require being logged in to the site, but purchasing or commenting/rating likely would require that. In order to make a purchase, there are likely steps that you need to take such as searching for and selecting products and entering or validating billing and shipping information.

Here is an example of a test class containing tests for purchasing products:

```
public class PurchaseTests inherits BaseTest {
```

```
    ClassSetup() {
       Users users = new Users()
    }

    TestSetup() {
       Driver driver = new Driver(...) // Create the instance of the browser driver
       LoginPage.login(users.default)
       MainPage.search('laptops')
       SearchResults.selectResult(0)
    }

    TestCleanup() {
       If (!testResult) {
          // Take screenshot on failure
       }
       Driver.quit() // Clean up the instance of the driver
    }

    Test1(){
       // Some test content
    }

    TestN(){
       // Some test content
    }
}
```

In this relatively simple example, without the use of setup and cleanup functions you would be copying and pasting about 8 lines of code for every single test case, and of course if anything were to change in those 8 lines of code, you would be looking at a large search and replace task. Make it easy for yourself, use setup and cleanup functions wisely.

3.5 Reducing the noise

Finally, in addition to everything else, there are a few things you can do to simply reduce the likelihood of your tests failing in the first place.

- Linking dependant test results. If the login test fails, there are probably a lot of other tests that will fail as well. You can create dependencies either within the unit testing framework, or simply within your reporting to tell you when a test's results should be ignored because it was really a failure due to the dependent test, and not the test itself.
- Making element discovery less brittle. You can ask developers to add ID's to elements and NOT change them. You can use CSS or XPath selectors that do not contain the entire path to the element (do you care if a div way up at the top of the DOM tree changes?) Have it be as concise as possible. For example, a CSS selector like this one: "table#messages > tr > td > div > ul > li > a[href='foo']" could be condensed to '#messages a[href='foo']". The smaller more concise selector is much easier to maintain, because the structure of the table and it's descendants could change, and your selector will still find what it is supposed to.
- Making element interaction less brittle. You should be using implicit waits everywhere. Implicit waits are built in to most UI automation tools, but an implicit wait basically will always try and retry any action on an element until successful, or until a timeout occurs. Explicit waits on the other hand you must call directly in your code. Set the time to wait for elements to something reasonable, and ALWAYS wait for the element to be ready first. If you are running into Stale Element exceptions, make sure to wrap those in a try/catch and retry.

4. Besides the well known Page Object Pattern [1], we will walk through effectively using setup and cleanup Logging and Reporting

4.1 Logging

I admit, I may be a little bit crazy when it comes to logging. My goal with logging is that I never have to debug an automation failure ever to figure out why it failed. I provide enough information to be able to peruse the log, and know within a few seconds exactly what went wrong. Whenever a test fails and there *isn't* enough data in the logs, part of me fixing the test is adding logging to make it easier to diagnose the next time there is a failure. This may seem like a lot of overhead, but when done right, it actually saves a dramatic amount of time in diagnosing failed tests, and makes all of the code more readable and understandable.

What should you log? You should have logging primarily at 3 different levels.

- Test execution level
    - Examples:

      ---------- Project setup beginning ----------
      ---------- Class setup beginning for class myTestClass ----------
      ---------- Test setup beginning ----------
      ---------- Test case abc123 beginning ----------

- Test case level (usually not very much, if any logging here)
    - Examples:

      Step 1: Log In
      The actual message "ERROR" did not match the expected message "SUCCESS"
- Page object level
    - I log the attempt and outcome of every single helper method, and often actions within the helper method. Remember the logging included in the example helper function?
    - Examples:

      Attempting to register user TestFirst TestLast
      Submitting registration
      Registration completed
      An unexpected exception occurred attempting to submit the form. Exception message: 'message'

- Logging at the element level
    - Examples

      Entering text 'TestFirst' into field with locator 'input#first-name'
      Clicking element with locator 'button.submit'
      Unable to find element with locator 'select.choose'

When you combine all of this logging, you get very specific details of what was happening in your test when a failure occurred, and combined with the exception and stack trace information, you should be in a lot better shape to determine the root cause of failure.

Logging at the element level for Selenium requires creating an event listener and listening for events. Other frameworks have their own preferred methods for doing so. If doing this yourself sounds daunting, here is a simple write up of how to do so for Selenium: http://elementalselenium.com/tips/55-wrapper [5].

Another important logging tool for UI automation is screenshots on failure. These can be infinitely helpful when you come up against that one inexplicably flaky test that doesn't fail when you run locally, or even on retry of a test. Looking at the screenshot is usually the first place I go after the log if I still need more information. Most frameworks have a built in function for taking screenshots, although depending on which test execution framework you are using, it may be trivial or tricky to determine the test result in a cleanup method where you can execute the test. Refer to your test execution framework documentation for details.

4.2 Reporting

Most unit testing frameworks produce a report for an individual execution of tests. This is the bare minimum you need to triage test failures for that particular run. Having historical test result data can make prioritizing and triaging test results much more efficient. Typically I have a way to store results only for "official" test runs, so you don't get your database filled with results from you debugging or creating tests locally. You can create reports to help you determine where you should be spending your time. Some interesting reports are:

- Your least reliable tests (change most frequently between pass/fail) over the last 10 runs – These should be given more attention. You can also group the failures by stack trace (see explanation below). If the top stack trace is the same, you know they are failing for the same reason each time.
- Status of the most recently added tests – New tests that fail soon after being added, should be looked at immediately.
- Tests that have had the most defects associated with them – This usually indicates an area of the application that is unstable, or likely to have failures. You can focus more effort on adding additional automated tests in these areas of the application.

Most unit testing frameworks support about 3 different result types: Pass, Fail and Ignore. I usually want more than that. One that I find extremely valuable to track is "KnownFailure". Basically, when a test fails due to a known defect, you want to know that it failed, but you don't need to waste your time triaging that test build over build, or include that failure as part of a quality gate that would prevent the build from being moved forward. I have extended 3 different Unit testing frameworks (Nunit [6], Junit [7] and Spock [8]) to support this. Usually you can annotate a test case method like this:

```
@KnownFailure(D12345)
public void MyTestCase(){
…
}
```

D12345 would be the defect ID, and my extension will be smart enough to query the defect tracking tool for the status of the defect. If the defect is not fixed, I would ignore running the test. If it has been fixed, I would run the test as normal. Depending on your defect tracking tool, you may be able to use other information as well, such as whether the defect is "ready to test", or even look at the "fixed in" build number to determine whether the build you are testing contains the fix. You may even want the result of the automated test to automatically move the defect into an accepted or rejected state, although I would be cautious with that.

When I store data in a reporting database, I always like to include the stack trace. One awesome thing you can do is group failures by the top stack trace. By top stack trace, I mean that you choose where to cut off the stack, usually you can ignore anything that happens in the test case code or prior to the page

object code, and save only the trace from your page object helper and later. These failures are very likely to have the exact same root cause, meaning if you have a test run where 20 tests fail, but they are bucketed into only 2 groups of top stack traces, then you only need to investigate 2 failures not 20. The idea is pretty simple, take your stack trace and draw a line in between the execution of a method from your test case and the execution of a method from your page object and throw out everything from the test case and up. Then draw a line between the last execution of a method from your test execution framework and your page object and throw out everything afterwards. What you are left with is your "top stack trace". You can use namespaces and other groupings in your code to make this easier to do in an automated way. If entering data into a form on a registration page is failing for some reason and you have a helper function in your page class for filling out the form, then if you remove the test case specific code from the stack trace, and the test framework code and the remaining stack trace should be the same for any test case that is executed and failing to fill out the form correctly.

Here is a simple example automating google search [15] with Selenium in Ruby[9]:

```
Selenium::WebDriver::Error::NoSuchElementError: no such element
  (Session info: chrome=44.0.2403.107)
  (Driver info: chromedriver=2.9.248307,platform=Mac OS X 10.10.3 x86_64)
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/response.rb:52:in `assert_ok'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/response.rb:15:in `initialize'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/common.rb:59:in `new'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/common.rb:59:in `create_response'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/default.rb:66:in `request'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/common.rb:40:in `call'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/bridge.rb:640:in `raw_execute'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/bridge.rb:618:in `execute'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/bridge.rb:586:in `find_element_by'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/common/search_context.rb:42:in `find_element'
/Users/samwoods/RubymineProjects/testGem/rspec/pages/google_home_page.rb:16:in `images'
./my_test_spec.rb:10:in `block (2 levels) in <top (required)>'
-e:1:in `load'
-e:1:in `<main>'
```

```
Selenium::WebDriver::Error::NoSuchElementError: no such element
  (Session info: chrome=44.0.2403.107)
  (Driver info: chromedriver=2.9.248307,platform=Mac OS X 10.10.3 x86_64)
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/response.rb:52:in `assert_ok'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/response.rb:15:in `initialize'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/common.rb:59:in `new'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/common.rb:59:in `create_response'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/default.rb:66:in `request'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/http/common.rb:40:in `call'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/bridge.rb:640:in `raw_execute'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/bridge.rb:618:in `execute'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/remote/bridge.rb:586:in `find_element_by'
/Users/samwoods/.gem/ruby/2.0.0/gems/selenium-webdriver-2.45.0/lib/selenium/webdriver/common/search_context.rb:42:in `find_element'
/Users/samwoods/RubymineProjects/testGem/rspec/pages/google_home_page.rb:16:in `images'
./my_test_spec.rb:17:in `block (2 levels) in <top (required)>'
-e:1:in `load'
-e:1:in `<main>'
```

You can see that the stack traces are nearly identical, the only real difference is the line number in the test_spec file. Two different test cases that both called the same helper function, and both failed in different places. It really is that easy, in fact the only line you care about from this particular trace is the line coming from the google_home_page.rb file. In more complex scenarios, there may be many lines you care about, primarily if you have helper functions calling other helper functions within the same page, or in the base page. Once you pull out that section, and determine it is identical between failures, there is a very good chance that all of those failures where the stack trace matches are due to the same root cause, and youre investigation time is dramatically decreased.

4.3 Code Coverage against functional tests

Code coverage can be a bit tricky to set up for functional tests. Historically you have needed an instrumented build, which is pretty easy when executing unit tests in the build pipeline, but a little trickier to deploy to a full test environment. These days there are a number of code coverage tools that you simply set up agents on the SUT, and point it to the source. Some examples are Clover [10] for Java [11] and NCover [12] for .NET [13]. Many web applications are also a lot heavier on the front end JavaScript

[14], making JavaScript code coverage more relevant and important as well. Ideally, the code coverage tool would support storing results per test case for reasons outlined below.

Code coverage on functional tests can be useful for the following reasons:

- Find areas of the code that are not covered that should be, and add automated tests.
- Find tests that overlap, or are equivalent. This is where storing results on a per test basis is handy, so you can compare coverage between tests. If you have test cases that do not execute any unique paths in the code, they are basically equivalent to other tests and are probably adding very little value and can be removed. The less tests you have, the easier they are to maintain and triage.
- A combined coverage with unit tests and functional tests, so you can work with developers to see what code paths should be covered, or are easier to cover with unit tests VS functional tests.

5. Conclusion

These tips all come from years of trial error and experimenting on my part. I hope that you have learned something new, or have been prompted to think about some new possibilities. If it all seems overwhelming, pick just a couple of things to try to start with. I am certain they will improve your automation and simplify the process of writing tests and investigating failures, just as they have for me.

I have identified a lot of low hanging fruit for you, such as implementing the Page Object Pattern and using effective Helper methods with proper data models, Some of the other recommendations like automating the process of checking your defect tracking tool, setting up code coverage, or setting up a reporting framework and building some reports may take a higher degree of effort, but I am confident that all of them will add value if the investment is made. Whatever you do, don't continue to build up technical debt! Set aside some time to implement some of the ideas I have shared, or that you have found elsewhere.

Automation code should be given the same level of scrutiny as the code it is testing. A failure in the automation that allows regressions through can be just as damaging as a failure in the system under test, so start treating it that way and watch how quickly it can transform into a lean, fast, powerful and absolutely invaluable tool for you and your team. Happy automating!

6. References

[1] Page Object Pattern, https://code.google.com/p/selenium/wiki/PageObjects (accessed August 1, 2015)

[2] SQA Stack Exchange. “User Reputation Leagues - Software Quality Assurance & Testing - All Time - Stack Exchange” http://stackexchange.com/leagues/111/alltime/sqa (accessed July 29, 2015)

[3] Selenium Webdriver, http://en.wikipedia.org/wiki/Selenium_(software) (accessed August 1, 2015)

[4] Sauce Labs, https://saucelabs.com/ (accessed August 1, 2015)

[5] Haeffner, Dave. 2015. “How To Add A Wrapper To Your Selenium Tests”. Elemental Selenium. http://elementalselenium.com/tips/55-wrapper (accessed July 29, 2015)

[6] NUnit, https://en.wikipedia.org/wiki/NUnit (accessed August 1, 2015)

[7] JUnit, https://en.wikipedia.org/wiki/JUnit (accessed August 1, 2015)

[8] Spock, https://en.wikipedia.org/wiki/Spock_(testing_framework) (accessed August 1, 2015)

[9] Ruby, https://en.wikipedia.org/wiki/Ruby_(programming_language) (accessed August 1, 2015)

[10] Clover, https://www.atlassian.com/software/clover/overview (accessed August 1, 2015)

[11] Java, https://en.wikipedia.org/wiki/Java_(programming_language) (accessed August 1, 2015)

[12] NCover, https://en.wikipedia.org/wiki/NCover (accessed August 1, 2015)

[13] .NET, https://en.wikipedia.org/wiki/.NET_Framework (accessed August 1, 2015)

[14] JavaScript, https://en.wikipedia.org/wiki/JavaScript (accessed August 1, 2015)

[15] Google Search, https://en.wikipedia.org/wiki/Google_Search (accessed August 1, 2015)

Using Machine Learning to Predict Bug Outcomes from Automation

Wayne Roseberry

wayner@microsoft.com

Abstract

Sure, automation is great because it can give you a whole bunch of information really fast. And yeah, it is a big pain because it gives you a whole bunch of information really fast. Sometimes the amount is too much to process, but you know you need to look because there might just be an important failure in the pile that has to be fixed.

What if it was possible to predict your decision before you read it? How could that save time? How could that help your team focus on important work first rather than dig through a huge pile of data.

This paper will talk about experiments the author has executed in the Microsoft Office team using machine learning to predict outcomes for failures found by massive automation suites. It will talk about the methodology, the problems involved with evaluating such results, the potential discovered in the experiments and some of the inherent challenges, difficulties and risks in trusting such decisions to a machine.

Biography

Wayne Roseberry is a Principal Software Engineer at Microsoft Corporation, where he has been working since June of 1990. His software testing experience ranges from the first release of The Microsoft Network (MSN), Microsoft Commercial Internet Services, Site Server, and all versions of SharePoint. Currently, Wayne works in the Office Engineering team focusing most of his time on test automation and automated testing systems. Previous to testing, Wayne also worked in Microsoft Product Support Services, assisting customers of Microsoft Office.

Previous to working for Microsoft, Wayne did contract work as a software illustrator for Shopware Educational Systems.

In his spare time, Wayne writes, illustrates and self-publishes children's literature.

Copyright Wayne Roseberry, June 8, 2015

1 Introduction

Large scale automation efforts offer a lot of value by running lots of tests, freeing people and engineers to perform other tasks. But the costs of triaging and processing those failures can exceed the capacities of teams to keep up with their incoming failures, assign to personnel to fix, make a decision on how to process them.

This paper describes an experiment run by the author exploring the potential of using machine learning algorithms to aid in the triage of bugs. It asks how well a computer would do predicting the decision that a human would make encountering the same failure, how difficult would it be to construct the models and tools to perform this task. It also asks what we can do with the results, whether we should trust a machine to make decisions about quality for us. Will the machine accelerate the worst part of our own decision making? Or perhaps we can use the information to improve the decisions we do make, executing them faster or more effectively.

This paper is targeted at readers and an audience with an intermediate to expert level familiar with software testing and test automation, but that may be new to machine learning concepts and methodologies. Some introductory material sufficient to explain the methodology and approach and results will be covered, although it intentionally avoids deeper issues of machine model tuning, increasing accuracy and comparison between different models and approaches.

The results show some real potential in using machine learning to aid the failure processing and triage process. The technique described in this document did not prove to be too expensive or difficult, and most certainly creating models based on existing test automation failure triage histories is well within the capabilities of available technology. The experiment never left the laboratory (that laboratory literally being the PC under the desk in the author's office), but certainly deployment to a production system is feasible. As will be described later in the document, there are certainly limits at this point to applying the results blindly, so applications are probably best selected that work in conjunction with human decision making rather than replacing it. This results in this document raise questions more than answer them, which is this author's intent.

2 Motivating Problem – Triaging with Large Volumes of Automation Failures

2.1 Test Automation in the Microsoft Office team

2.1.1 Office runs a lot of automation, with a lot of failures.

The Office automation system is used actively by approximately 2600 software engineers[1], all working on an integrated product with large numbers of dependencies. Engineers run automated tests that deploy to a lab of several thousand machines which install and configure the product before launching what is typically an end to end integrated test. These tests are launched either to validate changes before submitting to the source repository, or to look for bugs that may have accumulated in large blocks of changes. Individual product teams run constant looping builds of their own on team branches off the main source fork while daily builds of the entire main line Office branch also initiate several automation suite runs.

On a daily basis, the Office automation system typically runs over 2 million tests per day. The number of failures observed by the system on a daily basis tends to be about 130,000 – 150,000, which are distributed to product teams to triage for resolution (fix, not fix, etc.)

[1] Based on a query 7/19/2015 which identified 2619 distinct users launching automation jobs on this system over the prior 90 days. On a daily basis, the system will be used by as much as 700 users per day. Engineers in this case are all software developers, as the Office organization does not have a distinct job title for testers.

2.1.2 Failures are automatically classified as new or unique

In response to a large number of failures, many of which were symptomatically the same problem, Office introduced a failure bucketing mechanism that attempts to determine if a failure was seen previously (Robinson). The technology affords a great deal of time savings on the part of engineers and product teams as they no longer need to determine if a specific instance of a failure is new or not.

This document does not detail the failure matching mechanism beyond the following points: 1) it matches based on symptoms of the failure as root cause is unknown at the time failure, 2) the matching is based on a textual similarity metric inside the failure message emitted by the testing 3) the matching considers details of the call stack collected at time of failure.

Once a failure is identifies as new, it is entered automatically into the Office bug tracking database.

The auto-bucketing system and the eventual resolution of the bug entered are important inputs into the machine learning experiment that follows.

2.1.3 Triaging failures is expensive and error prone

Once bugs are in the bug database, product teams resolve them with one of the following states, “FIXED”, “WON’T FIX”, “BY DESIGN”, “POSTPONED”, “DUPLICATE”, “NOT REPRO” and “EXTERNAL.” The general disposition of this paper is that any decision other than “FIXED” represents energy and cost spent toward a desirable outcome, while all other costs are inefficiencies of the system. We would have preferred spending the cost on something driving to a fix.

Product teams spend a considerable amount of time and energy making decisions about bugs. As a bug is created by a test, it is often the case that the output of the failure message is not something immediately obvious to a person reading it without having to read the test code itself. And even inside the test code, the actual failure may be hard to discern. For example, a test might have been “Insert text, open Format dialog, applying formatting, dismiss, check if formatting matches expectation” and the failure might have been “The Format dialog never displayed” Any number of underlying causes, from legitimate product failures to issues in the test harness code, could cause such a problem. Investigation is difficult and expensive.

Even the seemingly simple act of eliminating duplicate failures (current rate of duplication in the system is ~40%) takes time. An anecdotal survey by this author of Office engineers reported an average of 20 minutes per bug to establish a credible case for a failure being a duplicate.

2.2 Questions of interest

2.2.1 How accurately could a computer model predict the final resolution of a failure?

The first question was whether or not a trained model could save time on bug triages. Would it be able to predict, accurately, the same resolution that humans would? Test automation failures messages tend to repeat the same text patterns, so I thought it was possible it might. But it was also far from certain, as failure messages are purely symptoms, and often underlying issues only seen after exploring the code in-depth drive the final decision.

But there is certainly an opportunity to save some sort of cost or energy if a prediction was possible. Even just hinting to a triage team what the model predicts could guide them in focusing their attention in specific directions more efficiently than without. Of course, it was also just as likely to waste time by pointing the triage team in the wrong direction.

2.2.2 How well could a computer model explain why it chose a given resolution?

Triage teams and engineers need justification for their decisions. Maybe a given failure represents a false assumption in test code, and does not justify a product fix. Maybe a given failure only occurs on conditions the test creates and deserves a lower priority. Maybe a given failure, while rare under test conditions, represents a severe failure state that would cause a customer great harm.

Likewise, we would want some sort of explanation from a computer model. This can be a difficult requirement, as many machine learning models get complex very quickly and determining why the model offered a specific solution or answer is sometimes unwieldy if not impossible. Further, the reasons for the decision are based on nothing more than the inputs to the system. If all a model looks at is occurrence rate of certain words in a failure message, then there is no direct relationship to other factors that might drive a team to make a decision, such as likelihood of occurrence in market, or severity of failure, or likelihood that the failure is a test artifact only. At best there is correlation of certain inputs with certain outcomes.

2.2.3 Would automatic prediction present any dangers or challenges we need to worry about?

Numerical analysis and machine learning are very good at allowing people to believe things that are not true. The reason why ML has to date been mostly within the realm of data science is because there is a need for disciplined statistical analysis to avoid drawing erroneous, and perhaps dangerous, conclusions. Consider:

Blindly trust an inaccurate model to resolve bugs: Imagine a model that achieves an 80% accuracy rate on predicting a bug would be something other than “FIXED”. This would mean that for every 100 bugs punted, 20 of them are bugs the team would have fixed. An 80% accuracy rate is actually a rather impressive feat, but for something like whether or not to fix a given bug, that 20% miss is likely much higher than a team would tolerate.

Reinforce bad habits: In the model described in this document, the training data was based on historical resolution trends. The model was being taught to predict team behavior, not necessarily best behavior. Suppose the team was making bad decisions by punting bugs that were important to the customer. The model could actually make such a phenomenon worse by deferring decision until a later point.

3 Classification via Machine Learning, Background

At this point, we will discuss a bit about machine learning and how it works for problems such as the one described here.

3.1 This problem is a supervised classification problem

A supervised classification problem means two things in machine learning terms:

1. “Supervised” means the model is trained by showing it example data with “correct” answers. In this case, the example data comes from test failure message text, the sequence of methods executing in the source code at the time of failure (call stack), and the answers to the data comes in the form of bug resolutions as chosen by product teams and engineers.

2. “Classification” means the goal of the model is to determine which category a given piece of example data fits into. In this experiment, the classifications were the actual bug resolutions assigned by product teams to bugs. The possible values are: “FIXED”, “POSTPONED”, “WON’T FIX”, “BY DESIGN”, “DUPLICATE” and “EXTERNAL”. A resolution, in this case, is the final decision about what to do with a given bug. In the case of “FIXED,” the resolution is assigned after an engineer has written, tested and submitted the code change. In all other cases, the resolution is typically assigned at the moment of triage.

3.2 Multi-class logistic regression partially explained

In this discussion, the word “term” refers to strings extracted from the failure message and call stack text in the training data.

It is almost sufficient to say multi-class logistic regression is *“…a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes*” (Wikipedia, “Multinomial Logistic Regression”) and leave it at that. There are many different types of machine learning models that are capable of multiple classification, and they all have their various strengths and weakness. They almost wind up being treated like a black box, and the engineer/scientist training the model will

typically try different types of classification models with their data and just pick the one, or combination of models, that consistently yields the highest accuracy. However, there are a few details worth mentioning because it is relevant later to the discussion:

1. Multi-class logistic regression is a very simple and fast classification technique

2. The heart of most logistic regression problems is a math formula that looks like the one below. In this case, "X_N" represents the frequency of some term in the training data and "Θ_N" represents a coefficient in the model that is multiplied by that frequency:
   $\Theta_0X_0 + \Theta_1X_1 + \ldots\Theta_{N-1}X_{N-1} + \Theta_NX_N$

So, the variable X_0 might be for the frequency of the term "window" and the variable X_1 might be for the frequency of the term "crash" and so on. The model is trained by looking at thousands of examples of data, and iterating until it finds a set of values for "Θ_N" that most optimally produce the same answer given for the data. A more thorough and very accessible explanation of the exact algorithm behind logistic regression is available from Coursera (Ng, Andrew).

The reason this is interesting is because it makes the reasons behind the decision the model makes very apparent. If a given term, such as "menu" is highly correlated with the resolution "FIXED" then it will have a "Θ_N" value associated with it that is very high. Likewise, if a given term, such as "timeout" is negatively correlated with the resolution "FIXED" then it will have a "Θ_N" value that is very low. We will refer back to this point later.

3.3 Understanding Classification Results,

3.3.1 Recall versus Precision

There are several metrics used to judge the accuracy of classification problems, but the two simplest and most commonly used are precision and recall.

Precision is the percentage of times that a prediction selected the correct answer. So, if a trained model were tested and said that 100 items were resolved "FIXED" and in fact only 99 of them were resolved "FIXED" then the model is said to have a 99% precision rate on the category "FIXED".

Recall is the percentage of total items in a given category that were correctly identified by the trained model. For example, if in the training data there were 200 items whose correct classification was "FIXED" and the trained model only classified 100 of them as "FIXED" then that model is said to have a 50% recall rate on the category "FIXED".

3.3.2 Dangers of over-fitting

Over-fitting is a classic mistake in machine learning. "*In statistics and machine learning, overfitting occurs when a statistical model describes random error or noise instead of the underlying relationship.*" (Wikipedia, "Overfitting"). It typically happens when there are so many variables in a given model that it is capable of extremely high precision and recall rates on its training data, but on nothing else. It can easily happen when the training examples do not generalize well or are too small relative to the large real-world examples. It also happens when the test data used to measure the model accuracy is really just a repeat of the data used to train the model instead of a suitable representative sample of the real world.

Over-fitting is relevant to the discussion partly because it is always relevant to machine learning. It is also relevant because as the results will show later, the experiment yielded surprisingly high accuracy. It became important to examine whether or not the sin of over-fitting had been committed.

4 Methodology

4.1 Toolset

For the experiment, I used a set of tools and library of machine learning APIs created and used at Microsoft. The tools are not available externally, but the code in the libraries are the same code that is

available via Microsoft Azure Machine Learning (Microsoft, “Multiclass Logistic Regression”). The rest of this document is not going to focus a great deal on specifics of the toolset.

4.2 Automation result collection and data preparation

Most of the effort in this experiment involved preparing the data for consumption. This is common in a lot of machine learning examples, as machine learning models are typically generalized learning systems, and the source data is usually specialized to whatever problem it relates to. The data needs to be converted from its specific, original form, to a generalized form for the model.

In this case, the data needed to be converted from something that looked similar to the following:

“Assert 'hy5k' occurred in WINWORDD.EXE. Message: This dll has failed to load…”

Into the following format:

Assert WINWORDD.EXE failed_to_load

This required the following steps

1. Running a query from the automation system database to get instances of test failure, the failure message text and the final bug resolution. This resulted in hundreds of thousands of rows of data.
2. Breaking the failure message text into single terms to serve as training “features” (the ML term used to indicate variables used to train a model)
    a. Split on common separators (comma, semicolon, etc.)
    b. Extract everything that looks like a filename or assembly name into a term (e.g. “WINWORD.EXE” or “Microsoft.Internal.Office.Automation.UI.dll”)
    c. Identify special terms that are interesting, e.g. “Assert” or “Crash”
    d. Remove anything that looks like noise (short words, dates, numbers, GUIDS, random strings used as filler test data)
    e. Concatenate remaining multiple words together into single terms (“Application Foo closed the Window” -> “Application_Foo_Closed_Window”). This was to reduce the number of variables training in the model, mostly for speed reasons.

Training data example

train_aug.txt

Instance	FailMessage Namespaces Modules Classnames Methods Filenames
1	Crash winword.exe mso!PushClipboardToAcb n:\src\mso\activeclipboard\acb.cpp INVA
6	UlsLogScanner OneNote_Service_log_scanner registered_violation__file TimeStamp 0
2	error Unable___property undefined__null_reference Microsoft.Oasys.Client.Motif
4	Test_left__window_open Visio_Professional owner_process_details Process_info VIS
2	Crash testmonitor.exe mita_logging! MS.Internal.Mita.Logging.LogManager.ReportEx
1	Assert vnph occurred WINWORDD.EXE mso\activeclipboard\ acb.cpp Message *WaitFor*
2	Process WINWORD.EXE crashed 100085692、種類 BEX64 Cab_ID 53ec1ef1 MSVCR120.dll 52
4	FAIL Method MS.Internal.Test.Automation.Office.Webtoad.WebtoadBrowser.WriteOffic
4	Test_left__window_open closing Visio_Professional owner_process_details Process_
4	Failed__start__DFF_Client Method MS.Internal.Test.Automation.Office.Tests.BigBut
1	Test_left__process_open terminating_process adb.exe Process_info CommandLine adb
2	ACTION_FAILURE Access__denied Failed_copying
1	Crash winword.exe mso!PushClipboardToAcb n:\src\mso\activeclipboard\acb.cpp INVA
2	Error Unhandled_exception__required_test_method find_Word_process WINWORD System
2	APPCRASH WINWORD.EXE Microsoft.Oasys.Client.MotifTestActions Microsoft.Oasys.

4.3 Result collection and comparison

The training data was collected from 90 days of automation data (selected because the automation system purges data older than 90 days), extracting only those results with resolved bugs. The bug resolutions were the real, actual decisions of product teams. 20% of the data was removed from the training set and used for model testing.

Product teams take a while to resolve automation bugs, on average 15 days (there is a division wide goal pushing to this). This means that toward the end of the training data sample there are fewer bugs in a resolved state to train the model. I was curious if this would affect the accuracy of the model with brand new failures. To measure this, an additional 15 days of data after the training data were evaluated and predictions recorded and compared against actual resolutions.

5 Results

5.1 Initial accuracy numbers

The initial results were surprising. Anecdotal accounts from other efforts by teams within the company attempting similar problems (automatically assigning bugs to appropriate owners based on the text on human created bug reports) were that achieving accuracy as high as 70% required considerable effort. The results of this experiment were quite different.

For the categories, “FIXED”, “DUPLICATE”, “WON’T FIX” and “NOT REPRO”, the model made a correct prediction (precision) 97%, 95%, 95% and 91% of the time, respectively. On the same categories, the model correctly identified 97%, 95%, 97%, 84% of the data in the category (recall). The other categories (“BY DESIGN”, “POSTPONED”, “EXTERNAL”) were so rarely used by the product teams that the model generated no predictions.

A diagram of the results is shown below. The results are shown in the form of a confusion matrix, which is a common format used in predictive analytics to compare recall against accuracy for a given population of data. The table below can be read to determine recall for a given classification by looking at the final column in the table and going to the row number for that classification. For example, “FIXED” is classification 1, and it has a recall value of 97.4%. Precision can be looked up by going to the final row of the table and looking at the number for the column of that classification. For example, “FIXED” (1) has a precision value of 97.3%. Prediction labels are shown on the top row of the table and the actual value are shown on the leftmost column of the table. Individual cells indicate how many predictions for a given classification hit how many items of a given, real classification. Where the column and row labels match, the model predicted correctly (e.g. 1:1 had 32454 hits for “FIXED”). Where the column and row labels do not match the model predicted incorrectly (e.g. 4:1 had 186 hits where the model predicted “FIXED” but the actual resolution was “WON’T FIX”).

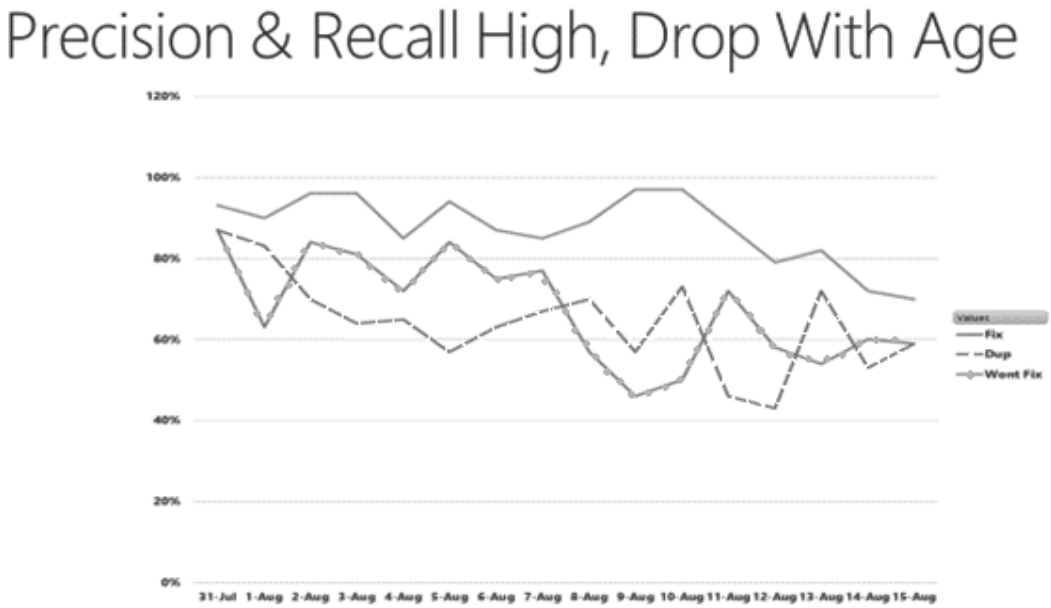
```
Feature Handlers:
          Added handler 'WordBag' with 120236 features at offset 0 for columns: 1,2,3,4,5,6

 Confusion table
          ||========================================================================
PREDICTED ||     0 |     1 |     2 |     3 |     4 |     5 |     6 |     7 | Recall
TRUTH     ||========================================================================
       0  ||     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.000
       1  ||     0 | 32545 |   518 |     0 |   249 |     0 |    81 |     0 | 0.974
       2  ||     0 |   489 | 21391 |     0 |   544 |     0 |    78 |     0 | 0.950
       3  ||     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.000
       4  ||     0 |   186 |   193 |     0 | 20589 |     0 |   106 |     0 | 0.974
       5  ||     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.000
       6  ||     0 |   174 |   326 |     0 |    42 |     0 |  3125 |     0 | 0.849
       7  ||     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.000
          ========================================================================
Precision || 0.000 | 0.973| 0.952 | 0.000 | 0.954| 0.000 | 0.919| 0.000 | 0.959 |

ACCURACY(micro-avg):     0.959595
ACCURACY(macro-avg):     0.516315
LOG-LOSS:                0.109485
LOG-LOSS REDUCTION:      91.701094

Time elapsed(s): 479
```

1=Fixed
2=Duplicate
3=By Design
4=Won’t Fix
5=Postponed
6=Not Repro
7= External

5.2 Accuracy changes over time

As stated previously, our bug resolution data tends to take 15 days or more to resolve, so I wanted to see how accurately a model built up to 15 days prior to a failure occurrence could predict the resolution of the bug from the failure. Would the accuracy hold up, or would it degrade with age? The results were clearly the latter.

Close to the training window, overall results showed high average of the measures for precision and recall, at approximately 90%, depending on which resolution was predicted for. The results dropped as the model aged relative to the failure occurrence. The daily results varied a greatly, but the overall slope was strong, show the predictions dropped in overall accuracy by > 20% points by the 15 day mark at the far end of the graph. This suggests that the accuracy of the model correlated to how close in time the model was trained relative to the failure itself. The following chart shows the drop rate over time from the last day of training data on the far left, and the last bug predicted against on the far right.



This drop in accuracy makes sense intuitively, as the new failures seen by the automation system, coming largely from changes to the product code base or to the test automation system, would not be reflected in the data in a much older set of training data. As you continue testing, you're exercising code that the training model hasn't seen yet and didn't train on. This had implications for the value of the model for dealing with brand new failures. The model accuracy was as much as 20% lower for any failure that was completely new than for a failure that was very similar to ones seen prior.

5.3 Term weightings and explaining results

As mentioned prior, the simple underlying structure of the way logistic regression based models are built makes it somewhat easy to evaluate how much a given feature, or in this case term, contributed to a given outcome. Every single term in the model is assigned a numerical weight that will be multiplied by whatever frequency it is given inside the instance data. If the number is large, the term is considered very important to the outcome of the bug resolution. If the weight is very small, or negative, it actually pulls away from the outcome of the bug resolution.

There was a strong correlation to certain terms appearing in a failure message and the tendency to fix or not fix a bug. Terms that teams related to test infrastructure components throwing a failure tended to receive a “WON’T FIX”, particularly components associated with simulating end users controlling and manipulating UI state. The image below shows a list of terms from the experiment that were heavily

weighted toward the “WON’T FIX” resolution. Two of the very interesting sub-strings in the terms that came up a lot toward the top of this data set were “MS.Internal.Test.Automation” and “MS.Internal.Mita.” Both of these are test tool and automation frameworks internal to Microsoft. It is impossible to tell from a numerical analysis of this sort the true cause of whatever failure was thrown. It may be that the presence of these strings in the failure message content push a team strongly not to fix a bug, although correlation does not always indicate causation. It is most likely the case that teams are interpreting the failure as purely a test system artifact, but that is not a point that can be made conclusively without further detailed investigation.

term	weight
4+0_0_FailMessage_WINWORD.EXE~Hashed_bucket	4.050632
4+0_0_FailMessage_Error~HRESULT	2.988678
4+intercept	2.980591
4+0_1_Namespaces_MS.Internal.Mita.Logging~MS.Internal.Test	2.692241
4+0_0_FailMessage_ACTION_FAILURE	2.577091
4+0_0_FailMessage_occurred__WordIm~word\core\	2.458367
4+0_1_Namespaces_MS.Internal.Mita.Logging~MS.Internal.Test	2.368129
4+0_0_FailMessage_otools~\\mso\memory\legacyoperatornev	2.279074
4+0_0_FailMessage_Microsoft.Office.Word.Interfaces.Verificati	2.267157
4+0_1_Namespaces_MS.Internal.Test.Automation.Office.Word	2.223034
4+0_1_Namespaces_MS.Internal.Test.Automation.Office.Public	2.107339

Meanwhile, terms that correlated with application thrown errors, such as crashes and faults, or with better information such as call stacks inside product code, tended to receive a “FIXED” resolution. The image below shows a list of terms that were strongly correlated with a “FIXED” resolution. Not the presence of the phrase “APPCRASH” in the data. These are errors that could only have happened within the product, and by no means could have been invoked by test code. Also interesting is the phrase, “Call_stack”. In this case a product engineer would be able to determine exactly what line of code threw the failure, making diagnosing and fixing the failure much easier. Another interesting phrase is “Mso~Telemetry” occurring at the top of this list. This refers to in-product telemetry markers, which suggests that telemetry reported data inside a failure is very strongly correlated with the resulting bug receiving a “FIXED” resolution.

term	weight
1+0_0_FailMessage_Mso~Telemetry	3.839336
1+0_0_FailMessage_APPCRASH~WINWORD.EXE	3.587338
1+0_0_FailMessage_microsoft.office.excel-1~int	3.219655
1+intercept	3.167692
1+0_0_FailMessage_Call_stack	3.142599
1+0_0_FailMessage_fioNoticonic~Method	2.934501
1+0_0_FailMessage_occurred__WordIm~word\client\shared\	2.664881
1+0_0_FailMessage_long_time__appears___unhandled	2.577774
1+0_0_FailMessage_n:\src\xl\shr\layer\win\glmisc.c~AirSpace	2.325094
1+0_0_FailMessage_occurred	2.323332
1+0_0_FailMessage_APPCRASH~pptim.exe	2.305285
1+0_0_FailMessage_pptim.exe~Hashed_bucket	2.305285
1+0_0_FailMessage_Unhandled_exception~TypeError	2.209667

Such an analysis is not possible with all classes of machine learning, which makes use of simple predictive models as logistic regression particularly advantageous. There are probably several ideas to

take away from this analysis, although they may be specific to practices and culture within the Office division at Microsoft. Some of those takeaways could be:

1. Failure messages that contain test harness and tool only data will tend to result in bugs that are resolved “WON’T FIX”
2. Failure messages with application specific failure content will fare better in terms of bugs resolved “FIXED”
3. Failure messages that aid in the investigation, such as providing stack trace information, fare better in terms of bugs resolved “FIXED”

6 Conclusions and Questions

6.1 Repetitive nature of the text input made the results unusually accurate

Over-fitting was an immediate concern because the numbers were so accurate. Particularly with logistic regression, it is common that with lots of variables (in this case, terms in the failure message) that the model contains a combination of coefficients capable of predicting exactly the result from the training data and no other. This concern was countered by the testing data. The 20% sample data used to test the model was never part of the original training data, and the results showed the same for the testing data as for the training data.

The falling trend also suggests an explanation for the very high accuracy rate of the model to begin with. The training data is coming from test automation, which produces nearly (but not exactly) identical results every time it is executed. If a given test fails with the message “The window failed to close after clicking OK” then it will say that same message every time the test fails in that way. The text of the failure may change somewhat, for example some tests display error messages with data that is derived randomly at test time, or might display call stack data which changes every build, or might record time based information in the text. But key words and phrases in the message will still be identical and in the same order every time. This makes the failure message highly predictable, and thus easy to train a model on. But those identical phrases hold less predictive value when they are new and the system has had fewer chances to see them. The words and terms are less strongly associated with a given outcome than other words, so the model does not emphasize them. This association gets weaker and weaker as the model ages.

The highly predictable nature of the text is contrasted against human generated text, where the string describing a problem might be “I cannot open the window” or “The app will not start” or “The app won’t run” all to describe the same behavior. By contrast, the repetitive nature of automation failures is much easier to classify. It is after model ages, and the value of the repetitive text diminishes by virtue of fewer instances in the sample that the predictive capabilities start entering ranges typically seen for matching human generated text.

6.2 The actual work was not ridiculously difficult

Build the experimental system took approximately two weeks. This was mostly time learning the APIs and tools, as well as time building the data preparation tools. A couple of days were given to experimenting with different model types to pick the model and settings for input. A couple of days were given to build a tool to iteratively call the model and generate the output.

Once built, the actual model took less than an hour to collect and prepare 90 days of training data, and approximately five minutes to re-train the model. The nature of logistic regression is that actual prediction is near instantaneous.

The point here being that the basic work was easily within reach of a typical software engineer with a modest amount of training in machine learning.

Another point glossed over here is the amount of work spent tuning the model. For sake of the experiment, I avoided tuning the model completely. The goal was just to see what a naïve approach could

generate and how accurate the results would be. In most business applications, achieving the highest results possible is an important goal, and this frequently consumes a lot of time and demands more than a simple exposure to machine learning to accomplish. Model training can involve anything from different data selection and preparation (usually a big factor on the results) to changing behavioral parameters on the model builder (requires sophisticated understanding of how model training works) or selection and maybe combination of different models.

The summary on all of this is that the starting costs of this problem were low and rather simple. The eventual costs could remain low, or drive high and complex, depending on what demand one wanted to place on the results and the accuracy required.

6.3 The results probably should not be trusted to let a computer automatically triage failures

Whether or not one should let the trained model in this experiment automatically make triage decisions about bug resolutions probably depends on one's tolerance for risk. But mathematically, the recall numbers on the "FIXED" category are pretty straightforward. At 97%, 3 out of every 100 bugs that would have normally been fixed by the product team would be rejected automatically by the system.

The number of bugs this represents on a regular basis to the Office team varies a great deal, but during the time the experiment was running, the weekly average bugs created by the automation system was 426, and the fix rate was (a somewhat low) 14%. This works out to 426 * .14 * .03 = 1.7 bugs a week that would have been fixed by the product team being automatically punted by the trained model.

One way of working with this might be to let the system communicate its recommendation and see if that at least makes product team triage decisions more efficient. Having a human see the decision that the model predicted might offer some protection against cases where the computer gets the prediction wrong while still leading the human to that decision more quickly.

But even that may not be a good idea. For reasons discussed next.

6.4 The results may reinforce bad decisions

There is a joke that says "To err is human. To really mess things up takes a computer."

The training data is based on historical decisions. There is nothing in the data that says the decisions were good decisions. Unfortunately, there is little, if any, record correlating automation failures, bugs resolved other than fixes and events that later changed that decision. Failures encountered later are sometimes traced back to the first time the failure was seen in automation, but even when done data is not available in the system that records this in a consistent and trackable fashion. It is not uncommon that a failure resolved "WON'T FIX" or "POSTPONED' occurs later during pre-release usage or even in the hands of customers, but in going through the process of reporting the failure and eventually fixing it nobody identifies the original automation bug that found the failure.

This means that if the team made bad decisions when resolving automation failure bugs that the training data will see those decisions, predict a similar decision for similar failures, and reinforce the bad decision in its prediction. A product team that errs on the side of blindly trusting the guidance of a computer trained based on their own bad behaviors may only get faster and more efficient and making the same mistake again.

Some ideas to consider which might mitigate against such problems:

1. track more closely errors resolved not "FIXED" which result in problems experienced by customers later
2. use other data, such as failure frequency rates, to temper the resolution, or to re-activate bugs that might have been previously resolved not to fix
3. train the model based on customer telemetry data to discover things like high frequency code paths, or features and product behaviors that have a high incidence rate in support traffic

6.5 The results can probably be trusted to guide teams toward further investigation

It is possible the output of the model could be useful for engineers needing to investigate failures. For example, if a model predicts that a result is very likely to be a duplicate of another bug, then an engineer is probably very likely to start first looking for other failures with similar text in the message rather than beginning an investigation into the reasons behind the failure.

If an engineer has decided that a failure might merit fixing, then the text weighting in the model could help as well. Knowing that specific words, or filenames, or maybe portions of a call stack caused the model to predict “FIXED” could offer a strong hint as to where the underlying cause of the problem is.

But even more interesting could be the value in helping teams target improvements in their test engineering. Any team with a suite producing failures that predict heavy for resolutions other than “FIXED” ought to make some sort of change in test automation or in the product to cause the suite to yield something more valuable. Either a very stable run of the product, or failures that are taken seriously and drive to fixes. So just the raw prediction hint is a clue that maybe there is something amiss. Maybe the team has a decision making bias toward “NOT REPRO” and needs to invest in better tooling. Maybe the wording in the test failure is using words that are too vague. Maybe a given test library needs to be made more robust. Maybe the product itself needs to provide more testability access so that failures are more obviously a product problem instead of appearing to be a test artifact.

7 Summary

The experiment overall was a success, and demonstrated the following things:

1. A trained model to predict bug resolution based on automation failure messages can achieve very high accuracy ratings
2. The trained model needs to be kept current to sustain the high accuracy ratings
3. Using simple models, the training data suggests potentially useful information regarding how certain words in the message drive different predictions
4. Despite the surprisingly high accuracy, it is doubtful that such a trained model could be trusted just yet to replace the human in the decision making process

Of course the experiment also raises many new questions, all available for exploration at another time. Most certainly, it seems human job security is still safe and intact, because if anything it appears that trained predictive models may prove to make people better at doing the job they already do.

References

Microsoft, “Multiclass Logistic Regression” https://msdn.microsoft.com/en-us/library/azure/dn905853.aspx (accessed June, 2015)

Ng, Andrew, “Machine Learning”, Stanford University online course at Coursera, https://www.coursera.org/learn/machine-learning/ (accessed September, 2014)

Robinson, M. P. Test failure bucketing, U.S. Patent 8,782,609 July 15, 2014

Wikipedia, “Multinomial logistic regression”, https://en.wikipedia.org/wiki/Multinomial_logistic_regression (accessed June, 2015)

Wikipedia, “Overfitting”, https://en.wikipedia.org/wiki/Overfitting (accessed June, 2015)

Sustaining in an Agile World

Don Hanson II

don@black-box.com

Abstract

The challenges associated with sustaining a successful product often take managers by surprise.

Most sustaining planning discussions follow along the lines of “that would be a wonderful problem to have, as it means we will have acquired paying customers! Bah! We’ll cross that bridge when we get there.”

When you acquire paying customers and are faced with the problem of resolving customer escalations in a timely fashion, the reactive nature of sustaining a product presents challenges with agile development methodologies. Over time these issues can chip away at a team’s velocity, morale and ultimately retention.

There are two common approaches for handling sustaining; whole team sustaining and dedicated sustaining teams. Each has strengths and weaknesses that must be actively managed and overcome.

This paper introduces a new, blended approach that provides many of the positives and minimizes the negatives of the previous two. We’ll discuss the warning signs of the two common approaches to help you decide if this new approach is right for you. Then we’ll cover the five simple best practices for easily apply this new approach with your team.

Biography

Don Hanson is bringing game changing new security management and protection capabilities to market. He is the engineering leader for the Data Exchange Layer, the foundation of Intel’s Security Connected vision, and Threat Intelligence Exchange open ecosystem.

Don Hanson founded his first company and began writing commercial software in the early 90's developing an evolving line of animation plug-in products. He has since worked on projects ranging from the mobile client for a wireless navigation startup to enterprise-level commercial software products. Don later spent several years delivering the next generation of McAfee’s flagship enterprise security management product and taking it to the cloud.

Copyright “Don Hanson II”, 2015

1 Introduction

Sustaining is the process of handling customer escalations involving your software product. The process may involve multiple levels of customer support personal attempting to troubleshoot the issue, or it could be a “Contact Support” form in your app that sends an email to a common account. In this paper we’re going to look at the sustaining process from the product development team leader’s perspective.

When a customer escalation reaches the product development team, whatever people, process, or sacrificial chickens you’ve put in place to handle dissatisfied customers have been exhausted. It is now up to you and your team to solve the problem posthaste.

When considering whom to assign to fix a particular escalation, there are two common approaches.

The first approach involve could be summarized as “you broke it, you fix it”. In this approach team members building the new, un-released version of the product are also assigned to fix escalations in the “old” version.

The second approach is on the opposite end of the spectrum in terms of who does the work to address escalations. This approach could be described as having a “dedicated fix-it team”, where specific team members work exclusively on addressing customer escalations. These team members do not work on the new, un-released version of the product.

Each approach has non-trivial advantages and disadvantages.

In this paper we’re going to discuss a third option that we call “rotating roles”. This approach combines aspects of the previous two to reduce the negative costs to the team while delivering outstanding customer support for escalations.

2 Welcome to Product Sustaining

The good news is congratulations are in order! You’ve created a product, released it and someone, somewhere is using it.

The bad news is something doesn’t appear to be working for them as expected. They would really appreciate it if you could drop everything you’re doing and make it work for them, say in the next five minutes or so?

Oh, and the clock is ticking. Whether the metric is a formal Service Level Agreement (SLA), lost revenue or simply customer angst, the sooner you can fix their issue the better.

3 Whole Team Sustaining – “You broke it, you fix it”

Whether your process is waterfall, agile or something in between, a common reaction is to assign the issue to the team member with the most experience in the general area associated with the customer escalation.

This is can often be the team member who originally wrote the suspect code.

With person most familiar with troublesome code working on the issue, the fix is most likely to be done in the best possible manner for the product. E.g. strengthening the architecture instead of piling on another band aid.

On the downside, the interrupt drive nature of customer support runs counter to the agile development best practices of keeping sprints atomic.

In addition, there are many situations that increase the mental cost of task switching between working on sprint stories and addressing sustaining issues beyond that of “simply” switching between stories. For

example, the new version of the product may use 3rd party libraries incompatible with the released version, requiring separate development environments. Or the test automation may have changed between the two versions.

It can be hard to sustain forward progress on new features when team members who working on critical functionality, that is needed by other team members, are taken offline to address customer escalations.

This can lead to frustration among team members. It's easy to see how those blocked due to missing dependencies can be frustrated, but those pulled to address escalations often feel equally frustrated that they're letting the team down by not finishing their stories.

This can lead to team strife as frustrations mount. Equally bad is the potential for low quality code to sneak into your product as team members try to do more than they should, in order to keep up the appearance of meeting the sprint goals *and* the sustaining SLA.

Figure 1 Summary of whole team sustaining approach benefits and challenges

4 Dedicated Sustaining Team

Another popular approach for handling product sustaining duties is to create a dedicated sustaining team. Sustaining team members work only on customer escalations. Typically they are rewarded for meeting the sustaining SLA, which is addressing customer escalations within specified time limits.

This approach generally starts off doing quite well. However, the drawbacks build up over time.

Three common challenges to this approach are decreasing knowledge of product & architecture, divergent rewards and staffing the sustaining team.

Dedicated sustaining teams are typically created in part or whole from mainline developers with current knowledge of the product and its architecture. Over time the sustaining team's knowledge of the product and architecture becomes outdated, as new features are created, technologies updated, design patterns changed by the mainline team.

This can result in sustaining fixes that use libraries the mainline team is trying to move away from, duplicate functionality available elsewhere in the code base, or negatively affect product performance or usability in another area.

Hearing about mainline team members backing out sustaining changes is a telltale sign.

Over time the divergent rewards for the sustaining and mainline product development teams tend to drive their cultures apart.

The mainline product development team is rewarded for making a product that provides useful new functionality to customers and is engaging to use. The sustaining team is rewarded for making customer complaints go away.

The problem is that it is often quickest and easiest to make customer complaints go away in ways that run counter to improving the product functionality and ease of use.

For example, it may be easier and quicker to write a monitoring program that restarts a web service when it crashes, than to eliminate the crash itself. This example is a bit extreme, but I've seen it used.

The third common challenge is keeping the sustaining team adequately staffed. Sustaining is not typically seen as an attractive role. It can be challenging to staff a dedicated sustaining team with appropriately skilled and talented team members.

Tactics to address this include using the sustaining team as a training ground for junior developers, or requiring all new team members to spend time on the sustaining team before joining the mainline team.

Unfortunately these tactics virtually guarantee team members unfamiliar with the product and its architecture and/or having entry level skills will be addressing customer escalations. Expect escalations to be fixed with band aids, instead of root level, structural changes.

Note that each of the above is a general trend. The quality, productivity and morale impacts typically build up under the radar. They may be first detected as a niggling sense of something being not quite right, or in the unexpected departure of a lead developer. There are exceptions, but managing these challenges over time is time consuming and hard to do well.

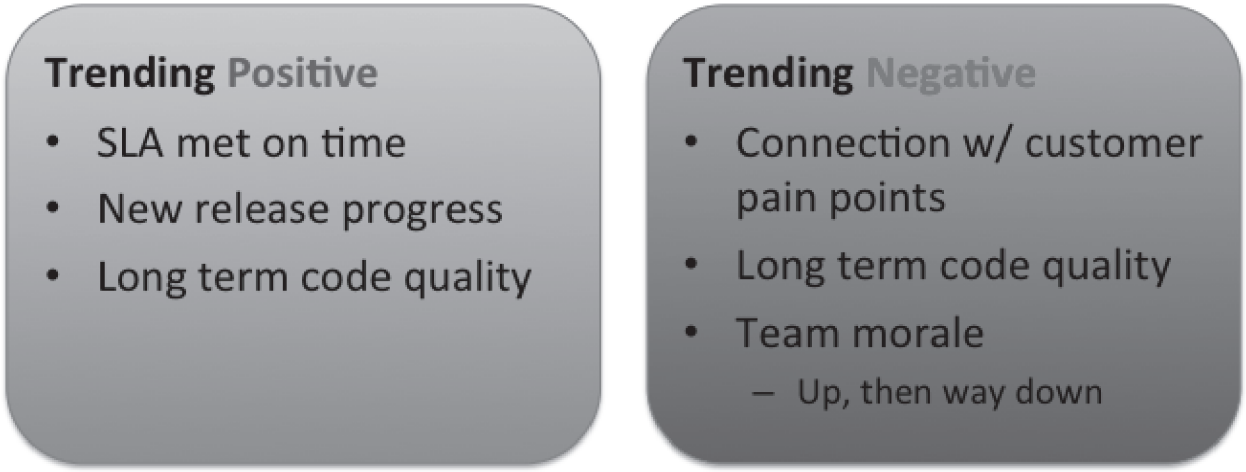


Figure 2 Summary of dedicated sustaining team approach benefits and challenges

5 Why Settle for Trade-offs?

While either approach may solve a number of problems in the short run, they can be difficult to maintain productively over time. At the daily team level, terrific work using team dashboards, Kanban and other systems to raise the visibility of the sustaining tasks and SLA metrics is being done throughout our industry.

As an agile advocate, at the program design level both approaches fall well short of what should be possible.

The challenge is we want the best from each approach. We want to meet the SLA and maintain new feature delivery velocity. We want team members to be aware of and feel responsible for the pain points encountered by customers. We want the sustaining team members to be familiar with where the next release is headed and how the issues should be fixed to maintain long term code quality. And we want engaged team members who believe their sustaining efforts are a valuable use of their time.

6 Rotating Roles Pattern

The rotating roles pattern combines aspects of the previous approaches to reduce the costs to the mainline effort while delivering outstanding customer support for escalations in the best possible manner for the product.

As an added bonus it can increase team morale by leveraging the optimism & energy from team members newly rotating into role to address structural and non-trivial process issues.

6.1 Concept

The rotating roles pattern is incredibly simple; team members take on a particular role for a period of time. Generally everyone on the team take a turn in the role.

You may already be practicing the rotating roles pattern. For example, many teams see value in having everyone take a turn as build master.

6.2 Best Practices

There are five best practices required to make this work optimally for your sustaining effort.

First, enough sustaining roles are needed to cover all aspects of the sustaining process. That may be a single person, one person to triage and one to fix issues, or a web developer and a database guru, whatever is appropriate for your team and situation.

Every team must decide where to draw the line for sustaining roles. For example, many teams do not create an installer sustaining role. They create a hotfix and/or patch installer template, and borrow time from the installer guru if needed. Some teams have their rotating sustaining roles handle hot fixes and creating patches, but involve the mainline team for validating patches (in those rare cases when the automation doesn't cover everything!).

Second, the sustaining role is 100%. Sustaining team members do not work on mainline stories. On larger teams they may have their own stand-ups in lieu of mainline stand-ups.

This helps achieve three things; it prevents conflict of interest decisions between sustaining and mainline development responsibilities, keeps the set of sustaining roles as few in number as possible, and provides potential time for sustaining team members to use their own judgment to "do good things" related to sustaining the product if there are no open escalations.

Third, sustaining team members are 100% responsible for the team meeting their SLA commitments. There are no fingers to point, no excuses to give, responsibility for meeting the SLA stops here. That's a lot of responsibility, so we give sustaining team members one special power to help meet the SLA goals

Fourth, sustaining team members can ask for help, on an interrupt basis, from any non-sustaining team member at any time.

But wait! Why do we bother with dedicated sustaining roles if they can interrupt the mainline team, you ask?

This is the magic sauce that makes it all work in the real world. In the real world team members are not fungible; product teams have a mix of senior and junior developers, newbies and old hands, specialists in different product areas or technologies. At pre-set levels before an escalation goes out of SLA

compliance, such as 60% and 90%, sustaining team members are advised to get additional help from the mainline team.

In most cases group dynamics work to motivate sustaining team members to do everything in their power to avoid requesting help. The team lead, architect, scrum master or manager can help subtly emphasize this by having out-going sustaining team members demonstrate what they fixed in the product, learned about customers or improved process-wise.

Fifth and finally, the length of service for the sustaining roles must be set. It is critical that the role lasts long enough for a sense of ownership to be established during each cycle, so think in terms of multiple sprints. We also want each sustaining team member to fully address far more escalations than they hand off to the next cycle's incoming sustaining team members.

Depending on the average length of time required to address an escalation, this could be one month, two months or even a quarter.

6.3 Benefits

Over time many different, fresh perspectives are applied the process. Items that merely annoy one team member may be an irresistible thorn for another that must be addressed so peace, harmony and productivity are restored.

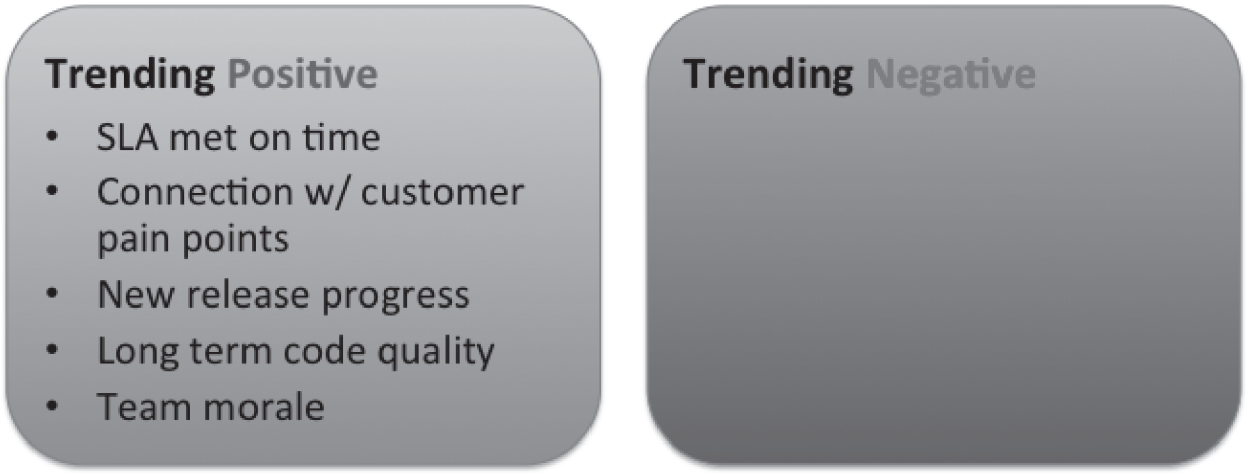


Figure 3 Summary of rotating roles sustaining approach benefits and challenges

7 Conclusion

Leveraging the rotating role pattern for your product sustaining efforts can help maintain team moral and enthusiasm while delivering new features and addressing customer escalations. Five essential best practices can help turn a serious challenge to team stability into a source of pride.

1. Create a set of dedicated, rotating roles to handle all aspects of the sustaining process.
2. Sustaining roles are 100% of assigned team member capacity.
3. Sustaining team members are 100% responsible for meeting the SLA
4. Sustaining team members can ask for help, on an interrupt basis, from anyone on the mainline team.
5. The sustaining role rotates on a longer timeframe, such as monthly or quarterly.

Thank you for your time. For questions, comments or insights I can be reached at don@black-box.com.

The ‘Toolbox’ Testing Approach

James Gibbard – Intel Security

james.gibbard@intel.com

Abstract

In an ever changing and increasing world of agile software development, automated testing approaches have struggled to keep up with the amount of software that can be developed. This has been exacerbated with the recent trend of using multiple development technologies to form a complex stack on which software products are created.

The usual approach to this problem from an automated test tooling perspective, is to make use of multiple, separate test tools/harnesses. These tools can only provide test automation for some of the complex development stack. This can be an expensive, heavyweight and inflexible solution, that struggles to provide testers the agility they need to implement robust, automated tests. Large COTS (Commercial Off The Shelf) software is normally rigid in its ability to be customized, or adapted to test different technologies it was not originally intended for.

This paper sets out a solution to the problem, with the use of a ‘toolbox’ test approach, leveraging the power of Robot Framework. The approach provides flexibility by allowing testers to utilize multiple automated testing techniques and technologies, using one easy to use, but powerful automation framework. The key and the purpose of this paper, is to help you understand how to fit all the tools together, to form your ready-for-anything toolbox.

The toolbox does not replace your existing tools; instead it helps you to utilize them more effectively. All the information, links and implementation examples are provided, so that you can have the flexibility to reach into your toolbox for whatever mix of technology comes your way to test.

Biography

James Gibbard is a Production Operations, QA Technical Lead at Intel Security, currently working at the Intel Security site in Aylesbury, Buckinghamshire, UK. Over the last 10 years, he has been involved in software testing of security software and web analytic solutions. This has ranged from hands-on testing of anti-malware engines with live virus samples, to managing QA teams, to setting software test vision and strategy for entire departments.

James has a BSc (Hons) in Internet Computing from the University of Hull, UK and is an ISTQB Advanced Level Test Manager.

1 Introduction

Software development teams are being asked to deliver value at increasingly faster rates, and to higher quality than ever. At the same time, software developers are making use of much more complex software stacks in order to supply this value to the customer, on time and to budget.

Software stacks comprise of layers of programs that work together towards a common goal. These layers are separated by logical boundaries, usually that coincide with the groups main function or purpose (e.g. database/storage layer). Each layer can be said to work 'on top' of the other, producing a stack.

The problem this paper sets out to help solve is the software tester's ability to effectively test this tornado of software, requirements and time constraints, in a flexible and pragmatic way.

Many software companies make use of large, expensive COTS (Commercial Off The Shelf) test automation tools. These typically don't have the adaptability to cope with the increasing complexity of today's software development stacks.

This paper will provide an approach and tooling, that can be used instead of these COTS applications.

The approach and tools have been used successfully and have helped to provide a great foundation for automated testing, which has led to multiple projects delivering with high quality.

2 The Challenges

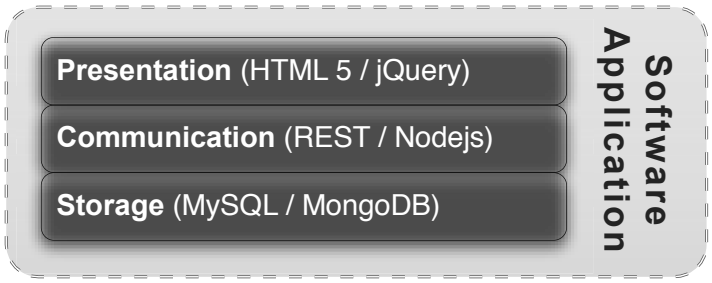
Software test departments that have purchased large test automation tools are starting to face problems, when asked to test a project that uses technology that is not normally used.

The test automation engineers are then normally faced with the daunting task of coaxing the tool to test aspects of the software that it doesn't support out of the box.

2.1 The 'Thin Vertical Slice' Challenge

The main challenge with COTS test tools, are that they normally specialize in one type of test automation or programming language (e.g. it can only test C# developed software applications). This is no good when a test department or agile team is asked to test a software application that uses multiple technologies (not just C#).

For example, many of today's web applications are made up of the following:



Each of the layers needs testing separately, and then the entire solution needs testing to ensure it all fits and works together as expected. This is a daunting proposition if your test automation suite only supports testing one or two of the technologies used in this stack.

Due to the cost of most COTS test tools, the test department or agile team, may be faced with writing custom code or extensions for the suite, as there may not be budget to start from scratch.

2.2 Cost of Maintenance

It's a technically challenging task to modify a COTS test automation suite, to test technologies it's not designed to. It's even more challenging to maintain that custom code over an extended period of time.

At any point the software under test changes, it often means that the custom code needs to be updated to ensure compatibility. This all makes for an expensive and labor-intensive solution to a problem that will continue to cost money and time.

2.3 Record & Playback Won't Cut It Any Longer

There is an ever increasing set of situations where the Record & Playback approach offered by UI test automation tools doesn't provide the level of inspection needed to properly validate a complex software application.

One such situation may be if you wanted to test a Web UI's ability to correctly enter data into a database. If there was no method of validating the inputs via the UI, a record and playback approach would only allow you to validate that the UI didn't return an error.

This just isn't good enough, especially for teams that are developing in an agile manner, who may not have all the functionality in each sprint, with which to validate against.

3 Test with a Toolbox Approach

The toolbox approach is just that, an approach. It's not a tool, it's not a framework, its not even something you can download. It could even be called a philosophy.

It's actually an approach to writing automated test cases that provides the ability to, for example, combine UI testing with DB testing. It has the ability to test various levels of the software stack in isolation, but also provides a flexible approach of the testing of the layers together.

3.1 Toolset Requirements

Although it was just said that it's not a framework, a toolset is needed to help implement the approach. The following is a list of requirements, which will help to ensure that the toolbox approach can be used successfully.

1. Be usable in multiple testing situations (Smoke tests, Functional tests, Regression tests, Acceptance tests, simple tests, and complex tests)
2. Be able to be used for testing multiple types of software and applications (e.g. Command-line tool testing, GUI testing, HTTP/REST testing, etc.).
3. Be able to aggregate the results of the tests into an easy to understand and presentable form automatically.
4. Be able to be plugged into a CI (Continuous Integration) system to execute test runs (or similar test runner) (e.g. Jenkins)
5. Be flexible enough to run on a test/dev box (standalone), and on a complex remote test automation system.
6. Be able to allow test case reviews by people who may not understand programming. This can be helped by having clear, concise and human readable test cases.

3.2 Introducing Robot Framework

In the example used for the basis of this paper, we used Robot Framework as the main toolset that enables the toolbox approach.

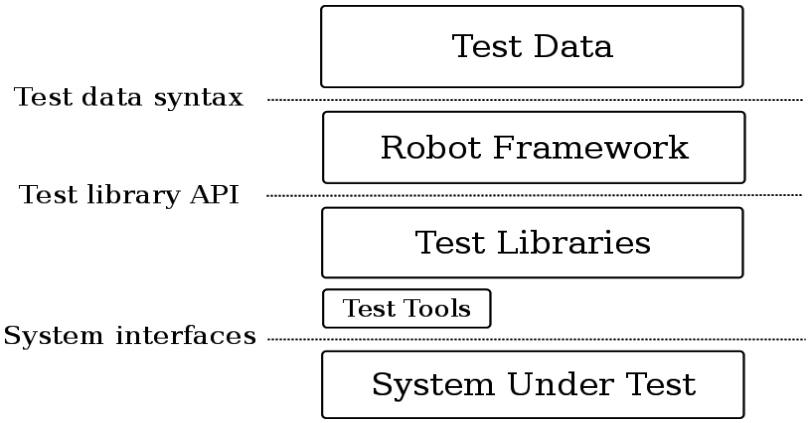
“Robot Framework is a generic test automation framework for acceptance testing and acceptance test-driven development (ATDD). It has easy-to-use tabular test data syntax and it utilizes the keyword-driven testing approach. Its testing capabilities can be extended by test libraries implemented either with Python or Java, and users can create new higher-level keywords from existing ones using the same syntax that is used for creating test cases.” (*Robot Framework*)

3.2.1 Multi-Platform Support

Robot Framework is compatible on both Linux and Windows, as it’s based on Python. It has a vast array of commands that work on both platforms. This is a great bonus for writing test cases once, which can work in any environment.

3.2.2 Modular Structure

Robot Framework’s power and beauty, and what makes the ‘toolbox’ approach possible, is its ability to allow users to create their own plugins or libraries that fit into the framework.



These libraries fit directly into Robot Framework by being loaded at the runtime of the test suite.

Libraries are called by your test suite by simply specifying the class name of your library as a configuration (*Robot Framework User Guide: Creating Test Libraries*).

3.2.3 Built-in Libraries

Robot Framework has many built-in libraries that can help to create test cases with no programming necessary. These include:

1. OperatingSystem (copy files, run commands, get environmental variables), great for running simple or complex system commands, that you might have done using a batch/bash script. (*Built-In Library Documentation: OperatingSystem*)
2. Collection (create list, add to list, remove from list), allows you to create arrays/lists of items that you test cases can make use of. (*Built-In Library Documentation: Collections*)
3. String (convert to lower case, get substring, get matching regex), provides powerful string manipulation functions without having to use awk or grep tools. (*Built-In Library Documentation: String*)

3.2.4 Custom Libraries

Custom libraries allow you to create your own keywords. These libraries can be implemented in Python, Java (using Jython) or C (using the Python C API).

3.2.5 Remote Libraries

Remote libraries are similar to the custom libraries, but they allow remote connections to external tools or applications that run in separate processes.

As an example (and one that I have used in the past), it is possible to create a remote library that can interact with a C# UI, by use of a remote C# service, that in turn uses the MS Automation Interface.

3.2.6 Combining Libraries

We have used both the Built-In and Custom libraries. They have proven very effective in allowing the creation of keywords that help to test the very complex stack of software used on our projects.

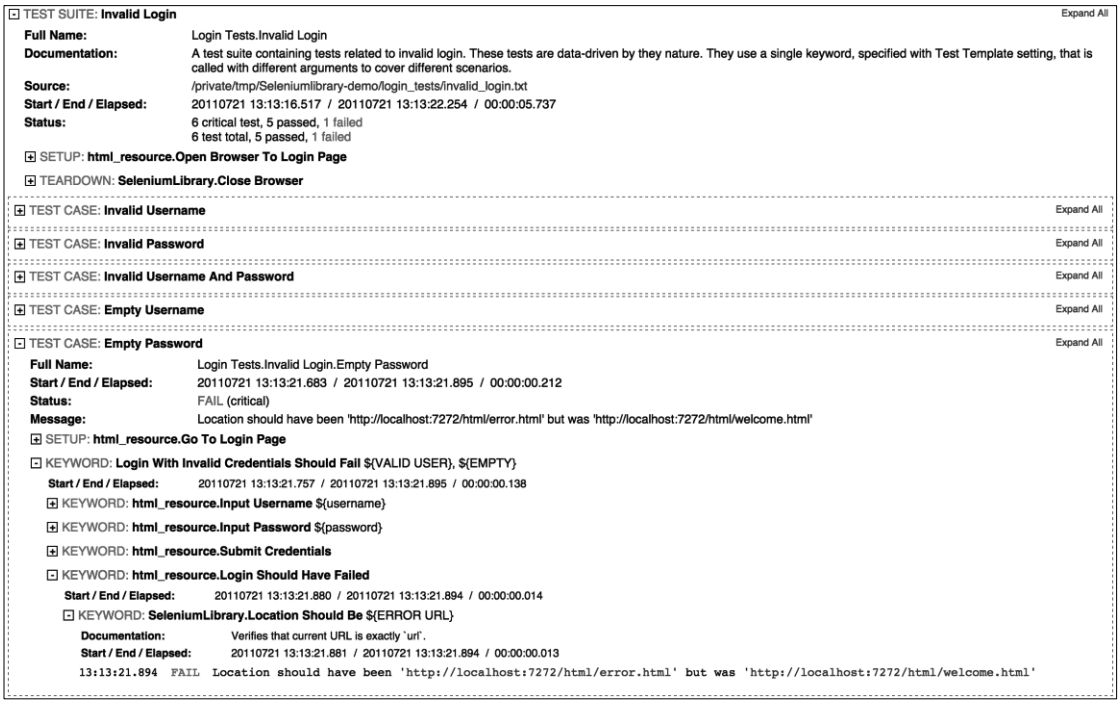
3.2.7 Other Benefits of Robot Framework

Some of the other numerous features and benefits of Robot Framework are:

Continuous Integration – Robot Framework integrates easily with Jenkins, using a freely available plugin. This provides result rollup, integrated logs and report, and much more.

Configuration Management – All test suites and test cases are saved as text files, custom libraries can be saved as Python files, so all elements can easily be version controlled.

Simple but powerful test result reporting/logging – One of Robot Frameworks best features, is its built in reporting. It rolls up all passed test cases, and automatically expands failed tests, to help aid investigation. An example is show in the following image.



3.3 High-level, Abstracted Language Automation Frameworks

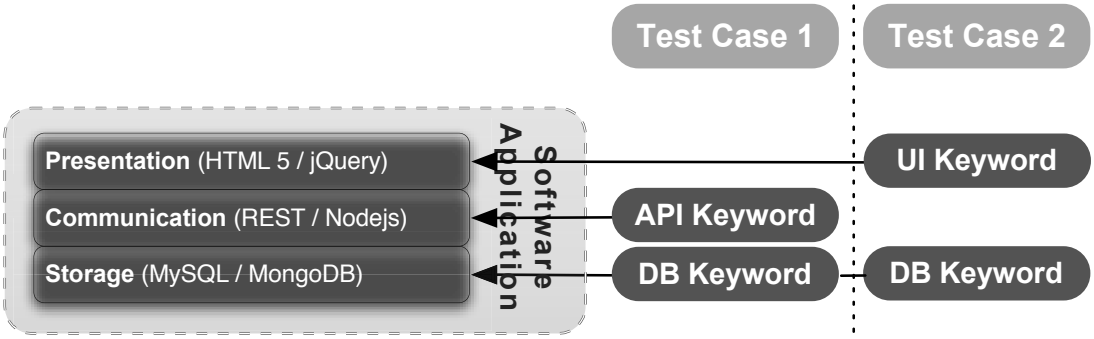
The whole reason why the toolbox testing approach is possible, is due to the introduction of high-level, abstracted language automation frameworks, such as Robot Framework and Cucumber (*Cucumber*).

They provide the ability to use keywords derived from different libraries or tools, in the same test cases. These test cases can then start to interact with the various levels of a software application (or stack).

What's more, due to the abstracted language that the test cases are written in, it allows the whole team (even members who are not familiar with code) to review, learn and suggest changes to automated test cases. This opens up a whole new perspective on test case reviews, and provides confidence that the test cases are testing the correct things, in the correct ways.

Additionally, in Robot Framework, test cases or parts of test cases can be re-used in multiple places. This provides a solution to keeping maintenance overheads down of commonly used test steps.

For example, you may have a test case that was initially created for testing an API & database, which can be re-used as part of a functional test case, once a UI is created. In the following diagram the 'DB Keyword' is re-used in Test Case 2.



4 Solving the 'Thin Vertical Slice' problem

As described earlier, the increasing uptake of agile in software development and the pressure on teams to provide value is getting greater. Most teams try to provide value by focusing on a thin vertical slice of the final software product.

Now that we have our framework, our toolbox can start to be developed, to help us test each vertical slice created by the team.

4.1.1 Identify & Standardize on Testing Tools

Firstly, start to identify your own team's software stack, and decide which tools you'd like to use to test them. This will require some level of standardization, but this is a good activity and can help to ensure that testing is done to a consistent level across your team.

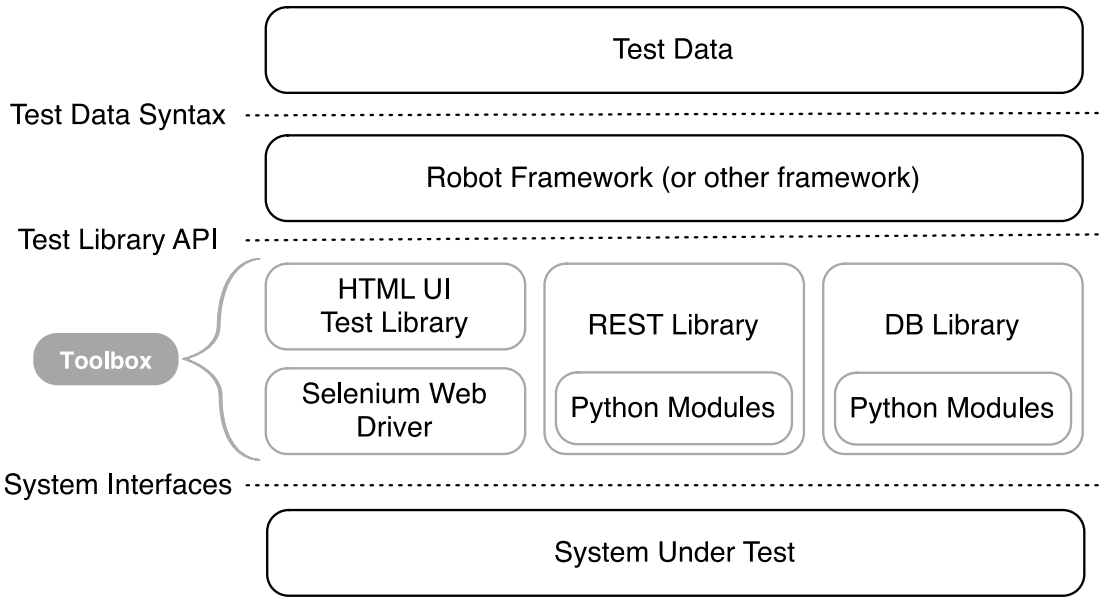
Tools can range from command-line tools (e.g. network latency emulators) to Python libraries that are not part of your test automation framework already. You may also feel that the libraries/tools available built in to your framework (e.g. Robot Framework) or available online, are adequate enough for your testing (e.g. Selenium Library).

4.1.2 Build Your Tools / Libraries

If you are going to write your own tools/libraries, once you've worked out which tools you are going to use, start to identify the ways that you want to interact with them. Which keywords are you going to need to utilize your test tool? You could use examples from other libraries and apply them to your tool.

In Robot Framework, building libraries involves creating Custom Libraries which can be imported into your test suite. (*Robot Framework User Guide: Creating Test Libraries*)

The diagram below shows how your toolbox contains various libraries, that are utilizing different technologies and tools. These are all loaded into Robot Framework and can now be called from your test cases.



4.1.3 Putting it all together

The key to the toolbox approach is to start to utilize your tools/libraries together when creating your test cases. Import your library for interacting with the API and import your library for connecting to the database.

Now you have the ability to easily validate that data inputted via the API is correctly stored in the database without having validate output on the UI (which may not have been implemented yet).

The following is an example of a Robot Framework test suite, containing two test cases.

```
*** Settings ***
Library     OperatingSystem
Library     Database_Lib
Library     UI_Lib
Library     Common_Lib

*** Variables ***
${API-URL}  http://localhost/api/user/create
${UI-URL}http://localhost/ui/users
```

Global settings, including importing your built-in and custom libraries

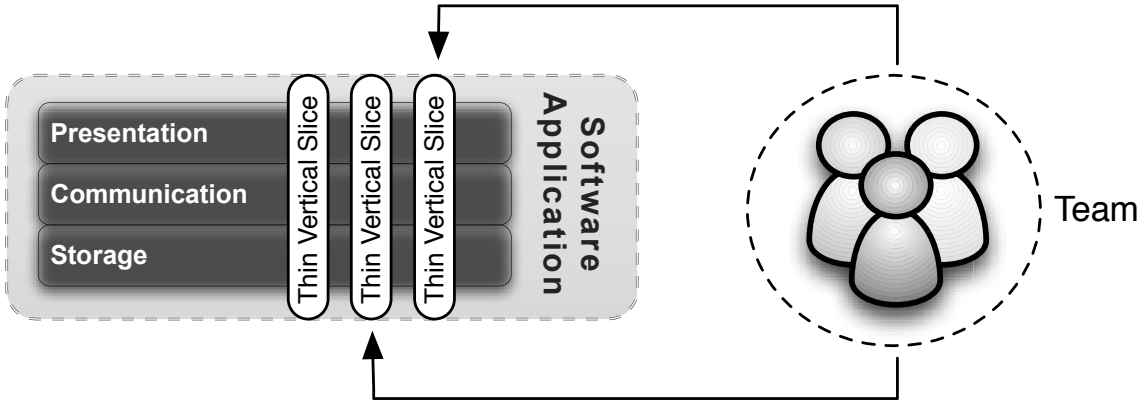
Global variables that can be reused in multiple test cases

```
*** Test Cases ***
TC01 - Verify new user is created (in Database) using API
   [Tags] API Regression
   Create User    qa-user     ${API-URL}
   Verify User in Database    qa-user
   [Teardown] Purge User from Database    qa-user

TC02 - Verify new user is created (in UI) using API
   [Tags] API Regression
   Create User    qa-user     ${API-URL}
   Open Browser   ${UI-URL}   Chrome
   Verify User in UI    qa-user
   [Teardown] Purge User from Database    qa-user
```

Test Case 01: Initially created to test the API & DB

Test Case 02: Reusing the 'Create User' keyword from TC01, but testing the UI

The whole team now has the agility to start testing elements (slices of the final solution) in an efficient and effective way.



4.1.4 Fill in the gaps / Augmenting your test cases

In some cases, a library may not always be as reliable or as able to interact with a software application as you would like (this could be due to bugs or poorly implemented functionality). In this case its possible to make use of another library to fill in the gap (i.e. augment your test case). This can ensure that you are efficient at getting test cases written and software tested on time. You can then return to the test case at a later stage to make it more robust, once bugs have been fixed in your primary library or when better features exist.

For example, your C# UI Test library is unable to click and drag UI elements to certain coordinates on the application under test. Having another library, using a very different technology (e.g. Sikuli, an image recognition and UI interaction technology developed by MIT) (*MIT. 2015*), you are able to import the Sikuli library (as well as your C#), which you can use for the click and drag step. Then switch back to your C# library for the remainder of the test case.

5 Future Direction

Due to the simplicity and readability of the libraries that can be created in these high-level, abstracted language test frameworks, future usages could include:

- Allowing customers or business stakeholders to be involved in test case reviews (of automated tests).
- Using the framework and custom UAT (User Acceptance Test) libraries, automated UAT tests

could be created by the user.
- Due to using one tool to test various technologies, a team will be able to write test cases faster instead of having to learn different test tools for different layers of the stack.
- Commonly used libraries and keywords can be refactored to be globally usable, providing a vast array of keywords to use on new projects.

6 Conclusion

In conclusion the Toolbox testing approach can enable teams to have a flexible, pragmatic approach to writing automated test cases, while still being able to adapt to changing requirements.

It no longer has to be a daunting task to create test automation for complex stacks of software.

With the ability of Robot Framework (and other keyword driven frameworks) to write automated tests in an easy to understand and reviewable format, it starts to break down the barriers to test case automation.

This approach has really helped teams I've worked with to implement testing that provides value and at the same time achieve high levels of quality.

References

Robot Framework, http://robotframework.org/ (accessed June 1, 2015).

Robot Framework User Guide: Creating Test Libraries, http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html - creating-test-libraries (accessed June 1, 2015)

Robot Framework User Guide: Using Test Libraries, http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#using-test-libraries (accessed June 2, 2015)

Built-In Library Documentation: OperatingSystem, http://robotframework.org/robotframework/latest/libraries/OperatingSystem.html (accessed June 2, 2015)

Built-In Library Documentation: Collections, http://robotframework.org/robotframework/latest/libraries/Collections.html (access June 2, 2015)

Built-In Library Documentation: String, http://robotframework.org/robotframework/latest/libraries/String.html (access June 2, 2015)

MIT. 2015. “Sikuli Script”, http://www.sikuli.org (access June 5, 2015)

Cucumber, https://cucumber.io/ (accessed June 7, 2015)

So You Think You Can Write a Test Case

Srilu Pinjala (Sridevi)

http://www.linkedin.com/in/SriluPinjala

Abstract

Does the **QUANTITY** of test cases parallel to **QUALITY** of the test cases and compliment testing?

Issues with test cases:

1. Most companies have vague test cases or no test cases.
2. Test Data, environment not stated in test cases, the tester has to guess these conditions.
3. Test steps are vague with multiple actions stated in one step.
4. Expected result description is vague or null with no GUI changes described.
5. No way to figure out which modules / flows / pages / components have test cases.
6. No way to measure the completeness of a test case.
7. Focusing only on the requirement and testing with a tunnel vision.
8. Redundant test cases costing more time and money but adding no value as per testing.
9. Adding more test cases to increase count for each requirement introduced.

We will look at typical test case examples. Identify what they lack. We will rewrite them as scenario based test cases with sufficient verification. We will work to understand and establish verification criteria for Software in general and for each product in particular. We will learn to write test cases that are robust, reusable, recyclable and ready for automation. We will learn to write QUALITY test cases.

Learning Objectives:

1. The test case belongs to the product.
2. Test steps are of two kinds – for Testing and for navigation
3. Test steps are user actions on a graphical user interface with test set up and re-setup
4. Expected result are the changes on the Graphical user interface not user perception.
5. Do one thing at a time, stop cramming.
6. Increase Scenario based test cases with complete flows
7. Reduce redundant actions and increase actual tests
8. Organize test cases based on application components / modules.
9. Write it once, and Write it accurately.

This paper introduces the concept that the test case should belong to the product not the requirement. The method coerces the tester to think outside the box to make the product consistent and to maybe fix the requirement too.

Biography

Sridevi Pinjala (Srilu) has been a QA professional for 10 + years. She worked at Porch, Amazon, IBM, etc. She came up with Green Lantern Automation Framework when the scripts are agnostic to the User Interface. She presented at PNSQC in 2011 and 2012. She is an expert at testing customer facing applications. More fun Stuff – http://qasrilu.blogspot.com

1 Introduction

In the age internet of all things, cloud computing, mobile communication, advanced extreme programming, exploratory testing, we still do not have clear definition of how to write a test case.

There are tons of literature online about how to write a test cases with types of test cases, review dates, created date, requirement it belongs to, approved by, etc. But no literature about how to actually approach the art of writing a test case.

Scenario	Test step	Expected Result
Verify cell phone charges	Charge the cell phone	Cell phone charged successfully

Now, that was not so hard. Any idiot can write a test case. Well, a couple of things –

Charge the cell phone –

- Did we test with the right cell phone?
- How do we verify we are using the right cell phone?
- How do we know it was plugged in correctly?
- Where do we get the cell phone under test?

Cell phone charges successfully –

- Define what successful means!! A blinking light?
- A straight light? (What if it a notification light for a different application was understood by a tester as the charging light and they have been marking it PASS all through the project cycle!!)
- What about scenarios like - How long it takes to charge the phone? How do we know if it failed?
- What is the obvious fail criteria for this test?

1.1 Who would benefit from this paper?

- You are new at a company where the test cases do not make sense or are incomplete or vague.
- Your product has been acquired, you have no test cases or requirements to work with.
- New testers are completely dependent on legacy testers for application testing and set up.
- Test cases are not helping to test the application accurately or completely.
- Existing test cases are missing several steps, data, environment, etc.
- Test cases written by one tester are not understood by the other testers.
- You are constantly adding test cases, but the testing needs do not seem to be meeting.

2 What is a Test Case?

As per IEEE Std. 610.12-1990 –

(A) "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program or to verify compliance with a specific requirement.
(B) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.

2.1 What is the purpose of a test case?

"A test case has two purposes: to expose an error, or to demonstrate correct execution." (Jorgensen, 2014)

1. Test case describes how to test a product or process.
2. Test case guides the user to use the product accurately.
3. Test case assists in identifying defects in the product.
4. Test case helps expose errors during regression.

2.2 What are the different kinds of test cases? Why are they different?

A test case has steps that describe the steps needed to test a feature (positive, negative, edge, boundary value, etc.) and steps that purely help navigate to the desired feature to test it.

2.2.1 Exploring / Testing

- These test cases explore the application, pages, and elements, the flows, the functions, to verify various criteria.
- These test cases have one user action at a time with one or more verification criteria.
- These test cases describe the testing methods or user action to the T.
- These test cases can have plenty of steps doing all kinds of testing on one specific feature.
- They take up more test steps.
- They are very specific.

2.2.2 Piloting

- These test cases provide adequate description to go through steps with adequate verification to get to the features that will need to be tested.
- These test cases can contain more than one user action usually a whole function with adequate verification (Ex: Create account, Login as user Joe, Logout, etc.)
- One function can be stated for execution per test step with adequate verification.
- Two or more piloting test functions cannot be crammed into one step.
- They need be specific but not detailed.
- It is important to make sure every pilot test case has adequate test coverage / test cases to test it.

2.3 What does the Test Case belong to? Requirement, Project, Product?

Many companies focus on showing the requirements, user stories, use cases, featurettes have been met through the test cases. So, the test cases are modeled to be traced to the requirements have been satisfied. This was the norm for a long time. (Merely making sure the requirement has been met does not complete the testing of the product.) But the requirements change, and sometimes they are no longer relevant.

The projects are born to fulfill certain requirements. But they sunset one day too. It does not make sense to keep tracking back to the project from 3 years ago.

Projects come and go. Requirements come, change and go. The core product is forever. The test case belongs to the Product. The test cases should target to test the product. Showing the requirements have been satisfied is part of the testing but not the only purpose.

2.4 Who are the stakeholders for the Test Cases?

Every product has stakeholders and target user base. Test cases also have a user base. Identify the users to be able to fully provide sufficient information to them.

a) Author of the test cases, Other testers in the company
b) User Acceptance testers, Beta testers, End users
c) Business Analysts, UX developers
d) Developers, Architects
e) Project managers, Scrum masters, Program managers
f) Upper management, etc.

Often the author and other testers are the users of the test case. Seldom other team members can and should be able to use the test cases to understand and test the product.

2.5 What is the ideal time to start writing Test Cases?

The norm is for the test cases be written as soon as the requirement is signed off on. But requirements talk about what an application should do and should not do. Their descriptions are vague and left to interpretation. Their implementation is kept neutral on purpose to be abstract like. So, requirements can give you an understanding of what to expect and be prepared for but not write exact steps for testing.

It is assumed the test cases should be written by the time the build is ready. But there is no way to know the exact steps and elements involved in the tests. Writing the test cases at this point would result with vague steps.

So, the best time to write a test case is – After the first round of testing is complete for a feature.

2.5.1 After the first round of testing

1. The interface is available to look at. Pages, elements, data, conditions become clear which makes describing accurate.
2. Working off of requirements limits the test step and expected result to assumptions
3. Assumptions will not enable the tester to test accurately and verify accurately.
4. Test case serves for many QA cycles all through the life of the product, not just the first cycle.
5. One gets a chance to explore the application before documenting the steps and verification criteria.

3 Key Components

As per many online sources – Test Suite ID, Test Case Id, Test Case Summary, Related Requirement, Prerequisites, Test Procedures, Test Data, Expected Result, Actual Result, Status, Remarks, Created By, Data of Creation, Executed By, Data of Execution, Test Environment, blah, blah. Depending upon the company's philosophy, some or all or more of these components are necessary components of a Test Case.

But the components that add value to the purpose of testing are the Test Step and Expected Result. The rest of the components are bells and whistles. Some of those components only serve administrative purposes. Today we are here only to talk about the **Test Step** and **Expected Result**.

3.1 Test Step

The Test Step is the user action. It is also the tester's actions to input data, set up application state, navigate to the feature in test.

3.1.1 Sample User actions and Tester steps

- Go to *URL* **something**
- Type in *field* **something**
- Type in *field* **username** – Excel.xlsx > tab1 > column1
- Type in *field* **password** – Excel.xlsx > tab1 > column2
- Select from *drop down list* **something** - data
- Click *button* **something**
- Click *link* **something**
- Check ON or OFF *check box* **something**
- Hover over *menu* **something**
- Go back one page, Refresh the page, Go forward one page.
- SQL - [`SELECT userID, userKey FROM Users WHERE userKey = 7`]
- SQL - [`SELECT TOP 5 * FROM Search WHERE searchType = 'service' AND searchResult LIKE '%ab%' ORDER BY searchResult ASC`]
- In DDL Search only, Check ON checkbox **People,** Type in *field* **Search –** data
- SQL – [`DELETE FROM Users WHERE firstName = 'john' AND secondName = 'jason';`] & [`SELECT * FROM Users WHERE firstName ='john';`]

3.1.2 Test Step – Dos

- Keep it clear, simple, specific (CSS).
- Do it right to test it right. So, explain it accurately.
- Avoid company jargons, abbreviation, etc. Treat the software elements as such.
- One step at a time / one function at a time / one action at a time.
- Instruct to use specific data sets for testing.
- Test steps should be present tense. The same tense the tests are being executed.
- Like talking to a Robot that does not understand logic and reasoning, only specific instructions.

3.1.3 Test Step – Don'ts

- Avoid instructions like – repeat the above steps or execute steps from 6 to 9. Tell the tester to execute specific functions instead.
- Don't cram multiple steps or functions into one test step. Verifications can be lost.
- Don't just mention a feature and expect the tester to know what to do with it.
- Don't just say use valid data or invalid data.

3.2 Pilot functions

It is best to execute one step at a time while exploring and testing a feature. But often we will need to execute certain functions to get to the feature we are going to test. These functions help navigate. They are crucial for this test case. But they do not need to be tested in this test case because they have been tested thoroughly in other test cases.

- Login to app – implies – go to URL, Log in and verify
- Log out – Click on header, click on log out link, verify logged out.
- Delete User – Delete a specific user from the data base using a SQL query.
- Register Account bankrupt – DQL for bankrupt Account, go to URL, and Register the account.

3.3 Expected Results

Expected Result is the changed state of the AUT. Typically involves GUI changes that are instantly visible, Data changes that need to be verified, other changes that have to be verified using tools.

3.3.1 Sample Verification Criteria

- Page appears with *title* **something**
- Header consists of *links* **something**, **something**
- *Error message* appears – "**something**"
- Dialog box **something** appears.
- SQL Verification - `userID = srilu.pnsqc@gmail.com`
- SQL Verification – 0 results.
- Logo - **International** (logo.png); Tabs - **View Saved Properties, Change Location, English (United States);** Text – **Global;** Field - **Search** with placeholder "Search"; Button – **Search;** Drop down list : ***(search only);*** Menus - **Research, Search Properties, Explore Our Services, Blog, About**
- "Page appears with Title – "Search | Global | Something International", CSS - (Home page elements), Header Search field is blank, Body - Search Results with field Search and Button Search, Links - Go to Property Search > Left Nav - Show Only"
- Auto-Complete appears with up to 5 matching suggestions with only Service names
- SQL Result - Top 5 rows match the search results

3.3.2 Expected Result – Dos

- Keep it clear, simple, specific (CSS)
- One step can have multiple verifications. List as much verification as possible.
- List and explain with IF and WHEN conditions where applicable.
- The description should be WYSWIG (What You See is What I Get)
- Verify page titles, logos, text, images, error messages, etc. verbatim.
- The results have to be tangible and verifiable, be it GUI or be it Data or both.
- Expected results should be in present tense. The same tense the tests are being executed.
- Like talking to a Robot that does not understand logic and reasoning, only specific instructions.

3.3.3 Expected Result – Don'ts

- Don't say "*it is successful*". Explain what defines successful.
- Don't say "*it has no errors*". Explain what no errors means.
- Don't say "system accepts". Explain how the system accepts, (new page comes up?)

3.4 Verification for Pilot function

Pilot functions may have multiple steps and multiple verifications. It is not necessary to verify all the steps. So, it is ideal to verify at least one unique aspect that are crucial for this test case.

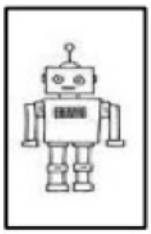
- Every function has at least one verification criteria.
- Verify page titles when on a new page.
- Verify login criteria that is unique to the credentials or account.
- Verify data on the GUI or in the Data base or both.
- The main reason to verify a pilot function is to make sure the application is in the desired state.

4 Go the Extra Mile

Test case is the place where the tester can be equipped with the tools and information they need, to get the testing done quickly and efficiently. Test cycle is not the place to test the tester's technical skills, do not assume the tester to know about the product automatically.

4.1.1 Like talking to a Robot

Enable the tester to execute the tests independently. The best way to achieve this goal By assuming the tester is a **ROBOT** who does not understand logic or assumptions. They only understand ***specific*** instructions.



Can you read the words in the Square?

If you can read them without issues, you have missed a lot of typos you could report!! (Human Error)

If only you copy pasted the text in a word document, you would have caught the typos easily. Suggest such ideas to the future test performer too.

Tihs praapagrh has mxeid up leterts but you can siltl raed it. It's aaizmng how the biarn wkors. I dno't konw who wloud hvae gseesud you cluod raed jlbuemd wdors. If you are eetmrelxy berod you cuold mkae yuor own ralely mexid

(Read the text in the square box?)

4.1 Wikis

Information gets lost over time. Create and maintain Wikis with information about various test components. Reference this information in the test cases where necessary. The Wikis can be updated periodically, the test cases need not be touched.

4.1.1 Environment

- Server operating systems, client operating systems, database servers, browsers, versions, etc.
- Don't just mention the OS, browser and its version, provide the location of these tools.
- Sometimes a browser needed for the test won't install on the standard operating systems. It will be helpful to provide information about which Virtual Machine to use and what credentials to use.
- If the test requires for connecting to a Database, provide information about names and credentials to be able to connect to the database.

4.1.2 Virtual Machines

Specify or list information about Virtual Machines with the necessary browser versions and other applications along with the credentials needed to use to log into them in the Wikis.

4.1.3 Databases

Specify or list the databases relevant for the product and testing in Wikis. Note down the credentials needed to access them. Also note down the owner for the database, along with where to get the credentials from, if needed.

4.1.4 Data Sheets

Create and maintain Excel spreadsheets (typically) that can be used for manual as well as with automated testing in an agreed location. Specify the location for each file in the wiki. If the location of the file name changes, the wiki needs to be updated but the test case does not. The data in the file can be updated without affecting the wikis or the test cases.

4.1.5 Test Data

- Test data may be listed in the test steps.
- It may be listed in a separate column too. That will make editing it a lot easier.
- “Valid account”, “invalid credentials”, “system admin”, etc. is not test data.
- Provide links to the Wiki. Provide the location of the Excel sheets with test data.
- Provide SQL queries with step by step instructions to locate the needed test data.

Provide steps and tools to create test data whenever necessary in the Wikis, which will make it easy to update the steps when needed.

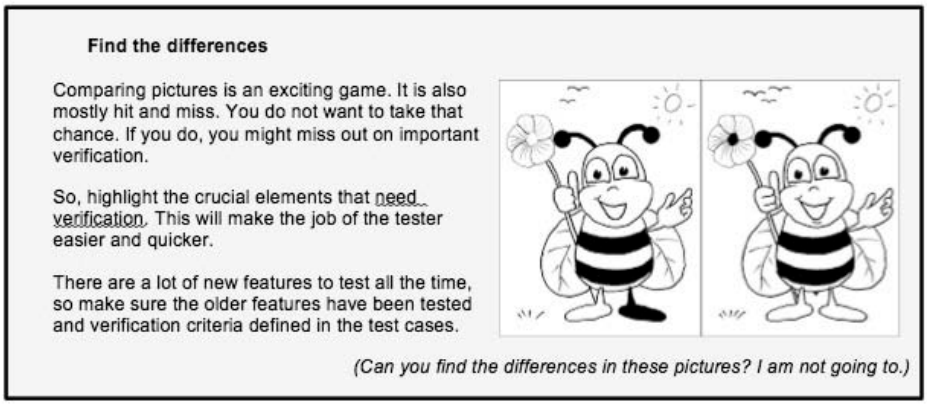
4.1.6 Test Setup and Re-Setup

- Test set up is not a one-time thing before the execution of the test. It could be done during too.
- Environment may be set up and updated and modified several times during the test execution.
- Data may be set up and updated and modified several times during the test execution.
- Example – Register through SQL -> Login through GUI -> Update the registration date through SQL -> test password reset through GUI.

4.2 Pictures

Create and maintain Wikis with pictures of screen shots that will need to be verified. It will be easy to update the Wikis rather than the individual test cases.

Providing a picture in the test case is an excellent way to describe the steps and expected results. But there are things one can do to make the process more efficient.



4.2.1 Highlight

Posting a picture is not efficient enough. Highlight what to look for in a picture and what to ignore.

Unless the tester has a good understanding about the product and what changes to look for and ignore, you will end up with a bunch of missed bugs or more annoyingly a bunch of unnecessary bugs. That is a lot of resources' time and money and energy (and bruised ego).

5 Where to start

Where to start in the absence of any requirements or test cases or documentation. Companies go through acquisitions, mergers, reorgs and major changes. Companies let go of QA teams during hard times. During which documentations related to an application are lost. What are you supposed to do?

Revise and Study the product, Mind map the application, Explore each element, and Design test cases by naming and organizing them, Integrate test cases as scenarios with accurately fitting data and other information. Revise the test cases from time to time.

5.1 Revise the product

Use the product that is in front of you. Explore the product. Treat an element like an element. Treat a widget as a widget. Go further and understand what Databases, tables and columns the elements talk to. Most applications out there are GUI based, so this example will be for GUI based products. Study the application.

- Authentication roles
- No. of pages
- CSS Elements and Unique elements in the application.
- Search features
- Email and communication features
- Session time out and other time based functions.
- Other features, Etc.

5.1.1 Pick a style

Decide if you want to model your tests around the GUI features, Core functions or flows. The style is based on your best judgment. What works for one product won't work for others. But, failing to plan is a plan to fail. So, pick a style. The style can be changed at any point very easily.

5.2 Map the Application

Map the application by the pages, and its elements based on the hierarchy. This is your **visual mind map** that can give you a visual for the elements you know about, their hierarchy, etc.

You may list all the known characteristics for each element in the following cells, like DB it talks to, field validation, how it reacts, etc. This visual mind map may be used for your initial round of testing as an unofficial test case.

5.2.1 Example 1 (MS Excel MAP)

Level 01	Level 02	Level 03	Level 04	Level 05
menu Home				
	group Clipboard			
		icon cut		
		icon copy		
			Copy	
			Copy As a Picture	
		icon format painter		
		icon Paste		
			section Paste	
				icon Paste
				icon Formulas
				icon Formulas & Number Formatting
				icon Keep source formatting
				icon No borders
				icon keep source column width
				icon Transpose
			section Paste Value	
				icon Values
				Icon Values & Number Formatting
				Icon Values & Source Formatting
			section Other paste options	
				(recently used option appear)
			section Paste Special	
				Dialog box Paste Special

MS - EXCEL

5.2.2 Example 2 (Login page)

Level 01	Level 02	Level 03
page Login		
	field **Email or Phone**	
		EM Enter your Username
	field **Password**	
		EM Enter your Password
	checkbox **Keep me signed in**	
	checkbox **Remember my password**	
	button **Sign In**	
		EM The username or password you entered is incorrect
	Link Can't access your account?	

	link **Sign in with a single-use code**	
	logo **Outlook**	

Hotmail – Login section

This visual mind map may be used for your initial round of testing as unofficial test cases. Use this map to track your testing status and coverage just by marking them cleverly with color or *font* type.

5.3 Explore each element

Once we have mapped out the visible elements, explore each element and understand what they do.

- URL - invoke it in multiple browsers and operating systems.
- Page – refresh it, resize it, and go back and forth, Note down the elements on each page.
- Go through as many pages as possible and group pages based on similarities.
- Separate CSS elements from unique elements on a page.
- Look for spelling and grammar slip-ups, consistency in font, colors, images, and error messages.
- Authentication – understand the login flows, the data, login types, user roles, etc.
- Profile information – set up and edit and delete profile information and the many ways to do it.
- Search – when does autocomplete populate, what kind of results show up.
- Session time out – Look in the source code, configuration files, data base or just observe
- Text Field – fill it up. Understand the field validation. What kind of characters it accepts and how many and what it does not and under what conditions.
- Links – click them and understand what they open up to. Do they respond to hover, or click?
- Buttons – click them and understand what they process.
- Check boxes – check ON and OFF each option and understand what they take into account.
- Menus and Sub menus – Observe where they navigate to.
- Explore - Dialog boxes, tool tips, help files.
- Databases – note down which tables and columns are talking to the UI elements.

5.3.1 Exploratory test cases for the Hotmail Login page

Page	Attributes
Go to URL **Hotmail.com**	Page open with title – Sign In Left section – Image – www.wiki.com/mycompany/signimage (fake wiki) Field – enter your phone number and we'll send you the download link; Button – Send My Link Right section – Logo – Outlook Fields – Email or Phone, Password Checkbox – Keep me signed in Button – Sign in Links – What's this?, Can't access your account?, Sign in with a single-use code Footer section – Links – Contact Us, Terms of Use, Privacy & Cookies, Link Disclaimer ©Copyright

Element	Attributes
filed **Email or Phone**	1. DB – SELECT email FROM users; 2. Placeholder text – Email or phone 3. If registered email is typed – Page Inbox appears. 4. Field Validation – Accepts letters, numbers and characters (period, hyphen, underscore), 256 digits. 5. If field is left blank – login page persists with the typed in username and error messages in red font "Enter your Username" 6. If unregistered email ID is typed – error message "The username or password you entered is incorrect".

Element	Attributes
filed **Password**	1. DB – SELECT email FROM password – (is encrypted) 2. Placeholder text – Password 3. If registered email is typed – Page Inbox appears. 4. Field Validation – Accepts letters, numbers and characters (period, hyphen, underscore), 256 digits. 5. If field is left blank – login page persists with the typed in username and error messages in red font "Enter your Password" 6. If unregistered email ID is typed – error message "The username or password you entered is incorrect".

Element	Attributes
check box **Keep me signed in**	1. ON – the user stays logged on the particular browser when navigated to the Login URL. 2. OFF – the user will have to have to login when navigated to the Login URL

Element	Attributes
button **Sign In**	1. Unregistered email and password typed in – Error message "The username or password you entered is incorrect" 2. Gibberish email and password typed in – Error message "The username or password you entered is incorrect". 3. Registered email and gibberish password typed in – Error message "The username or password you entered is incorrect" *4.* Registered email and registered password typed in – 5. *Page opens with title –* ***Inbox*** 6. Registered email and registered password typed in (using a new IP address) – Page open with title – verify your email

Element	Attributes
link **Sign in with a single-use code**	Same page persists (title – Sign In) with message – "A single-use code lets you sign in without entering your password. This helps protect your account when you're using someone else's PC" with link "Learn more" Fields – Microsoft account, Phone number, Links – What's this? Learn more, Already have a code? Button – Text me the code

5.4 Design

Design test cases for all possible scenarios based on the findings. These test cases are simple. These test cases can be used individually or in test sets with multiple iterations and data too. Keep them ***CSS***.

As per the mapping, it is easy to remember each element's hierarchy. These test cases should have an information as to what the pre-requisites are for using them. Some may have none, all will need the product to invoke. Post a picture and highlight what to click on and what to look for.

5.4.1 MS-Excel

Sample test cases for Excel.

5.4.1.1 Click Icon CUT

Test Step	Expected Result
Select any amount of text. Click icon CUT (Home > Clipboard > Cut)	The selected text is cut (disappears) from the page / cell.

5.4.1.2 Click Icon COPY

Test Step	Expected Result
Select any amount of text. Click icon Copy (Home > Clipboard > Copy)	The selected text persists on the page. It does not disappear from the page.

5.4.1.3 Click Icon COPY: Formula

Test Step	Expected Result
Type random numbers in any column's 10 cells. Type in formula to add the numbers in the 11th cell [=SUM (A1:A10)] and press enter.	The numbers in all 10 columns are added into the cell 11 (no need to verify the calculation)
Select the 11th cell and Click icon Copy (Home > Clipboard > Copy)	The selected text persists on the page. It does not disappear from the page.

5.4.1.4 Click Icon PASTE

Test Step	Expected Result
(After using cut or copy with selected	The selected text is pasted (appears) where the cursor is

text) Click icon PASTE (Home > Clipboard > Paste)	placed.
Place the cursor in a new cell and Paste	The selected text is pasted (appears) where the cursor is placed.
Place the cursor in the middle of text in a cell and Paste	The selected text is pasted (appears) where the cursor is placed.

5.4.1.5 Click Icon PASTE: Formula

Test Step	Expected Result
Execute test case: Icon Copy: Formula	
Click in the 11th cell of the next column Click icon Paste Formula (Home > Clipboard > Paste > Formula)	Zero appears in cell.
Type random numbers in the first 10 cells of the same column	The Zero value is replaced with the value of added numbers in the 10 columns.

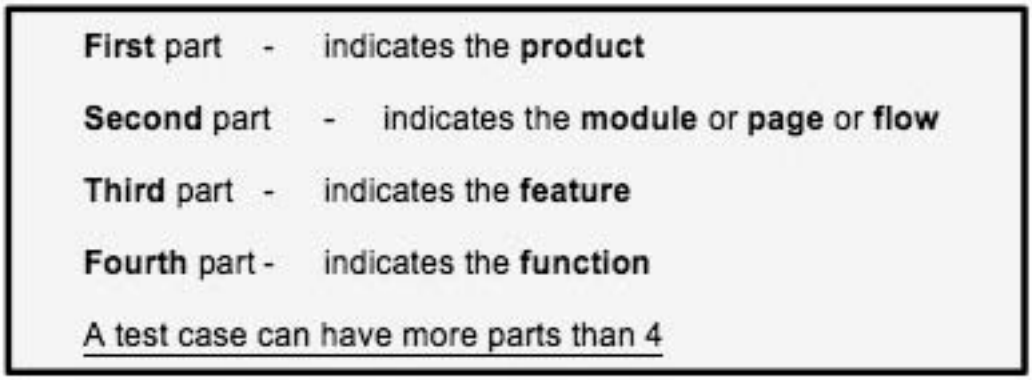
5.4.2 Naming

Naming and organizing the test cases is a crucial part of the design phase. The name of the test case must help in recognizing what product, flow or feature it belongs to and help organize it too.

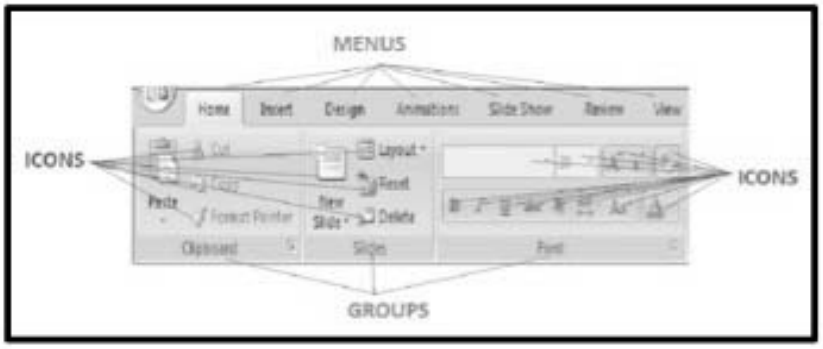
Listing the project name in the test case name does not add any value. Naming the test case after the requirement creates a traceability, but does not ensure total test coverage. The name of a test case should be able to tell the audience what area of the product the test case belongs to.

Name the test case after a feature, element or function or flow of the application.

The test case name can be divided into parts. Each part indicates a certain aspect of the product.



5.4.3 Test Case names for each features



- MS.PPT.mHome.iCut = Microsoft > Power Point > menu Home > icon Cut
- MS.PPT.mHome.iPaste.oPasteSpecial = Microsoft > Power Point > menu Home > icon Paste > option Paste Special
- Mail.fUsername = Mail > Username
- App.Search.AutoSuggest = App > Search > Auto Suggest

5.4.4 Test Cases / Test set names for Pilot functions

Pilot functions may also be written separately. This eliminates the redundant steps while navigating. Remember, there is little value in testing over and over. Test it once and test it right.

- Mail.Login = Username + Password + Sign In + Verify Login
- Mail.Register = lots of elements and fields + Verify Registration
- MS.PPT.Close.withoutsaving = exist + click no on dialog box save

5.4.5 Organize

Organize the test cases as per your plan (Flow or Feature or Functions or Element). What makes most sense to your team and what is best suitable for the product. It is all about being organized and being able to locate a test case easily for execution, editing, merging or deleting as needed.

5.4.5.1 Sample folders and test cases

Home – test cases for common elements on all pages.

Create Profile – Register through email, register through Facebook, register half way, etc.

Authentication – with email, Omni authentication, merge with Facebook, etc.

My Profile – Forgot password, change address, etc.

Upload images – Upload image files, data files, documents, delete, edit images, etc.

Communication – Send request, forward profile, announce, etc.

SQL – Select UserID7, Delete Registered users, Reset Password, etc.

5.4.5.2 Organizing test cases like this gives

- Visibility of coverage
- Minimizes execution time

- Eliminates redundancy
- Saves time when time comes for test sets.

5.5 Integrate

Test case/test set with all possible combinations as scenario based test cases are integrated in this phase. Typically the blank fields, valid credential and invalid credentials flows are separated into 3 test cases, there is no reason why the tests cannot be combined into one test case or test set. Combining tests has to be considered on a case to case basis. The obvious advantage of writing test cases like this – reduction in execution time.

This is where the test cases/set are updated a little more to tell a story with each scenario. These test case/sets can be linked to one or more requirements. They are basically user flows at this point with specific or various outcomes that have been defined. (Remember, it is not just about the product, enable the tester too, don't forget to provide them the specifics and the tools to get the testing done.)

Rule of thumb for organizing test cases as test sets maybe based on -

- Test execution order
- Priority
- Dependency

5.5.1 Example Login test case

Test Step	Expected Result
Go to URL **mail.com**	Page open with title – Sign In Left section – Image – www.wiki.com/mycompany/signimage (fake wiki) Field – enter your phone number and we'll send you the download link. Button – Send My Link Right section – Fields – Email or Phone, Password Checkbox – Keep me signed in Button – Sign in Links – What's this?, Can't access your account?, Sign in with a single-use code Footer section – Links – Contact Us, Terms of Use, Privacy & Cookies, Link Disclaimer
(Blank fields) Leave *fields* **Email** and **Password** blank. Click *button* **Sign In**	Error messages appear in Red Font – **Enter your email, Enter your password**
Refresh the page	Error messages do not show anymore.
Type anything in the *field* **Email** (asdgkdf) Leave the *field* **Password** blank. Click *button* **Sign In**	Error message appears in Red Font – **Enter your password**
Refresh the page	Error messages do not show anymore.

(invalid credentials) Type in the *field* **Email** (example1, example2, etc.) Type in *field* **Password** (asdfggdg34@#$) Click *button* **Sign In**	Error message appear in Red Font – **The username or password you entered is incorrect**
Refresh the page	Error messages do not show anymore.
(Registered credentials) Type in the *field* **Email** [SELECT userID, userKey FROM Users] Type in *field* **Password** [userKey = 12, use password **Pass1;** userKey = 13, Use password **Af14!!**] Click *button* **Sign In**	Page appears with Title - **Inbox *(number of emails, if any)* – *(email)* – Gmail** Header elements….. Left Nav….. Footer elements…. Body elements ……
SQL – SELECT * FROM accessTracker WHERE userID = '***email***' ORDER accessTime ASC	accessIN = '**current time**' (00:00:00)
Go back one page	Same page persists. Login page with login fields does not show up
Click *button* **Sign Out**	Page appears with *title* – **Sign In**
SQL – SELECT * FROM accessTracker WHERE userID = '***email***' ORDER accessTime ASC	accessOut = '**current time**' (00:00:00)
Click *link* **Can't access your account?**	Page open with *title* – **Why are you having trouble signing in**
Click *button* **Cancel**	Page appears with *title* – **Sign In**
Click *link* **Sign in with a single-use code**	Page persists (*title* – **Sign In**) with message – **A single-use code lets you sign in without entering your password. This helps protect your account when you're using someone else's PC** with *link* **Learn more** Page screenshot - www.wiki.com/my/signimage Fields – Microsoft account, Phone number, Links – What's this? Learn more, Already have a code? Button – Text me the code

More steps may be added to this test case for forgotten password, code, logging in with phone number, etc. (I am keeping this short for this paper)

Integration is NOT JUST ABOUT grouping with other test cases. This is the phase where the test data and test environment information is added to the tests. Links to Wikis, locations to Excel sheets, specific test data are added to the test sets.

5.5.1.1 Test Data

- List SQL queries where necessary to enable the tester to get to the Data needed.
- Maintain clean test data in Wikis or in Excel sheets that are accessible to all.
- Log in as Valid, registered, system admin means nothing.
- What qualifies as a valid, registered, etc. means everything.

- Suggest the sample test data. Provide SQL queries to help the test executer find the data needed for testing. Going the extra step saves time in the long run.

5.5.1.2 More

- Enable the tester to get back from the changed state (refresh the page, go back one page, etc.)
- If the tester needs to update things in Database for next steps, list the instructions to do so, step by step.

5.5.2 Pilot Test Case for Login

Test Step	Expected Result
Log in as System Administrator [SELECT userID, userKey FROM Users WHERE userKey = 7] or [Wiki – http://wiki.com/proj/pass] Or [Users.xlsx > tab – 2015]	Page appears with Title - **Inbox *(number of emails, if any) – (email)***

The pilot login function serves the purpose of navigation only. Verifying the page title suffices. These test cases do not test. They help navigate to the desired destination and the desired state for testing other features. The login pilot can be used in combination with other test cases.

5.5.3 Test case with Data Sheet

Search is a feature that can be used with multiple keywords. In such cases, use a data sheet as below. This way one test case can be used with multiple data values with obvious results.

Test Step	Data	Expected Result
`SQL - SELECT TOP 5 * FROM Search WHERE searchResult LIKE IN ('Data') ORDER BY mostRelevant ASC;`	Type in	0 to 5 records appear. (Note down the results and records from `searchResult` as Step 1)
`SQL - SELECT * FROM Search WHERE searchResult LIKE IN ('Data') ORDER BY mostRelevant ASC;`	Click On	(Note down the results and records from `searchResultLinks` as Step 2)
`SQL - SELECT COUNT * AS count FROM Search WHERE searchResultLink LIKE IN('Data');`	First Result	(Note down the count as Step 3)
Type in *field* **Search**	Type in	If Matching results exist – Auto-complete appears with up to 5 matching suggestions. Compare the results from SQL Step 1 If no Matching results exist – Auto-complete does not appear at all
If auto- populate appears Click on *search Result*	Click On	Auto-complete closes. Selected text appears in the *field* **Search.**
Click on *button* **Search**		Page appears with title – “Search”. Typed in text persists in the *field* **Search**

		Body appears with – • Text– Search Results ***(count) [SQL step3]*** • Footer text – Page 1 of ***(SQL step 3/10)*** • VCR buttons for pages appears if more than 10 results count from SQL step 3 • 10 links appear as search results.
Verify the first link in the results		Matches the ***First Result*** in the data sheet.

5.5.3.1 Test data sheet

Type in	Click on	First Result
sa	san Francisco	san Francisco jeans
bob	bob chodos	bob chodos profile
zzzzzz		
123	123 Studios	123 Studio Seattle
car	Car dealers	Volkswagen
Act	Acting	Mel Brooks
Sail	Sailing boats	Seattle boats

Using a data sheet in this scenario reduces the number of test cases and increases the iteration count.

Use If conditions in the Test cases.

Every element has multiple attributes and purposes and reacts differently based on some logic. If this logic is clearly defined in a test case, several tests can be run in a single flow.

5.5.4 Pilot Test Case for Search

Test Step	Data	Expected Result
Type in *field* **Search** Click on auto-complete **txt** Click *button* **Search**	Type in, Click on	Page appears with title – **Search Results** First link – **Something**

5.6 Repeat the cycle

Test cases are not a onetime thing. The test case will be documented and revised and refined several times through the life of the product.

- Set up time to go over the sets of test cases periodically. Review and update as necessary.
- New test cases may be added to keep up with the new requirements and new features.
- Two or more test cases may be combined. This depends on a case to case basis.
- Some test cases could be deleted due to redundancy. You don't need to test one thing twice.

5.6.1 Change in Features

An existing feature of the product has a new requirement it needs to satisfy.

- New test cases may not be necessary.
- Add or update steps to existing test cases as necessary.

5.6.2 New Feature

A new feature is being introduced in the product.

- New test is necessary.
- Add a new test case for testing and pilots.
- It may also be necessary to update other existing test cases to ensure consistency.

5.7 Owner

Test cases are part of knowledge management. Just like the knowledge management can be successful with an owner, the test cases can also be managed and kept up to date by an owner. It is necessary to have an owner for the Test Cases of any given product.

The test case author does not become the owner by default. It could be a Lead, or Manager or a product owner who is the owner. Typically it is the QA team that has the ownership.

The owner is in charge of the document's upkeep. They may or may not physically edit and update the documents. They will have trained testers in place to update and edit the test cases and concerning documents.

6 Typical Test Cases

6.1.1 Typical test case for Logging

Test Step	Expected Result
Login with valid user ID and password	Logged in successfully
Login with invalid User ID	Not logged in

In this case, what is a valid user ID and what is an invalid User ID? Unrequested? Invalid chars? What does "logged in successfully" mean? How can we be sure we verified log in correctly.

6.1.2 Typical test case for search

Test Step	Expected Result
Search for "San Diego"	Results for San Diego are found
Search for "!@#$$"	Results for !@#$$ are Not found

The test step is not specific but it somehow makes sense. But this is not enough when testing.

6.2 Revise a typical Test case

6.2.1 Typical test case for invoking URL

Test Step	Expected Result
Go to URL – https://www.google.com/	Page loads successfully Or Page throws no errors

This test case seems like a good enough test case. But what defines the criteria for being "successful"? What does "no errors" mean for the Application under Test? What is successful for Yahoo home page is not the success criteria for Google or some other home pages!! So, define the success criteria.

6.2.2 Revised test case

A simple Go To URL – (something) suffices as a test step.

Most companies have multiple environments for every application. It is a good idea to maintain one test case with all the environments listed, so that the test case uses one of those URLs based on the test cycle or test plan. All the information we need to test and verify in one location. Described simply.

Test Step	Expected Result
In Chrome browser incognito. Go to URL – https://www.google.com/ Or Go to URL – QA1 = http://qa1.com QA2 = http://qa2.com Prod = http://prod.com Stage = http://stage.com	Page opens with title – Google Header consists of – • Links +You or +*(Username)*, Gmail, Images • Buttons – [app group buttons], Sign in (blue) Body consists of – • Logo – Google • Text Field – with placeholder text Say "OK Google" • Buttons – Google Search, I'm feeling Lucky Footer consists of – • Links – Advertising, Business, About, Privacy, Terms, Settings

6.2.3 Pilot Test case

Test Step	Expected Result
Go to URL – https://www.google.com/	Page opens with title – Google

We have verified the page elements in the main test case. We are using this step as a pilot. Hence, it is ideal to do some verification. Verifying the title of a page is the best and simplest verification.

7 Test Case Life Cycle

Life Cycle of a Test Case

Revise – the application and understand the best approach for testing it and managing test cases.

MAP - the application based on the pages, elements, flows, functions or features.

Explore – the code, interfaces, databases, environments, application, etc.

Design – the test cases with appropriate names and organize them in the appropriate category.

Integrate – the mini test cases with various test data and environment information, etc.

Note

- Test Case is mere ***data***. It is instructions and verification in its simplest form.
- Test step and Expected result is ***information*** about the test. It is like a journal with information about what was tested and how it was tested and what tools were used for testing it. If the data is not clear and complete the information is not complete for future test execution.
- Test Case execution status becomes the ***knowledge*** that helps understand the health of the application, product or feature.
- The knowledge helps, us the humans, make an informed ***decision*** about the release, sprint, project and most importantly the product.

Summary

Test case is not a document that assumes what could be tested when the product is developed. It is a document that describes how it was tested and how it works. If the document can be furnished with the tools necessary for testing, supporting the product through its life will become stress-free and doubt free.

Test case supports the product through its lifecycle. Not just the first release.

Test case **belongs** to the Product, not the requirement or the **project**.

The best time to write a test case is after the **first round of testing.**

Talk to the Robot. Robots do not understand much. They need specific instructions.

Test cases are two types – **Explore, Pilot**

Exploring test cases explore the feature thoroughly with specific verification.

Pilot test cases navigate through the steps they do some verification.

Specify **one** user action or pilot at a time.

One test case can have **more** than one verification criteria.

Don't tell the tester what to do, tell them **how to** do it.

Provide the **tools** to get the job done accurately and rapidly. Go the extra mile to **equip the tester**.

Provide exact **SQL** queries for execution and verification in the test case.

Picture says a thousand words, so **highlight** what to look for in a picture.

Periodically **revise** the test cases to add, update, delete and merge as necessary.

Have an **owner** who will oversee the test creation process and standards for the test cases.

Following this method of documentation won't give job security, but it will give job satisfaction!!

Works Cited

Jorgensen, P. C. (2014). *Software Testing: A Craftsman's Approach 4th Edition.* Boca Raton, Florida, USA: Taylor & Francis Group, LLC.

API Testing: Picking the Right Strategy

Asha KR

Asha_kr@mcafee.com

Shwetha D J

Shwetha_naik@mcafee.com

Abstract

A right testing strategy for **Application Programming Interface (API)** is crucial for developing a successful product. However, many businesses fail to see a complete picture during project management phase. Changes in product and project requirements are inevitable, which can potentially disrupt the timelines or quality of the product. Eventually, more time and resources are required to accommodate the change requests. It is challenging to deliver a good quality product to the end users in such situations. Testing of API after it integrated with the UI is not an easy task especially when we deliver a quality software as a service to our customers and partners.

API driven testing is a testing framework that uses a programming interface and the application to validate the behavior under test. Typically API driven testing overcomes application user interface altogether. API driven testing is also being widely used by software testers as it serves additional benefits as compared to other testing strategies. Programmers or testers write scripts using a programming or scripting language that calls interface exposed by the application under test. These interfaces are custom built or commonly available interfaces like COM, HTTP. Present development world changes every day with new technologies and this is directly proportional to UI changes. In this era we cannot rely only on UI testing for functional verification. The test scripts created are executed using an automation framework to compare test results with expected behavior of the API. The journey was more challenging while bringing up a startup project to a stabilized API automation framework with fair number of test cases automated in it. There were many initiatives and ideas put forth for converting manual testing to stabilized automation testing.

Outcome of API automation is increased test coverage by 100% with quality deliverables. 20% of early detection of defects and scope for manual testers increased to become Test Automation Engineers. There are nightly builds running with automation for Build Validation and Functional Validation, which has helped testers pitching in early defect analyses with in less time.

Biography

Asha KR *is a Senior Software QA Engineer at Intel Security, currently working in the Intel Security India Center in Bangalore. She has been working for the past 5+ years in different QA roles SaaS products. Asha holds Bachelor of Engineering in E&C from VTU, Karnataka, India.*

Shwetha D J *is a Software Development Engineer for Test, currently working in the Intel Security India Center in Bangalore. She has been working for the past 1+ years on SaaS products. Shwetha holds Bachelor of Engineering in E&C from VTU, Karnataka, India.*

1. Introduction

Web services extend the World Wide Web infrastructure to facilitate a software to connect to other software applications. Web services (sometimes called as application services) are services usually including a combination of the software program and data, but possibly including human resources as well that are made available from a business's Web server for Web users or other Web-connected programs. Applications access Web services via Web protocols and data formats such as HTTP, XML, and SOAP. Web services combine the best aspects of component-based development and the Web. They are a cornerstone of the Microsoft .NET programming model.

In this era of **agile development**, we are moving to a model where UI is ever changing and the consumption of web services is taking the center stage and hence giving this layer the attention it truly deserves by shifting the onus from UI testing to Web Services based validation.

API stands for **Application Programming Interface**, which specifies how one component should interact with the other. It consists of a set of routines, protocols and tools for building the software applications. An API is similar to **user interface**, the difference is that, instead of a user-friendly collection of windows, dialog boxes, buttons, and menus, the API consists of a set of direct software links, or calls, to lower-level functions and operations. APIs can look formidable, but they're designed to be accessible to trained, knowledgeable programmers.

API testing in many respects is like testing software at the user-interface level, however instead of testing by means of standard user inputs and outputs, the testers use software to send calls to the API, get output, and log the system's response. General steps involved while performing API testing are mentioned below:

- Details of API information is found in Specification Document which lists the signatures of each API function (the input parameters, the function or method name, and the return type)
- Identify the software to be used for API testing
- Add the web service call to be tested
- API call will have request and response parameters
- Prepare the inputs for the request
- Invoke the method with all provided inputs
- Analyze the output response

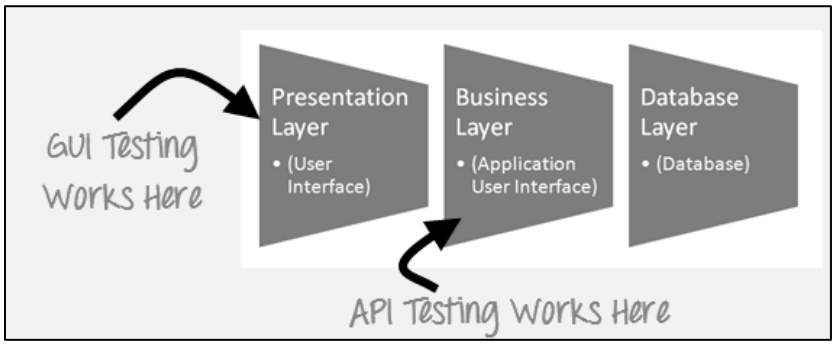


Fig 1.1: API Testing in Business Layer

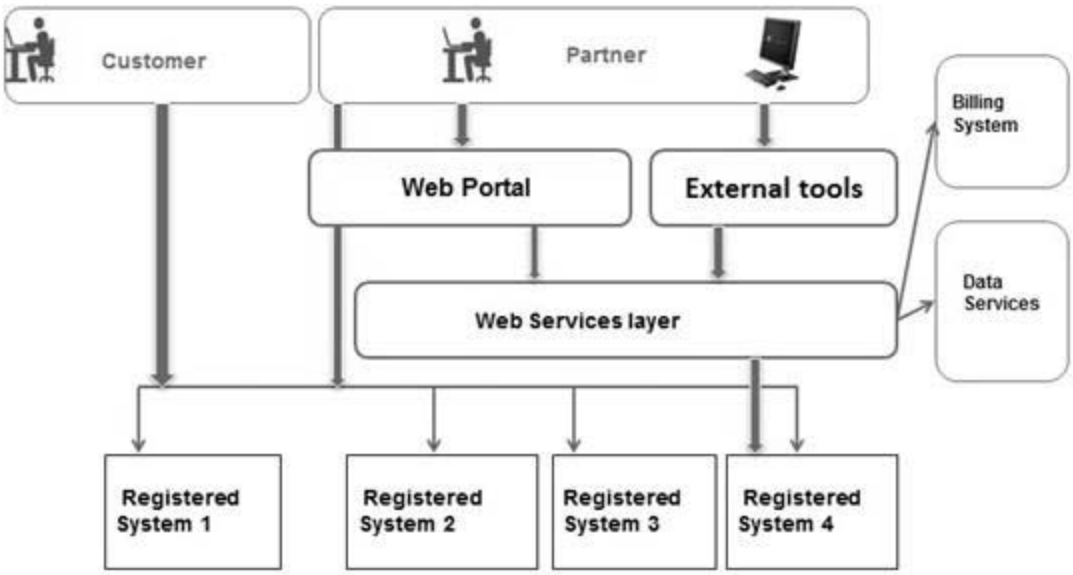


Fig 1.2: Web Services and API Representation

The above diagram Fig (1.2) represents architecture of where the Web services, API come into the big picture of the solution under test. Our team is responsible for development of Web Services. There is a **Web Services layer** that hosts all the web services. **External tools** block is one in which the methods are present and they can be accessed from Web Services layer. Web Service layer consists of two sub layers:

1. **Data Service Layer**: This layer consists of all Data related APIs which provide the business logic for the application. This layer also in turn communicates to the data source or the data layer.

2. **Management Service Layer**: This layer consists of Management related APIs which provide the management solutions to the applications that consumes the corresponding services in this layer.

User requests reaches management service layer, which communicates to data service layer, in turn it fetches data from data source. Each layer has multiple web services and each web service has multiple methods called APIs. Different systems access these API's for their business purpose. **Web portal** is the UI layer which consumes all the web services and presents the relevant data to the end user. Partners can make use of external tools to directly invoke the APIs.

2. Problem Statement

Testing of an application which integrates multiple systems is not an easy task. When we promised to deliver flawless product to customer, detailed testing of the individual components, systems and subsystems that integrates to main product is very crucial. If we are planning to test the end to end scenarios that spans across all the systems and subsystems in consideration it may lead to product which is not impeccable. If some functionality fails at UI then analysis of failure at different system is more time consuming. Having API testing one step before the UI can help in managing different systems without a glitch. However when we start to test multiple different systems as one, lot of manual effort and time is required and also maintaining the product will eventually become tiresome.

When products grow more complex and interacts with external systems, we cannot rely on manual testing because of human errors and a natural tendency to miss few regression test scenarios or some feature validations. The strategy that we followed should suffice all the requirements mentioned above with the effective outcome.

2.1 Challenges in API Testing

- Main challenges in API testing are Parameter Combination, Parameter Selection, and Call Sequencing
- There is no GUI available to test the API which makes it difficult to provide input values
- Validating and Verifying the output in different system is not very easy for testers
- Parameters selection and categorization required to be known to the testers
- Exception handling function needs to be tested
- Coding knowledge is necessary for testers

3. Approach

Considering the problems that arise during API testing, choosing an approach for an effective and efficient API testing depends on a lot of things. Will the API be a public API that will be consumed by some external users/systems, or is it a part of a larger product's infrastructure? API is a general term that is sometimes used to describe anything from a COM interface, to a DLL or JAR you can reference, to a REST web service. Different approaches can be applied to testing these different categories.

3.1 Points to keep in mind while performing API testing

Below flowchart represents the steps to consider while performing API testing.

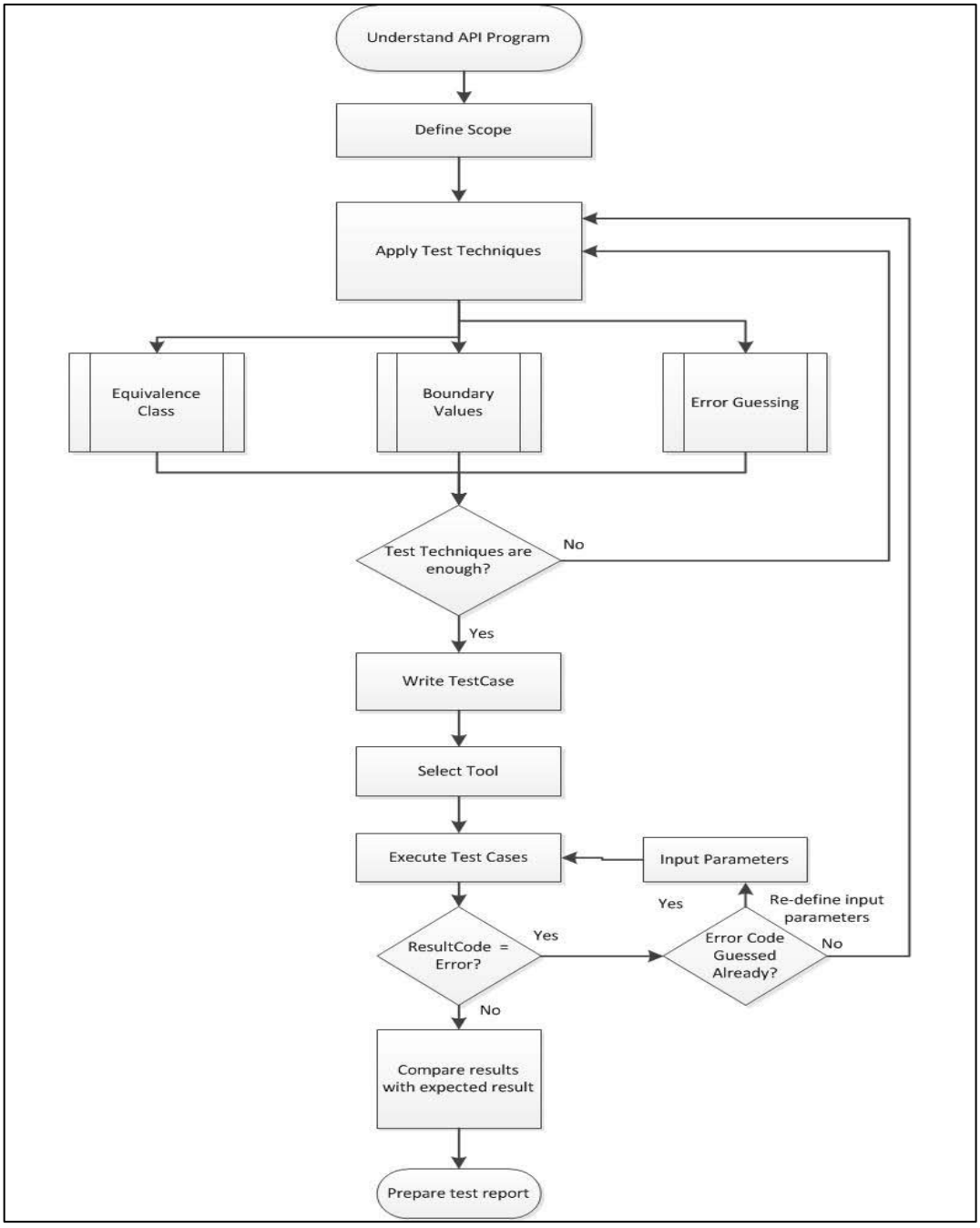


Fig 3.1: Flowchart depicting the API Testing steps

- To know how the API works, it is important to understand the functionality of an API and clearly define the scope of the program.
- Apply testing techniques such as equivalence classes, boundary value analysis and error guessing and write test cases for the API.
- Input Parameters for the API need to be planned and defined appropriately.
- Execute the test cases and compare expected and actual results.
- Use the appropriate tool to test API which gives more effective results

When we understand and realize the importance of Web Services Automation, the next challenge is to select the right Test approach and Test Type before jumping into automation design and implementation.

3.2 Test Objective

It is important to answer the basic question "Why do we want to do Web Services Testing?" and that will help us to decide the right kind of Test we should choose to automate them. Let's begin with few possible answers

- To validate the functional behavior of your application / APIs
- To validate the performance aspect of your application / APIs
- To first validate the functionality and then also test the performance aspect of it.

Once the objective of the test is clear, we need to look at the technologies that can make a difference to the way we test the Web Services.

3.3 API testing Manual

Manual API testing is one where in which testers needs to give inputs and invoke the method for response, analyze the results and report them.

E.g. AuthenticateSession API method in SessionData Service.

Inputs for AuthenticateSession method are: SessionId, Username and Password. Output for this call will be 'Success' or 'Fail' [With fail reason as message]
There are many steps involved in execution of API methods manually, which also consumes lot of effort and time. Identifying the tools to suit the testing is also one of the challenge. There are many tools available for executing manual testing such as

3.3.1 WCF (Windows Communication Foundation) Client
3.3.2 SOAP UI etc.

WCF is the tool which comes with Microsoft .net framework and user friendly. **Soap UI** is the open source cross platform functional testing solution with an easy to use graphical interface. Soap UI allows easy and rapid creation and execution of test cases.

3.3.1 WCF Client

Windows Communication Foundation (WCF) Test Client (WcfTestClient.exe) is a GUI tool that enables users to input test parameters, submit that input to the service, and view the response that the service sends back. It provides a seamless service testing experience when combined with WCF Service Host

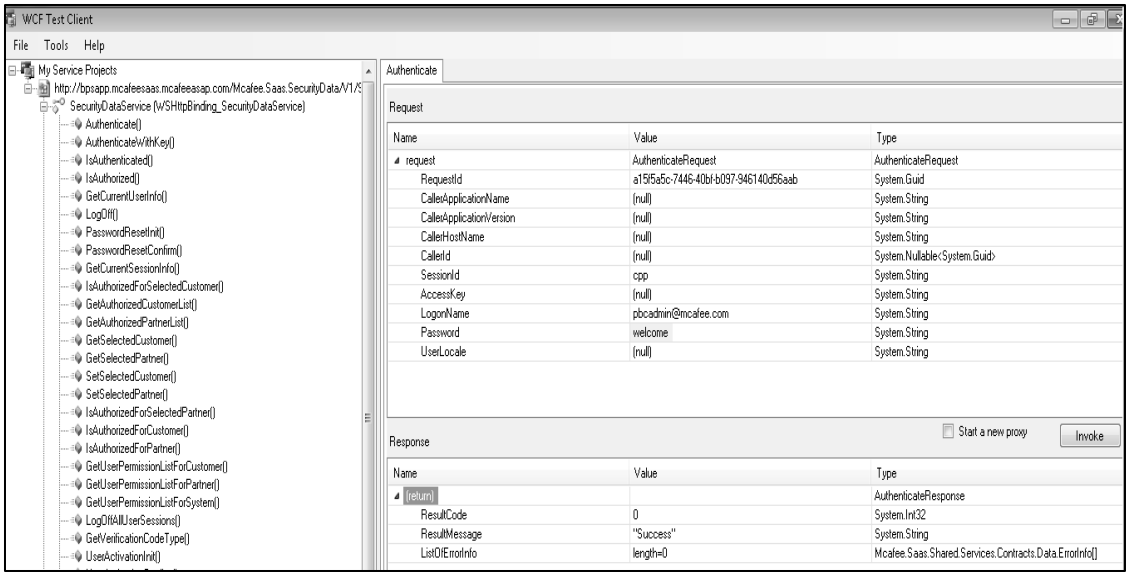


Fig 3.3.1: WCF Client

The Fig (3.2.1) shows a WCF client window, where left pane represents the Web service requests and the API methods included. Right top part represents the request with input parameters and down part is the response of the API invoked.

3.3.1.1 Advantages

- **Interoperability**: A single platform used to exchange information using various network protocols and platforms. Typically, an Asp.Net web service uses HTTP to communicate between client and server. Similarly, in .Net Remoting the client and the server must use .Net applications to share or exchange information. WCF services are interoperable, which uses a variety of network protocols such as HTTP, TCP and MSMQ etc.
- **Security and Reliability:** WCF provides better security and reliability as compared to web services or ASMX services. Security is a key element in any Service Oriented Architecture (SOA), and it is provided in the form of auditing, authentication, authorization, confidentiality and integrity of messages shared between the client and the service.
- **Support for Plain XML, Ajax and REST:** WCF libraries now support other formats for sharing or exchanging messages (data) across the network. We can now configure WCF to share plain XML messages between clients and services, formats that are no more controlled by SOAP anymore. We can now build WCF services using **REST**, also known by the name Representational State Transfer. It is simply an architecture to design distributed applications on a network, where clients can make requests for services.

3.3.1.2 Disadvantages

- The Service Host instance is not shut down gracefully – thus any pending requests are aborted when the WCF Service Host is closed.
- Cannot customize Service Host initialization – and it is common to programmatically initialize fixed behaviors and provide exception handling.
- The WCF Test Client does not provide a way to save and reload inputs to service operations – thus must retype those inputs each time you run the test client.
- The user interface for supplying test values is automatically generated and cumbersome to use.

3.3.2 SOAP UI

SOAP stands for **Simple Object Access Protocol**. A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

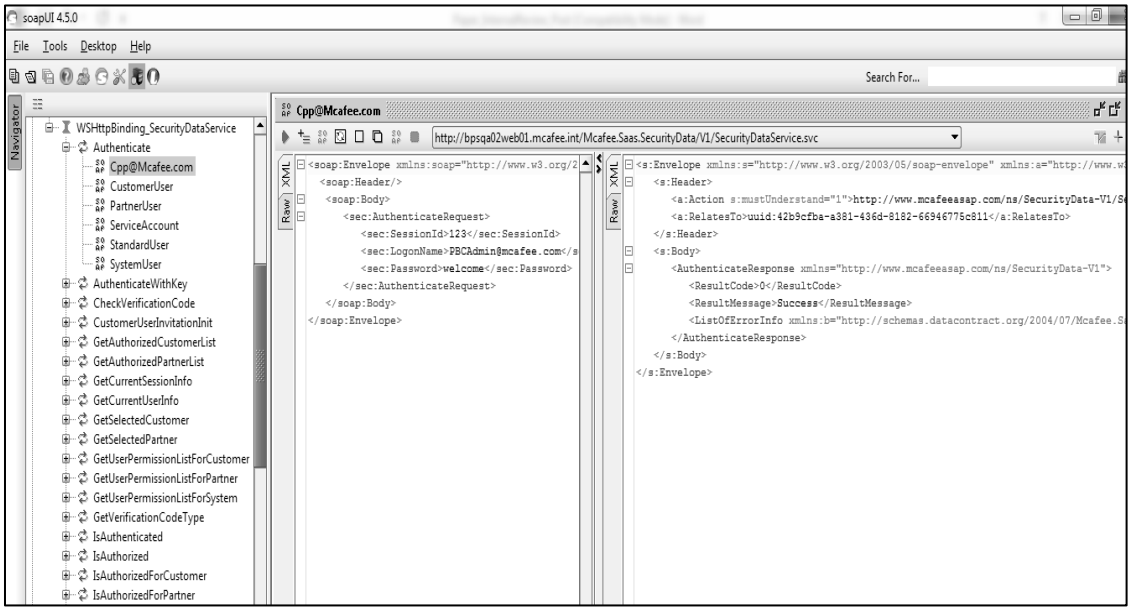


Fig 3.3.2: SOAP UI

3.3.2.1 Advantages

- SOAP is a Communication protocol designed to communicate via Internet.
- Extend HTTP for XML messaging.
- Provides data transport for Web services.
- Can exchange complete documents or call a remote procedure.
- Can be used for broadcasting a message.
- Platform- and language-independent.
- XML way of defining what information is sent and how.
- Enables client applications to easily connect to remote services and invoke remote methods.

3.3.2.2 Disadvantages

- The SOAP specification contains no mention of security facilities.
- SOAP specification does not specify a default encoding for the message body. There is an encoding defined in the spec, but it is not required that you use this encoding to be compliant: Any custom encoding that you choose can be specified in the encoding Style attribute of the message or of individual elements in the message.
- Because SOAP deals with objects serialized to plain text and not with stringified remote object references (interoperable object references, IORs, as defined in CORBA), distributed garbage collection has no meaning.
- SOAP clients do not hold any state references to remote objects

3.4 API testing using Automation

Automation approach for API Testing: When we need to test a feature multiple times because of frequent modification in the dependent subsystems, then automation is the best of testing. **Regression testing** is made easy with automation. Automation can be done with some tools or if someone is very good in programming can do API automation without any tool. This is explained in below sections

3.4.1 SOAP UI

Automation to execute test suites developed using Soap UI, without using the user interface of SoapUI, and instead trigger it from some kind of automation build tool. There are many different tools that could trigger the execution. Most of them have in common that they are good at triggering anything from a command line prompt. This could be a bat script in Windows, a shell script in UNIX, or it could be Maven project in a Java build environment.

Command line: Command line tools are good if you want to be able to execute something from a script that you have created yourself. SoapUI has support for executing from a command line. Execute it like this:

Testrunner project_file

Where project_file is a SoapUI project file that could look like this c:\ projects\my-soapui-project.xml if you are on Windows. If you want help with running the exact same execution from a command line that you executed from the TestRunner, then try this:

- Execute the test you want from the test runner.
- Scroll back in the execution log until you find the launch of the test runner in the beginning of the log.
- Copy the command and use it in your own script or from a command line

Automate the execution after creating your SoapUI functional tests with test suites and connecting SoapUI tests to your build tool, test suite will get executed, as soon as a change has occurred.

3.4.1.1 Advantages

- We will save a lot of time and be able to focus on the real problem, testing, instead of the problem of creating your test tool.
- Delivery of high quality APIs is assured, compared to when testing the services with manual effort.

3.4.1.2 Disadvantages

- The SOAP specification contains no mention of security facilities.
- SOAP specification does not specify a default encoding for the message body. There is an encoding defined in the spec, but it is not required that you use this encoding to be compliant: Any custom encoding that you choose can be specified in the encoding Style attribute of the message or of individual elements in the message.
- Because SOAP deals with objects serialized to plain text and not with stringified remote object references (interoperable object references, IORs, as defined in CORBA), distributed garbage collection has no meaning.
- SOAP clients do not hold any state references to remote objects

3.4.2 Microsoft Web Test

Web tests allow simulating a user performing a set of operations – typically a defined use case – on ASP.NET Web application, and validating the responses to see if the application is working as expected.

Web test is a better when we have relatively simpler web services where our objective is to test simpler / straightforward functionality and primary focus is to test their **performance** down the line. We are working on legacy app that is based on ASMX kind of services which have a better support in Web Test.

Steps to write Web Test: Web test can be created from Microsoft visual studio.

- Start Visual Studio as Administrator.
- Go to the Test menu and select New Test.
- From the list of project templates, select the Web Performance Test template
- When the project is created a new IE window will open and using a plug-in we can record the session.
- Browse a number of pages of the publishing portal and then press the Stop button. While you are browsing, notice how the requests will be listed in the Web Test Recorder plug-in.
- Press the Stop button and the requests will be added to your Web Performance Test.
- Press the Run Test Button in Visual Studio to run the recorded test.
- Once test run completes double click on the test name to see details.
- By using validation rules, we can determine what conditions are necessary for a test to pass. For example, we can consider that all requests that take longer than 5000 Milliseconds have failed.

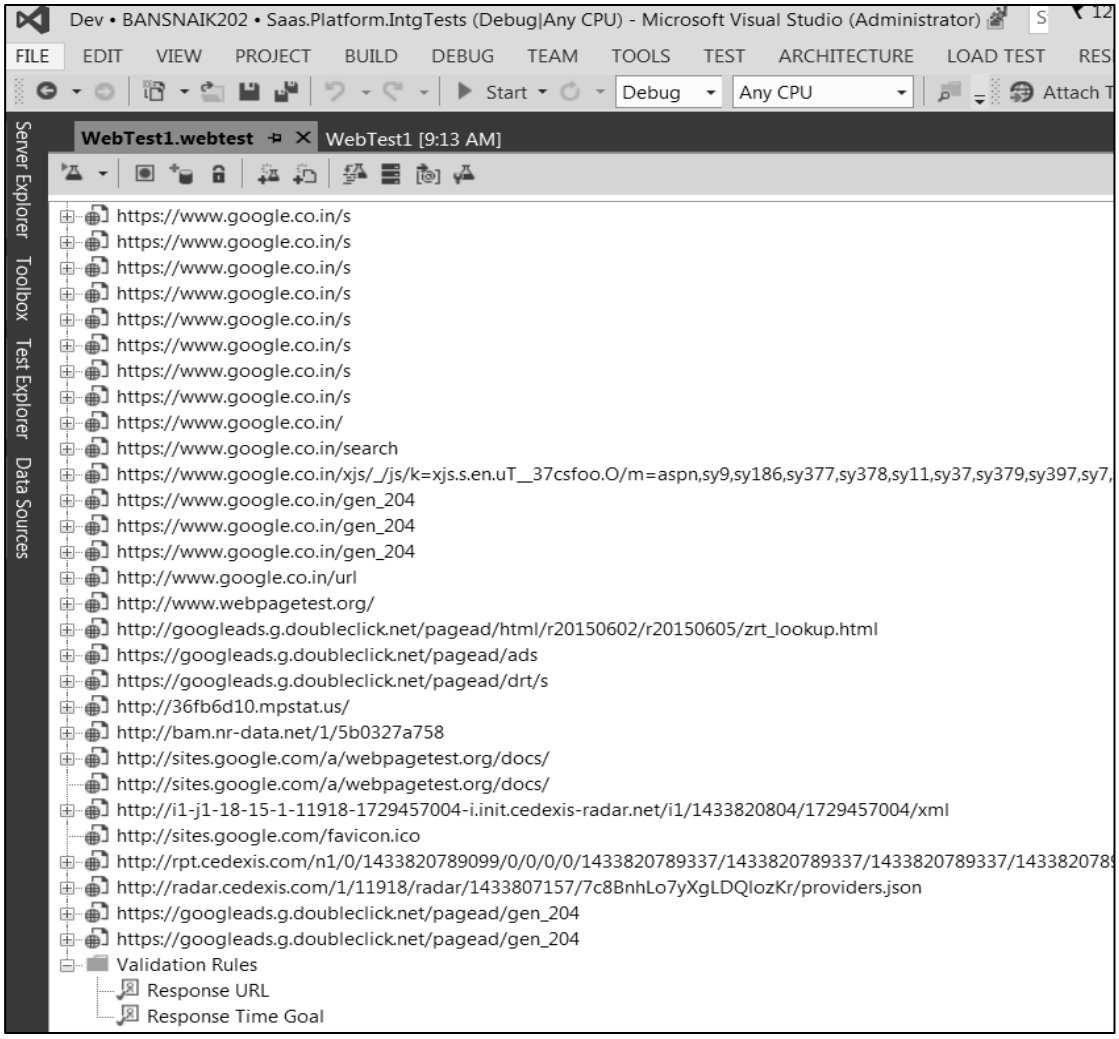


Fig 3.4.2a: Web Test

If we don't want add web test by record and playback, we can add service requests manually and write test cases.

Performance testing: We can use test results obtained by running webtest for performance testing. This can be done by analyzing metrics on the server and on the client, explore the HTTP traffic requests and performance counter data. Double click the test name in test result pane to see the details.

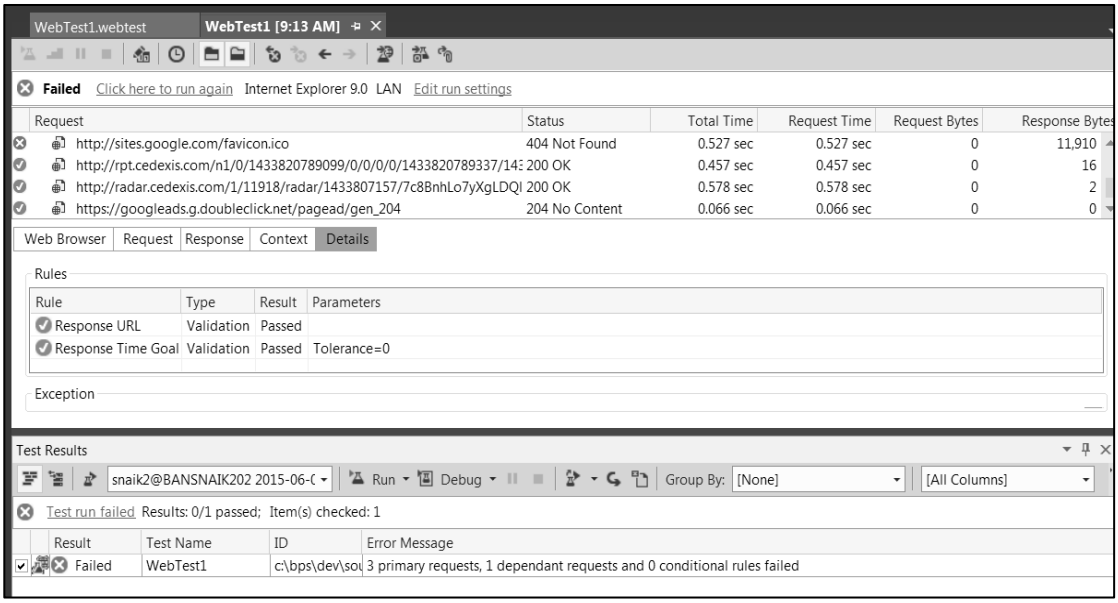


Fig 3.4.2b: Test Result pane for webtest

3.4.2.1 Advantages

- Creating web tests without writing code using the Web Performance Test Recorder is good for beginners.
- We can edit recorded tests to tailor to specific needs.
- Simplify JavaScript interaction by automatically promoting dynamic parameters to editable web test parameters
- Load tests can be created by aggregation of existing web tests

3.4.2.2 Disadvantages

- Not much validation logic can be put into the tests: Validation can be put only on validation rules available in webtest.
- Maintenance is difficult: A small change in contract, would impact in changing all affected test cases
- Data Cleanup was not done properly
- High Volume tests were not at all executed in the test suite

3.4.3 Using Programming Language

Limitations of Microsoft webtest and dependency of third party tool like, SoapUI led us to write an automation framework on **Microsoft .Net** platform by using programming language **C#**.

Reasons for selecting C#:

- It is a simple, object-oriented language.
- Anyone with basic knowledge of C++ and C# language will be able to drive the framework.

Framework: When we talk about test framework, there is lot more things comes in picture. Before starting to develop framework we had to think about below aspects

- Suitable Programming Language
- Suitable Tool [If Applicable]
- Programming knowledge of team members
- Time required to develop a framework

Choosing a programing language is the key thing to start developing test framework. Because all the team members should be educated in that language or proper training should be given to educate them. If there is any tool available for automation it can be adopted. With all these boundaries C# can be used as programming language to start with framework. A framework should be an integrated system which integrates different libraries, test data sources, object details, reusable components and different utility modules. Once framework is developed integrate the framework with testing environment where we can run all the API tests.

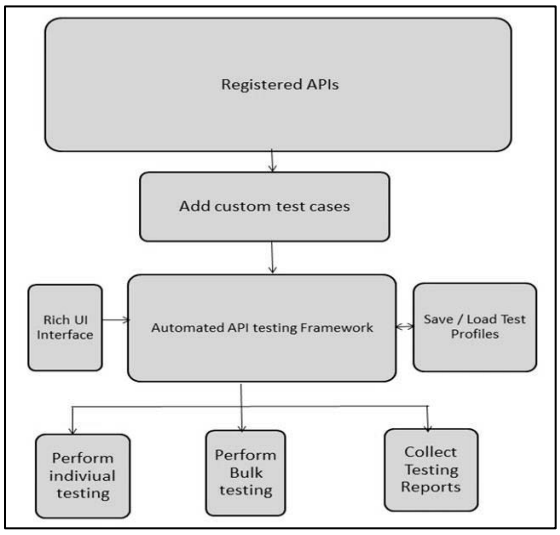


Fig 3.4.3a: C# Automation Framework

API testing is often compared with **unit testing** because the purpose served by both API testing and unit testing is same in most of the scenarios. Unit testing is covers some block of

application code and the test cases can only verify its behavior with respect to boundary, valid input data and incorrect input values. Whereas API testing works for the same kind of testing but API testing can be taken to test end to end flow where we have to testing APIs which internally call one or more APIs to get the response. Unit testing has limited in scope whereas API testing has broader in scope as it can run for end to end testing.

API testing can be done by,

1. Create a new project of unit test in Microsoft visual studio
2. Adding Service Reference:
   - If service endpoint URL is available, then adding service reference is easy. Example of service endpoint: http://localhost/Mcafee.Saas.SecurityData/V1/SecurityDataService.svc?wsdl.
   - Once service is added to project, prepare required test data and consume that service client by providing test data as input.
3. Writing Test Method:
   In UnitTest.cs add test methods which validates required service. Create new test method and add service client which exposes all APIs available in the test method.
4. Invoke API and validate Response: Using Service Client invoke required API by providing the input data. Validate response of the API call with suitable result message and error codes as show in below diagram Fig 3.3.3b.

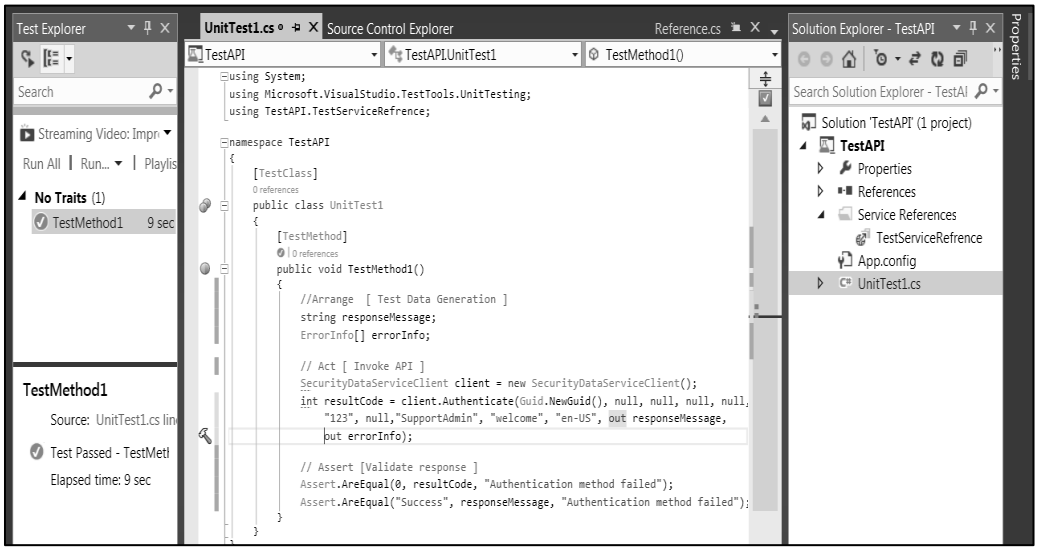
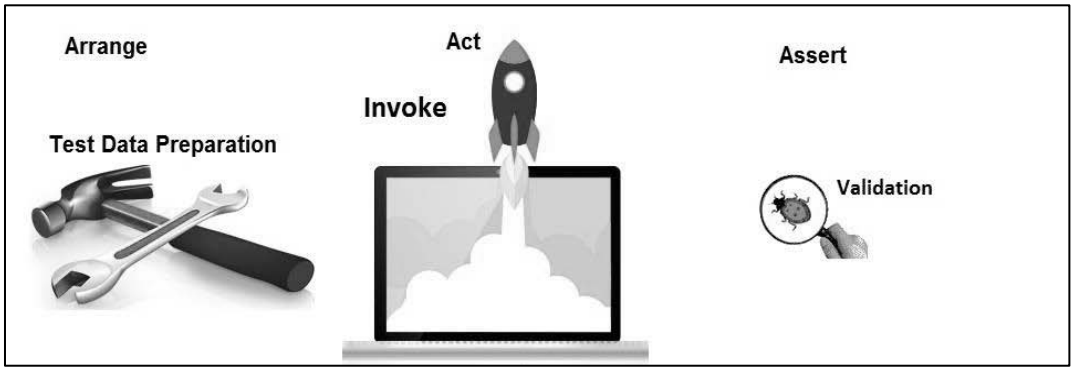


Fig 3.4.3b: Sample C# test case

Test Case: Any test case written for API testing will have below 3A's (as shown in Fig 3.4.3b below) as mandatory steps:

- **Arrange**: Arrange all requirements. Create test data required for API request.
- **Act**: Act on the API. With the test data created in step 1 invoke any API or invoke required APIs and get the response.
- **Assert**: Assert on the response obtained from the step 2 Validate the response from the API request with the local or remote data source.

In Simple words



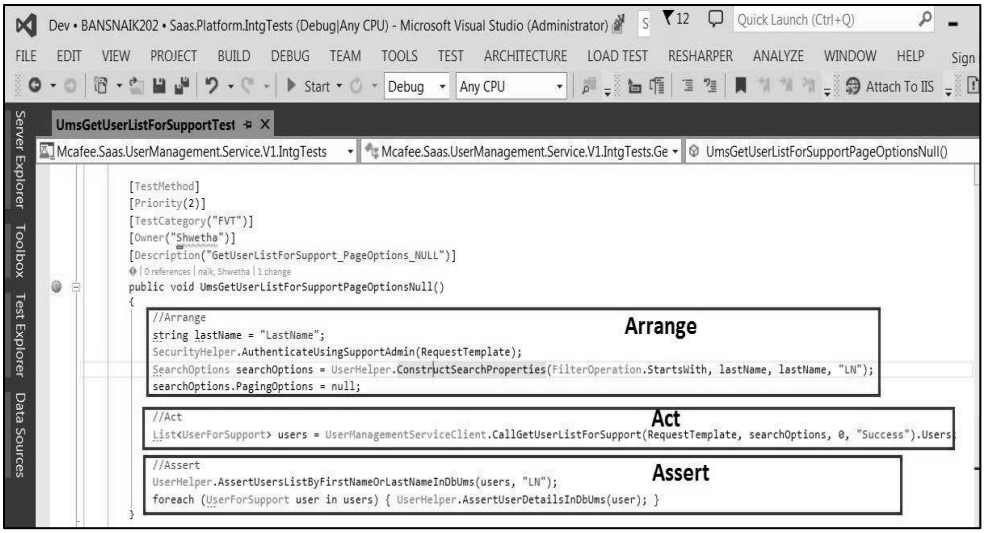


Fig 3.4.3c: 3A's mandatory steps

Writing appropriate helpers, utility libraries, reusable methods one can design a good automation framework which can be used by different systems as platform of writing test cases of their own.

When we are dealing with the APIs more than one in one test case we have to make sure first API is not blocking the invocation of the second or rest APIs. When we call multiple APIs in one call or in multiple call it is good practice to have meaningful error messages at all steps so that tester can easily analyze error message or result message and can log **defect** according to that.

C# framework for End to end testing: API testing framework can be used for end to end testing before going to UI. To do this end to end testing user should be aware of sequence of APIs to be called and amount of time wait required between each API. For end to end 'Arrange' part should be kept as simple as much so that it should not lead to overburden for any person who joins later.

Test Result: Reporting test execution results is very important part of testing, whenever test execution cycle is complete, tester should make a complete test results report which includes the Test Pass/Fail status of the test cycle. If manual testing is done then the test pass/fail result should be captured in an excel sheet and if automation testing is done

using automation tool then the HTML or XML reports should be provided to stakeholders as test deliverable.

By writing some scripts for running all the automated test cases at some cycle and generate proper test result. Test result of the run should include,

- Test cases pass / fail count
- Complete assert message for failed test cases
- Test Run time – This should include time for service call for each and every test case
- Metrics – metrics for test case pass fail statistics compared from last few runs
- % of test coverage and code coverage
- Major areas where failures are more.

3.4.3.1 Advantages

- C# framework is tool independent. There are n numbers of third party tools available for API testing automation but having a framework with C# will act independently without relying on any tools.
- Anyone with good basic knowledge of C# language will be able to manage framework.
- Programmers who are familiar with C++ concepts can easily learn C#.
- The Assembly concept of C# solves the versioning control problem of application
- Ease-to-development, the rich class library makes many functions easy to be implemented.
- Working with C# will give access to Microsoft .NET Framework libraries, which are extensive.
- As C# is .NET language it supports interoperability that means code written in C# can easily transformed to other languages of .NET [E.g., Visual Basic, C++ etc.]

3.4.3.2 Disadvantages

- C# mainly depends on .NET framework. Any library which is not found in .net will be difficult to implement
- C# programs are largely for Microsoft windows environment but C# doesn't come up with open source technologies and operating systems like Linux.

4. Result

Moving from manual testing to automation framework for API testing, increased lot of scope in testing. The journey from manual to automation and stabilization of code gave good results in managing the web services.

4.1 Opportunity for every QA to learn and enhance programming knowledge
4.2 With the stable framework an application can be more stabilized and can deliver good product to customers.
4.3 API testing also gives the deeper insight into the product which in turn helped QA to increase the product knowledge.
4.4 Analyzing the failed test cases helped in detecting defects at the earlier stage. Early detection of defects reduced rate of defect detection at the GUI level.

Percentage of early defect detection: 20%

4.5 Manual validation effort reduced because of the 100% automation of smoke test cases and 80 % automation of functional test cases.

Build Validation Tests: (Fig 4.1a)

- No of Test Cases: 390
- Frequency of test run: Every night with latest build
- Time: All 390 test cases take 3hours to complete and generate test result report. This saved 75% of manual effort and 12hours of resource time.

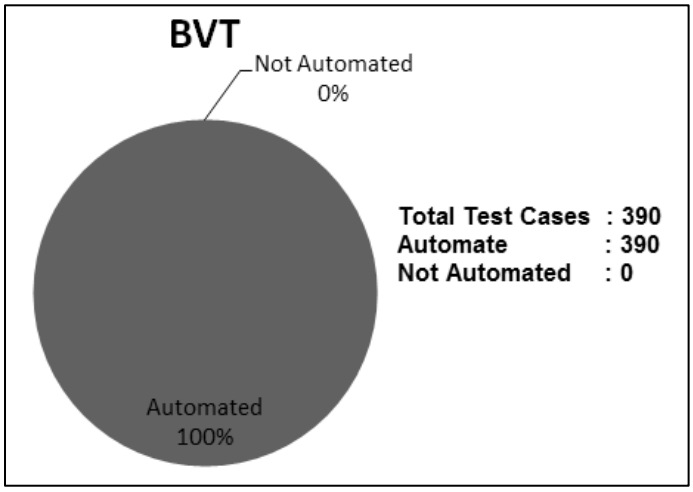


Fig 4.1a: 100% BVT Automation

Functional Validation Tests: (Fig 4.1b)

- No of Test Cases: 1550
- Frequency of test run: Weekly twice with latest build
- Time: All 850 test cases take 5 hours to complete and generate test result report. This saved 75% of manual effort.

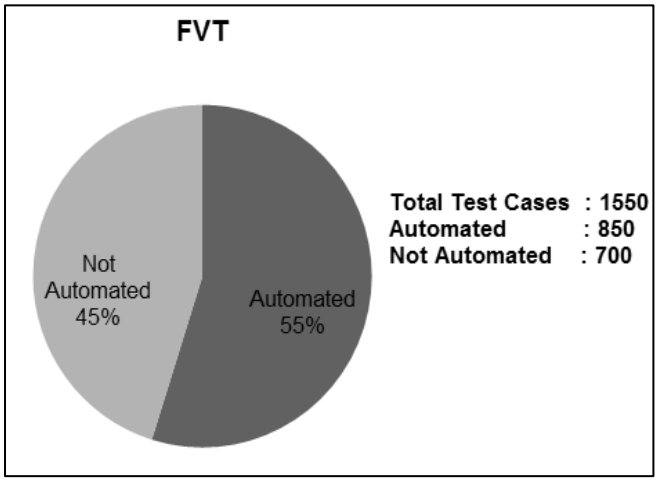


Fig 4.1b: 55% FVT Automation

Below graph depicts the overall automated test coverage across BVT, FVT and End to End Tests carried out till date. End to end testing for all BVT scenarios has started and is covered 10% as of now. Scope to cover more test cases in future.

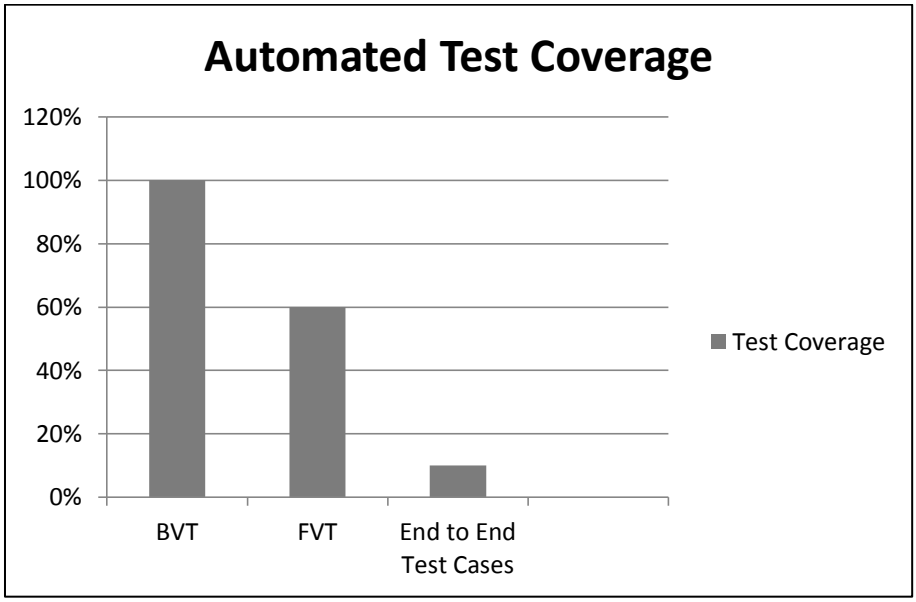


Fig 4.1c: Test Coverage

5. Summary

API consists of set of classes / functions / procedures which represent the business logic layer. If API is not tested properly, it may cause problems not only the API application but also in the calling application.

5.1 Advantages

- More effort into API testing leads to much healthier final product, ensuring the data access through the API only.
- API testing simplifies security and compliance testing and thereby certification, since there is only one interface.
- Ensuring that all the required business rules are being enforced at the API tier allows time for much more complete user-experience tests once the UI is released, and not having to concentrate on testing every single business rule and path through the application near the end of the project.
- Ensuring that the API offers complete functionality allows for easy future expansion of the application as new business needs arise.
- As API testing is structured way to testing, it makes automation more feasible.

5.2 Disadvantages

- No encryption - All requests and responses are visible to anyone between the requesting server and the API server.
- Increasing Tooling needs: API testing can be done only using different tools available. Cannot be tested without the tool like in GUI.
- Straightforward reason for failing API is not known, until and unless debugged into the code, unlike in GUI where it gives user-friendly errors.

Though automation is 100% possible, still there remains a question whether automating 100% test cases is really profitable? Because of few reasons like priority of test cases, frequency of those test cases run and effort calculated in automating them.

5.3 Future

With all the merits and de-merits kept in mind, we have strategy planned to automate End to End tests and Integration tests.

- **End to End Testing**: Plan to extend all BVT scenarios to cover end to end testing. This is one step ahead from normal API testing where we just test one functionality. End To End testing make hold of the scenario which user performs from the UI. Calling different APIs in single method is little exasperating because failing of some API will mark whole test case to fail but with good coding style and capturing exact result at all points we can develop a robust test suite.
- **Integration Testing**: Integration of different systems in testing is little difficult. We have a roadmap of covering different systems from the BVT perspective. Capturing behavior of each system for one single functionality and automation of this will lead us to have non-defective system.

References

Book:

James D. McCaffrey. 2012. .NET Test Automation Recipes: A Problem-Solution Approach. California. The Expert's Voice

Web Sites:

Marc van't Veer. 2013. "Testing the API behind a mobile app", https://www.polteq.com

Sulagna. 2009." Introduction-to-API-Testing", www.scribd.com on Jan 07, 2009," http://www.scribd.com/doc/9808382/Introduction-to-API-Testing#scribd

"Software Testing - API testing", http://www.tutorialspoint.com/software_testing_dictionary/api_testing.htm

Michael-churchman, 2014. "API Testing: Why It Matters, and How to Do It". https://blog.udemy.com on 2014. https://blog.udemy.com/api-testing/,

Joe Colantonio, 2013, "UFT – What is an API test type?" www.joecolantonio.com on February 19, 2013 http://www.joecolantonio.com/2013/02/19/uft-what-is-an-api-test-type/

http://sqa.stackexchange.com/questions/6745/how-do-you-test-a-backend-api

http://sqa.fyicenter.com/FAQ/Testing-Techniques/What_is_API_Testing_.html

http://www.encodedna.com/wcf/tutorial/advantages-of-wcf.htm

http://www.soapui.org/

http://www.codeproject.com/

http://en.wikipedia.org/

http://www.guru99.com/api-testing.html

Brewing Next Generation Identity

Kalman C. Toth

kalmanctoth@gmail.com

Abstract

The growing popularity of mobile apps, the "bring your own device" (BYOD) phenomenon, cloud computing, and big data, seem to have created the perfect storm for traditional identity technologies and solutions. Service providers - and certain users too - are increasingly aware that the features and benefits offered by an identity solution are worth nothing if a crafty attacker breaks through critical design elements, exposes secrets and private information, and thereby facilitates user impersonation and fraudulent transactions. This paper provides a synopsis of the identity problem (as I see it), discusses essential weaknesses of legacy identity technologies, and puts forward a plausible vision and operational concept for a next generation identity solution that overcomes many of weaknesses of these legacy technologies.

Biography

Kal Toth *has over 30 years of technical, consulting and management experience working for small, medium and large-sized technology companies including aTrust Inc., Hughes Aircraft, Datalink Systems Corp., the CGI Group Inc., the Software Productivity Centre (Vancouver, BC), and Intellitech Canada. He is past Executive Director of the Oregon Master of Software Engineering (OMSE) program at PSU and has delivered systems/software engineering and project management courses at PSU, OSU, TechBC, Simon Fraser, U of Alberta, and UBC. He completed his Ph.D. in systems engineering at Carleton University (Canada) and is a registered professional engineer with a software engineering designation in British Columbia.*

1 Introduction

For some time now I have advocated and refined a user-centric model (see refs.[21-24]) for personal identity to counter certain weaknesses of server-centric identity solutions. Rather than allocating the responsibility for storing and managing personal identities to service providers and applications in the cloud, I have suggested that next generation identity solutions enable individuals to embed their identifying information into their personal devices, for example, their smart phones, smart cards, tablet PCs, and laptops. Naturally, such an approach opens up many new challenges for the software quality engineer.

Although privacy laws oblige enterprises to safeguard private and identifying information of customers and employees, their record has not been stellar. 2014 was not a good year for big targets like Target, Home Depot, JPMorgan Chase, Athenahealth and others [1]. We have plenty of evidence that the server-side of the web is rapidly losing ground in its battle against hackers, malware and other types of electronic abuse.

I believe the root of the problem is that enterprise servers and server farms are, by definition, massively complex, collectively containing virtually all of the identities and private data of our global population. It is no wonder that servers are the primary targets for online hacking, breaches, and identity theft that enable fraudulent use of identities. In comparison, end-users, numbering 270M in the US alone [3], are collectively much more numerous, are widely dispersed, and are mostly low-yield targets. With the exception of notables like Bill Gates and Warren Buffet, the average return-on-hacking effort against individuals will be much less than that realized by attacking enterprise and service provider repositories. The tide should be shifted from a server-centric identity model to a user-centric one that strengthens user control over their identities, while reducing opportunities for exploit on the server-side. Such models have merit and should be widely discussed.

2 Elevated Identity Assurances Mitigate User Masquerade

Fraudulent web transactions arise by way of “user masquerade” (a.k.a. impersonation), such e-fraud damaging targeted victims, their associates, and their e-business providers.

For example:

- Scam artists register pseudonyms to obtain accounts and credentials to defraud victims
- “Dumpster-divers” acquire records and credentials of victims to obtain accounts in their names
- Hackers and phishers acquire electronic credentials of their victims to break into their accounts to mount various fraudulent transactions.

Recommended impersonation prevention measures include elevating identity assurances by:

- Conducting thorough due diligence/vetting of users and their asserted identities (a.k.a. “proofing”)
- Vetting in proportion to perceived transactional risks
- Deploying technologies, policies and administrative procedures specifically designed to collectively prevent fraud and exploits enabled by way of identity theft and server-side breeches.

3 Weaknesses of Traditional Password/PIN Authentication

Traditional schemes have long been demonstrated to present ample opportunity for user masquerade. Employing a number of fairly straight-forward exploits and/or readily available software tools, a malicious attacker can defeat traditional PIN/password authentication by acquiring the victim’s secrets and fraudulently using such private knowledge to access and tamper with their electronic accounts. At a minimum, it is essential to adopt best password management procedures which include appropriate procedures for creating sufficiently strong passwords and PINs; their safekeeping; augmenting them with

non-guessable security questions; safe account reset procedures; and elevated awareness of the risks of social engineering attacks and scams.

Of course, the routine reuse of the same and/or similar passwords/PINs greatly exacerbates the above risks. This is because the number of opportunities for a malicious attacker to acquire similar passwords/PINS increases together with the number of accounts protected by these similar passwords. Identity federation and single sign-on (SSO) together with multi-factor authentication (mFA) schemes can significantly reduce this password reuse risk.

4 Potential and Limitations of Single-Sign-On and Federation

Pioneered by Liberty Alliance and other players in the late 1990s and early 2000s, single-sign-on (SSO) consolidates identity management at a single point, a federated identity service, which enables the user to be authenticated in one place (or at least in a small number of places) while provided access to multiple web resources. This greatly reduces the number of PINs and passwords required. SSO and identity federation thereby reduce the need and motivation to reuse the same password or PIN for multiple purposes and reduces the opportunities for password/PIN compromise. OpenAM, an open source solution supported by RockForge, Oracle, PingIdentity and others, can be used to implement such identity federating solutions.

5 PKI and PGP: Positive Features and Shortcomings

Public Key Infrastructure (PKI), underpinned by digital certificate technology and extensively deployed across the Internet, automates the deployment of public-private encryption key pairs for secure communications, message transmission, and document safe-keeping. A digital certificate, conforming to the X.509 digital certificate standard, includes a public encryption key embedded in the certificate that is paired with a private key stored outside the context of the digital certificate. A remote party who does not have knowledge of the private key can perform tests to verify that the party claiming ownership of the certificate and contained public holds the matching private key. PKI implements a hierarchical trust model wherein certificate authorities successively distribute digital certificates to dependent certificate authorities, Internet servers, and end-user devices. Digital certificates and their corresponding private keys are distributed by certificate authorities to other certificate authorities, to servers, and to end-user devices. Certificate authorities have the option of employing qualified human agents for 3rd party identity proofing and verification.

I have observed the following deficiencies of PKI:

(a) Using qualified independent certificate authorities, effective for verifying and tracking the identity of service providers, does not scale for human beings who outnumber servers by orders of magnitude;
(b) Because public-private key pairs are generated by certificate authorities and subsequently distributed electronically, such key pairs are vulnerable to compromise during distribution;
(c) Because X.509 digital certificates only specify the certificate holder by a common name or identifier, identities of persons cannot be specified comprehensively for commercial and other such applications;
(d) Digital certificates do not readily bind with other personal identifying information of an owner such as digital photographs or personal identifying information (e.g. passport, driver's license, certifications);
(e) Although digital certificates enable relying parties to verify that the digital certificate owner has the private key that matches the public key of a digital certificate, PKI does not incorporate personal identifying information that reliably distinguishes the certificate owner from other users;
(f) PKI does not provide assurances that the private key is strongly bound to the certificate owner;
(g) PKI does not incorporate identity proofing and binding capabilities that provide objective evidence to relying parties that an independent party has attested to the identity of the digital certificate holder;

(h) Because X.509 certificates are associated with a single public-private key pair, typically multi-purposed (e.g. used for digital signing, encryption, email, FTP, etc.), the risks of encryption key compromise are elevated over other approaches.

Pretty Good Privacy (PGP), meanwhile, was introduced to automate the deployment of public-private key pairs among persons (peer-to-peer) to secure communication channels, transmitted messages, and documents among PGP users. In contrast to PKI, PGP implements a web of trust model wherein individuals issue digital certificates to each other. An end-user, having installed PGP software on their personal computer, creates an X.509 digital certificate containing a single public key with matching private key stored on the user's computer. PGP enables an informal process whereby a first user can send such a certificate to a second PGP user who digitally signs and returns the certificate to the first user. By retaining the single private key of a digital certificate within the owner's computing device, PGP reduces the risk of exposing and compromising this private key. This approach for creating and sharing digital certificates can be replicated among users with PGP software on their computing devices. PGP users can present one or more signed digital certificates to relying parties (users) which elevates identity assurances when presented to other parties.

PGP has the following deficiencies:

(a) Because X.509 digital certificates only specify the certificate holder by a common name or identifier, identities of persons cannot be specified comprehensively for commercial and other such applications;
(b) Digital certificates do not readily bind with other personal identifying information of an owner such as digital photographs or personal identifying information (e.g. passport, driver's license, certifications);
(c) Although digital certificates enable relying parties to verify that the digital certificate owner has the private key that matches the public key of a digital certificate, PGP does not incorporate personal identifying information that reliably distinguishes the certificate owner from other users;
(d) PGP does not provide assurances that the private key is strongly bound to the certificate owner;
(e) PGP does not incorporate a formal identity proofing process whereby relying parties are provided objective evidence of a user's identity;
(f) Because X.509 certificates are associated with a single public-private key pair, typically multi-purposed (e.g. used for digital signing, encryption, email, FTP, etc.), the risks of encryption key compromise are elevated over other approaches.

6 Role of Multifactor Authentication

Multiple factors can be applied jointly to reduce the probability of failed authentication due to the compromise of any one factor. For example, the following factors could be applied in various combinations to achieve more than one factor of authentication:

- What the user knows (like a PIN or password)
- What the user has or holds (for example, possession of their smart card, smart phone or tablet PC)
- What the user is (facial, iris, fingerprint, hand geometry, voice print, or key stroke biometric).

Probably the best known examples of 2-factor authentication are using a PIN together with a banking debit card, and using a hardware token that generates a one-time-password (OTP) for remote terminal logon. Another possibility is 3-factor authentication using one of the above biometrics together with knowledge (a PIN or password), and possession of the access device (say a smart phone).

When used together, MFA and SSO/federation have the potential of significantly scaling back the need to manage large numbers of passwords/PINs by decreasing the motivation for specifying and using many passwords. Next generation identity should closely integrate multi-factor authentication schemes with federated SSO frameworks to reduce the risks of reused, weakly specified, and poorly managed password/PIN systems.

7 Role of Biometric Authentication

Biometric authentication is an increasingly critical ingredient for preventing user masquerade and elevating authentication assurances. Fingerprint, facial, signature, voice, iris and other biometric authentication schemes are commercially viable for deployment on user platforms (e.g. PCs and smart phones). Ma in [20] reports the relative accuracy of available biometrics in terms of false positive rates with facial recognition at 43%, fingerprint at 30%, signature at 28%, voice at 20%, and iris recognition at only 0.47% (which explains the growing interest in iris biometrics). Meanwhile, emerging biometric schemes leveraging the body's venous, nervous and DNA systems are being researched.

Relevant Observation: User preferences for each biometric scheme, matching accuracy, matching performance, human risks factors, and compatibility with the individual circumstances can vary. This implies that:

- Next generation solution architectures should be designed specifically to accommodate a range of biometric options for remote user authentication;
- Biometric scanning/matching should be embedded in the user's personally held and controlled platform;
- The user's platform should be resistant to tampering to protect the user's internally stored biometric minutia (a.k.a. biometric templates);
- The identity service should acquire objective evidence that the user is in control of their platform;
- The system architecture should be extensible, accommodating add-on biometrics in a modular fashion as they become available.

8 Browser Vulnerabilities

Well-documented by the Open Web Application Security Project (OWASP) [4], browser vulnerabilities can be mitigated by implementing best programming, configuration, and usage practices. However, in the face of constant feature creep and fixes, externally mounted browser exploits are unlikely to abate any time soon rendering browser-based user authentication to be of unacceptably high risk in many business contexts. This has stimulated considerable interest in a variety of "out-of-band" authentication schemes that avoid in-browser authentication risks.

9 Benefits of Out-of-Band Authentication Schemes

Out-of-Band (OOB) Authentication over an alternate channel between the user's platform and the identity service provides the user an independent path for user authentication, reserving the browser channel for transaction flow. A compelling risk mitigation strategy is to re-authenticate the user immediately prior to committing a critical transaction (e.g. high value electronic funds transaction). For successful compromise, the attacker must be able to simultaneously penetrate both the OOB authentication channel and the browser channel.

Certain SMS-based OOB schemes using smart phones have been shown to be vulnerable primarily because SMS text messaging typically runs in the clear [15]. To take advantage of operating as an independent authentication channel, messages running over the OOB channel must be encrypted and digitally signed, and the communications protocol must be resident to Man-in-the-Middle (MITM) attacks.

10 Benefits and Limitations of Fast-Identity Online (FIDO)

Fast Identity Online [29] was launched by a consortium of technology companies providing hardware-oriented authenticators (e.g. OTP and smart card tokens) designed to positively authenticate the device

holder. FIDO has been developing a standard set of methods and protocols by which such devices can integrate with online web services, the aim being, to replace existing password usage.

FIDO authenticators generate public-private key-pairs on the client where the public key and an associated “handle” is registered with the online service/application during initial password-based authentication. Subsequently, login access is accomplished by way of a public/private key protocol and challenge that, in effect, substitutes the public/private key pair for the password being currently used (private key is the user’s secret; the public key and challenge are used to verify proof-of-possession). While FIDO reduces dependence on passwords, their authenticators do not support the specification or proliferation of user identities characterizing the user’s attributes and/or life events. The proofing of users remains the responsibility of the service/application, and personally identifying information continues to be held in server-side identity repositories which are vulnerable to large-scale breaches.

11 Relevance of Identity Assurance

While authentication assurances can be elevated by deploying multiple authentication factors, they do not identify other attributes that characterize the individual being authenticated. They only confirm that the person being authenticated is, indeed, the same person. In contrast, identity assurance involves life events, observations and endorsement made by independent parties who can attest to such aspects of the person’s identity. For example, birth date and place, contact information, education, citizenship, skills, financial instruments, business affiliates and so forth are attributes of a given individual that cannot be captured by biometrics, and are (or should be) independent of their secret knowledge (PINs/passwords).

Note that many of these attributes can be found in existing physical credentials such as birth certificates, citizenship certificates, driver’s licenses, passports, diplomas, credit cards, and business cards.

When communicating with a remotely located persons or services, collaborating parties need assurances as to the true identity of the parties. To support this requirement, the identifiers and attributes of a person (a subject), including legal, common, and pseudonyms, should be independently attested by other persons who may elect to issue an identity artifact to that person. The level of identity assurances achieved by such an issuer depends on the extent to which the subject person is known by the issuer (familiarity), and the vetting and proofing competencies of the issuer. Relevant competencies for an issuer include proofing and vetting skills, objectivity, questioning skills, professional oversight by a governing body, and applicable code of conduct possibly sworn by oath - notary publics are exemplars. Identity assurances increase as the number of years that an issuer has personally known a subject, though not necessarily linearly. Identity assurance levels are also proportional to the above listed range of vetting and proofing competencies. Because objectivity and independence may conflict with familiarity, certain professionals, such as notaries and agents of credential issuing organizations, may be obliged to decline proofing and vetting a person who is too closely related to the issuer by way of family and employment.

NIST and Kantara in [8] and [9] respectively recommend implementing progressively increasing levels of identity proofing and verification thereby significantly and prudently reducing the risk of bogus identity issuance.

Although, document proofing and in-person verification procedures are time-consuming activities, there is little doubt that they significantly reduce the risk of fraudulent credential issuance and consequential transactional risk. Such procedures should be an integral part of any identity solution mandated to govern value transactions in the banking, finance, healthcare and similar industry sectors. Solutions should bind the level of identity assurances performed to the credentials issued, and then use this information to mediate high value transactions and access to critical information.

12 Role of Trusted Execution Environments and Modules

User Authentication can be performed by a software component (an “app”) running on the user’s platform that runs independently, but in cooperation with, the browser. Such a component could be designed to implement biometric and/or multi-factor authentication schemes independent of the browsing channel thereby avoiding in-browser authentication vulnerabilities. Nonetheless, such a component would be vulnerable to malware running on the user’s platform leaving open the possibility of remote hackers, phishing attacks, man-in-the-middle (MITM) attacks, and BOTs breaking the embedded authentication mechanisms. For example, malware could lurk within the run-time operating system of the user’s platform waiting for an opportunity to activate and tamper with authentication software thereby exposing private and secret user information.

A trusted user platform capable of isolating the authentication logic and the user’s sensitive information store from malware injected into the platform’s operating system would greatly mitigate such risks.

13 Characterizing Next Generation Identity Solutions

Next Generation Identity Architectures must overcome the range of vulnerabilities and technology challenges with which today’s legacy solutions are barely able to cope. They must support the highly complex interplay between multi-faceted applications running in various run-time contexts, multiple identity repositories from different vendors, and disparate user platforms including PCs, tablets, smart phones (e.g. Android, I-phone, Blackberry) … all exacerbated by today’s “bring-your-own-device-to-work” trend.

The critical design attributes that must be incorporated into the identity architecture should include the following functions and features:

- Multi-factor authentication of the user on their computing platform that includes PIN/password knowledge, proof of platform possession, and at least one biometric authentication factor. This mitigates the risks associated with single-factor authentication schemes and elevates identity assurances for both e-business providers and users;
- Hardware and/or software mechanisms that isolate user authentication processes and private user data from malware that may penetrate the user’s run-time environment executing on their computing platform;
- Procedures that vet users by way of document proofing and verification, and mechanisms that bind users’ physical credentials to their identities and electronic credentials. These credentials are locally available on the user’s platform, and also remotely to authorized identity and service providers;
- Architectural elements that isolate remote user authentication schemes and processing from primary application access and transaction flows;
- Components and mechanisms that federate user authentication for the benefit of multiple service provider applications, centralizing, hardening, and off-loading critical authentication processing from core information processing services;
- Provisioning the appropriate mechanisms and protocols needed to ensure that the communication channels connecting users to service provider applications and identity services, and application services to identity services, are reliable and highly resistant to MITM, MITB, and malware exploits mounted by phishers, hackers, BOTs and others.

The total risk posture of an identity solution is only as strong as the system architecture’s binding strength. For example, a reliable biometric authentication technology is not very useful if an attack can be mounted that bypasses this functionality. It is therefore essential that the software components and protocols implementing the architecture’s binding mechanisms also be highly trusted - resisting

determined Man-In-The-Middle (MITM), Man-In-The-Browser (MITB), and malware attacks by phishers, hackers, BOTs and other attack agents.

In the final analysis, the identity solution's architecture is the essential glue that binds distinct roles and responsibilities of each technology element into a cohesive whole, mitigating the baseline risk of electronic fraud by way of the various user masquerade/impersonation scenarios.

14 Vision and Operational Concept

The above-articulated analysis has led me to formulate the following identity vision and operational concept which I offer to the reader. Your constructive feedback is invited.

a) Users own personal identity devices that contain their electronic identities with associated encryption keys which they use to (a) identify themselves and (b) secure collaboration with other parties.
b) Users (e.g. consumers, employees, admins, and managers) install trustworthy identity applications (apps) on their personal identity devices and servers to safeguard and manage their identities.
c) Users can exchange electronic identities with other users and servers having an identity application.
d) Identities are specified by electronic artifacts (I call them "e-credentials"). Each e-credential of an owner includes personally identifying information chosen by the owner including identifiers, attributes and images associated with the owner. The identifiers may be pre-existing (e.g. social security number) or may be created by the owner. Attributes can be physical characteristics and life events. Images selected by the owner may depict the user themselves or physical artifacts with which the user openly or secretly identifies (e.g. an article of clothing, a favorite object, or a super-hero).
e) The identity device owner can specify "true", pseudo-anonymous and anonymous identities. A "true" identity could be a driver's license, a credit card, or a business card; a pseudo-anonymous identity is one that is known only to selected friends and associates; an anonymous identity is a secret handle known only to the owner (e.g. for web blogging purposes).
f) After installing their identity app, the owner enrolls their authentication data (e.g. PIN and/or a biometric) and defines their personal profile including their identifying images and contact info.
g) Subsequently, the owner submits already specified e-credentials to collaborating parties requesting them to proof and attest to their identities including accompanying personally identifying information. The owner's identity app cryptographically binds their identity to requests such that the owner cannot repudiate having submitted the request. Both application servers and users can attest identities.
h) In response to receiving an e-credential request, an owner uses their identity app to inspect and proof the provided e-credential and personally identifying information. If satisfied, the recipient uses their identity app to attest to the requester's identity, cryptographically bind their identity and attestation to the e-credential subsequently issued to the requester. The issuer cannot repudiate this action.
i) Once e-credentials are exchanged, identity apps can use the cryptographic properties of e-credentials to establish mutually trusted channels for transactions and document notarization.
j) When initially exchanging e-credentials, the identity apps can generate and exchange one-time-passwords (OTPs) over alternate channels (e.g. in-person, phone, texting, email, etc.).

15 Why this Operational Concept is Promising

a) Given the identity app controls both the owner's enrolled authentication data and the owner's e-credentials, the app strongly binds owners to their identities and, in turn, enables the identity app to employ the encryption keys associated with each e-credential to remotely bind collaborating owners.
b) As suggested by Asokan [6], identities should have multiple public/private crypto key pairs, each pair designated to perform distinct cryptographic functions. For example, cryptographic functions

designed to achieve confidentiality, integrity, originator authentication, and non-repudiation should use distinct key pairs. Breaking of one cryptographic key does not break the others.

c) Private keys associated with any given e-credential are “secrets” of the owner and must not be revealed by the identity device and app. This ensures that an owner’s identity acquired by way of identity theft or a server-side breach cannot be used to impersonate the owner. Remote parties (users and servers) will be able to conduct tests to verify proof-of-possession of these private keys.
d) Ideally, a given owner’s e-credential should be proofed and attested by multiple parties. Such a strategy elevates assurances that the e-credentials actually represent the person specified. Furthermore, the likelihood that an identity thief could conspire to issue such multiply-attested e-credentials, each of which cannot be repudiated without detection, would be significantly reduced.
e) Once e-credentials have been successfully exchanged between collaborators, man-in-the-middle (MITM) and phishing attacks are prevented because the attacker does not possess either collaborating party’s private keys. Exchanging an OTP out-of-band to bootstrap e-credential exchange similarly thwarts MITM and phishing attacks.
f) An e-credential, attested to using an e-credential of the owner and stored on a backup device/server of the owner, or held in escrow, can be used to create a “poison-pill” that only the owner can activate, for example, to disable or wipe their device if lost or stolen.
g) Of course, identity app must be trustworthy and isolated from tampering by unintentionally hosted malware and faulty software. This should be achieved by employing a trusted operating system and platform (e.g. trusted execution environment, trusted platform module (TPM) or trust zone). The identity device should also safeguard the user’s private information, authentication data, and secret keys in a protected memory store that can only be accessed by the identity app.

16 How this Approach Compares to Other Solutions

In contrast to Public Key Infrastructure (PKI) [13], e-credentials can be specified richly (suitable for consumer identities); have multiple public/private key pairs, each for well-specified purposes (hence less vulnerable), and e-credential can be attested and issued by multiple collaborating parties (other users as well as service providers/applications). This significantly elevates identity assurances by reducing the risk that a compromised identity will proliferate across the web.

Like PGP [14], I have proposed that private keys associated with an owner’s e-credentials be strongly protected by the identity app and not revealed to other parties. I also advocate that other parties be obliged to routinely execute remote proof-of-possession tests that verify that the e-credential owner controls the associated private encryption keys. In contrast to PGP I have also advocated that both ordinary users and designated identity authorities be capable of proofing, attesting and issuing identities to other users as well as to service providers.

While FIDO [29] reduces dependence on passwords, their authenticators do not support the specification or proliferation of user identities. Personally identifying information continues to be stored on servers where they remain vulnerable to server-side breaches. The approach I have advocated specifies e-credentials that, if stolen, cannot be used to easily create fraudulent identities because the private keys are controlled by owners and collaborating parties are obliged to conduct proof-of-possession tests. This strategy would prevent identity thieves employing stolen identities because they would not be in possession of users’ private keys.

17 Role of Software Quality Engineering

It is reasonable to argue that experienced software quality engineering practitioners are ideally placed to implement such a mission-critical identity vision and concept. Certainly, information security expertise, including experience with the application of cryptographic components and related security protocols will

be essential. However, we should not forget that other software quality engineering skills and processes will also be critical success factors, including: functional and operational requirements analysis and specification, architectural design know-how, reviews and walkthroughs, static code analysis and inspections, multiple levels of testing, penetration testing, independent quality assurance, and independent (3rd party) verification and validation (IV&V).

References

[1] Lt. Dan, A 'Perfect Storm' for Data Breaches, Experian, Dec. 17, 2014

[2] Draft NIST Special Pub. 800-157, Guidelines for Derived PIV Credentials, March 2014

[3] Miniwatts Marketing Group, 2014

[4] Open Web Application Security Project (OWASP) Top 10 Project, https://www.owasp.org, 2013

[5] National Institute of Standards and Technology SP-800-63-1, "Electronic Authentication Guideline", 2011

[6] Asokan et. al., "On the Usefulness of Proof of Possession", PKI workshop, 2013

[7] Zaker Soltani, "Improving PKI: Solution Analysis in Case of CA Compromisation", 2013

[8] NIST Special Pub. 800-63-2 Electronic Authentication Guideline, August 2013

[9] Kantara Initiative, "Identity Assurance Framework, Service Assessment Criteria", April 8, 2010

[10] Adeoye, A Survey of Emerging Biometric Technologies, Int'l Journal of Computer Appl, Nov. 2010

[11] ITU, X.1252, "Baseline Identity Management Terms & Definitions", April 2010

[12] RSA Laboratories, B. Kaliski, PKCS #5: Password-Based Cryptography Spec'n, V2.0, Sept 2000

[13] Internet X.509 PKI and CRL profile, Network Working Group, 2008

[14] Finnet, IETF, Open PGP Message Format, Network Working Group, Nov. 2007

[15] Gary Blair, SMS-delivered two-factor authentication will be dead in three years, 2007

[16] Kalman C. Toth, A Practical Identity Management Reference Implementation, CATA, Honolulu, Hawaii, 2007

[17] K. Toth, Identity Management Systems, tutorial for IEEE COMPSAC, Chicago, September 2006

[18] Toth, Persona Concept for Web-Based Identity Management, Int'l Conf on Privacy, Security & Trust, 2006

[19] Sarbanes – Oxley Act of 2002, top management accountable for financial accuracy of corporate information

[20] Ma, L., et.al., "Efficient iris recognition by characterizing key local variations". *IEEE Trans. Image Processing. 2004*

[21] K.C. Toth, M.Subramanium, Req'ts for the Persona Concept, RHAS'03 workshop, Monterey, 2003

[22] Toth, Subramanium, Persona Concept: MobEA, Budapest, 2003

[23] Toth, Subramanium, Persona Concept for Privacy and Authentication, In'l Bus & Eco Res. Jrn,2003

[24] Toth, Subramanium, Chen, Persona Concept for Privacy and Authentication, IABR Conf, 2003

[25] K. Toth, Information Security Architectures, AFCEA, Hawaii, 1990

[26] K. Toth, Towards an Improved Information Security Model, 1st Can. Comp. Security Conf, Jan'89

[27] K. Toth, Data Encryption Equipment Specification, Internal report specifying CryptoNet, 1986

[28] Implementing Mutual Authentication Using TLS/SSL: http://en.wikipedia.org/wiki/Transport_Layer_Security

[29] Fast Identity Online (FIDO) specification: www.fidoalliance.org

[30] FIPS PUB 201-2, Personal Identity Verification of Federal Employees and Contractors

[31] DoD Common Access Card Standards, http://www.cac.mil/common-access-card/

[32] Payment Card Industry Standards Site: https://www.pcisecuritystandards.org/

[33] RSA Encryption Standards: see http://en.wikipedia.org/wiki/RSA_%28cryptosystem%29

[34] Elliptical Curve Encryption: see http://en.wikipedia.org/wiki/Elliptic_curve_cryptography

Misusing the Type System for Fun and Profit

Ian Dees

me@ian.dees.name

Abstract

It's amazing what a type system can do at compile time. This paper offers a peek at the C++ type system and standard libraries, which can make certain types of bugs difficult or even impossible to express in code. By brewing more robust code interfaces, engineers can save their QA teams from chasing down rare crashes, and let them focus instead on usability and performance.

Writing good software is difficult; doubly so in an unsafe language like C++. A simple off-by-one indexing error can cause code execution vulnerabilities. A couple of lines of cut-and-paste code can lead to memory leaks that slowly rob users of good performance.

Whenever an error strikes, one's reaction should be, "How can we make it harder for this bug to happen again?" Typical solutions involve more tests, or code review. These are both good things. But what if engineers could make errors disappear from a code base forever—that is, prevent them at compile time? With its support for generic and functional programming, C++ is finally becoming an interesting enough programming language to explore this style of bug prevention.

Programmers may be familiar with using smart pointers to reduce memory leaks, or with using constructs like `boost::optional` to prevent data corruption. This paper will take these familiar ideas to the next logical step, using a technique called *denotational design*.

With denotational design, as espoused by Conal Elliott, a developer can first sketch a design in a type-safe language like Idris—even mathematically proving parts of a design correct—before translating to C++. One can dramatically change a C++ API by looking at it through the lens of denotational design.

Biography

Ian Dees was first bitten by the programming bug in 1986 on a Timex Sinclair 1000, and has been having a blast in his software apprenticeship ever since. By day, Ian slings code, tests, and puns in Portland, Oregon. By night, he dons a cape and keeps watch over the city as Sidekick Man. In his (heh) "spare time," he converts espresso into programming books, including (with others) Seven More Languages in Seven Weeks*.*

Background

Writing correct software is hard—doubly so in a systems programming language like C++. The same language features that offer low-level control over the machine are hard to code and debug.

With a slip of the keyboard, a developer types <= instead of <, and tomorrow the application is on the front pages with a zero-day security vulnerability—buffer overflows are among the oldest classes of exploits (Litterio). An engineer forgets to check just one memory access among thousands, and the program crashes to a rude halt—or worse, silently corrupts data that will only reveal itself days later.

1.1 Traditional Advice

Pundits are eager to explain how to solve these problems. Their solutions typically fall into one of the following bits of advice:

- *Just be extra careful*. If everyone could simply never commit boundary errors, never forget to check a return value, and so on, these errors would vanish. But developers already are being careful! These errors are creeping in despite everyone's efforts.
- *Just use another language*. C++ may unsafe, the argument goes, but switching to Haskell, Rust, Go, or the next language *du jour* could eliminate these problems. Changing languages may indeed be a workable approach for a new project. But teams rarely have the luxury of starting from scratch. A typical project has legacy code, portability constraints, or performance requirements that restrict the available language choices.
- *Just test more*. What the programming language can't do, the development team pawns off on testing. Coders write unit tests to verify what they couldn't check at compile time. The QA team hopes to catch the rest of the problems with exploratory testing. These test techniques are valuable. But they do not prevent writing the errors in the first place. "You can't test quality in," as the saying goes (Mack).
- *Just review your code*. This is the one piece of advice that stands the greatest chance of preventing bugs. Even the most conservative estimate is that peer code review can catch 60% of errors (Glass). Code review is fully compatible with the techniques discussed in this paper, and can even help identify opportunities to use them.

All of these recommendations—careful coding, safe languages, testing, and peer review—are useful. But they do not all apply to legacy systems, nor do they all help prevent bugs in the first place. Prevention requires a new approach to code.

1.2 A New Approach: Lean on the Type System

With languages like C and older dialects of C++, engineers are used to thinking of types as expressing what the code can do. For example, one may say, "This function multiplies two floating-point numbers." "This data structure holds an employee ID and a name."

There is, however, another way to look at what type systems do. Not only do they describe what operations *are* allowed on a piece of data, they also express what's *not* allowed. One may not take the square root of a string, nor concatenate characters onto an integer.

By combining this "can't do" mentality with a sophisticated enough type system, one can make certain classes of errors *inexpressible*. In other words, one can move errors from runtime, where tests might catch them, to compile time. A good type system can prevent the following kinds of errors:

- *Null pointer accesses*, where a program attempts to access a memory location that doesn't exist
- *Unclear allocation policies*, which end up skipping cleanup code and leaking memory
- *Indexing errors*, where a routine may corrupt data outside a container
- *Unchecked return values*, where a coder assumes a computation succeeded without actually looking

As it turns out, mainstream systems programming now has a language and type system sophisticated enough to advance this goal.

The C++11 language and its libraries offer a number of ways to reduce or prevent certain kinds of errors. Reference types have all the memory efficiency of pointers, but can never be null. Smart pointers make memory leaks much more difficult to write. The `optional` class from the add-on Boost library can nudge a forgetful developer to check a return value.

This paper will explore similar ideas through a technique called *denotational design*. It will offer a peek at how even more advanced type systems work, and will show how to apply a few of these lessons in C++.

Denotational Design

In his 2009 paper “Denotational design with type class morphisms,” Conal Elliott explains a technique for designing mathematically sound APIs. His core recommendation is, “When designing software, in addition to innovating in your implementations, relate them to precise and familiar semantic models” (Elliott).

In other words, he is urging architects to design their programs in two steps:

1. Sketch the design in clear, expressive notation
2. Translate this idea into the implementation language

In Elliott's paper, these two notations—the sketching and implementation languages—can be the same programming language. Indeed, he gives examples of a design that begins in a mathematically rigorous Haskell style, and ends in more idiomatic, performant Haskell.

However, one can also use two different languages for these two steps. This paper will offer a glimpse of API design improvement techniques using the following two languages:

1. A dependently typed language called Idris for the sketching language
2. C++11 for the implementation language

Much of the expressive power of Idris’s type system will carry over into the C++11 translation.

1.3 Dependent Types in Idris

Idris is a Haskell-like programming language with an even more powerful type system based on the idea of *dependent types* (Brady). In Idris, functions can depend on types—a bit like C++'s template functions, but more expressive. The type system is so deep that one can even encode correctness proofs directly into source code.

A simple example may help illustrate the concept of dependent types. Here’s an Idris function that takes a type as an input parameter and returns an ordinary Boolean value:

```
isNumeric : Type -> Bool
isNumeric Int    = True
isNumeric Float  = True
isNumeric String = False
--
-- and so on...
```

To use this function, the caller does not pass in an individual integer such as 42, or a string such as “hello world.” Instead, the caller passes in the `Int` or `String` types themselves. The function returns `True` if the passed-in type is a numeric type, and `False` otherwise.

This ability to “do math on types” lets developers express constraints *at compile time* on what functions are or aren’t allowed to do. One encodes these constraints directly into the function declaration. If the implementation ever violates the restrictions, the program will not compile.

For example, here’s the declaration for a function that adds together the elements from two vectors of length *k*, resulting in a third vector—also of length *k*:

```
add : Vect k a -> Vect k a -> Vect k a
```

The compiler will check that every caller correctly passes in two lists of the same length, *and* it will verify that the function implementation (not shown here) returns a list of the same length.

Idris's type system can express much more sophisticated properties than just collection size. The following snippet from Ben Sherman declares a quicksort function that is guaranteed by the compiler to result in a list that's a sorted permutation of its input (Sherman):

```
quicksort : TotalOrder a lte
          -> (xs : List a)
          -> (ys : List a ** (IsSorted lte ys, Permutation xs ys))
```

In a mainstream language, one would need to write several test cases to be confident in the implementation of the sort routine. In a design-by-contract language, one would have to add runtime preconditions and post-conditions all over the code. But with Idris, the *compiler* can prove that the `IsSorted` property will hold for all possible inputs.

Not all of these concepts will translate directly to C++. But stepping into a more rigorous language has other benefits, even if one can't bring all of Idris into day-to-day work. In particular:

- Shifting notations helps clarify the essential parts of the API and avoid class bloat
- Sketching in a math-like language encourages more rigorous models than pseudo-code does
- Some of the type-checking features still apply in C++

The next section will walk through a couple of examples of API improvement where Idris can inform the design of the C++ code.

Taking It to (C++)11

Designing C++ APIs in dependently-typed languages is far from a theoretical exercise. In his talk "The Intellectual Ascent to Agda," David Sankel provides practical advice for C++ implementers sketching in Agda, a language with similar power to Idris (Sankel). This paper will adapt some of Sankel's techniques to Idris, which has more tutorial information for newcomers than Agda does.

Some examples of dependently typed code are directly expressible in C++. For example, recall the signature of the Idris function for adding two vectors of length *k*:

```
add : Vect k a -> Vect k a -> Vect k a
```

Simple constraints like collection size are fairly easy to translate. Here's a C++ function similar to the Idris one, using the `array` template that comes with the C++11 standard library:

```
template <typename A, size_t K>
array<A, K> add(array<A, K>, array<A, K>);
```

More sophisticated concepts may not map so directly into C++. Even so, expressing an idea in Idris can still help a C++ design. This section will demonstrate this technique using a couple of fictional examples based loosely on real programming errors discovered in the wild.

1.4 Example 1: A Simple Matrix Data Type

Consider the following type representing a two-dimensional matrix of numbers, indexed by row and column:

```
template <typename T>
class Matrix
{
public:
    T operator()(size_t row, size_t col);

    // ...
};
```

This API looks straightforward enough: the caller passes in a row and column, and the method returns the value located there. But what happens when a maintenance programmer forgets that the parameter order is `(row, column)` and instead passes indices as `(x, y)`, the opposite order?

```
Matrix<double> m(3, 5);

size_t const x = 4;
size_t const y = 0;
cout << m(x, y) << endl; // wrong order!
```

Depending on how the matrix values are stored, the method could walk right off the end of an internal array and corrupt memory. The program will either crash right away, or—if the developer is unlucky—silently and mysteriously overwrite data belonging to a different part of the program. These kinds of data corruption issues are notoriously hard to track down.

1.4.1 Sketching in Idris

At this point, it's worth stepping back from the C++ code and building a mental model of what a matrix should look like. A simple Idris function declaration will help sketch the idea out.

A matrix is a two-dimensional collection of values, arranged in a specific number of rows and columns. In Idris terms, one would write a function taking two natural numbers for the dimensions, plus the type of data being kept in the matrix:

```
Matrix : Nat -> Nat -> Type -> Type
```

The matrix would provide access to individual elements through a function—call it `get`—that takes two natural numbers for the row and column and returns an element:

```
get : Nat -> Nat -> Matrix rows cols a -> a
```

This is a decent first cut. It's roughly the equivalent of the C++ version from earlier, with no range checking. But one can describe a matrix's properties more thoroughly.

For retrieving an element, the row index must be less than the total number of rows, and the column index must be less than the number of columns. One can model this behavior using a built-in Idris data type for bounded numbers, called `Fin` (for "finite").

```
get : (Fin rows) -> (Fin cols) -> Matrix rows cols a -> a
```

Now, the `get` function takes a row index bounded by `rows`, the number of rows in the matrix. A similar constraint holds for the column index.

If a caller tries to use `get` with an out-of-bounds index, their program won't pass the type checker. Rather than having to lean on code review or runtime tests (though both of these things are good!), one can ensure at compile time that we are using legal row and column indices.

1.4.2 Back to C++

How might this idea translate to C++? If C++ had a `Fin` type like Idris's, one could make the row and column sizes a part of the `Matrix` data type:

```
template <size_t R, size_t C, typename T>
class Matrix
{
```

```
public:
    T operator()(Fin<R> row, Fin<C> col);

    // ...
};
```

Then, the compiler would catch an accidental out-of-bounds use of the array:

```
Matrix<3, 5, double> m;

auto x = Fin<4>();
auto y = Fin<0>();

// error: no known conversion from 'Fin<4>' to 'Fin<3>' for 1st argument
cout << m(x, y) << endl;
```

This is a decent proof of concept. However, the C++ type system is not quite expressive enough to define `Fin` universally. (It is possible to define a limited version that works for this example.) It is time to step back a bit and rethink the data type.

1.4.3 Stepping Back from the Brink

If it's impossible to constrain the index bounds at compile time, perhaps there is another way to prevent using a row in place of a column. One approach might be to define `Row` and `Col` as two different types:

```
data Row a = aRow a
data Col a = aCol a
```

These declarations define two new types, plus constructors `aRow` and `aCol` to create values of those types. Here's how one might adapt the `get` function to use them:

```
get : Row (Fin rows) -> Col (Fin cols) -> Matrix rows cols a -> a
```

Now, the type checker will notice if a caller tries to pass a row index for a column. This design, as it turns out, is implementable in C++. First, a simple class, Row, wraps an integer in a type-safe way:

```
class Row
{
public:
    explicit Row(size_t value) : value_(value) {}
    operator size_t() { return value_; }

private:
    size_t value_;
};
```

Because the constructor is marked as `explicit`, it's impossible to use a raw integer by accident in place of a row. The `Col` class will be nearly identical; it makes sense to abstract these two types into a template:

```
enum class Dimension
{
    Row,
    Col
};
```

```
template<typename T, Dimension D>
class Wrapper
{
public:
    explicit Wrapper(size_t value) : value_(value) {}
    operator size_t() { return value_; }

private:
    size_t value_;
};

typedef Wrapper<size_t, Dimension::Row> Row;
typedef Wrapper<size_t, Dimension::Col> Col;
```

Now, the signature for the matrix indexer can take `Row` and `Col` types, just like the Idris version:

```
T operator()(Row row, Col col);
```

The compiler will catch accidental switching of the row and column:

```
// error: no known conversion from
// 'Wrapper<[...], 1>' to 'Wrapper<[...], 0>'
cout << m(x, y) << endl;
```

It will not, however, catch deliberate misuse, such as indexing a 3×5 matrix with `Row(1000000)`. Still, the new design uses the C++ type system more fully than before.

1.5 Example 2: Audio Clips

Sketching in a type-safe language is not just about preventing accidental API misuse. It can also help simplify and clarify a library.

Consider the following class that represents a clip of audio data:

```
class Audio
{
public:
    size_t count() const;

    int16_t  operator[](size_t index) const;
    int16_t& operator[](size_t index);

    double timeAtZero() const;
    double sampleRate() const;

    double verticalScale() const;

    std::string fileName() const;
    std::string comments() const;
    bool isStereo() const;

    void loadFromFile(const std::string& filename);
    void saveToFile(const std::string& filename);
};
```

This class uses a fairly common C++ style. The method names are clear, but the class is somewhat large for what it does. It mixes together unrelated concepts: element access, scaling parameters, metadata, and file I/O.

One can simplify this class slightly by moving the metadata and I/O routines elsewhere:

```
class Audio
{
public:
    size_t count() const;

    int16_t  operator[](size_t index) const;
    int16_t& operator[](size_t index);

    double timeAtZero() const;
    double sampleRate() const;

    double verticalScale() const;
};
```

The improved version still has issues, though. It exposes the internal representation of the data: 16-bit integers. The class also depends on the caller to handle conversion and scaling from integer values (indices and samples) to their real-world values (times and voltages). First, the caller would have to convert their time value into an index:

```
Audio audio;
double time  = 0.5;
size_t index = (time - audio.timeAtZero()) /
               audio.sampleRate();
```

Next, they would have to retrieve the *nth* integer sample and convert it to a real-word audio level:

```
int16_t sample = audio[index];
double  level  = static_cast<double>(sample) *
                 audio.verticalScale();
```

Doing this kind of conversion in every caller is tiresome and error-prone. Moreover, error checking (such as making sure the index value is within bounds) is left up to the caller.

One could provide these conversions as methods on the class, but doing so would make the class even larger and more difficult to understand. Instead, it's worth taking a step back and rethinking this API using Idris.

1.5.1 Sketching in Idris

To simplify this design, one can ask the question, "What is an audio clip?" It is tempting to leap straight to the representation and answer something like, "An array of evenly spaced 16-bit samples, scaled by a common factor." But those are implementation details. They don't describe what an audio clip really is from the user's perspective.

A better answer would be, "An audio level changing over time." This answer does not mention bit widths or sample rates. These are important details, but they are not relevant to the fundamental question.

If an audio clip is a changing audio level, what is an audio level? There doesn't need to be a single answer to this second question. It could be a floating-point number, a custom data type representing audio levels, or some other type.

In other words, an audio clip is a dependent type! This idea is easy to express in Idris. One could write an `Audio` function that takes an audio level type, `a`, and returns a new type:

```
Audio : (a : Type) -> Type
```

The resulting type represents the entire audio clip. It needs to provide a mapping from time values to audio levels. This mapping could simply be a function from `Float` to `a`.

```
Audio a = (Float -> a)
```

But this approach does not consider time values before and after the audio clip's contents. The API needs a way to return nothing if the time value is out of range. Fortunately, Idris provides just such a construct, called `Maybe`:

```
Audio a = (Float -> Maybe a)
```

If a caller passes in a valid time value, the result will be an audio level. Otherwise it will be a special value called `None`. Crucially, the Idris compiler will not allow callers to forget to check for `None`. Their code must explicitly consider this case.

1.5.2 Back to C++

It is possible to express the key ideas of this design in C++. The dependent `Audio` type becomes a class template, taking the audio level type as an argument:

```
template <typename T>
class Audio
{
public:
    // mapping goes here
};
```

All that's left is to provide the mapping from time to audio level. The Idris version does so with an ordinary function. A C++ class can provide a similar interface; it can act like a function call by defining the call operator.

What about the return type? The function needs to return an audio level only if the time is in range, and some kind of placeholder value otherwise. Fortunately, the add-on Boost library provides a type called `optional` that works much like Idris's `Maybe`:

```
boost::optional<T> operator()(double t);
```

Now, callers have a much simpler interface for getting the audio level for a given time:

```
double time = 0.5;
auto level = audio(time);
```

Moreover, the C++ compiler will remind callers to check the return value; if they try to use `level` directly, they will get a compilation error:

```
// error: invalid operands to binary expression
// ('boost::optional<double>' and 'double')
double result = level + 1.0;
```

Instead, callers will check that `level` has a value (by using it in a conditional statement), and then get that value by dereferencing `boost::optional`:

```
if (level) { double result = *level + 1.0; }
```

Although one could maliciously dereference the value without checking, the compiler will guard against accidentally forgetting to check.

Next Steps

This paper has shown how changing one's perspective helps focus on the essential parts of the problem being solved. The Idris sketches have hinted at what's possible with a notation that supports expressing thoughts in code—not just how many arguments a function has, but the relationships between those arguments.

The point is not to abandon day-to-day languages and switch to something new like Idris or Agda. The point is to bring a little rigor from strongly typed languages into one's mental models, and then return to C++ with a renewed understanding. Some of the power of dependent type systems will inevitably be left behind, but C++11 is expressive enough to preserve many of the concepts.

As a next step, readers should watch David Sankel's talk on Agda, and see how he was able to clean up several C++ APIs by modeling them in simple mathematical notation. The following questions can then guide future exploration:

1. Have you encountered a hard-to-use C++ library in your daily work?
2. Was it crash-prone if passed bad arguments?
3. Did the parameters have extra rules that weren't obvious in the function signature?

Readers could then try making these hidden constraints explicit, by describing them in a dependently typed language. What would a new and improved version of the C++ API look like?

References

Brady, Edwin, et al. "Idris: A Language with Dependent Types," University of St. Andrews, http://www.idris-lang.org (accessed July 24, 2015).

Elliott, Conal. "Denotational design with type class morphisms." *LambdaPix Technical Report 2009-01*.

Glass, Robert. *Facts and Fallacies of Software Engineering*. New Jersey: Addison-Wesley Professional, 2002.

Litterio, Francis. "The Internet Worm of 1998," http://web.archive.org/web/20070520233435/http://world.std.com/~franl/worm.html (accessed July 30, 2015).

Mack, Wayne, et al. "Quality Assurance is Not Responsible for Quality," Cunningham & Cunningham, http://c2.com/cgi/wiki?QualityAssuranceIsNotResponsibleForQuality (accessed July 24, 2015).

Sankel, David. "The Intellectual Ascent to Agda," CppNow 2013, http://2013.cppnow.org/session/the-intellectual-ascent-to-agda (accessed July 24, 2015).

Sherman, Ben. "Quicksort in Idris," https://github.com/bmsherman/blog/wiki/Quicksort-in-Idris (accessed July 24, 2015).

Brewing Quality in Android Robots

Nikhil Murthy[1], Anshuman Radhakrishnan[1], Simon Chow[1], Siddharth Suri[1], Sameer Suri[1] and Kingsum Chow[2]

batteriesinblack@gmail.com[1], kingsum.chow@intel.com[2]

Abstract

The emergence of versatile embedded processors and the Internet of Things bring new opportunities to the robotic world. The organization For Inspiration and Recognition of Science and Technology (*FIRST*) [1] that runs worldwide robotics competitions for K-12 students has recently announced changes in the programming platform for the *FIRST* Tech Challenge (FTC) robot game. They now adopt Java and Android running on commonly embedded devices that are similar to smartphones. The new programming platform not only makes the robot platform better, it also allows us to study the challenges of testing software quality in the modern robotic world. Using the case study of the FTC robot game, this paper describes the taxonomy of robot testing processes for software quality, covering several key areas: (1) testing robot software and sensor quality in the presence of uncertainties, (2) a systematic approach to identify defects in the performance of Android robots and (3) developing and testing machine learning programs for autonomous robots that improve decision making.

Biography

Batteries in Black is a robotics team formed in 2007. After four years in FIRST LEGO League (FLL), a LEGO robotics competition for elementary and middle schools, the team moved to FIRST Tech Challenge (FTC). The team has been recognized many times for being an exemplary team in the robot competition, community outreach, and software development. In the past two years, the team was nominated for the Control Award, an award that recognizes the programming and software achievements of the team, at the World Championships. In 2015, the team received the Sensor Savants Judges Award at the World Championships, recognizing the extensive research and effort put into developing and programming custom sensors.

Kingsum Chow is a Principal Engineer at Intel. He works in the System Technology and Optimization division of the Software and Services Group. He joined Intel in 1996 after receiving a Ph.D. in Computer Science and Engineering from the University of Washington. Since then, he has been working on performance, modeling and analysis of software applications. In his spare time, he volunteers to coach multiple robotics teams to bring the joy of learning Science, Technology, Engineering and Mathematics to the students in his community.

[1] FTC Team 4855 Batteries in Black, https://www.facebook.com/FTCBatteriesInBlack
[2] System Technologies and Optimization, Software and Services Group, Intel Corporation

1 Introduction

There are many programs and organizations around the world that aim to promote learning science, technology, engineering and math (STEM). Martínez et al [2] conducted an experiment with Preschool and Elementary students to determine how one's age affects the ability to learn basic robot programming. Additionally, the success of learning science through robotics is explored by a couple of publications by Niu et al [3] and Chow et al [4]. Phelim Dowling and Kevin McGrath [5] also describe an approach using free and open source tools to manage the efficiency and quality for planning and creating software projects. Another major organization that promotes STEM around the world is FIRST. FIRST stands for "For Inspiration and Recognition of Science and Technology" and FIRST is a worldwide robotics program that is aimed towards kids ranging from kindergarten to 12th grade. By using their knowledge gained from FIRST, the authors describe how a team should divide into specialized sub teams in order to increase the overall efficiency. The use of freely available resources and integrating those tools to maintain software quality is crucial as the popularity of learning science and engineering through robotics increases.

The First Tech Challenge (FTC) is one of the two options for high school or experienced middle school students who want to participate in the FIRST program.

The FTC robot game is composed of two phases: (1) the autonomous phase and (2) the user control phase. In the autonomous phase, the robot independently makes decisions about movement and scoring actions (e.g. picking up balls, or putting balls in goals). The challenges of testing robots in this phase include testing combinations of objects in the environment to simulate a great number of scenarios, and monitoring the robot's motions and decisions. In addition, sensor errors and variations of robot motions make testing the quality of software quite difficult. In the subsequent user control phase, team members use a pair of game controls and a tablet running a user-written software application to control the robot. In this phase, the main challenge is to be able to react and adapt accordingly based on the opponents' movements in order to outscore them.

1.1 Introduction to Robot Programming

Robots utilize many different types of computer controllers, making use of devices that have been specifically designed for the application of robotics. In FTC, the robot's motions are controlled by Android smartphones and tablets, which are programmed in Java. The Android device sends commands to different types of motors in order to make the robot move. To aid with decision making, the phone also can receive input from internal and external sensors.

The Android device on the robot is programmed from a computer, using Android Studio and the FTC SDK, which adds functionality to control motors from the phone. Both autonomous and user control programs are loaded onto the robot, and then are run from the Android device. During the autonomous phase, the robot moves based solely on its autonomous program. During the user control phase, the Android device on the robot is connected to another Android device through Wi-Fi Direct. This second Android device is connected to two joysticks and transfers joystick input to the Android device on the robot. From there, the drivers of the robot can use their joysticks to drive the robot. Teams use the Android software platform on the mobile phones located both on the robot and with the driver to run their programs that control the robot. A diagram illustrating the interaction between the different components of this system is below:

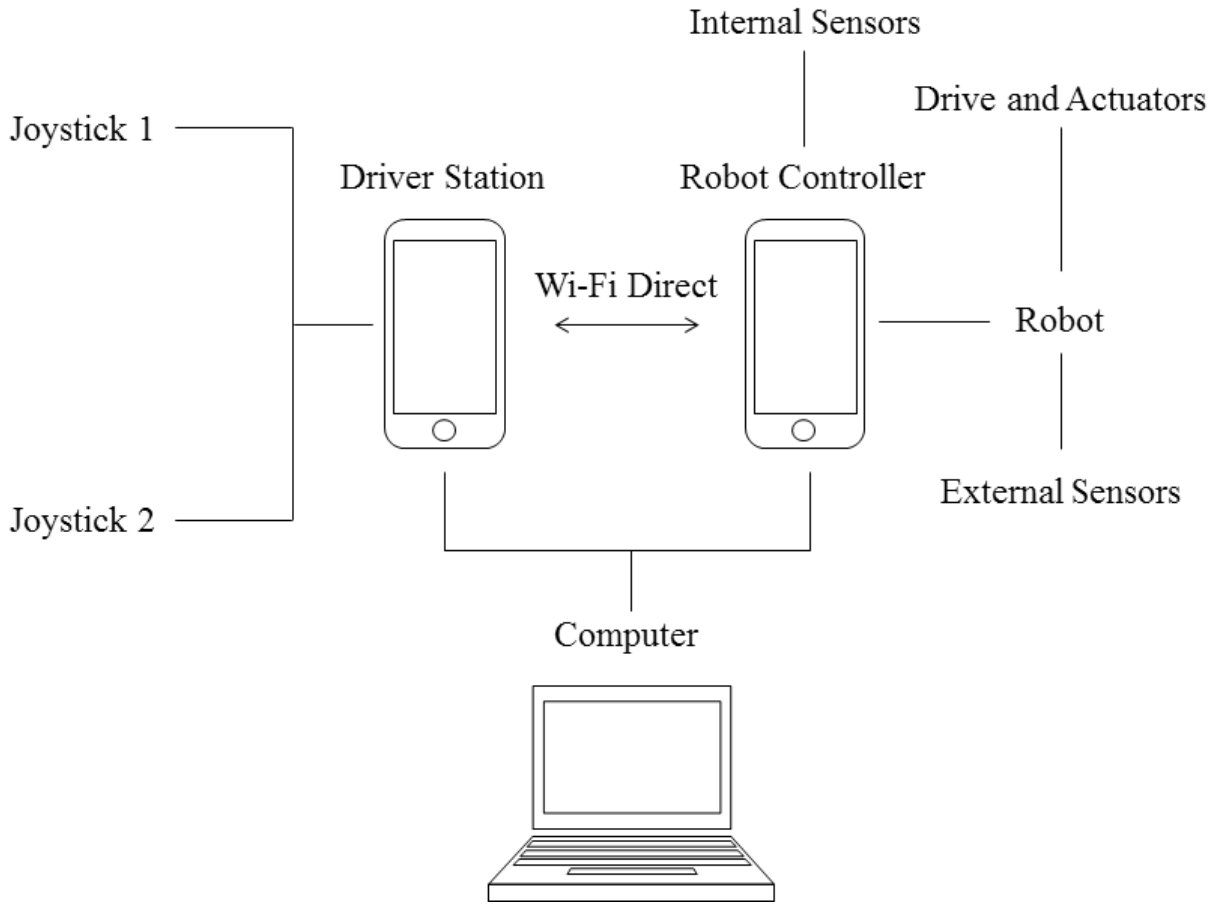


Figure 1. The control system of an FTC robot and its subsystems consisting of joysticks (used by drivers), Android smartphones with internal sensors, a robot with external sensors and actuators, and a computer for programming.

1.2 The Robot Components

The robot can be broken down into several components such as the hardware, software, internal sensors, and external sensors, and the communication among them. The hardware consists of the physical mechanisms that allow the robot to interact with the playing field and objects on the field. For example, actuators and drive trains are both part of the hardware. The software consists of the instructions given to the robot, whether the robot is operating autonomously or under human influence. These instructions can be altered by the use of sensors and these instructions can identify potential decisions. Internal sensors are built into the Android phone on the robot and thus are consistent between all robots on the field. These sensors include magnetometers and accelerometers. External sensors are attached to the robot but not to the phone directly. The communication components present sensor data from multiple sources to the robot controller. Android phones contain a multitude of built in sensors [6]. Some are used to collect motion data, others measure conditions in the environment, and some are used to measure the physical location of the device. For example, the Android phone used in FTC has a three-axis accelerometer. The light sensor and proximity sensor collect environmental data.

Each of these sensors collects different types of data which can be used by the robot to know its location in the environment, as well as states of the environment. The three-axis accelerometer records the current acceleration along the x-, y-, and z-axes of the phone. This is useful to be able to find the motion of the phone, based on changes of acceleration over time. The gravity sensor returns a vector that gives the direction of gravity, which shows the orientation of the phone in respect to the earth. The three-axis

magnetometer records the magnetic field of the earth, also on its x-, y-, and z-axes. This can also be used to find the motion and orientation of the phone. Using motion and orientation data from these sensors, it is possible to constantly calculate the position of the phone as it moves.

The light sensor measures ambient light in front of the phone, which can be used to understand the conditions of the phone's environment. The proximity sensor is a low-resolution distance sensor which returns whether or not an object is within 5 cm of the phone. This sensor gives a simple indication if your phone is close to an object. These two sensors allow the phone to get some simple information about the environment, which can help the robot know how to interact with its environment. In addition to sensors built into the phone, external sensors such as ultrasonic sensors, inertial measurement units, and infrared seekers can be useful.

2 Challenges of Brewing Software Quality in Robots

Testing and developing software for robots is inherently difficult due to the difficulty in isolating the precise point of failure, whether it is present in the hardware, software, or electrical systems. Robots also constantly evolve in terms of hardware and physical capabilities, thus software must be reliable and adaptable. In addition, field conditions vary, so the robot has to be able to adapt to a variety of conditions and make reliable decisions. This presents two main challenges in developing software for robotic competitions.

The first challenge is ensuring that internal sensors are reliable. Internal sensors, as defined above, are sensors built into the Android phones that are in use for the robotics competition, such as accelerometers and magnetometers. It is key that these sensors are reliable and can be characterized so that failure can be identified quickly for any of the components. To ensure that the variance between sensors on different phones is minimal, different phones need to be tested.

The second challenge is in ensuring that the robot can accurately make decisions in a variety of situations. This challenge is difficult to solve, because the robot potentially needs to accumulate knowledge from previous scenarios, and use the data from the current scenario to make an informed decision. Since so many factors change between each run of the robot, decision-making has to be tested thoroughly to ensure that the robot can consistently make the right decision.

3 A Taxonomy of Robot Testing Processes

Internal sensor reliability is a key challenge for the robot testing process. Internal sensors consist of the built in sensors in Android phones in the case study laid out in this paper, but these concepts can be applied to a general case in which sensors are used. By looking at the inherent error built into the sensors from stationary testing, the reproducibility of sensor readings through multiple trials, and the variance across multiple devices and sensors by performing similar tests on multiple Android phones, testing of the sensors is fully characterized. Robot function reliability consists of the robot's performance itself, its reproducibility, and the interaction between the software and physical elements. The robot decision reliability is critical in order to perform autonomously and minimize error, and consists of machine learning algorithms and decision-making techniques.

By using several case studies including the development of several robots in the context of the FIRST Tech Challenge (FTC) competition, the robot testing process and taxonomy is revealed. A black box approach through data logging is important in order to effectively and efficiently characterize the stimulus (input) and response (output) of the behavior. Testing within each category should cover three main classifications: the Physical World, the Software World, and the Interaction World (the interface between the other two worlds). The use of real and simulated hardware allows for extensive testing both before and after hardware has been developed.

3.1 Testing the Reproducibility of Sensor Readings on Mobile Phones

A case study involving ZTE Speed Android phones and their built in sensors will further illustrate the taxonomy of the robot testing processes. These phones contain numerous sensors including an accelerometer, linear accelerometer, and magnetometer. Phones were evaluated on a test bed that consisted of a rotating platform that spun the phone in a consistent manner while the phone recorded sensor readings. We also recorded data of a stationary phone as our control, to compare to the data from the spinning phone, and to see inherent error in the sensor.

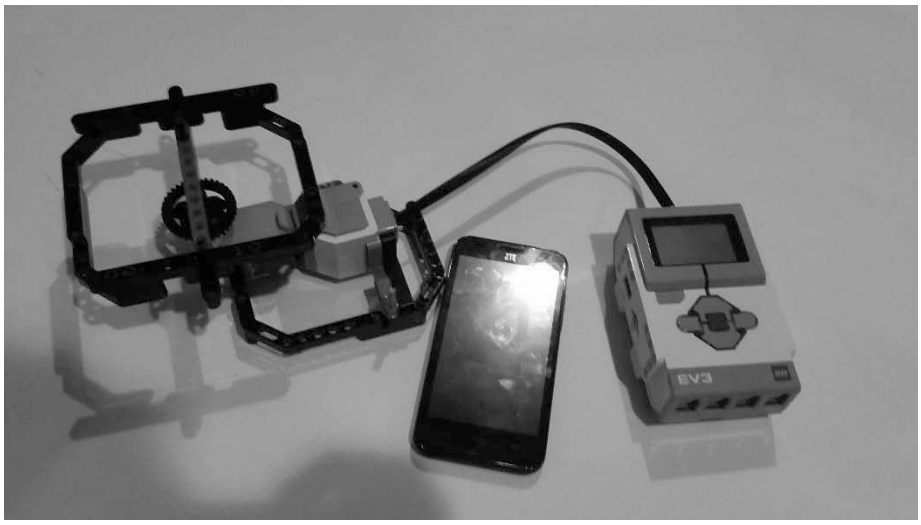


Figure 2. The test bed used to collect sensor data while the phone is rotated. A LEGO Mindstorms EV3 system is used to drive the rotation.

Figure 2 illustrates the testing rig connected to an LEGO Mindstorms EV3 system [6]. To test the reproducibility of movement with the case study of the ZTE speed phones, the consistency of data across multiple trials and different phones was tested. The test bed above was constructed to minimize error during data collection. For example, the turns that move the phone are powered by a LEGO motor, therefore allowing for controlled motion by another robot in comparison to movement by hand which brings in human error. While collecting data from the many sensors, the phone was rotated to different sized turns to fully test sensor readings ranging from stationary (0 degrees) to one rotation (360 degrees). These tests were repeated many times to analyze the consistency of the readings, and were repeated on another Android phone. Each phone was turned four times at each 90, 180, 270, and 360 degrees while collecting data. Both of these phones had the same sensors including an accelerometer and a magnetometer, allowing the devices to be directly compared. The data collected on each was compared to look at the reproducibility on different devices, a variable important in robotics.

Stationary tests in which the phone did not move were conducted. Here, the deviations from the mean were calculated for each sensor, allowing for an approximation of the error due both to environmental and non-environmental causes (i.e. sensor noise) that would be present in other tests. Table 1 illustrates the stationary tests (across multiple trials and phones) and each sensor's corresponding standard deviation:

Table 1. The standard deviations of internal Android sensor data of stationary tests. Tests were conducted on two phones and across 12 sensors.

Sensor	Standard Deviation Phone 1	Standard Deviation Phone 2
Accelerometer x-axis	0.22370 m/s^2	0.02261 m/s^2
Accelerometer y-axis	0.07897 m/s^2	0.02655 m/s^2
Accelerometer z-axis	0.16040 m/s^2	0.06752 m/s^2
Gravity x-axis	0.02313 m/s^2	0.00978 m/s^2
Gravity y-axis	0.02003 m/s^2	0.00568 m/s^2
Gravity z-axis	0.00057 m/s^2	0.00010 m/s^2
Linear Accelerometer x-axis	0.08300 m/s^2	0.02487 m/s^2
Linear Accelerometer y-axis	0.04811 m/s^2	0.02273 m/s^2
Linear Accelerometer z-axis	0.12271 m/s^2	0.04518 m/s^2
Magnetometer x-axis	0.23700 μT	0.34770 μT
Magnetometer y-axis	0.20070 μT	0.17742 μT
Magnetometer z-axis	0.24363 μT	0.28269 μT

In addition to stationary tests, repeated tests of different sized turns were done. These turns are 90, 180, 270, and 360 degrees and are done by a NXT robot with a high level of accuracy. This eliminated error that could arise if the phone was turned by hand. The sensor values were analyzed and compared to trigonometric functions due to the circular nature of the turn. The following graph, for example, shows the x-axis accelerometer data for a full 360 degree turn and the corresponding trend line:

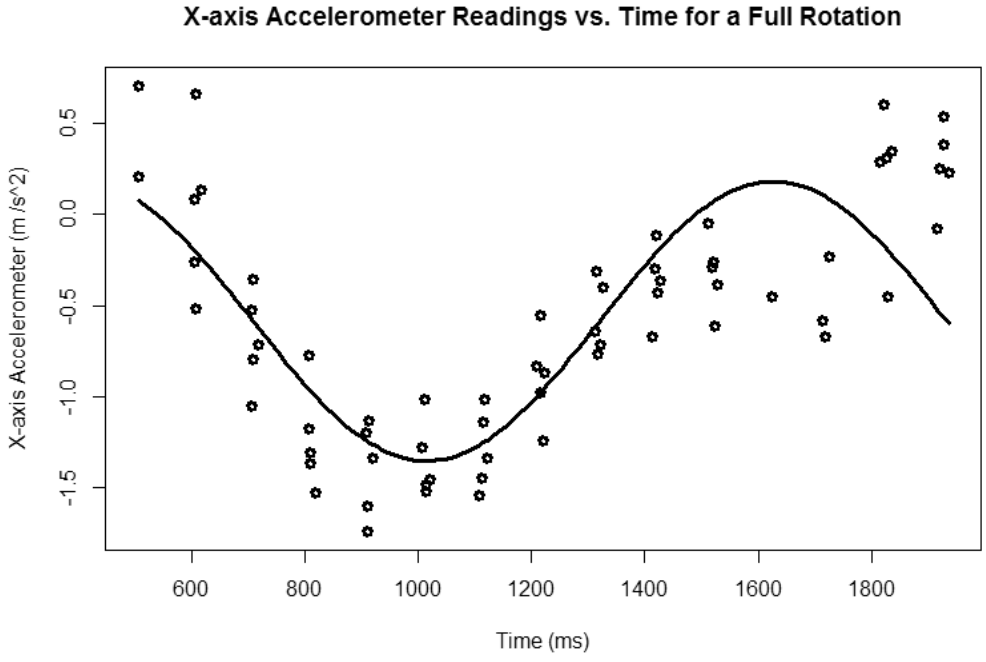


Figure 3. Sensor data of the x-axis accelerometer versus time over a full rotation on the first Android phone. The scatterplot highlights the trigonometric trend, which is further backed up by a regression: y=-0.58645-0.76761sin (0.4998-0.005136x) with a coefficient of determination of 0.9465.

The trend line is modeled by the equation: y=-0.58645-0.76761 sin (0.4998-0.005136x) and shows high correlation. The coefficient of determination for this model is 0.9465, signifying its high trigonometric correlation. Similarly, the same was done for the following sensors that showed a trend: Accelerometer (three axis) and magnetometer (three axis). The other sensors, such as the gravity z-axis sensor, had a constant value around 9.81 m/s^2, as expected. In addition to modeling these sensors' data and looking at their correlation, different phones and their reproducibility were analyzed. The graph below highlights the x-axis accelerometer readings on a second phone and a trigonometric model of the data:

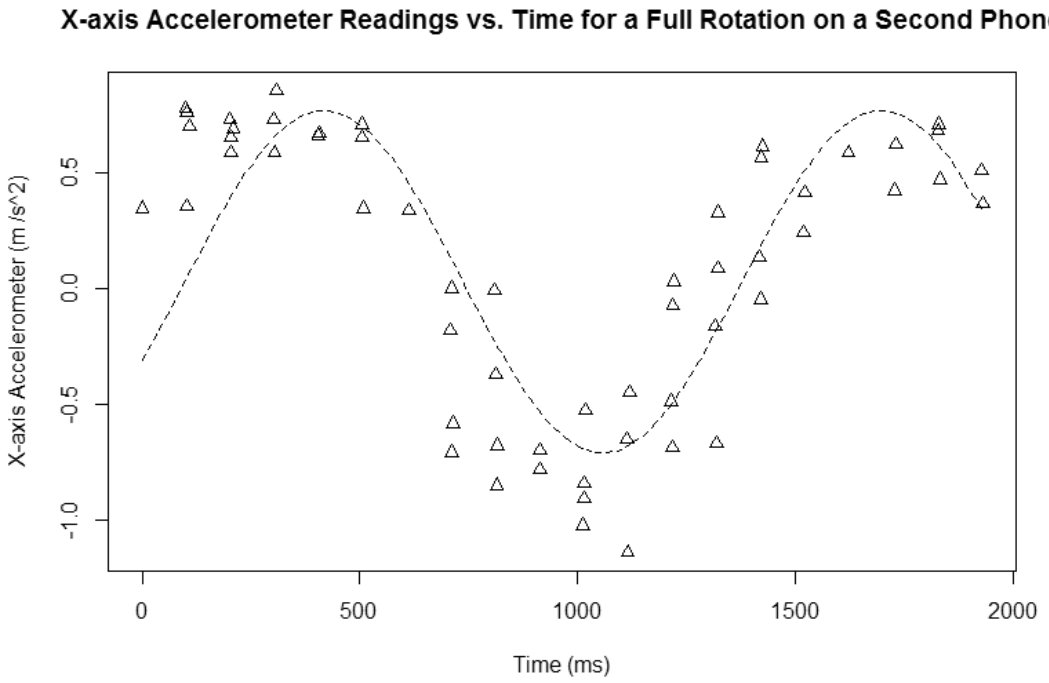


Figure 4. Sensor data of the x-axis accelerometer versus time over a full rotation on the second Android phone. The graph features a trigonometric trend, which is also shown by the trend line: y=-0.58645-0.76761sin (0.4998-0.005136x) with a coefficient of determination of 0.9465.

The trend line above is modeled by the equation: y = 0.02785-0.73899 sin(0.4800-0.00491x) and has a coefficient of determination 0.805135, highlighting its high correlation. The error in both phones from the theoretical sine wave to the data can be explained by multiple factors. First, the average standard deviation of both phones of x-axis accelerometer stationary data is 0.123155 m/s^2, which highlights the error of the sensors either deriving from environmental or inherent causes. Secondly, some error can be explained by the irregularities of the turn by the robot. In theory, the sine wave approximates a turn in which no acceleration occurs (by the NXT motor). This is physically impossible, and is indicated on the graphs by data points towards both the beginning and end that are farther away from the trend curve.

The phone's trend curves can be directly compared to one another by looking at their coefficients and graphs. The following table describes the coefficients of the general sine wave equation:

For the equation, y = a + b sin(c + dx):

Table 2. Table of coefficients of each phone's trend line in the general form of: y = a + bsin(c + dx). The percent difference between the two is also listed, allowing the two curves to be directly compared.

Coefficients:	a	b	c	D
Phone 1	-0.58645	-0.76761	0.4998	-0.005136
Phone 2	0.02785	-0.73899	0.4800	-0.004910
Percent Difference	2206%	3.87%	4.13%	4.60%

As shown in Table 2 the main difference between the two models is the coefficient a as all other coefficients have a percent difference of less than 5% (versus over 2000%). The two graphs also highlight this difference, seen by the apparent upward shift derived from the change in the coefficient, a:

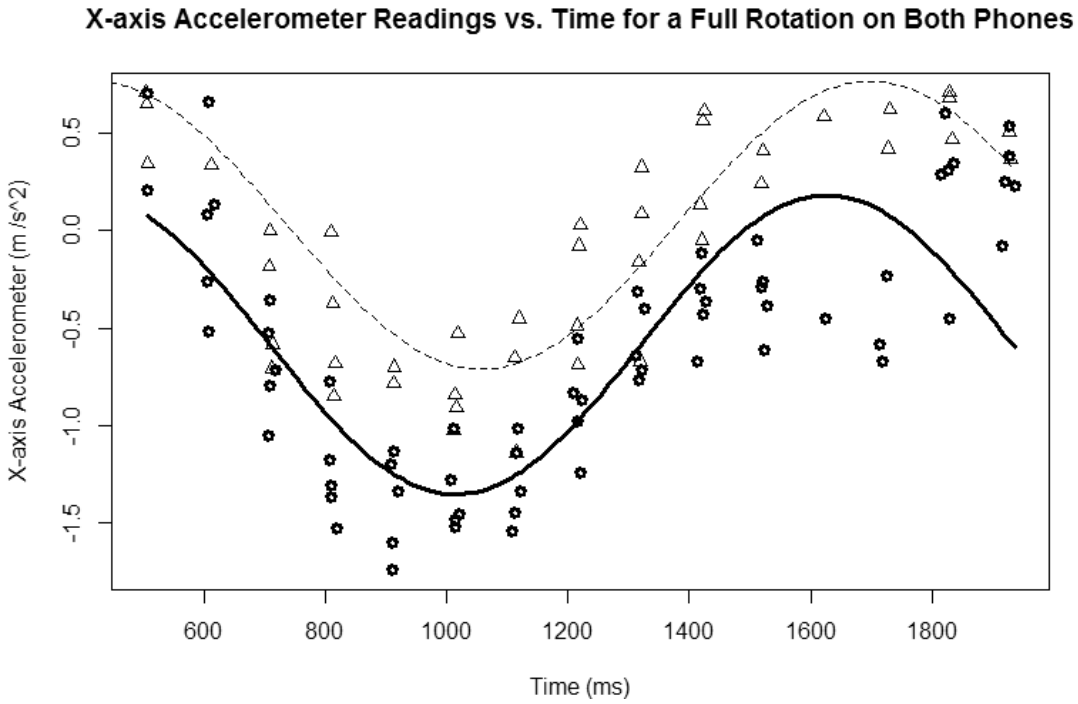


Figure 5. Sensor data of the x-axis accelerometer versus time over a full rotation on both Android phones. The first Android phone data is represented by the solid line and circles while the second Android phone data is represented by the dashed lines and triangles.

From looking at the variation in sensor data for circular motion, it was found that this data is very reliable and consistent across phones and across multiple trials. Although this is true, values are initially offset and therefore should be calibrated by accounting for the offset, which is roughly equal to the coefficient a in the general sine wave equation listed above.

Unlike testing with circular motion, testing for arbitrary motion was not very accurate. For testing arbitrary motion, we performed several experiments that involved two phones held together with rubber bands. The goal of banding them together was to negate the possibility of one phone moving away from the other. By negating the change in position between the two phones, we inferred that any discrepancies between the two data plots would have come from the uncertainty within the phone's data collection systems.

After finishing the data collection on our phones, we realized that another source of error for the experiment was that the two phones did not begin data collection at the exact same time, thus resulting in slightly inaccurate data. To fix this problem, we included a timestamp in our data analysis. This changed the independent variable of our data from time elapsed since data collection began, to time elapsed since January 1, 1970. This timestamp allowed the data from the two phones to be lined up with each other as they have the same starting point and this minimized the uncertainty and error in our data analysis.

After implementing the timestamp, we proceeded to graphing our processed data. Each phone had a separate graph for the x-axis, y-axis and z-axis accelerometer values. Since we were experimenting with two phones, this resulted in six individual graphs. It was observed that there was little difference between the data of the two phones.

In order to clearly show the error between the two data sets, we took the difference between each of the corresponding data values that were collected by the two phones. We also performed the same set of actions for the magnetometer x, y and z values. This resulted in six additional graphs as there are two sensors with each having three components to it. We analyzed the spread of the data for each of the six data tables by finding their standard deviation. Once we saw the standard deviations greater than three for the accelerometer data, we were surprised, as that means that our accelerometer data is very spread out and inaccurate. However, after researching a bit more about the phone sensors and the inaccuracy of the accelerometer we decided to analyze the gravitational sensor as well. The gravitational sensor seemed to be a more processed and therefore accurate version of the accelerometer and once we created the “difference” graphs for the gravitational sensor, we could clearly see the gravitational differences were much smaller than the accelerometer differences as there were more points that had zero difference on the graph with the gravity sensor.

Table 3. Standard Deviation Table of the Differences between the Two Phones for the Three Sensors and Their X, Y and Z Axis Values

Sensor	Mean	Standard Deviation
Accelerometer - X (m/s^2)	0.1489	3.560
Accelerometer - Y (m/s^2)	-0.1677	3.290
Accelerometer - Z (m/s^2)	0.2718	3.193
Gravity Sensor - X (m/s^2)	0.3771	0.8091
Gravity Sensor - Y (m/s^2)	-0.1863	0.7193
Gravity Sensor - Z (m/s^2)	0.5159	0.7990
Magnetometer - X (uT)	0.7935	13.70
Magnetometer - Y (uT)	-0.7022	19.36
Magnetometer - Z (uT)	0.5822	13.67

As illustrated in Table 3, the standard deviations for the magnetometers are comparatively high while the gravity sensors’ standard deviations are extremely low and accurate. From our data tables and graphs, we conclude that the phone is accurate, but not accurate enough for robot and software testing. In

robotics, one only needs a few data points to be way off in order to declare the sensors not reliable enough. One way we can increase the accuracy in the future is to disregard the outliers. For example, when running robot or a software and collecting data, we could ignore data values that are over two standard deviations away, or over one standard deviation away, if we want it to be more accurate. However, we will have less data points if we implement a change like this.

3.2 Testing the Decision Making Process of a Robot

The increase in the number of sensors available to the robot and the complexity of the changes in the robot's environment create opportunities to use machine learning to make decisions. Machine learning [1][9] has been shown to perform the job well for human activity recognition. We extend the work to robotics. In FTC, sensor usage scenarios were subject to variation in motions on the field and environment changes. We specifically use machine learning to classify the position of a field element into a certain position out of a given number of possible positions. In using infrared seekers, we found that they can have readings offset by bright lights and variations in the battery level of the target object associated with an infrared beacon. We use a classification machine learning algorithm to smooth out the effects from these sources of variation.

We used one of the machine learning methods called the logistic regression [6]. Our approach, as illustrated in Figure, consists of two phases, the offline training phase and the online scoring phase. With logistic regression, we found it was particularly easy to implement the scoring phase.

Figure 6. Process for creating Logistic Regression Algorithm

In FTC, a robot may be in an unstable position during the autonomous phases if it bumps into other objects on the field. If the robot is in an unstable position, we want to return it to a flat position and regain balance. This involves first classifying the robot's current orientation, and then implementing a correction procedure. Using built-in sensors like the accelerometer and magnetometer, data samples were collected for the robot tilted in various directions: no tilt, tilted forward, backward, left, and right. Sensor data was developed into features and logistic regression was used to classify the orientation of the robot.

We implemented a logistic regression with regularization **Error! Reference source not found.**. Essentially, a sigmoid function is fitted to the data by updating the parameters so that a cost, or error, function is minimized. This predicts the probability that the input is in a certain class. A common problem with Machine Learning Algorithms is that they either suffer from high bias, where the prediction function underfits the data (an example of this would be when a linear function acts as the prediction function when the data is roughly quadratic), or they suffer from high variance, where the prediction function overfits the data (an example for this would be if the prediction function is a fifth-order polynomial, and the data is roughly quadratic). In essence, underfitting models the data too poorly, and is thus unable to generalize to new inputs, and overfitting fits the data too well, and is also unable to generalize to new inputs. These problems can be solved in different ways-- bias can be solved by increasing the number of features, and variance can be solved by reducing the number of features, or collecting more data. In order to avoid overfitting the data, as a preventive measure, we used a method called regularization.

Regularization helps solve the overfitting problem by increasing the value of the cost function when the parameters have large values. This forces the parameters to shrink, and thus reduce complexity- in essence, reducing the number of features. Thus, it minimizes the effects of overfitting and results in an optimal algorithm.

As we plan to use the same algorithm in multiple phones, we conducted the experiments to quantify sensor variations within a phone and across phones. We mimicked the changing of the orientation of the robot by changing the orientation of the phones. Five phones were used to collect data to quantify sensor variation across phones in addition to sensor variation within a phone. Each phone was identical in model, and each phone was tested in the same way. The robot is essentially a rectangular prism. Figure illustrates one of the five orientations of the phone mimicking the robot. We designate five classes corresponding to the normal and four different off-balance positions. The five classes are: Class 1 - Flat (Normal, No Tilt), Class 2 - Tilted Head Up, Class 3 - Tilted Head Down, Class 4 - Tilted Right Up, and Class 5 - Tilted Left Up.

Each phone was placed into these five positions. Data was collected in each of the positions at 10 times a second, leading to over 50 samples for each data set (a data set corresponds to data from one phone, for one orientation) for a large number of built-in sensors on the phone. After data collection, we had 25 data sets. A picture of our testing apparatus for orientation 3 is shown in Figure.

Figure 7. An example of data collection for machine learning. Orientation position 3 is shown.

Before analyzing the algorithm, several features were eliminated, as data from certain sensors was identical between different orientations. This was a rudimentary feature selection, though more complex methods [11] of eliminating and weighting features are available.

Another element of data processing before the machine learning algorithm was the simplification of the remaining features. We recognized that since the phone was stationary for each data set, the values never strayed considerably from their initial value. Thus, we took the initial value from twelve sensors, leading to twelve features. Our reduced data set for one phone looked like this:

Table 4. Reduced data from one phone: initial value for each attribute.

Accelerometer (m/s^2)			Linear Accelerometer (m/s^2)			Magnetometer (uT)			Orientation (°)			Position
X	Y	Z	X	Y	Z	X	Y	Z	Yaw	Pitch	Roll	
0.01	-0.08	9.45	0.00	0.01	-0.35	-36.96	1030.4	-123.48	2.06	0.40	0.02	1
-0.10	1.13	9.39	-0.01	-0.07	-0.38	-43.08	1027.5	-120.96	2.44	-7.00	0.50	2
-0.08	-1.32	9.39	0.02	0.02	-0.36	-26.16	1035.7	-126.66	1.55	8.17	0.58	3
2.05	-0.10	9.25	-0.05	0.05	-0.33	-25.56	1036.7	-123.90	359.95	0.67	-12.27	4
-2.42	-0.09	9.22	0.03	0.03	-0.32	-10.80	1038.6	-124.92	2.33	0.43	14.66	5

The data was then used to train the machine learning algorithm. Data from three phones formed the basis for training, and the remaining 10 datasets were reserved for testing. The selection of the phones was arbitrary. We wanted to ensure that the algorithm would be accurate enough for any phone's data, so fifteen data sets were used for training. The remaining 10 data sets were tested to see if the algorithm

could correctly classify the orientation of the robot. For example, data collected when another phone (a phone that was not used in training) was placed in Orientation 4 is shown below.

Table 5. Data from one Phone, in one Orientation. The values shown represent the first value of each feature, the same as the training data.

Accelerometer (m/s^2)			Linear Accelerometer (m/s^2)			Magnetometer (uT)			Orientation (°)		
X	Y	Z	X	Y	Z	X	Y	Z	Yaw	Pitch	Roll
2.242	0.185	9.474	-0.021	0.004	-0.066	9.853	-12.367	-53.473	242.440	-1.016	-13.320

The algorithm finds the function that outputs the highest probability for the data and predicts that the robot is in that class. Here is a table that shows the actual orientation of the phone, and the predicted orientation. As shown below, the algorithm was 100% accurate.

Table 6. This is a "Confusion Matrix", which shows the actual orientation of the phone, and then the predicted orientation. Values outside the diagonal represent errors, but as can be seen below, since all values are in the diagonal, the algorithm was 100% accurate for the ten testing examples.

		Predicted Class				
		Orientation 1	Orientation 2	Orientation 3	Orientation 4	Orientation 5
Actual Class	Orientation 1	2	0	0	0	0
	Orientation 2	0	2	0	0	0
	Orientation 3	0	0	2	0	0
	Orientation 4	0	0	0	2	0
	Orientation 5	0	0	0	0	2

After testing the algorithm on data from other phones, a quick review of the prediction functions and the original data sets revealed that the data sets were similar for a large number of sensors, and there was no appreciable difference in the readings of the sensors except for a select few sensors. Thus, the majority of features were not very useful. In future applications of this algorithm, the number of features will probably be reduced to minimize overfitting, especially since only a small amount of features is necessary for accurate classification.

The main challenge with this application of machine learning was the similarity of the data sets, which led to the outputs of the prediction functions being very similar between classes. For example, when a data set for position 4 was tested, the algorithm gave a high probability of the robot being in either position 3 or 4, as these positions differ largely only in the readings for a few sensors. Ultimately, the probability of being in position 4 was higher, so the algorithm could still correctly classify the test data. As mentioned above, further iterations will either eliminate some features or develop a way to weight the importance of features.

Further action would involve more advanced feature selection in our machine learning algorithms, along with the implementation of different statistical methods to experiment with eliminating overfitting. Other applications of machine learning include the analysis of the tilt of the robot to determine its exact orientation in space-- in other words, to determine its pitch and roll (which would both be zero if the robot were flat). This would act as an extension of the above application of machine learning. Another machine learning application would be in image processing, an area that we have researched before, but which we have yet to apply machine learning to.

In conclusion, the decision making of the robot can be improved by using machine learning algorithms, which allows the robot to learn from past decisions and make informed and accurate decisions in a

variety of situations. This decision-making process is important to test in order to ensure high performance when the robot runs autonomously.

4 Summary

Our approach to brewing quality in Android robots that integrates robot testing processes. Case studies were used to examine quality in Android devices in the context of the FIRST Tech Challenge robot competitions and develop a testing taxonomy. Internal sensors within Android phones including the accelerometer and magnetometer and their data were analyzed and collected under controlled conditions such as precise turns. With each movement, a theoretical model of the expected sensor output was generated based off of the characteristics of motion. For circular motion, this model was a trigonometric sine wave for both the magnetometer and the accelerometer. These motions were carried out by a robot in order to ensure reliability. It was found that by comparing a regression of the data (in the form of the mathematical model) and the data, the precision of the sensor data could be quantified. In addition to controlled motion, arbitrary motion was conducted in order to test reliability of sensor readings across phones (and sensors). By applying these concepts to the reliability of a robot, the consistency of performance was determined by utilizing the established testing taxonomy. Finally, the decision making of the robot that allows it to function autonomously was refined by employing machine learning algorithms. Statistical methods were employed in order to ensure that prediction algorithms could accurately make decisions based on prior data. By testing in many different scenarios, we ensured that our robot wouldl be able to use an accumulation of data from past scenarios to adapt to new and unexpected situations. All of these methods can be extended to industries that require the usage of sensors with minimal adjustments.

The contributions of this paper are:

- Established a taxonomy of robot testing processes
- Developed an approach to integrate robot testing processes with regard to inherent sensor, robot, and robot decision reliability.
- Demonstrated a prototype in FIRST Tech Challenge robot competition

Acknowledgments

The authors wish to express their gratitude to their reviewers, Aaron Hockley and Moss Drake, whose assistance and feedback have been invaluable.

References

[1] http://www.usfirst.org/
[2] M. Cecilia Martínez, Marcos J. Gómez and Luciana Benotti. A Comparison of Preschool and Elementary School Children Learning Computer Science Concepts through a Multilanguage Robot
[3] Vicki Niu, Ethan Takla, Ida Chow, MacLean Freed, Jeffery Wang and Kingsum Chow. Engineering Quality in the Robotic World. Proceedings of the Pacific Northwest Software Quality Conference, 2012. Portland, Oregon, USA.
[4] Kingsum Chow, Ida Chow, Vicki Niu, Ethan Takla and Danny Brillhart. Software Quality Assurance in the Physical World. Proceedings of the Pacific Northwest Software Quality Conference. 2010. Portland, Oregon. USA.
[5] Phelim Dowling and Kevin McGrath. Using Free and Open Source Tools to Manage Software Quality - An agile process implementation. ACM Queue April 2015.
[6] http://developer.Android.com/guide/topics/sensors/index.html
[7] http://www.lego.com/en-us/mindstorms/about-ev3
[8] Pierluigi Casale, Oriol Pujol, and Petia Radeva. “Human Activity Recognition from Accelerometer Data Using a Wearable Device”. Pattern Recognition and Image Analysis, Lecture Notes in Computer Science, 2011, pp 289-296 http://dx.doi.org/10.1007/978-3-642-21257-4_36

[9] Mannini, A.; Sabatini, A.M. Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers. *Sensors* **2010**, *10*, 1154-1175.
[10] http://www.holehouse.org/mlclass/06_Logistic_Regression.html
[11] http://homes.soic.indiana.edu/natarasr/Courses/AML/Readings/FeatureSelection.pdf
[12] http://qwone.com/~jason/writing/lr.pdf

Challenges in Testing an Intelligent Software

Anurag Sharma, Satish Yogachar

(anurag.sharma@intel.com, satish.yogachar@intel.com)

Abstract

Artificial Intelligence is always a fascinating field. An intelligent system can perceive its environment, learn, adapt and take action to maximize the success. AI has always been considered a highly specialized field, mostly limited to research labs. Researchers are using various techniques such as Genetic Algorithms, Fuzzy Logic Approach and Data mining etc. to build problem solving, logical deduction and reasoning capabilities into machines, with limited success. On the other hand, movies such as Terminator (1984), Artificial Intelligence (2001), Matrix (1999) and Her (2013) etc. have stretched our imagination and fascinated us with possibilities of Artificial Intelligence.

Deep Blue chess playing system, IBM's Watson question answering system and more recently, intelligent person assistants such as Siri and Cortana are more close to an intelligent software. When, more and more intelligent applications become mainstream, the traditional software verification principles and practices will require a paradigm shift. Verification of these applications will present new and unique challenges. For example, a simple test case which is expected to return "FALSE" may work correctly today but may return "TRUE" tomorrow, because application learned and started behaving differently. Thus, tests will not be repeatable. Similarly, we may need to think differently while developing a regression test suite and designing automation framework. It may even lead to the test system being an intelligent system. When such intelligent systems integrate with intelligent sensors and/or web as a part of intelligent network, the testing becomes even more complex.

This paper is a commentary on testing challenges which we may face in verifying intelligent software. The purpose is to stimulate discussion on how traditional testing concepts may evolve to cater to intelligent software verification, without going into technicalities of artificial intelligence techniques such as neural networks, machine learning etc.

Biography

Satish H Yogachar *is a Senior Principal Engineer at Intel Security located in Bangalore. He has over 11 years of experience in Software product development. Prior to Intel Security, Satish was an Architect in product development in BFS domain at Tata consultancy Services and Symphony Teleca Corp. Satish holds Bachelor's degree in Computer Science Engineering.*

Anurag Sharma *is a Senior Principal SDET at Intel Security, Bangalore. He has over 9 years of experience in embedded as well as application software verification. Prior to McAfee, Anurag lead the Software Common Components verification team in SWCOE domain at Honeywell. Anurag holds Bachelor's degree in Electronics and Communication Engineering and MBA with specialization in Software Project Management.*

1 Introduction

As a general definition, an Intelligent System is a machine with an embedded Internet-connected device that has the capacity to gather and analyze data and communicate with other systems.

Intelligent systems exist all around us in point-of-sale (POS) terminals, digital televisions, traffic lights, smart meters, automobiles, digital signage and airplane controls, among a great number of other possibilities. Home Automation [1] Solutions such as Google Nest; Internet of Things (IoT) network [2]; Smartphones as well as wearable devices such as Jawbone, Fitbit etc. are all part of this intelligent ecosystem.

The Turing test [3] is a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human. Today's intelligent systems may not pass the Turing Test, but these systems demonstrate intelligent behavior close to humanness in certain domains and will certainly get through the Minimum Intelligent Signal Test (MIST) [4], a less strict variation of Turing Test.

Throughout this paper, the terms "Intelligent System" and "Intelligent Software" have been used interchangeably, because, the word "System" is more relevant when referring to whole ecosystem of intelligent software. Still, the focus of this paper is on software testing aspects. "Conventional Software" terminology is used to describe the traditional software applications such as payroll processing software.

The content of this paper are organized as follows:

- Section 2 describes how testing an intelligent software is different compared to conventional software testing and challenges faced in doing so.
- Section 3 describes some approaches for verifying an intelligent software.

2 Why testing intelligent software is challenging?

An intelligent system is a combination of complex machine learning algorithms, contextual understanding, natural language processing and deep database represented in the form of knowledge graphs. Generally, an intelligent system initially boots up with a set of complex software algorithms and a limited input set. At this time, the capabilities of this system are limited. The longer this system is in service, more it learns about the user preferences, user behavior etc. based on user input patterns and other external stimuli: let's call it the *Learning Phase*. Thus, the database and knowledge graph of the system expands and it gains predictive capabilities, *the Analysis or Testing Phase*.

Let us consider, Google Now on Tap, an intelligent virtual assistant, as an example. When installed, it presents a few basic questions about the user like, home and office locations, hobbies and interest, etc. and the information provided by it is limited to route map and expected time of arrival at destination. Over the period of time, based on the interaction with the user, context of information and other linked google apps, the Google's knowledge graph about the user expands. Thus, it becomes more and more intelligent, tracking parcels for you, suggesting movies to watch, suggesting restaurants to order food and even making reservations by reading text messages from your wife, etc.

Testing an intelligent software is quite different compared to a conventional software. The conventional software testing principles and practices that we have known and understood for decades do not directly apply while verifying an intelligent software. Also, various software testing methods such as manual testing, white-box testing, automation, usability testing, etc. are either not applicable or have to be performed differently for an intelligent software. Each intelligent system offers unique testing challenges. Some of those are describe in the paragraphs below.

2.1 Psychology of Software Testing

Human beings are highly goal-oriented, therefore, setting the correct goal post has a psychological effect. Conventional Software Testing is the process of executing a program with the intent of finding errors. Testers select the test data which has high probability of finding errors. Equivalence class partitioning, boundary value analysis, etc. are used in designing the positive and negative test cases.

While testing an intelligent software, the intent is to demonstrate the learning ability of the system, to understand how the algorithm behaves, besides testing it. The test data and test cases designed are incremental in nature, such that the learning capability or pattern of the software is understood and verified. There is an implicit expectation that the software may legitimately behave differently (output) in each iteration.

It's an important distinction and it also implies that conventional testing is a destructive process while intelligent system testing has signs of a constructive process. This difference also has an impact on how test cases and test data are designed and how to test the program.

2.2 Test Case Design

Designing test cases for an intelligent software is difficult and more challenging in many aspects:

1. Usually, when input or output equivalence classes are applied to developing test cases for a conventional software application, the expected output for a given input is known in advance. Thus, it is pretty easy to design a test suite around it. For example, an MD5 secure hash algorithm [5] always produce the same 128 bit hash value for the same input. On the other hand, for an intelligent learning software, the expected output for a given input may not be known in advance. It depends on situational factors such as the context of the software at that point of time, the output from the previous tests and on the time when test is performed. For example, "Google Now on Tap" virtual assistant will produce different output for the same question "Who is the singer?" when different song is playing. Similarly, the answer to the question, "Who is the president of United States?" is different if the question is asked now or back in year 2007. Therefore, designing repeatable tests for an intelligent software is more challenging.
2. No reliable "test oracle" (or test suite) can be designed to indicate the correct output for any arbitrary input, therefore, it is challenging to detect subtle bugs. Although the repository of data-sets created over time can be used during regression runs to compare the result quality (code changes are not degrading the quality of output) but not for testing in strict sense.
3. As we know, testing a conventional software with all possible inputs is not always practical. For example, for a simple algorithm of identifying if a number is prime or not, the valid input dataset is infinite. Instead, by following equivalence class partitioning and boundary value analysis, a reasonable test suite capable of detecting any error in the algorithm can be designed. However, for a very basic intelligent software such as the SoundHound app (natural language processing engine), which detects the song currently playing, this task can be daunting.

2.3 Test Outcome

1. Conventional Software Testing is focused on the final output of execution. For an intelligent software testing, even though we may or may not know what final output should be, inspection of intermediate results with respect to an algorithm may reveal bugs. For example, Siri virtual assistant will not be able to provide answer to every question (perfectly) in near future, but, understanding how Siri software is breaking down the spoken question into chunks, how it is creating the knowledge graph etc. is an important step for fine tuning the software.
2. For an intelligent software, classification of output in a binary sense ("Pass" or "Fail") is not sufficient because given enough time and with a large number of executions, the probability is that every test will pass (*or fail, depending on how test vector is designed*). Therefore, the distinction between a positive or negative (boundary value analysis) test cases changes over a

period of time. Probably, a “Partial Pass” or quality of output may also need to be considered (See section 3.3).

3. Another major challenge is to figure out what test cases have best chance of identifying the errors. While performing tests for a conventional software, the requirements are generally well defined by the time software is available for formal QA validation. Therefore, the test cases with the best chance of revealing bugs, can be designed based on the requirements themselves. For a learning software, the intent is to identify the bugs in the specification, even before reaching implementation. Therefore, during the learning phase, the test data is designed carefully after analyzing the implemented algorithm such that the output can be predicted with high confidence. The expected output during execution of the algorithm ensures that the implementation follows the algorithm correctly, and any deviation can be due to the incomplete/incorrect specification of the real-world problem.
4. Most of the time, when interacting with an intelligent software, a user doesn’t have time to scan through all the available output results, therefore test scenarios should be designed to ensure that the best answer is reported (something like, Google “I am feeling lucky” answer [6]). For example, while driving you ask “OK Google, what is the route to reach A from my current location?” Then you expect that google will provide the best route (shortest distance, less traffic etc.) out of all available routes.
5. Results of an intelligent software testing are open to interpretation because of the way the requirements are defined, the number of features offered, how the tests are conducted and what dataset was used.
6. For conventional software, desktop as well as web applications, the minimum quality metrics in terms of response time, memory footprint, etc. are well established. There are standard tools available to measure these quality factors. Intelligent software are still evolving, therefore, no such standards are in place. Therefore, the results of any such tests are more prone to being biased.

2.4 Security and Privacy

In conventional software testing, the software has well defined entry points and user inputs. Most software uses industry standard cryptography algorithms for user authentication, confidentiality, encryption and integrity of data in transit/at rest. The security testing follows standard processes of vulnerability scans using standard tools, threat modelling, security assessment, security audit and security review. Privacy testing includes review of user data storage/retrieval methods (credit card information, user passwords etc.).

For intelligent software flooding the market today are built on ideas such as:

1. Proactively provide information, even before user asked for it (Google Now, Microsoft Cortana), or
2. Make tasks easier by integrating/eliminating steps from the process (For example, Apple Pay, Google Wallet for quick checkout at the store), or
3. Recommend things to buy based on previous search history, or
4. Connect the dumb devices sitting around home (like coffee maker, refrigerator, TV, locks etc.) with the internet/mobile so that these devices can do things automatically according to user behavior (Internet of Things).

To make it possible, these software apps are continuously connected to Internet and upload data to the cloud. Moreover, they collect much more data about users such as location (GPS), social friends’ contact information (Facebook contacts, LinkedIn contacts etc.), addresses, credit card information, previous shopping lists, access to e-mails, messages. It is easier for hackers to steal data or to commit fraud online by exploiting these vulnerabilities or through social engineering. Users are also becoming aware about the security risks. A recent backlash against Facebook’s privacy policies is one such example [8].

Today, with millions of internet connected devices which continuously uploading user data to cloud; traditional firewalls at the datacenters are not enough. Application design needs to consider data security

and user privacy upfront, not a side activity. Similarly, test plans should emphasize security testing as much as functional testing.

2.5 Automation

Automation designed for an intelligent software should also be intelligent in certain aspects.

1. It should be capable of not only generating test cases to cover branches and paths, but also capable of understanding the relationships between the inputs and outputs so as to simulate real world behavior.
2. It can simulate the large number of external inputs such as user actions, GPS location, network, etc.
3. Intelligent applications are inherently non-deterministic in nature, similar to us, the real people. While making decisions, we correlate various stimuli, behave accordingly. Therefore, to test intelligent applications, automation should be able to generate various sequences and permutations of inputs and to randomize the inputs, so that software correctness can be investigated in more detail.

2.6 Programming Languages and Test Tools

Intelligent software developers prefer specialized languages for development [9], which are strong in mathematical notation of computer programs, declarative programing, symbolic reasoning and language parsing, for example, Lisp, Prolog, Haskell, and MATLAB, etc. Experienced test engineers for these languages are in short supply.

Available test tools do not exactly apply when testing intelligent software. Conventional tools such as memory leak analysis tools, static analysis tools, unit test tools, etc. may be applied partially. But, most of the tools or scripts such as test data generators, tools to compare output data models, etc. need to be designed specific to each application under test.

2.7 Localization

Localization refers to the actual adaptation of the product for a specific market. It involves language translation, adapting graphics, adopting local currencies, using proper format for date and time, addresses, and phone numbers applicable to the location, the choices of colors, and many other details, including rethinking the physical structure of a product. All these changes aim to recognize local sensitivities, avoid conflict with local culture, customs and common habits. User facing portions of the software such as log messages, the user interface, etc. are candidates of localization. Localizing any product and its testing is a highly technical process. When it comes to intelligent software, especially the virtual assistants on mobiles (i.e. Apple Siri) which rely on spoken or typed natural language processing, it is still more challenging.

3 Test Approaches

Few approaches for testing intelligent software:

3.1 Testing by End User

Beta testing is very popular in the application software world. For example, beta versions of new operating system versions are released periodically for Microsoft Windows and Apple OS. Early adopters and developers, even enterprises share their feedback and bugs to Microsoft, which ultimately boils into the final release to general public. This approach is most suitable for intelligent software testing where technological boundaries are being pushed.

Testing by end users helps in understanding the use cases of new breakthrough innovations being developed and to generate curiosity among general public. Google Glass [10] Explorer program is a prime example of it. Google Glass is nowhere near the final product, but at 1500 dollars apiece, it was given to technology experts for obtaining the reviews, understanding the use cases and to acclimate the world with this new way of interacting with this system.

Testing by end users is also adopted for scenarios where it is not feasible for the developers to perform exhaustive testing because of infinite number of combinations. For example, you cannot test Google Maps, "Get Directions" feature for every possible route in the world (and where each input source and destination combination has multiple route options). Therefore, once it is tested with a very large sample set against a reference model and all obvious issues are fixed, it is rolled out to users for User Acceptance Testing.

3.2 Equivalence Classes of Test Design

While designing the test datasets, the software requirements and functionality of the real world application (to be emulated by intelligent software) should be kept in mind. Partitioning the input data into various classes helps in focusing testing and is also useful for automating the tests. For example, test datasets can be divided into:

1. Predictable Output Behavior vs. Unpredictable Output Behavior datasets. Predictable output behavior is where the answer is always known beforehand such as a mathematical calculation or a well-known fact such as "*Who was the 16th President of United States?*" On the other hand, unpredictable behavior is where the output cannot be decided on previously known information and requires modelling and analysis of facts and figures such as "*Who will win the 2018 FIFA World Cup?*" A predictable behavior test dataset covers the natural language processing and search capabilities of the software, while an unpredictable behavior dataset verifies the mathematical processing capabilities of the intelligent software.
2. Incremental Learning Behavior vs. Random Input dataset – A test dataset designed to verify the incremental learning behavior of the software takes into account recent history to make its predictions. A good example of an incremental learning software is the weather forecast; if it has been sunny and 80 degrees for the last 2-3 days, it is a pretty reasonable prediction that it is not going to snow tomorrow. In this particular case, the underlying data source is not changing. Another example is the sales model of a retail chain where the predictive model has to evolve as the company grows. For an incremental learning software, Data Horizon (how quickly new input data should be considered) and Data Obsolescence (when old data is still relevant for prediction) are important factors, which should be kept in mind while designing the test dataset. On the other hand, a test dataset designed to verify a computation search engine "Wolfram Alpha" [11] will cover a broad range of inputs covering various topics such as statistics, astronomy, mathematics, finance, etc.

3.3 Ranking the Output

As described in section 2.3, the output of a test cannot always be a "Pass" or a "Fail". Therefore, a ranking system should be developed for ascertaining the quality of the output. Most appropriate results can be ranked higher, while the most irrelevant results can be ranked lower. The testing should verify that the best result based on this ranking is presented to user.

For example, Google Maps may suggest the subway is the best method (fastest) to commute from point A to B in New York City, while in any other city, a cab/taxi may be ranked higher. Similarly, Google maps updates the best route ranking using of external factors such as diversion due to construction work or traffic situations. The reference model and the sample test dataset designed for Google Maps' "Get Directions" feature should handle and verify the correctness of this ranked output.

3.4 Comparison Test

One of the most common approaches to evaluate two or more intelligent software apps is by pitting them against one another. For example, tests such as these have been conducted to find out which of the iOS and Android personal assistants (namely, Siri and Google Now), is better [12]. This can be a very effective test when the test dataset is very large and different areas of software features are covered.

4 Conclusion

This paper is an attempt to demonstrate how traditional software testing is becoming more challenging with the increasing sophisticated software applications. Increasing sophistication demands rigorous software development and testing. Software testing has to evolve beyond just functional testing and cover other aspects such as security, user behavior etc.

In a thought provoking video “Humans Need Not Apply” [13], CGP Grey argued that every profession that relies on creativity, decision-making, and pattern-recognition could eventually get replaced by Artificial Intelligence. However, we firmly believe that the future of software testing is secure and brighter since more complex automation (Artificial Intelligence) doesn’t cause automation, it causes more testing (Human Intelligence) needs.

References

1. Wikipedia article on “Home Automation”, [Online] https://en.wikipedia.org/wiki/Home_automation (accessed July 5, 2015)
2. Wikipedia article on “Internet of Things”, [Online] https://en.wikipedia.org/wiki/Home_automation (accessed July 5, 2015)
3. Wikipedia article on “Turing Test”, [Online] https://en.wikipedia.org/wiki/Turing_test (accessed July 5, 2015)
4. Wikipedia article on “Minimum Intelligent Signal Test”, [Online] https://en.wikipedia.org/wiki/Minimum_intelligent_signal_test (accessed July 5, 2015)
5. Wikipedia article on “MD5”, [Online] https://en.wikipedia.org/wiki/MD5 (accessed July 14, 2015)
6. Wikipedia article on “Google Search”, [Online] https://en.wikipedia.org/wiki/Google_Search (accessed July 18, 2015)
7. “An Approach to Software Testing of Machine Learning Applications” by Christian Murphy, Gail Kaiser and Marta Arias [Online] Available at http://www.psl.cs.columbia.edu/publications/pubs/Murphy-SEKE2007.pdf
8. “Facebook’s new privacy policies and your data security”, Avast blog entry posted on December 11, 2014, https://blog.avast.com/2014/12/11/facebooks-new-privacy-policies-and-your-data-security/ (accessed July 5, 2015)
9. Wikipedia article on “List of programming languages for artificial intelligence”, [Online] https://en.wikipedia.org/wiki/List_of_programming_languages_for_artificial_intelligence (accessed July 5, 2015)
10. “Project Glass: Live Demo At Google I/O”, https://www.youtube.com/watch?v=D7TB8b2t3QE (accessed July 5, 2015)
11. Wolfram Alpha computation search engine [Online] - http://www.wolframalpha.com/
12. “Next-gen Siri versus Google Now on Tap: The battle to read your mind best”, CNET blog entry posted on June 9, 2015, http://www.cnet.com/uk/news/proactive-siri-versus-google-now-on-tap-compared/ (accessed July 5, 2015)
13. YouTube video “Humans Need Not Apply” [Online] https://www.youtube.com/watch?v=7Pq-S557XQU (accessed July 5, 2015)
14. “Will Software Testing Ever Succumb to Artificial Intelligence?”, posted on November 22, 2014, http://www.testuff.com/blog/will-software-testing-ever-succumb-to-artificial-intelligence/