Testing Circus

Volume 7 - Edition 3 - March 2016

Magazine for Software Testers

Interview with Matt Heusser

www.TestingCircus.com

serious about software quality

At Doran Jones, our mission is to help technology organizations improve their ability to deliver software and add business value. We believe the best way to do this is through hands-on delivery, working alongside our clients. Let us show you life at the intersection of talent and opportunity.

- **Software Development**
- **Software Testing**
- **Training and Coaching**
- **Recruitment**
- **Urban Onshore Outsourcing**

DJ Doran Jones

as seen in: WIRED

www.doranjones.com

Testing Circus

Volume 7 - Edition 3 - March 2016

Table of Contents

Testing Circus Team

Founder & Editor – Ajoy Kumar Singha

Team -

- Srinivas Kadiyala
- Sanath Kumar
- Dwarika Dhish Mishra
- Pankaj Sharma
- Jaijeet Pandey
- Vikas Kumar

Editorial Enquiries: team@testingcircus.com
Article Submision: article@testingcircus.com

Testing Circus India
Chaturbhuj Niwas, 1st Floor,
Sector 17C, Shukrali,
Gurgaon - 122001
India.

Edition Number : 66 (since September 2010)

*On the Cover Page - Matt Heusser

From the Keyboard of Editor

On the eve of April Fool's Day, we bought a new domain ThisWebsiteIsHacked.com, created a two page website which looked like it was hacked by some anonymous group of hackers. Then we added a temporary redirect instruction to our domain TestingCircus.com and forwarded to the new domain. So whoever visited the site, they will be automatically redirected to the new website. Everyone found the website was hacked. We had included a link where web-admin can find out how to get the hacked website back. There, we revealed it was an April Fool prank. Here are some of the facts about this prank.

It was a simple domain redirect hack. This is possible when hackers break into your domain registrar account and add unauthorized information to your domain's records. So, keep your domain information safe. Use domain Whois guard/privacy protection and don't expose your actual email id associated with the domain name. Use two factor authentication where available.

We also found people retweet, comment, send private messages without verifying information first hand. Many of our fans thought we were actually hacked and offered to help and also recommended some security experts for help. Few offered ideas to reset the configs to get back the website from the hackers.

We would love our readers and social media fans to react only after visiting the actual link and finding out the truth. Else, we know this is how rumors are spread and half baked opinions are formed.

We are also happy that, in case we ever get hacked, we will have enough help from our network to recover the site quickly. Thank you all who offered to help us. Thanks to the readers who found out the prank and helped others fall for it. We had fun.

In this edition we have an interview with Matt Heusser and also some valuable articles for testers from Ravi, Michael Larsen, Raj Subramanian, Arslan Ali, Jari Laakso and Brendan Connolly. More news next edition. Happy testing!

- Ajoy Kumar Singha

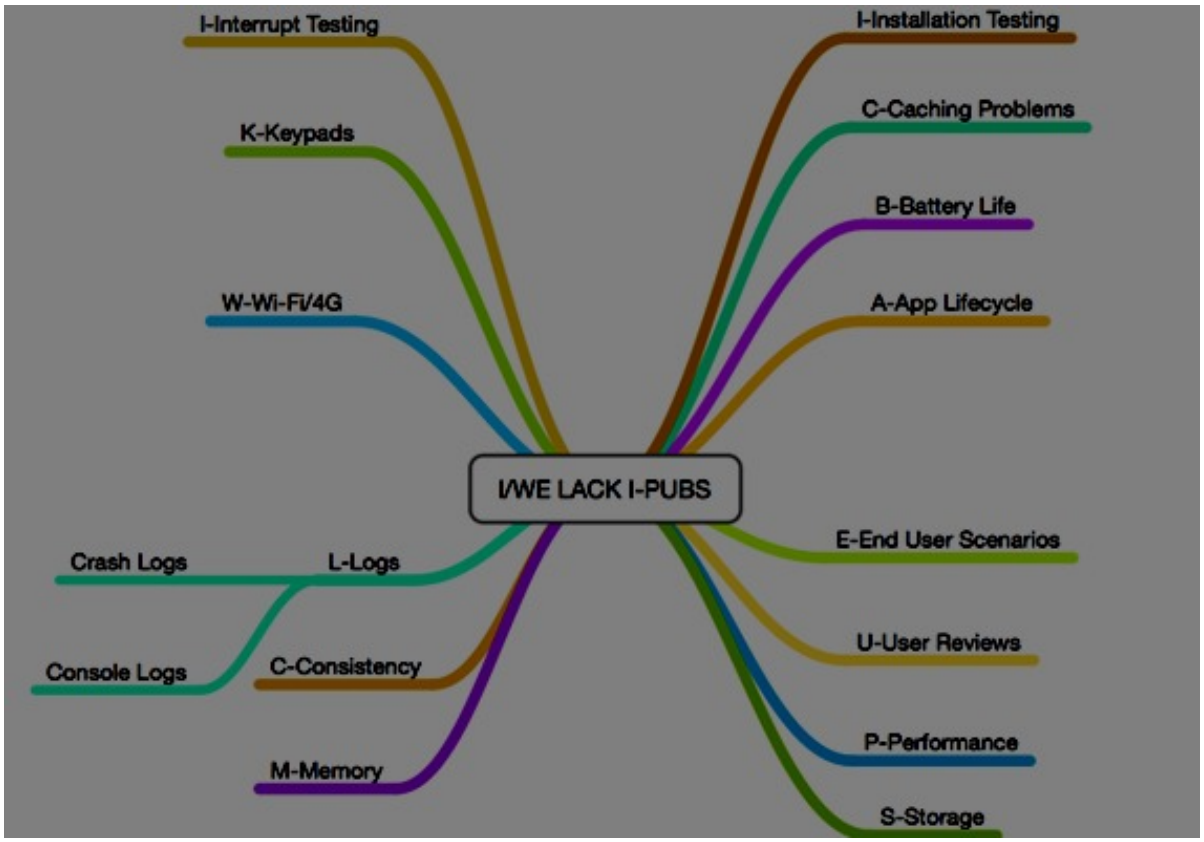
@TestingCircus // @AjoySingha

Feedback please! team@testingcircus.com

Mobile Testing Cheat Sheet – Taking Quick Tours on your Mobile Application

- Raj Subramanian

During my career in the mobile testing field, I've come across numerous bugs related to mobile native applications. Looking at these bugs, I started categorizing them, and have since come up with this mind map. This mind map helps to do a quick tour of your mobile application and find vulnerabilities as quickly as possible. This could be used for doing smoke testing, acceptance testing or even production testing after you application is live on the different App stores.



I came up with this Pneumonic -**I/WE LACK I-PUBS** to help testers get quick feedback about their application.

Each attribute is described below-

I – INTERRUPT TESTING

We often fail to realize how mobile applications would be used by an end user. Testers often just stick to testing their mobile application without considering the fact that it is just one of the many applications an end user would use on a daily basis. Most users don't use an application until done with it, and close it, before opening another application. It is really important to mimic real life scenarios, and observe the mobile application behavior when, as the user:

- I send or receive text messages
- Someone calls me, or I make a call while using the application
- I get notifications or updates from other apps while using your application
- I open a mobile browser and then come back to your mobile application

W – Wi-Fi/4G

Imagine a typical morning. You wake up, and of course the first thing you do is check Facebook in bed. Now, you are on your home Wi-Fi network. Then, you get

ready for work and get out of your house to get in your car, Now, the connection switches from Wi-Fi to 4G. When you get to work, you get out of your car and enter the building, and the connection switches from 4G to your company Wi-Fi network. You take the elevator to your floor and inside the elevator you do not get any signal at all. Now there is no network connection. Finally, you get out of the elevator and go to your desk, reconnecting to the company Wi-Fi.

Almost everyone has a regular routine similar to this scenario. In our typical morning, the mobile connection changed 5 times! Therefore, it is important to see how your mobile application reacts when switching between different connections.

E – End user scenarios

It is really important to think like an end user while testing your application. Try switching between applications, locking your phone, unlocking your phone, putting it in airplane mode, turning on/off bluetooth and so on. Think like a user and put yourself into their shoes. Think about scenarios which the real user would face when using your mobile application.

You may also want to create different user "personas" such as a non-technical user (someone who has little smartphone experience); a power user (who switches between apps consistently and quickly) and impaired users (those who may have bad or no eyesight, or those who have sharp hand movements).

L – Logs

It bothers me that there are still testers out there who do not realize the value of log files. These files usually have bugs in them, hidden gems that are waiting to be discovered. While testing our mobile application, note that there are 2 types of logs:

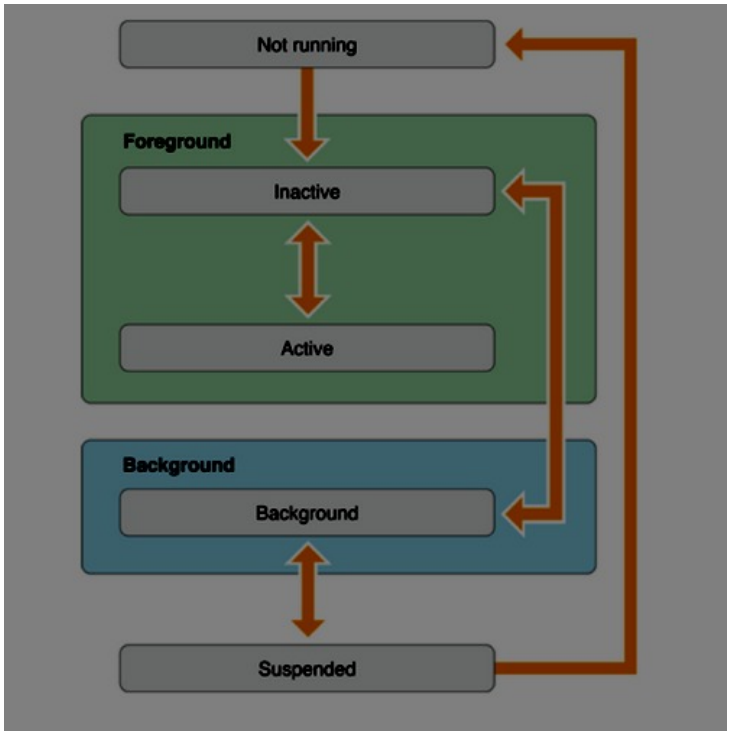
- Crash Logs
- Console Logs

Crash logs are a great way to find out what may be causing your application to shut down unexpectedly, whether user input, device issue, connection issues, memory leaks, or code related issues. For example, there are logs which are part of your IDE's (Integrated Development Environment) like Xcode and Android Studio where you can find information about the different kinds of crashes that happen while using your mobile app.

Console logs, while often including information that was in the crash log, also contains kernel-specific information related to the specific device and OS and how they work with your application. For example, in the Mac, *Applications->Utilities->Console* is where we can find different information about the mobile app when the phone is being used while connected to a computer.

A-Application Lifecycle

It is essential to understand the different states the mobile application goes through while using the app – *The Application Lifecycle*



The above diagram illustrates the different states of a mobile application. Consider how the application behaves in the four states:

- Foreground
- Background
- Suspended
- Idle

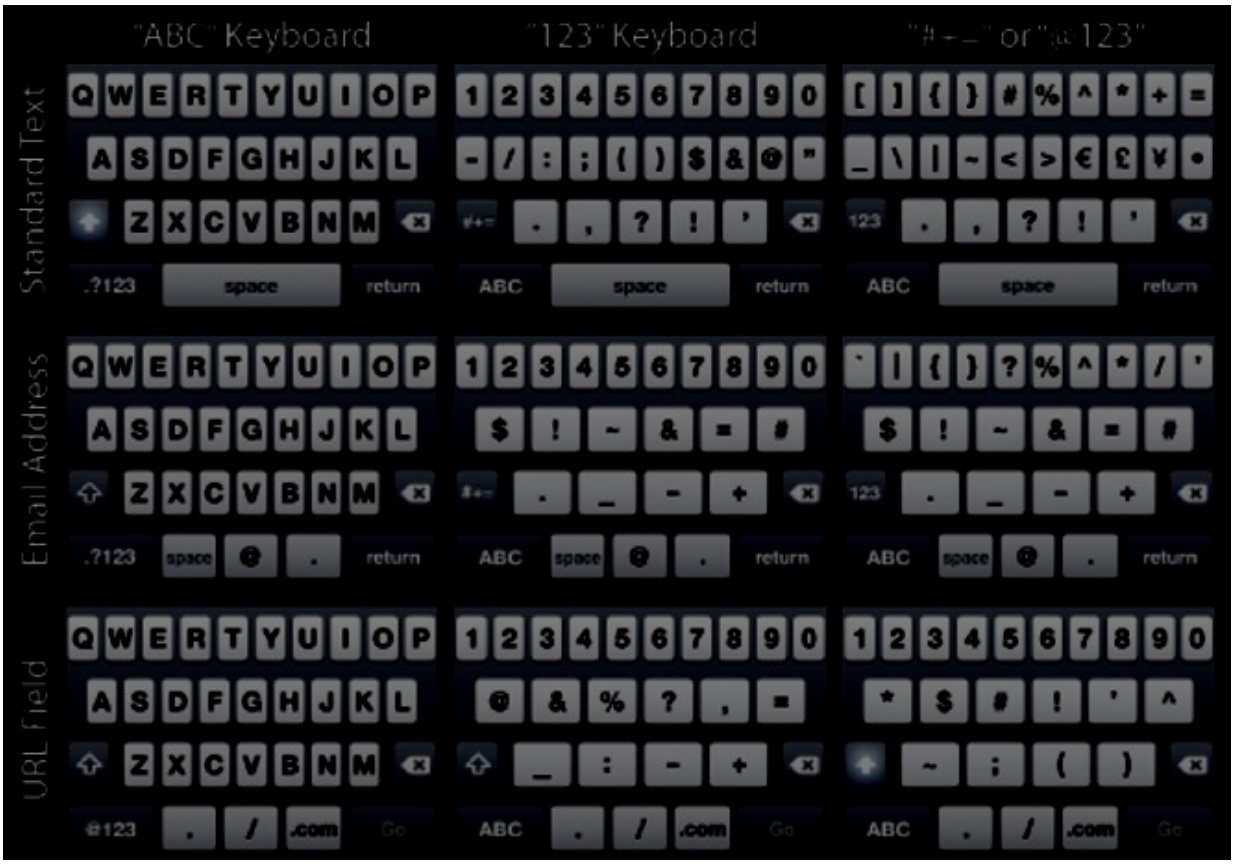
C – Consistency

Companies often have their mobile applications running across different devices and OS's. It is important that the application is consistent across devices and OS's, as the user should get a seamless and consistent experience while using your app. The app options/logos should not be different in different devices unless there is valid reason for doing so. Users should not need to learn an entirely different app every

time they switch devices.

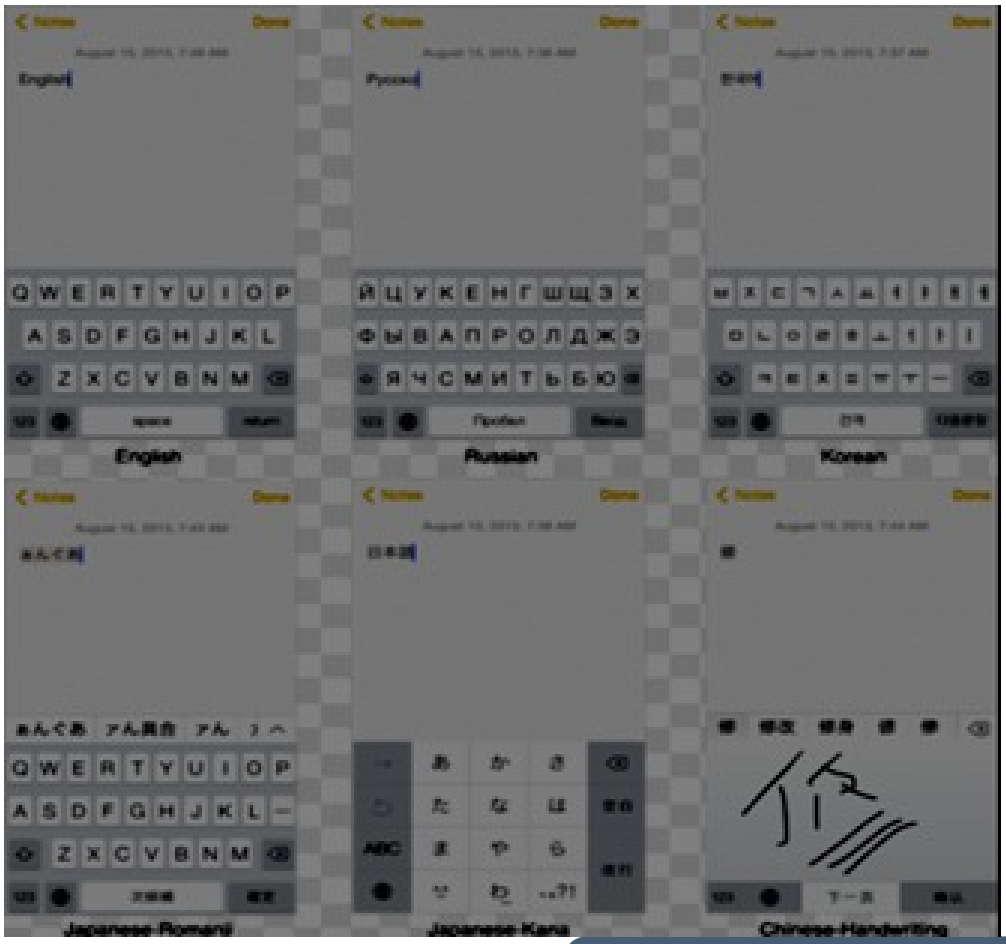
K- Keypads

This is the most overlooked area of testing; one that often breaks applications, and testers don't even know to check it.



You subconsciously know there are different keypads; you see them when entering email address, creating emoticons, and dialing phone numbers. But do you actively test all of them?

Apart from the above keypads, you have keypads supporting different languages as shown below.



Consider these variables when testing, as not everyone may have the standard English keyboard.

U – User Reviews

User reviews can make companies go bankrupt. The review score your app has is a critical aspect of your company's mobile application, such as those reviews on the App Store or Google Play. This is often a factor when users have to choose between similar apps. Pay attention to what users are telling you about your app; looking through the user feedback, you can find lot of issues/ improvements to be made on your app.

I – Installation Testing

Have you ever tried to:

- Delete the app and install a new version of your app

- Delete the app, uninstall the app, and install the app again
- Install the app over an existing version of your app
- Install the app over Wi-Fi/4G/Bluetooth/USB

By doing the mentioned actions, you can find many vulnerabilities in your mobile application. These actions are what the end user is going to do when they get their hands on your app.

P – Performance

People try to load a number of features into their mobile apps to help the user, but they fail to realize that it can come at a cost – performance.

Possible factors which could affect performance are

- Use of dynamic Images
- Use of maps
- Web service calls
- Client side validations

There are free tools for mobile app performance testing, including *"Instruments"* (part of the Xcode IDE) and *"adb tool"* (part of Android Studio and IntelliJ.)

B – Battery Life

No one likes a battery-sucking vampire app. Pay attention to different daemon processes that may run because of your app. Other aspects which could consume battery life include not clearing idle sessions and the use of dynamic visual components.

S – Storage

Years ago, when 8GB phones first came out, users had to make tough decisions regarding the number of photos on a phone vs. which apps to download, due to the limited memory. Although many devices have larger storage now, users may still be hesitant to download an app due to the size of it.

Compare the size of your app to those of your competitors'. This is an easy way to see whether your app is a bloatware or not.

Those are the main issues with mobile native apps, remembered easily with the pnemonic I/WE LACK I-PUBS. (Just think about how we don't have any digital bars yet, and you'll remember all the tips I gave you!)

About the Author

Raj Subramanian, a former developer for a payroll processing company, moved to testing to focus on his passion. Raj currently works as a mobile test lead for a large product and services company. He actively contributes to the testing community by speaking at conferences, writing articles, blogging, and being directly involved in various testing-related activities. He currently resides in Chicago and can be reached at raj@rajsubra.com. His website is www.rajsubra.com and twitter handle - @epsilon11.

Can We Teach Curiosity?

- Michael Larsen

For the past few years, a comment has held true for me, yet I have also wondered if there was a way that we could potentially prove it wrong. That comment is one I saw Jon Bach write a number of years ago:

"As a trainer, manager, and coach, I had fun teaching technical skill and product domain knowledge. But what I CAN'T train is curiosity. I cannot train someone to have a hunger to learn and discover and explore. Either they have it or they don't."

This comment has long intrigued me, and part of me hoped it could be disproven. Some years later, I had a chance to work on some introductory materials for the SummerQAmp initiative with a number of other software testers and trainers. Part of the goal of these materials was to help teach a broad set of skills to software testers, but more to the point, my top desire was to see if we could actually spur on curiosity in the participants. Was John true that we could not train curiosity? Was it an innate ability, or was it possible that we could help spark that curiosity early in the process?

I had a chance to participate in my church's "career night" for high school students in our community. Along with a number of other industries (architecture, financial services, engineering, nursing, government, dentistry, etc.) I was the lone representative for both software development and software testing. Everyone else had these posters, graphs and charts talking about the basics of their career and numbers for participants. I decided to go for a different approach.

I brought in two MacBookPro laptops, and I pulled up several of James Lindsey's Black Box Puzzles to display. Then I waited. Other than these two laptops, and a handwritten sign that said "Software Development and Testing", that's all that was visible. As people walked up and were looking to ask me questions about my job, I asked them if they'd like to try it out for themselves. I'd have two people sit down at the laptops, look at the puzzles, and then see what they did. Some people just looked, pushed a button or two, and then moved on. Others sat down and really tried to understand what the puzzles were about. I gave an arbitrary time limit of five minutes, but in reality, I never stopped anyone at the allotted time. If they were engaged, I'd let them go as long as they wanted to. If others came to watch or ask questions, I'd encourage them to get involved and participate with those already sitting at the computers.

What I found interesting was seeing who became engaged and who showed the most interest. We had a group of kids between the ages of 12 and 17, with a mix of roughly 60% female and 40% male. Most of the participants that stopped and participated were female, and of those, about half of them exceeded the time limit and asked multiple questions. I tried my best to keep leading questions out of my dialog, and to answer just what they asked me. The participants also had a list of questions they could ask, and I would answer those as well (such as what was a typical day like for me, how long I had been involved in my job, etc.).

By having people test software first, before going into greater details, I could see who was genuinely interested in doing it vs. those just going around the tables and filling in their question sheets. It was interesting to note that younger participants seemed to be more engaged than older ones, and I had many more young women come and participate and ask me questions than I did young men. Also, the young women hung around the area, engaged in the simulations and asked roughly twice as many questions

as the boys did. Again, this is not representative, and I'm not trying to draw any inferences from this one experience, but it definitely made one thing clearer. Jon was right. The people who became the most engaged and showed curiosity at what they were looking at were much easier to talk with about software testing, and they seemed to "get" what I was talking about quickly. Is it possible that we may see some future software testers emerge from this group? Maybe, but I'd be very curious to compare notes sometime in the future and see if the ones that showed the greatest level of curiosity and focus in those first few minutes will indeed be the testers of the future.

Growing up in and around the San Francisco Bay Area, and having raised my children here as well, I tend to take for granted the level of interconnectedness and the technological understanding that many kids have, and as such, I also thought that just everyone took this for granted, that they were invested in their tools and their apps. I watch my own children use technology with a dexterity and natural ease that sometimes confounds me. Yet they also recognize problems when they see them. They share their frustrations when tools are ineffective, when performance is slow, when they cannot do something they wish to, or that they could do some time before, but because of an update, they no longer can. Their frustration is understandable, but it also showed me that many people just take it as part of the process. When they have problems, they either figure out a way around it, or they just stop using the app altogether. Perhaps this might be part of the challenge when it comes to determining who the really curious ones are. If the solution much of the time is to dump the app and move on to another, maybe they really do not care if an app has good quality or not. If it doesn't work for them, they just move on to something else. Still, I found it both interesting and heartening to see that there were a few people who genuinely were interested in understanding how things worked, were willing to explore and learn about applications and their behavior, and were willing to ask questions and dig a little deeper.

I found that this experience taught me quite a bit about the way that a certain group of teenagers responded to a challenge placed in front of them. Some were engaged, some asked questions, some just poked around and then got bored, choosing to go do something else when they had the chance. Here I had hoped that I would see if I could teach curiosity up front, and give a jump start to those interested. What I found out was a little more nuanced. While it's possible I may have set up the environment to help teach curiosity, instead what I saw was that those who were already curious responded well to the setup, and those who weren't or were not as engaged remained as such. I'm still hopeful that I can refine and develop the materials that we have to help engage more people and, if not teach curiosity, at least help trigger the curiosity they already have, and bring it out more. Looks like I still have some work to do on that front, but I feel I'm getting closer.

About the Author

Michael Larsen is a lead instructor for the Black Box Software Testing courses with the Association for Software Testing (AST). Michael is also a Senior Quality Assurance Engineer with Socialtext in Palo Alto, California, USA. Over the past two decades, he has been involved in software testing for a range of products and industries, including network routers & switches, virtual machines, capacitance touch devices, video games, and client/server, distributed database & web applications. Michael tweets at @mkltesthead.

The Rise of the Coordinators

- Arslan Ali

If you are part of the solution delivery industry then it should not come as a surprise that besides being highly technical in practices and terminologies, the end user of this industry falls beneath the average IQ line. Since the creation of the first line of code to the first ever smiley pasted in a chat room, this fact was kept well hidden, but it has befall inevitable and now, we have to cope up.

Contextual side of the client complaints:

In this industry, the client's resentments are contextual. The nature of client grievances shall depict different meanings to different stakeholders in diverse situations. But the real question that hangs like a sword on a solution provider heads is; what makes a client angry and how can we manage that complaint?

Under the general world context, it can be anything from a fire on a factory floor, someone in need of a first aid, or a tweaking tyre in a car; all these situations throw the general best practices out of the window and require immediate skilled solutions. Similarly, in the business of software development, it could be anything from an unfulfilled marketing claim, a glitch in the requirement understanding, a business rule being coded wrongly, a module crashing with a single click, an unanswered support call or, overbilling for chargeable activities.

Each context carries its own weightage and requires surgical precisions to handle it accordingly. Question is how to make that guy on the other end of the line satisfied?

The human management:

I don't know if most of us have noticed, but within the revolutions of the software industry, in regards to technologies and life cycles, the human side has furthermore evolved. We have to admit the un-denying fact, that software applications are not something highly technical anymore, in fact, have become delicate mix of codes, screens, dialogue boxes, touches, clicks, swaps and user experience.

With Cloud serving as the ultimate library of resources, the barriers in user's choice has diminished. Considering the numbers, there are currently 1.5 million apps available on both Apple and Google Play stores. Consequently, keeping a customer loyal has become the prime challenge for the software development industry.

In order to keep a client coming back for more and stay loyal to your products requires simpler and understandable human thinking. If you set out to this being excessively technical or by being obvious, then smart user on the other end are going to roll their eyes somewhere else, and in a matter of click, you will be searching for another sellable idea, or eventually going back to the design board.

The right resources:

The domains of marketing, development, business analysis, testing, quality assurance, customer support and implementation have their own specialized roles. Altogether, these diverse functions are serving one entity, the customer. For the large software houses, these domains become independent organizational structures and usually share large pool of resources as per their technical and non-technical needs.

To organize further, the functional areas share a common platform to manage and share resources. For an outsider, it seems like a very simple activity comprising of developers and computer screens, but the perspective of solution delivery is way more complicated than anticipated by the eyes of an end user.

Rise of the coordinators:

There is now an imminent requirement for specialized coordinator role to work within these complex software delivery structures. Usually, this role serves well within the domain of business analysts and customer support people, but since the evolution of the latter into more technical aspects, the gap re-emerged for more human oriented actors.

The communication and coordination areas work on either side of the picture, whether it is a solution delivery company, or a client in search of a solution. Both require people with extensive capabilities in communication, human psychology, change management and project coordination.

Internationally speaking, already the business sector has started to acquire similar resources encompassing the same skill set. These setups are either in process of getting a customized solution or a large enterprise resource solution with regards to integrating their practices.

In order to achieve this, these large setups require special resources to manage change, maintain accessible communication lines amongst functional areas, management and human resources. The resources they seek out are not overly technical, nor are they specially trained computer professionals.

Who can fit in?

These resources are mere management techies with a tinge of human understanding in their line of career. They do have extensive experience supporting their skills, but usually that parachute carries too many colors.

The best suited people for this kind of role are software testers and business analysts. The reason being is that they work at the both ends of the bridge between the developers and the clients. Their comfort zone lies within this boundary. Both these titles have no constraints of either being getting technical with the developers to clarify business process issues, or, by getting psychologically effective with the clients and their complaints, and eventually bringing both sides on a comfortable common ground.

In my opinion, these people are firefighters, the last line of defense, and the unnamed rescuers. You send them where the things are not in your control anymore, and they can resolve these anomalies for you.

The direction in which the computer industry is headed, the need to hire for coordination professionals shall rise with good factors. As the word “user experience” is creating more ripples amongst the minds of the techies, the pressure however cannot be exerted on the technical teams, as they will break apart. To get things driven in accordance with the set goals, we will sooner or later call for Firefighters.

It is only a matter of time.

About the Author

Arslan Ali is a Software Testing and Quality Assurance professional from Pakistan. He has been part of the IT industry past 18 years. His career spans from Software houses, industry, and training institutions. Arslan has been an active member of context driven testing community, he has branded his training as #Being Tester and have trained several professionals, students and career seekers in this regards. He tweets @arslan0644

The Evergreen Tester

- Brendan Connolly

Feeding the revolution in web technology and development has been the advent of *evergreen* browsers. These browsers will update automatically as new features are added or bugs fixed often without any intervention from the end users.

This change has unshackled web based applications from chains of supporting legacy environments. While software has always grown and evolved, since evergreen browsers arrived on the scene, it has reached a frenetic pace. Couple this with the increasing adoption of agile/incremental development and the result is software development and technological innovation is occurring at record breaking speeds.

But that's the Developers Problem

So why should testers care, unless you are focused on automation then an app is an app right? Don't the technical implementation details just provide some additional insight to testing? They aren't the primary drivers of test design or considerations. As testers we care about the behavior, the business logic, the usability and functionality. Isn't the rest just a means to an end?

That may be true, but this shift in development has also been changing the process in which software is being designed and developed. Agile methodologies, DevOps, the shift to the cloud... All of these things and more are changing where, when and how testing is integrated into the software development lifecycle.

If they haven't passed completely, the days of testing beginning after development of project has completed are numbered. In some cases these days, time to market/release is more important than a completely polished or fully featured experience. This means imbuing quality earlier and/or in different ways than formal test plans and detailed procedural test cases. There just isn't time for a mini test waterfall inside of an agile sprint.

When the practices around the act of testing introduce more risk than reward, the testers value within the organization falls. We need to be aware of these changes internally within our organization and also within the context of the software industry.

The Sky Isn't Falling

This isn't a fire and brimstone story, I'm not here to tell you QA is dead or that all testing jobs are going away. If you have been in the game for a while, there seems to constantly be some doom and gloom projection that testers will be replaced by some technology or practice. There's been the impending doom of being replaced by automation, being outsourced, or even developers unit testing us out of a job.

The yarn I'm spinning is the polar opposite, it is possibly the best time to be a tester. Things are changing but the software world is always changing. While testing is changing, so much of what a tester has always dreamed of is within reach or for some is already here.

In the days where waterfall application development was all the rage, testers often begged for someone to open up the doors for them at design reviews. We were often physically separated from development teams. Segregated to maintain the sanctity of test from the perilous development influence. Segregated to alleviate developers frustration at the interruptions from testers vandalizing their coding efforts. While those statements are written in jest, there are plenty of veteran testers out there that can regale you with tales that may sound eerily similar to what I have described above.

We've wanted a voice in the conversation, a place at the table, an opportunity to show we can improve quality in ways beyond bug reports. More than ever, we have that. Testers are pairing with developers while they are writing code. Not to do the coding but to be a voice for quality, to share our mindset and how we would

approach using the software. Testers help groom the product backlog, adding acceptance criteria and setting expectations for how software will need to function before code is written. Developers are even making changes to help make testing easier, sometimes even asking testers what they might need in advance. While it's not happening everywhere and maybe not occurring all the time, it is happening and it's not rare like discovering a unicorn.

Just like everything else this change comes with a price tag. Tester's have to put their money where their mouth is and have to bring meaningful contributions to these conversations.

What it means to be an Evergreen Tester

In an interesting twist of fate testers face a similar set of circumstances to that of the state of web applications and development in the days prior to evergreen revolution.

From the craze of browser wars, to the weight and baggage associated with supporting multiple platforms and versions, the browsers, applications and people that used them carry physical and emotional baggage from the journey. Don't believe me bring up Internet Explorer 6 in your next team meeting and watch people physically cringe.

Similar pain and baggage runs as deep or even deeper in the testing world. There seems to be a never ending stream of people selling the next product or practice that will eliminate every teams testing problem. While that takes a toll on the testers, it also warps the perspective of developers and management and fosters misunderstanding and misgivings over testers practices. So we often feel under the gun to not rock the boat, not speak up and avoid the spotlight while still show value.

Just like the web browsers evolution to automatic self-upgrading pieces of software has fueled software revolution and innovation, testers need to evolve and shake loose from the related baggage.We do this by becoming evergreen and continuously and proactively growing and learning independently. Upgrading ourselves in order to take full advantage of the changing landscape of software development.

We do this by:

Building A Solid Root System

A strong, healthy root system is what secures a tree and allows it to stand tall and straight. First and foremost an evergreen tester needs to be rooted in a solid foundation of testing skills and methodology. The roots also allow a tree to take in nutrients from its surroundings. These nutrients are what enable the trees growth. By fostering growth in our core competency it enables a tester dig deeper into their craft and become exposed to new areas and possibilities.

With this solid foundation a tester can evolve and adapt to fit into the current environment. As dedicated test teams shrink and testers are increasingly embedded on cross functional teams you will be expected to stand tall as your teams ambassador of test. We can't know only one way to approach testing or be locked into a single way of doing things. Showcasing our competency and adaptability is what opens the door to tester's influencing other areas of the business.

If you aren't sure how build up your testing roots, the testing community is incredibly welcoming. You just need to plug in and there is a forest of resources available including this periodical, the Black Box Software testing classes offered by the Association for Software Testing, the Dojo at Ministry of Test, Weekend Testing sessions, even Twitter. These all allow you to stay informed, educated and even practice testing in contexts you may not have available to you in your day job.

Branching Out

This new level of access across the team means testers will be more deeply involved in conversations that while directly related to the product are only tangentially related to testing. This isn't a fancy way of saying that testers need to learn to code. It is reasonable to though to assume the more a tester understands about a system and its design the greater insight they can provide into increasing its quality. Coding isn't the darkside for testers anymore, it's just another way you can contribute to your team. You don't need to be an expert or go around giving performance tips to current developers. It goes a long way to strengthen the connection between developers and testers to just be partially code literate and be able to follow more technical conversations.

It doesn't have to be code though, we are just as involved in the product, program or project management processes. Read up on agile, scrum or

whatever your team is using so you are better prepared to chime in with some insights or maybe serve as scrum master for your team. Go deep on your softwares domain or your customers needs, it'll help you be test better and you'll be of greater benefit to your product team.

There isn't a specific predetermined set of skills here, but to be a successful tester you need to love software. So let your branches stretch and grow and follow your passions.

It's Not Easy Being (Ever)Green

That's a lot to ask of anyone, but we are the quality people so the standards for ourselves (and our craft) should be as high as the those we hold the software we test to.

Beyond that honorable call to action is a pragmatic reaction to the fact that testing will always be a target because it is a proactive practice that potentially saves money but generates none. There is also this dangerous sentiment out there that anyone can test. That sentiment derives from a lack understanding of what testers do and the value that they bring to a team. So please join me, whether out of noble pursuit of craft or out of self preservation, in seeking to change those assumptions one interaction at a time by striving to be an evergreen tester.

About the Author

Brendan Connolly is a Software Design Engineer in Test based out of Santa Barbara, California with over 7 years of testing experience in a variety of different roles. He is responsible for creating and executing testing strategies and using his coding powers for developing tooling to help make testers lives easier. He's writes tests at all levels from unit and integration tests to API and end to end UI tests. He doesn't believe automation solves all problems and prefers to take a context driven approach to his testing.

Brendan is a member of the Association for Software Testing and proud graduate of the BBST Foundations course. He also enjoys attending Weekend Testing sessions and does his best to contribute to the testing community through his blog at http://www.brendanconnolly.net and tweets @theBConnolly.

1st Annual Test Leadership Congress
"Test Management Agility"

We brought you years of practice, expertise, innovation and thought leadership on stage.

Opening Keynote by Fiona Charles "The Dying Art of Test Management"
Speakers from Spotify, Etsy, Huge, Johnson and Johnson, major financial institutions and more.

Masterclasses

Well-known educators will teach full day and half day masterclasses in the area of their expertise.
The classes are covering in-depths automation, mobile testing, agile practices, leadership and even the basics of testing itself.

VISIT

TestMastersAcademy.org

MATT HEUSSER

Managing Director
Excelon Development
Allegan, Michigan (USA)

Interview with Testers

As the Managing Director of Excelon Development, Matt Heusser, consults, trains, and does software delivery while helping others do it. Probably best known for his writing, Matt is the lead editor of "How to Reduce The Cost of Software Testing" (Taylor and Francis, 2011), editor for Stickyminds.com, and recipient of the 2015 Most Popular Online Contributor to Agile at the Agile Awards. A 2014 recipient of the Most Infuential Agile Test Professional Person Award (MAITPP) in Potsdam, Germany, Matt also served as the lead organizer of the Software Testing World Cup. He is a former member of the board of directors of the Association for Software Testing and creator of Lean Software Testing family of methods.

* Interviewed by **Ajoy Singha**

1. How did you become a software tester? How long has it been as a tester?

Wow. That's a short question with a long answer. so let's start at the beginning - I started as a programmer in 1997, with a degree in Math and a concentration in computer science. It was the end of the 1990's, the middle of the tech bubble that would end in a dotCom bubble … then a crash. I worked developing telecommunication software. My first real assignment was to create an electronic audit for a telecom bill - basically to re-calculate all the amounts owed on every account, including calls, then compare that to what the billing software generated. If you think about it, it was a sort of model-driven testing.

We didn't have testers at that job, and I would keep "programmer" or "project manager", in one shape or another, on my business card until 2008. Over the years, there has been a lot of angst, a lot of disappointment, between test and programming, but I didn't feel it. I thought a safety net, people to help me check, would be a good thing.

On my first annual review, under areas to improve, I wrote "too many bugs"; my boss said not to worry, everyone has bugs. I did not accept that, and went to school at night, earning a masters degree in CIS, and studying software engineering, distributed systems, and a lot of test and quality. In 2003 I went to the open source conference and met Danny Faught. Danny told me about a magazine called STQA, Software Test and Quality Assurance, the starconferences, and stickyminds. He told me that if I wanted a free subscription to STQA, just go to a STAR, and If I can't afford STAR, then I should speak at one. So I did - I spoke at STAREast in 2004 and wrote for sticky minds that year. In 2005 I went back to STAREast and STARWest, and my little part-time writing and speaking earned me a couple of thousand dollars. The company still didn't have a testing role, but I got more involved in test strategy, to the extent that we had such a thing. I remember around 2006 I was talking to Sean McMillan about the "program with you programmers" - my business card still said programmer!

Sean said "wait a minute, Matt, do you self-identify as a tester?" and I realized the answer was yes! I left that role in 2008 for Socialtext, which wanted me but at the time only had a programming role available. I said "I have a programming job. I wouldn't need to leave for that. I want to be in test!" They found a test lead role for me eventually.

MATT HEUSSER

So I became a tester in 2008, 2006, or 1997. Take your pick.

2. If you have to start your testing journey all over again, what changes would you make in your plan?

Honestly, it all turned out so wonderfully - I don't know what I would change. Two ways to do it: I could have gone to a better school, and paid more attention in school as well as more attention to social situations. I graduated in 1997, just as tech was booming and the dotCom bubble was coming. I could have gotten that education started out in programming - or maybe not. In 1995 I saw Yahoo was exploding and briefly considered dropping out of school, moving to San Francisco, and getting a customer service job at a then-rapidly-expanding Yahoo.

Of course, if I had done that, I wouldn't have had the ten years of programming experience, and, possibly, the experiences I got dealing with conflict, where I was perceived as lower-status because I didn't go to a great school or dress well.

Either way, who knows? I might have been laid off as the silicon valley bubble imploded in 2001. The truth is, to consult and advise, I needed that ten to fifteen years of work/life experience. Maybe I could have earned an MBA at night, instead of the CIS degree, maybe I could have written a book earlier instead of going to school - but really, sometimes, the food needs to cook before you can serve it.

On one consulting assignment I was hemmed in - the company wanted me to do something very clearly defined, that I wasn't excited about. I would be a very expensive contractor. The money was good and they said I would have a chance to have an impact on strategy, so I took the work. They ended up cancelling the project because, surprise surprise, it seemed like I was an overly expensive contractor. So there are plenty of little mistakes like that I made. If I did it again, I would have said "no" a little more. Something else would have come along.

3. You started Excelon Development. What motivated you to start this?

As I implied above, it started as a sort of holding corporation for this income I was earning writing, speaking, and consulting. When I left my position at Socialtext, most of the companies I was talking to wanted me to be an employee. I realized that I didn't want that; I wanted to contract-to-hire at the very least, if not run my own shop indefinitely. Nearly five years later, we seem to be doing okay.

4. What kind of services does Excelon offer?

I'd say that if a company would like to improve their software delivery, it's worth a conversation. The major categories of our work lately has been in contracting (actually doing the work, often testing), writing (technical and for the public), consulting, training on continuous delivery or lean software test/delivery or "LST", and occasionaly, full-time employee placement.

5. What kinds of training do you provide? What are the challenges do you face when imparting training to a group of testers who are new to your school of thought (such as CDT)?

We teach a course called lean software delivery and another one called lean software testing. The testing piece is about pursuing more valuable ways of doing testing. Of course we talk about exploring and managing work in 'lite' ways like mindmaps and work in progress boards - but we also talk about more generalized ways to reduce work, ways to manage regression/release process, etc. We teach Scrum for teams and for testers; in 2016 we are adding a course on transitioning toward continuous delivery. So many companies see it as a light switc that is either on or off. We see concrete steps a company can take to move in the right direction, each that add value in and amoung themselves. I

MATT HEUSSER

think that's enough commercial for one day, though?

6. What is lacking in today's commercialized testing training industry?

The big challenge is that you have to sell what people understand. People understand 1-to-3 day courses. The problem isn't the new concepts; it is the change management piece, the transitioning to a new concept - and the context. We like to work with companies to understand the problem and customize our course materials - along with talking about what to do next. There just isn't enough of that out there. So people come take the course, learn some new free tools (for example), or techniques. Then they go back ot the office, and their user ID does not have admin rights, so they can't install the tools. They mention the techniques to the boss, and the boss says "GREAT. Of course, you can do that ... in addition to everything you had to do before."

This actually adds to the work.

To go faster, we need to take away.

In the lean course, we talk about identifying and removing waste, in a way the entire team agrees with. We talk about how to communicate that to management and make the cost explicit. We want to help people go /faster/.

7. If you were to train a "High School dropout Starbucks employee", how would you start teaching testing?

I'd probably start with some analytical brain teasers. Some people are really good testers, but bad at math - or at least, uncomfortable with math, especially geometry. If they hate geometry, we will NOT start with the triangle problem. Then I'll likely have them watch me test - we'll pair. After awhile, I'll suggest they try. So mostly I describe it, I explain it, they watch me do it, I help them do it, they do it under observation. Basic repeatable skill building, 1-on-1, kind of like teaching soccer - only It would be much more pairing.

8. In 2010 you participated in the American Society for Quality's "Influential Voices" program. Tell us more about that program.

After graduate school and before going independent, I used to blog about twice a week, every week, for several years in a row. My blog "creative chaos" attracted a reasonable following, and I was a member of this group, the American Society for Quality, or ASQ. For a year, once a month, the president of ASQ, or the executive director, would put a blog post out, and we would respond to it on our blog - then the blog would link to our responses.

The worldview of ASQ comes from manufacturing, which in the USA leans heavily on standards and defined process. We had more than a little bit of back and forth. It was fun, but a reasonably large amount of work.

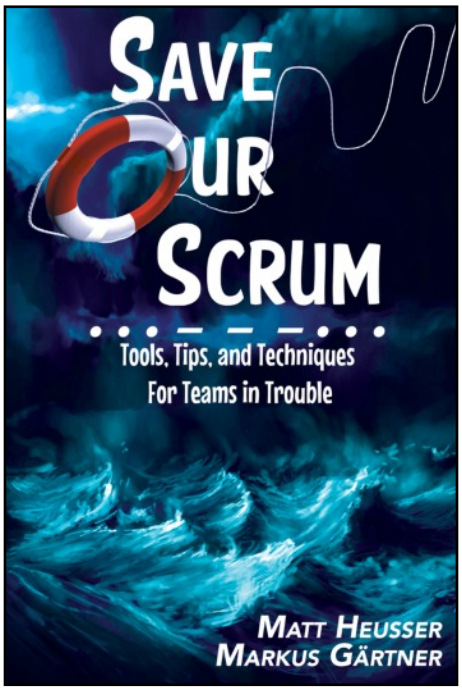
Matt @TestBash, New York

MATT HEUSSER

9. You were one the co-editors of the book "How to reduce the cost of software testing". How was this book conceptualized? Tell us more about it.

That is a fascinating story, and a lot of the credit goes to my co-editor, Govind Kulkarni. He started with a linkedin group, asking how to reduce. There were something like 180 answers. Eventually he suggested we do a book. I had some contacts and offered to do a chapter and help out with the proposal. Eventually I complained I was doing a lot of work here, I should be co-editor, he said okay. Then I said I'm doing A LOT of the work, I should be senior editor - and he agreed!

Books don't make a ton of money. For this one, we took a very small advance. I'm very pleased to say that we split it up among the contributors. There was $500 left over, which I sent to Govind. He was great to work with.



The books contributors, I think there were something like 27 people and 24 chapters - for the most part, these people were not professional writers. At all. It is a bit dry to read. It is a bit awkward. The authors do not always agree - but we wanted that. We wanted the reader to struggle with it.

Some of the ideas that I hint at in the book - about obstacles, throughput, about time on task - these became a big part of our lean software testing philosophy. Perhaps, if the book had that title instead, it would have sold a few more copies (laughs) - but it turned out okay.

10. Why "Save Our Scrum"? Where is the problem with Scrum today?

We could do a whole interview on the system forces trivializing Scrum and re-interpreting it as command and control. For today, let's just say there is a lot of it - "Scrum" becomes Stories, Sprints, and Standups. Standups become a status meeting, and a boring one at that. The breath of fresh air that was supposed to be Scrum has become just more work - sort of like I mentioned about lean before. But when you combine lean with things like the Scrum Retrospective, where we make plans for improvement - that's magic, man. It's also a better way to work, a more human way to work. I feel more alive doing it. I want to tell people about it.

11. What is your next big plan?

I want to stop starting and start finishing! Markus and I need to finish Save Our Scrum. Maik Nogens and I are doing the Software Testing World Cup again this year. Qualitest Group just launched the Testing Show http://www.qualitestgroup.com/resources/the-testing-show/ in partnership with my Company, Excelon Development. CAST 2016 is going to be in Vancouver, and I am running a test retreat before it, this time as an official part of the conference. And of course we are talking about a humanistic, realistic vision for purusing continuous delivery. We've got a new contractor starting full-time next week ... I think that's plenty for now.

12. Name few people in testing, who influenced you directly or indirectly in your career as a software testing professional.

Oh gosh. So many. Early influences included my old professors - Homer Austin, Kathleen Shannon,

MATT HEUSSER

Dean X. Defino (he doesn't have a middle name, but salisbury's email system requires one, so they gave him an X) - Mary Lou Malone suggested I read The Mythical Man-Month, which made a world of difference.

Outside of School, DeMarco and Lister's PeopleWare had a huge impact on me early on, and also Ron Jeffries and Chet Hendrickson's XP:Installed. After those books I ran into context-driven, and Dr. Kaner and James Bach. Michael Bolton was very helpful to me early in my development as a context-driven tester; I appreciated his sort of reducto-ad-absurdum method of asking questions. Not my style, but great to see example of the thinking, almost like a math proof. Karen Johnson is another great example of someone i've worked with, a little here, a little there, over the years.

It was in Miagido, around 2009, that we started to see a new group come up - the Markus Gaertners, Justin Rohrmans, Michael Larsens, and, more recently, people like Carol Brands, Tim Western, Tony Guiterrez and Claire Moss remind me of why I fell in love with testing in the first place.

13. Five things nobody knows about you.

- Like literally? No one at all? Not even me? That doesn't sound right.
- I'm pretty good at deconstructing sentences that are not, well … precise.
- Generally, when that happens, I'd rather embrace the intent of the question than pull a "well, actually", as I did in answer #1.
- I was a scout as a child, which some people may know, and we learned the twelve points of the scout law - on being Trustworthy, Loyal, Helpful, Friendly, Courteous, Kind and so on (https://www.meritbadge.org/wiki/index.php/Scout_Law). As a child, I did not take those things seriously, but as an adult, I realized what my leaders had been trying to do. They were trying to teach us virtue. Today, I can appreciate them for it.
- I got into computers because they were consistent. Humans … I could not figure out. They would lie to me about something then be upset when I took them seriously, or care about the value of my shoes over everything else. Eventually I figured out that, in order to be successful in most things, I would need the willing cooperation of other people - so I got into the "people stuff." I used to feel bad about that, unitl I read Jerry Weinberg say something almost identical in Quality Software Management Volume I.

14. Usually our last question. If you had to run Testing Circus, what changes would you make to improve it?

I've read it from time to time; it's hard to keep up! It seems like over the past five years or so there has been an explosion in the test literature produced. To be fair, I'm partially to blame for that. One thing I've noticed is the sheer number of blogs and magazines with "test" and "quality" I'm their name — it's hard for me to tell them apart. What makes "testing circus" different from other magazines? I do not mean this as a harsh criticism. I just think exploring your competitive differentiation - literally what makes you unique/different, both in your DNA and in format of the articles, might add some more value to the community.

Blog/Website: http://xndev.com/

Twitter : https://twitter.com/mheusser


“Doing business without advertising is like winking at a girl in the dark. You know what you are doing but nobody else does.”
https://www.testingcircus.com/advertise

What is the 'Right' Testers to Develops Ratio in Agile?

- Ravi Kumar BN

Over the years, there has been much interest in finding the 'right' answer for a long-standing question in the software development world is: what is the correct ratio of testers to developers? Obvious answer you get would be "It depends". You may or may not find this response useful. But it does depend. It always has and will do. The context-driven people provide a little more information - "it depends on context". But this doesn't answer the question of course – we still get asked by people who really do need answer.

Why do project managers ask this question?

Project Managers are the people who really do need to plan and to resource teams.

- Planned budget is low and needs to articulate a case for justifying more OR
- Planned budget is too high and wants a case for spending less

Other possible reasons could be that to give him/her reassurance that he/she has the right answer already or want personal, direct, relevant experience and data or want to understand estimates and recommendations or he want references to industry 'standards'.

What are Industry Standards?

Microsoft employs a 1-to-1 ratio of testers to developers, according to the book 'Microsoft Secrets.' In an informal poll of participants in a conference session, it was found that the most common ratio was 1 tester to 3 developers. A paper published by well-known authors found that ratios such as these are surprisingly meaningless. The responsibilities and tasks assigned to each of these roles vary greatly from project to project.

You can find many rules of thumb for the ratio of QA to developers if you do a Google search with the words in the title of this blog entry. You will find people talk about 10 developers to 1 QA tester, 3 to 1, 1 to 1, and many others. None of these can possibly be correct. They can't be right, because they don't take into account the abilities of both the developer and the tester. Highly capable developers may be 10 or more times quicker at producing the same code as less capable team members. The same will hold true for QA testers. Few people observed that a ratio of 6 testers needed to absorb the work of one highly productive developer.

What are the pitfalls in adopting Industry Standards?

Let it be clearly understood that you shouldn't completely discount the use of ratios in planning if they are your ratios, based in your experience, your technology and your organizational structure. However the risk is when an organization takes another organization's ratios and applies them to their project without regard to differences in technology, process maturity, and skill levels. Some are some possible differences between projects.

1. Same test team may be supporting parallel projects/programs with overlapping verification timelines.
2. Lack of automated sanity and regression test suites, leading to huge manual verification efforts.

3. Same test team might participate in multiple verification cycles such as formal sub-system verification and system verification.
4. Verification timeline could be fixed time box such as formal sub-system verification should be completed in 4 weeks and system verification in 2 weeks.
5. Verification team may need to conduct regression for huge set of legacy applications including 3rd party integrations.
6. To meet regulatory compliance verification team may have to create test docs and evidences for each milestone (short time boxed increment in agile).
7. Test suites for few applications could be very complex and humongous
8. Few applications such as 3rd party integrations might do very limited dev, but testing is required.
9. Verification team has to cover breadth of configurations and hardware/software/upgrade variants.
10. Major population in verification team could be contractors which is a revolving population.
11. Verification team may be supporting maintenance releases or patch fixes on need basis.
12. Verification testers are involved in preparing test docs, testing new features, conducting regression on all legacy apps including defect fixes and merges from previous releases & system/platform changes.
13. Delays in build releases for verification may lead to peaks in verification
14. Constant verification effort builds up due to multiple build releases in verification (repeated tests for each build)
15. As product architecture is complex with multiple systems, integrations and platforms, regression is always a huge effort.
16. Testing same application with different multi-vendor datasets effort is also huge.
17. Manual test could be the only way to find defects with the current maturity in development practices.

What are the influencing factors?

The 'best' dev/tester ratio is possibly the most context-specific question in testing. It is found that variations in project circumstances make comparisons between projects less relevant. What are the influences on the answer?

- What is the capability/competence/experience of the developers and testers respectively and absolutely?
- What do dev and test WANT to do versus what you (as a manager) want them to do?
- To what degree are the testers involved in early testing (they just system test? Or are involved from concept thru to acceptance etc.)
- What tools are available for testing and to what extent?
- What is the risk-profile of the project?
- What is the incoming reliability of the software?
- Do stakeholders care if the system works or not?
- What is the scale of the development?
- What are the quality standards that must be maintained?
- What is the ratio of new/custom code versus reused (and trusted) code/infrastructure?

- How trustworthy is the to-be-reused code anyway?
- How testable will the delivered system/software be?
- What is breadth of configurations that must be tested
- Do resources come in integer whole numbers or fractions?

How does Agile impact this ratio?

In fact agile teams can do more testing, with fewer testers. On agile teams, testers are able to add much greater value by doing exploratory testing, creating test automation, and working closely with product owners to refine requirements and acceptance criteria.

Agile teams function effectively with significantly lower tester to developer ratios. This doesn't indicate that testing is less important, however. Agile teams need testing skills at least as much as traditional teams. The difference is that these skills, and the responsibility for ensuring quality, does not rest with a few people called testers. The entire team is working to build quality in to the product, as opposed to counting on a QA group to test quality into the product.

We can't win this battle for correct ratios between QA and developers by simple rules of thumb. But we can fall back on the values and principals of Agility. We need to focus on the team's people and interactions (specifically, QAs, BAs, and developers) for as much work as possible up front to increase quality (the most efficient and effective method of conveying information to and within a development team is face-to-face conversation). Keep WIP sizes small (working software is our primary measure of progress). Test our code not just during development (continuous attention to technical excellence and good design enhances agility), but get it into the hands of customers quickly and often (customer collaboration).

Rather than going that rule of thumb route, consider getting closer to a single stream flow on individual things that the software needs to do and employ the "Three Amigos" model. Here, we get the BAs, QAs, and developers at the start to write automated tests that serve as the functional requirements for the work to be done. If we keep the rate of production of these collaboratively-developed tests in line with the actual rate of production that satisfies the tests, we never have to fear that we have something off balance. If we find, perhaps with a Kanban analysis, that we can't produce enough tests to keep our developers happy with enough work to do, we may find that we don't have enough BA types or enough QA types available for us, and can adjust accordingly.

And, yes, there will always be a place for QA exploratory testing on integrated code. But that should be automated as well into regression suites that are automated and repeatable.

What are the possible choices for project manager?

Suppose the PM says he has 4 developers and needs to know how many testers are required. Here are the possible choices:

- **4 dev – 1 tester:** onus is on the devs to do good testing, the tester will advise, cherry pick areas to test and focus on high impact problems. PM needs to micro manage the devs, and the tester is a free-agent.
- **4 dev – 2 testers:** testers partner with dev to 'keep them honest'. Testers pair up to help with dev testing (whether TDD or not). Testers keep track of the coverage and focus on covering gaps and doing system-level testing. PM manages dev based on tester output.
- **4 dev – 3 testers:** testers accountable for testing. Testers shadow developers in all dev test activities. System testing is thorough. Testers set targets for achievement and provide evidence of it to PM. PM manages on the basis of test reports.
- **4 dev – 4 testers:** testers take ownership of all testing. But is this still Agile?

Perhaps it's worth asking the PM for dev and tester job specs and working out what proportion of their activities are actually dev and test? Don't hire testers at all – just hire good developers (i.e. those who can test). If he has poor developers (who can't/won't test) then the ratio of testers goes up because someone has to do their job for them.

Conclusion

Let's change the conversation from one asking for rule of thumb ratios into one that asks for collaborative development, better quality, and faster market realization of smaller and smaller chunks of valuable software. We can measure and find where WIP is causing resource constraints, and do the traditional 5 step Theory of Constraints solution to fix things. In other words, let's turn around the faulty question for an answer (that is bound to fail in practice) into a new quest to deliver, measure, and deliver more of what works.

About the Authors

Ravi Kumar BN is a Product Verification Manager for Imaging Clinical Applications & Solutions, HealthCare IT, Philips Innovation Center, Bangalore. He is a master's graduate from IIT Kanpur, UP, India, and BE (CSE) from SDM College of Engg & Technology, Dharwad, Karnataka, India. He has 13 years of experience in software quality processes and testing at Honeywell. He is a six sigma, lean and agile testing expert and a six sigma techniques and tools trainer. He is actively involved in building and deploying testing strategies for various platforms such as ERP (Peoplesoft), CRM (Siebel, SFDC), BI (Cognos, OBIEE), emerging technologies such as Mobility, Cloud, Analytics, Voice, Responsive Web Design and Wearables. He has attended design thinking workshop and has expertise in deploying design tools in problem solving and usability testing. He has authored and published few testing articles in QAI conferences and online magazines such as Testing Experience, Testing Circus etc.

Govind PK

Expert Exploratory Tester

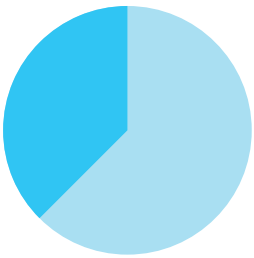
India

236	1653
Following	Followers

Level 7 (870 points)

30 more points needed to became a validater

4 test cycles won



83 bugs reported

Receivable amount in this month

₹19,500/-

What is 99tests?

99tests is a crowdsourced testing platform consisting over 12,000 testers who have logged around 75,000 bugs. Testers are exposed to various types of testing and domains. 99tests is a platform where you can earn, learn and grow by login bugs, connecting with testers across the globe.

Why should I sign up with 99tests?

1. Work whenever you wish to and from any place you want to
2. Choose the products/ software you want to test
3. Collaborate with the testers across the globe, share your skills and grow with them

Meet our community


India


USA


Russia


UK


Philadelphia


Austin

Join our community today!

99tests.com

"Being in the field of software testing for a while, I always wondered whats my stand in the testing skills I possess compared to the other testers. This was more of a positive thought as I was more keen on identifying what I lacked and what I needed to improve."

AWARDED BY

NASSCOM GSF productnation BIG IDEA Small Business TRENDS

#LearnFromHere

Codecademy

The easiest way to learn to code.

https://twitter.com/codecademy

Udemy

he world's largest destination for online courses.

https://twitter.com/udemy

Coursera

We provide universal access to the world’s best education.

https://twitter.com/coursera

edX

Free online courses from @Harvard @MIT & more of the world's top schools.

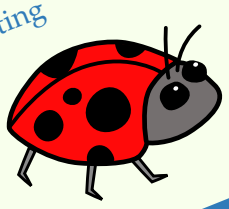
https://twitter.com/edXOnline

https://Twitter.com/TestingCircus

Testing Circus

Testing Circus is a world's leading English language magazine for software testers and test enthusiasts. Monthly editions since September 2010. #testing

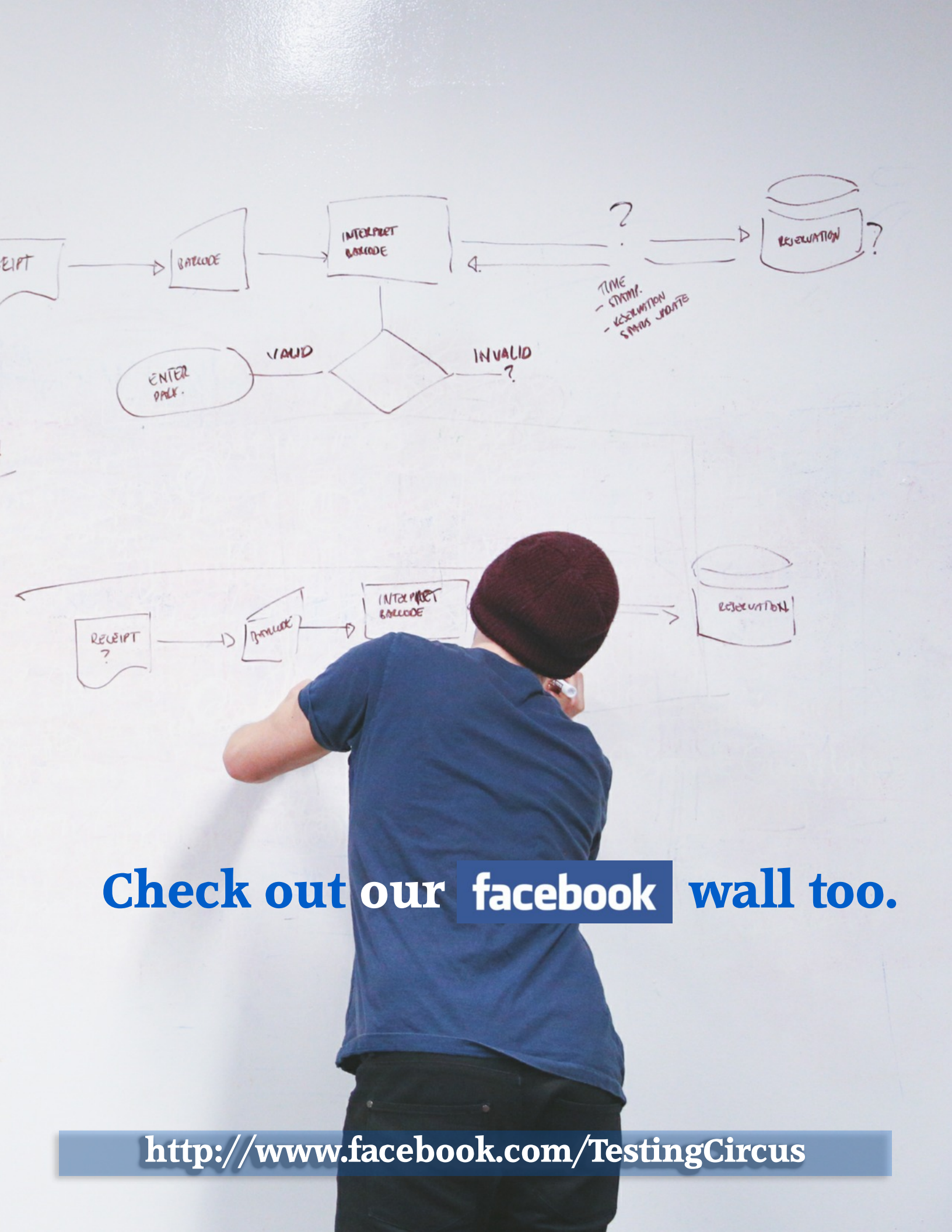
Follow us at https://twitter.com/TestingCircus



#Follow Us On Twitter

https://Twitter.com/TestingCircus

200+ testers to follow on Twitter -
https://www.testingcircus.com/testers-in-twitter



Check out our facebook wall too.

http://www.facebook.com/TestingCircus

The Simplest Web Automation Tool

Simple

Powerful

Productive

Simple powerful scripting

Smart Object identification - No complex Xpaths

Powerful record and playback -Any Browser Any OS

Fast parallel batch playback

Ajax? No time out issues

Become our fan -
https://twitter.com/_sahi
http://www.facebook.com/sahi.software

Request a free demo by sending us an email at support@sahi.co.in

http://www.sahi.co.in

Tutorial 3

Automation with Selenium

- Learn Selenium with Mohit Verma

How to handle dropdowns using Selenium Webdriver

This is our third tutorial on Automation with Selenium series. Till now, we learned about Selenium Suite - how to install and use selenium for automation, how to locate elements on a webpage and use those elements for automation. In this tutorial, we will learn how to select the value from dropdown using Selenium Webdriver.

We will be using **Select** class from Selenium Webdriver to handle the dropdowns. There are three simple steps to select a value in dropdown

1) Locate the dropdown you want to automate

2) Create the object of Select Class

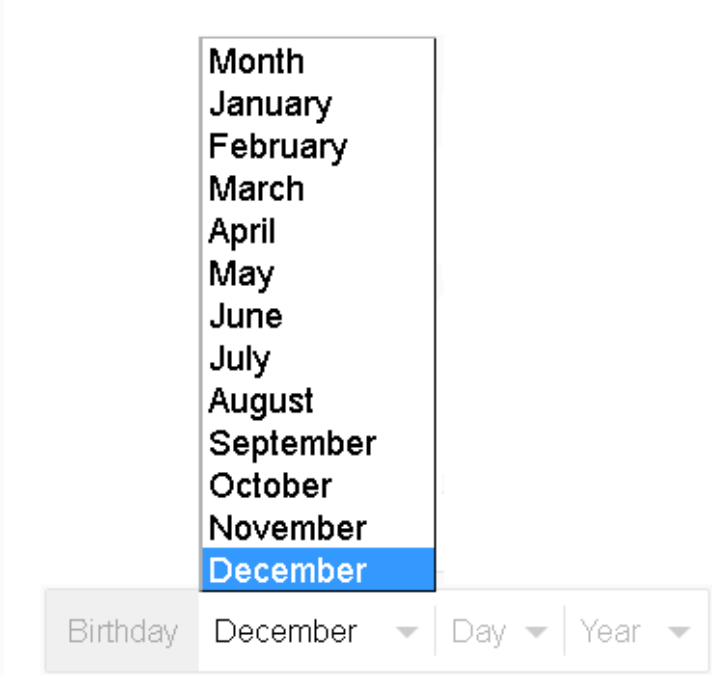
3) Use the Object to perform the action on the dropdown

For using the Select class, you would need to import the following package:

```
import org.openqa.selenium.support.ui.Select;
```

There are three ways to select the options in a dropdown

- **SelectByVisibleText** – User selects the option by using the text displaying in dropdown. For example – Use December for choosing the month December in dropdown

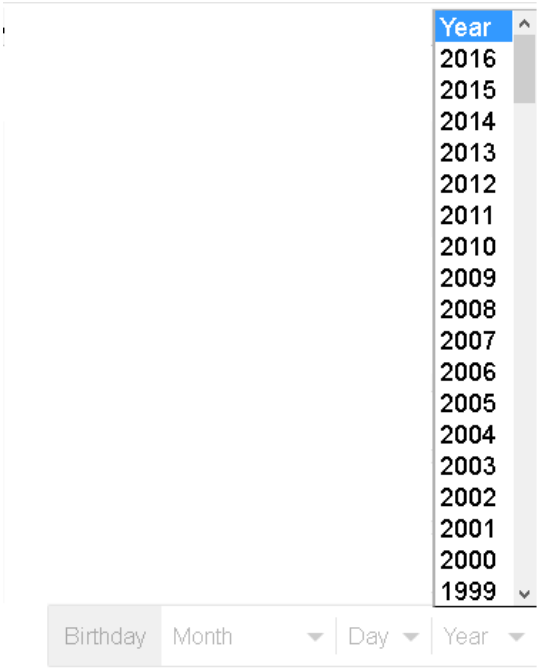


- **SelectByValue** – In this method, user selects the option in the dropdown using value against that option.

```
<select id="month" data="birthday-drop-down" name="mm" style="opacity: 0; position: relative; z-index: 100;" aria-invalid="false">
    <option value="">Month</option>
    <option value="1" title="January">January</option>
    <option value="2" title="February">February</option>
    <option value="3" title="March">March</option>
    <option value="4" title="April">April</option>
    <option value="5" title="May">May</option>
    <option value="6" title="June">June</option>
    <option value="7" title="July">July</option>
    <option value="8" title="August">August</option>
    <option value="9" title="September">September</option>
    <option value="10" title="October">October</option>
    <option value="11" title="November">November</option>
    <option value="12" title="December">December</option>
</select>
```

For example – The value 2 shall be used for choosing the month February in dropdown

- **SelectByIndex** – In this method, the dropdown option is selected based on index value. The first option in dropdown has index value 0. So, in below example, the index value for year 2014 will be 3.



In our example, we will be using all these three methods to select the option in the dropdown.

Scenario to be automated:

1. Launch the web browser and open the Yahoo Registration Page
2. Select value in Month dropdown using SelectByValue method
3. Select value in Day dropdown using SelectByVisibleText method
4. Select value in Year dropdown using SelectByIndex method
5. Close the Browser

Sample Code:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class UsingSelect {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
	WebDriver driver = new FirefoxDriver();
	String AppURL = "https://in.edit.yahoo.com/registration";

	//Open Yahoo Registration Page
	driver.get(AppURL);

	//Maximize the Browser Window
	driver.manage().window().maximize();

	//Identify the Month dropdown
	WebElement monthList = driver.findElement(By.xpath(".//*[@id='month']"));

	//Instantiation of Object of class Select
	Select month =new Select(monthList);

	//Selecting the value in Month dropdown by selectByValue method
	month.selectByValue("5");

	//Instantiation of Day dropdown - Element is located in same statement
	Select day = new Select (driver.findElement(By.xpath(".//*[@id='day']")));

	//Selecting the value in Day dropdown by selectByVisibleText method
	day.selectByVisibleText("12");

	//Locating the Year dropdown
	WebElement yearList = driver.findElement(By.xpath(".//*[@id='year']"));

	//Instantiation of Year dropdown
	Select year = new Select(yearList);

	//Selecting the value in Year dropdown by selectByIndex method
	year.selectByIndex(3);

	//Close the Web Browser
	driver.close();

	//Terminate the Program
	System.exit(0);
	}
}
```

Code Explanation:

Locating the dropdown

```
WebElement monthList = driver.findElement(By.xpath(".//*[@id='month']"));
```

Here, we are locating the Month dropdown and storing the value in monthList variable of type WebElement.

To use the WebElement, we would need to import the following package:

```
import org.openqa.selenium.WebElement;
```

Instantiation the Class

```
Select month =new Select(monthList);
```

Here we have created an object month for the class Select and instantiate it with **monthList** identifier.

Selecting the option from dropdown

```
month.selectByValue("5");
```

Here we are selecting the option with value 5 in dropdown.

Summary: In this tutorial, we learned that **Select** class is used to handle the data in dropdown. We have three methods to select the desired option in dropdown

- SelectByValue()
- SelectByVisibleText()
- SelectByIndex()

About the Author

Mohit Verma has 8 years of experience in Software Testing. He has experience testing projects in Insurance, Health, Retail, Social Media, ERP and many other domains which include Fortune 500 clients and Government agencies.
Mohit holds Master degree in computer applications. He is currently working as Project Lead with SPAN Infotech (EVRY India), Bangalore. He can be contacted at mohitrajanverma@gmail.com or follow him at Twitter @MohitRajanVerma

Unit Testing & TDD are for Developers

- Jari Laakso

A tester, a programmer, and a developer were sitting in a bar

I am sometimes inconsistent with my use of the word "developer". I use it mainly to describe people who are part of developing a (software) product, but sometimes I might say for example "dev", when referring to programmers. As an example, I might use "dev" in Twitter to save those precious characters that we claim limiting what we want to say ("140 char!" in Twitter Lingo), and "developer" when talking about programmers while having a strong feeling of confidence I am talking with someone who uses the word in the same way. In this article, I am using "developer" to point out "tester" is a (proper) subset of "developer", and also how testers can collaborate with the other kind of developers - even if there is a terminology difference.

Unit testing and TDD

For the sake of this article, unit checks are the codified outcome of the unit testing process. Similar to how writing happens mostly inside the mind of an author and the outcome can be typed, unit testing is mostly about the thinking involved in creating unit checks and how the source code should be written. With TDD, I'm referring to writing a failing automated check, then coding until the check passes, and then refactoring the code. Unit testing is necessary while writing code (or maybe someone can shut down the brain), but the checks are written before, during, and/or after the unit has been coded. TDD, however, is using checks to drive the development, so the checks should be written before the source code.

I am making some assumptions here that I find necessary in order to keep the article on track. Thus, I am going to ignore: what is the difference between "test first" and TDD, what if the automated check is wrong, what if the code passes but is wrong, does passing mean anything for the product, do we always refactor the code, what if we add checks, and so forth.

Let our powers combine

No Captain Planet here, just a bunch of testers wanting to help. We can't always get all what we want, even if there are rather obvious potential synergy benefits, in our opinion. We aim to do the best we can with what we are given. Surely we can explain the benefits we see, but people make decisions based on feelings, and, more often than not, we don't know all the reasons behind the decisions.

But let's assume we can move our testing earlier - maybe even before any specific code is written. What could we do, you might ask. We could review user stories, for example. We can ask questions:

- "Are we sure all our users have only numbers in the postal code"
- "Do we want to allow only male and female as gender options when registering"
- "Do we want to ask the gender at all"
- "Do we want to ask a user to register just to browse our shop"
- "How will we detect if the uploaded picture contains a virus"

We could explain the programmer how we would go around testing what they will build:

- "I have tools to generate the weirdest inputs"
- "I will modify the POST request after submitting the form"
- "I will go back and forward with navigation buttons of the app, as well as from the browser"
- "I will hit my keyboard like there would be ants all over it"
- "I will try it in safe mode"
- "I will use Selenium IDE for finding race-conditions"

We could help with checks:

- Codify unit checks, e.g. check if piece of code works according to design specification
- Introduce a new unit checking framework
- "It's Gherkin all the way down"
- Review checks from the programmer

The power is yours!

The non-existing Captain Planet returns to the planet, restoring the Planeteers' powers. It's really up to us to try to make the change. Very few people, in my experience, have come to ask testers to do this stuff. Just like I've seen very few people deliberately increase testability without anyone asking for it.

If you have tried all the things you find reasonable (find a programmer who likes you, bribe a programmer, ask the project manager for a small-scale piloting, present the slideshow you created over the weekend, give them a recording of your talk in a conference about TDD…) and there's still no traction, consider focusing your energy on something else. You can do other wonderful things besides helping people with something they are not interested in. However, I recommend trying this out, not only for direct product improvement reasons, but also for example because it can enable you and the programmer learn from each other.

About the Author

Jari Laakso is a software testing professional who constantly strives to become better. He likes to talk and write about software testing, which has enabled him to connect with hundreds and hundreds of testers all around the world. His pursuit of deeper understanding of testing requires, in his opinion, studying various sciences, including linguistics and semantics. Jari has spent about a decade in testing, out of which most on software side. Since 2007, he has worked in different managerial positions, but always tried to keep up with his hands-on testing skills. In addition to this, he is a proud father of one.
Jari tweets at @jarilaakso.

Still relying on reading Testing Circus from tweets & facebook updates?

Subscribe with your email id and get the magazine delivered to your email every month, free!

To Subscribe

Click Here

Testing Circus

www.testingcircus.com

IT'S CHECK AUTOMATION, AND NOT TEST AUTOMATION.

WE FOCUS ON TEST COVERAGE & CHECK AUTOMATION COVERAGE.

WANT TO WORK WITH US?

— Write to our Founder at st@testinsane.com —