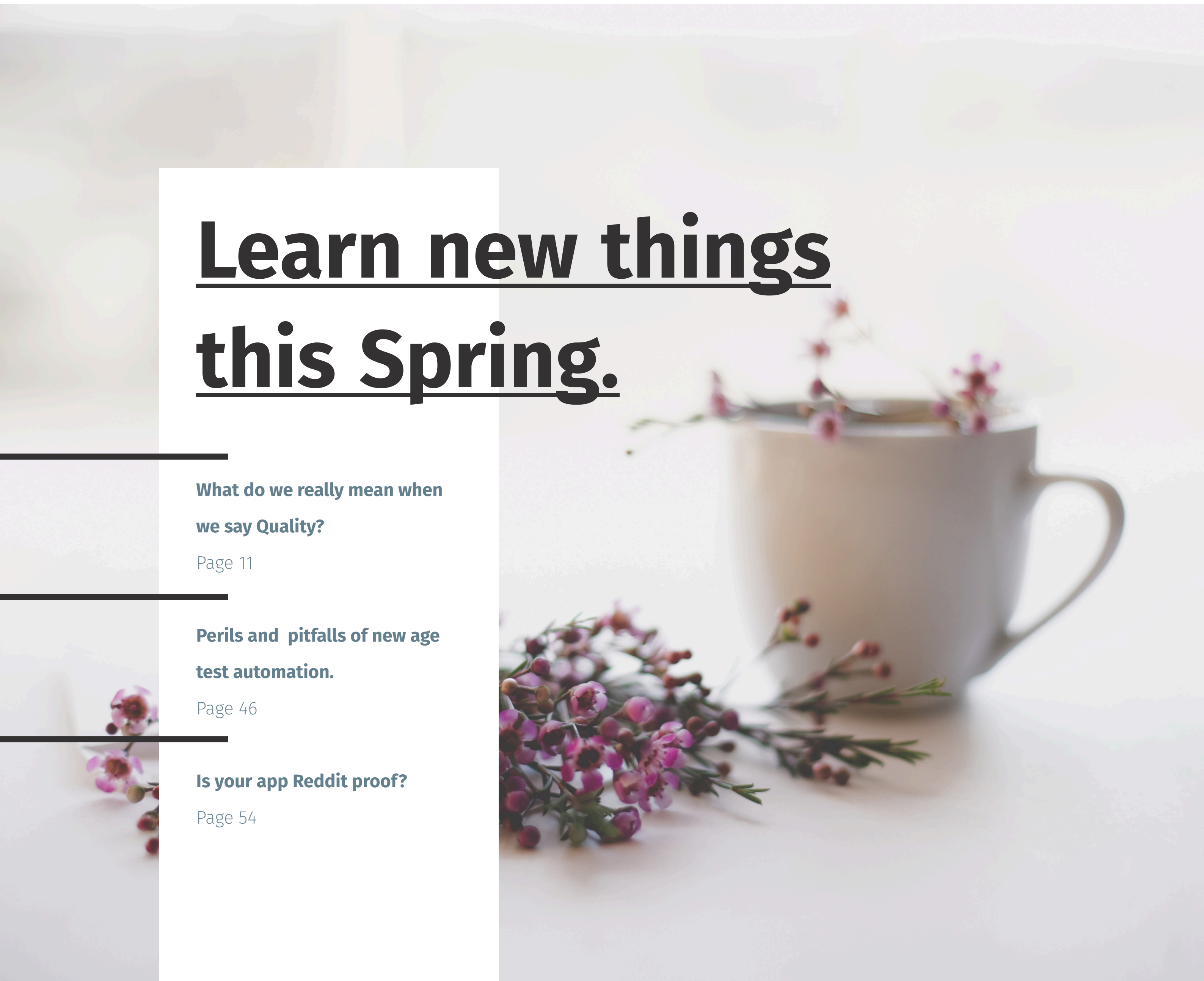
A SPECIAL EDITION //
CELEBRATING 10 YEARS OF AWESOMENESS ❤

ISSUE #01/2021

TEA-TIME WITH TESTERS

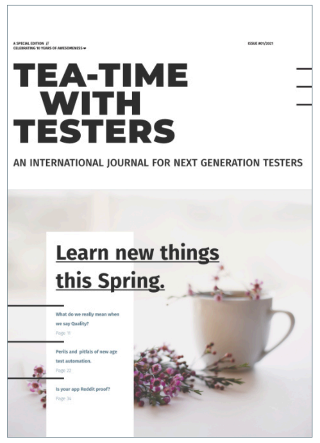
AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS



Learn new things this Spring.

PUT THE CRAFT BACK IN TESTING!

TEA-TIME WITH TESTERS

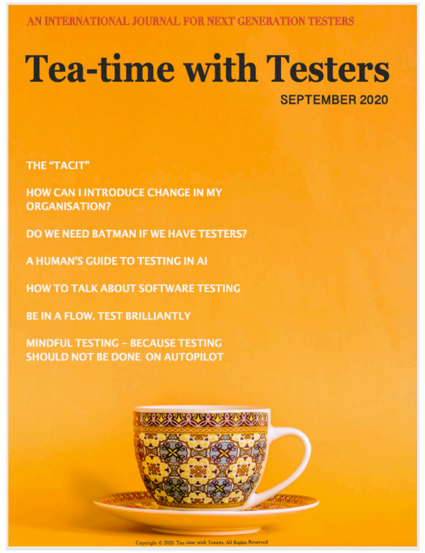


TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

Learn new things this Spring.

TEA-TIME WITH TESTERS


AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

Tea-time with Testers

SEPTEMBER 2020

THE "TACIT"

HOW CAN I INTRODUCE CHANGE IN MY ORGANISATION?

DO WE NEED BATMAN IF WE HAVE TESTERS?

A HUMAN'S GUIDE TO TESTING IN AI

HOW TO TALK ABOUT SOFTWARE TESTING

BE IN A FLOW. TEST BRILLIANTLY

MINDFUL TESTING – BECAUSE TESTING SHOULD NOT BE DONE ON AUTOPILOT

Copyright © 2020. Tea-time with Testers. All Rights Reserved

A NEXT GENERATION MAGAZINE

FULL OF CONTENT AND TIPS FOR TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

Tea-time with Testers

DECEMBER 2020

WHOSE QUALITY IS IT ANYWAY?

THE FUTURE OF TEST AUTOMATION MUST BE INTELLIGENT

WHO CARES?

DEVOPS DEFECT CATALOG FROM 1988'S CEM KANER

EXCELLENT SOFTWARE TESTING

10 THINGS TO BE SENSITIVE TO DELIVER BRILLIANT CODE

PERSPECTIVE – THE SECRET WEAPON WE SELDOM USE WHILE TESTING

Copyright © 2020. Tea-time with Testers. All Rights Reserved

Welcome to this 10th Anniversary Issue!

By publishing this ezine we promise to offer an open platform where we all can discuss, share, suggest, contribute, criticize, guide, relax, and many other activities related to the world of software testing.

We will be more than just happy if we succeed to bring the same confidence and never-fading smile on yet another new tester's face.

In January of 2011, Lalitkumar Bhamare and Pratikkumar Patel launched a brand new online testing magazine with that pledge. Produced, edited, and staffed by willing volunteers and stuffed with interesting, useful, and entertaining articles, the first issue established Tea-time with Testers as a must-read resource. Excepting 2019, there has been at least one issue every year since, and sometimes as many as 12, though publishing monthly turned out to be too difficult after the first few years.

Today I have the great pleasure of welcoming you to this 10th-anniversary issue of Tea-time with Testers.

Since Lalit invited me to write this guest editorial, I've been browsing through different issues of Tea-time, often stopping to read or reread articles that intrigued me. (I even found one that I wrote a few years ago and had forgotten). In this time of plague when we are all worn down with WFH and the tedium of Zoom meetings and isolation, it's been a delight to renew my acquaintance with Tea-time and its community of bright, challenging, funny, sometimes even cantankerous, contributors.

I love Tea-time's quirkiness and the variety of articles and points of view:: the fun stuff, the serious, and everything in between. I didn't need to be reminded that there are engaged, thoughtful testers all over the world doing interesting and innovative work and eagerly sharing what they learn and invent, but I enjoyed the reminder all the same. It jumps off the virtual Tea-time page: from thought pieces by legendary authors and testing luminaries like Jerry Weinberg, Karen Nicole Johnson, and many others whose names you will readily recognize*, to hands-on practical how-to's—and also thought pieces—by testers whose names may be new to some of us, but from whom we will surely hear more quite soon. Visiting the Tea-time site and reading previous issues is an activity I highly recommend.

In this issue you will find—as you have in every issue of Tea-time from the beginning—excellent articles by testers representing a wide range of interests across the testing world: to think about, learn from, and argue with. Some may reiterate things you know already, but that are new to others. Some may challenge your existing beliefs and ways of working. I hope so, anyway!

I hope also that you will join me in thanking and congratulating Lalit, Pratik, and their collaborators for fulfilling their original promise and giving us a terrific 10 years of Tea-time with Testers. Congratulations! (and many happy returns).

**I realized that once I started naming authors there'd be no good stopping place, so I arbitrarily picked one man and one woman and stopped there.*

GUEST EDITOR - FIONA CHARLES

–

Fiona is a software test consultant, teacher, writer, speaker, iconoclast. She leads workshops "beyond process", coach, renovate & rescue testing of any scale.

HEAR FROM MORE LUMINARIES ON THE OCCASION

~People

ENLIGHTENING IN TESTING

Something about knowledge

What you cannot know

E. W. Dijkstra expressed this principle of testing in 1969:

Testing shows the presence, not the absence of bugs.

Dijkstra was discussing testing with researchers focused on finding ways to prove that code was working. They were in a conference organized by NATO and although I was barely born when it took place, I would have loved to be there as according to the transcript Dijkstra terminated the discussion with this statement. It takes courage, but some of the wisest people around are those who tell others what they cannot know.

Computers are machines. We can assume that there is always a root cause for any undesired behaviour we may see. Even unpredictability would be a problem with root-causes.

Computers are also reliable: Unlike testing a human, testing a program running on a computer potentially, but objectively shows whether there is a bug in it. But the objectivity comes with a price: The test will never be able to say anything else about the program other than the fact that there is a bug.

Dijkstra's statement relies on concepts of objectivity and reliability. It also relies on something else, however, namely an assumption about what a bug is.

Bugs are different

Bugs in computer programs in 1969 were quite simple: The program was loaded on the computer, it was fed some input which was then processed according to the program, and when the program had completed running, results were produced and stored on magnetic tape, punched cards, paper tape, or merely printed on the console or line printer. Defining a bug in that context follows simple reasoning: A bug equals an incorrect result.

For about 20 years, I have worked on enterprise systems used by users to create, access, and manage cases of varying kind. Such systems are used to manage logistics, social services, customers, pensions, taxes, etc. These systems always have some automation running to process payments, messages, requests, claims and more. These automated processes handle interactions with other systems some of which may be internal to the organization running and developing the system, while some will be external. Compared to software in 1969, software systems today are more complex as for example the relations between inputs and outputs in a modern software system is far from trivial.

The concept of bugs has therefore become complex.

Do bugs carry meaning?

I vividly remember some of the discussions we had around the dining table at home. My father was a computer engineer in the 1970's and he read piles of code like it was poetry identifying bugs in it without even running it on the host computer. He worked on several pioneering computer projects. The highlight of his career was writing the library functions and microcode for a vector processing unit for a mainframe computer. It was custom designed for cartography.

Although I followed his path and became an engineer like him, I have always found the kind of logic reasoning he was so fluent at and which enabled him to read code the way he did difficult and unintuitive.

Fortunately, the industry had changed by the time I graduated engineering university during the 90's: My first job was programming a multimedia games title where the success of my work depended more on collaboration with the graphic designer and the composers for the music and sound effects we used than on my abilities to write perfect code. Later I changed to enterprise systems and became a tester. Where the software my father talked about at the dining table could be evaluated by its ability to produce a consistent output, evaluating software in the 90's was much more complicated. Performing these evaluation is still my profession.

Let me generalize a bit: 50 years ago, scientists discussed software as data processing. We still call what computers do "data processing", but note that the where the processes used to be linear and procedural, they are more complex today.

Dijkstra's statements limits the scope of a proof that testing can perform to falsification. At the essence, that is still what a test can do when testing is evaluating outputs based on inputs. But is that what we do today?

No. We automate that.

So why is it that testers can still be great at finding bugs in software by running it and trying out different things it is supposed to do and should not do?

You know what a bug is, don't you? You are a tester, aren't you? Ok.

Then ask yourself the following questions: What is the meaning of the bugs you find? What do the bugs mean for users? What do they mean for stakeholders? What threats to the value of the system are you looking for?

Do you still know what a bug is?

I am asking you those questions about to make you doubt what you think you know.

ANDERS DINSEN

–

Anders thinks of himself as a driver of learning and development testing complex software. His title is test manager and he work for KMD, a NEC company in Copenhagen. He leads people in agile, waterfall, and hybrid contexts, is critically minded, and enjoys the short power-distances part of Danish organizational culture as it enables effective influencing and driving quality for those who matter. He has 25 years of experience as a developer, tester, manager, facilitator, and coach, and tested his first piece of code in 1982.

Changing the meaning

Meaning is another complex concept.

It is intuitively easy, but when you start thinking about meaning, things are often messy. Let me be honest: I have found that when I thought I knew for sure what a bug meant, I was always wrong. At least to some degree.

So why am I still testing to find – prove! – those damned bugs? I do it too, so let us make it a "we" question: Why are we still testing to prove bugs?

Could we change our ways and avoid problems in testing?

My answer is that we can, and we do. I see testers changing testing every single day by asking themselves similar questions to the question I asked you above: What is the meaning of this?

The answer is often: "I cannot know for sure, but I think..."

Improving testing

A skype call with a remote tester

We had only briefly met each other before, the person in front of me, and myself. He was in fact not sitting in front of me, but in an office space 6300 km away. We were wearing headsets and staring on blurred images of each other on our screens. He was onboarded on the team before I started as the manager. I knew very little about him or his skills. I remember how I reminded myself of the cultural differences before greeting him welcome to this our first one-on-one meeting. I remember I was a bit anxious about the whole situation.

Working with people from different skillsets, backgrounds, and culture is one of the most fascinating things about working in tech.

I have worked in tech for more than 25 years now, and that kind of experience is worth a lot. I am not thinking in terms of money, but I see that being where I am today is a good place: Where others might see chaos, I often see patterns. That has made me more robust and happier dealing with complexity and uncertainty.

When I was younger and less experienced, I looked to my senior colleagues to learn from them. I often look at leaders with a combination of awe and frustration, especially when they organized meetings. I have always been impatient, and my impatience had given me some lessons in the past, so I started observing their behaviour and asked myself: "Why are they taking time out for these conversations that seem to go nowhere?"

It took a while before I understood.

Coming from different places

The conversation does not flow easily. I notice we are coming from quite different places and that getting a common understanding of his role is annoyingly difficult.

A: "The bug you reported earlier. Tell me about that."

T: "I prepared the scripts according to the specification. I had to wait a long time until the feature was released from the developer. Executing the script, step 7 failed. I reported that."

A: "Has the developer reached out?"

T: "No"

A: "I'd like you to reach out to him to ensure the bug is fixed."

There is silence on the call. My team member at the other end of the network call clearly does not feel comfortable.

My next sentence emerges with a slightly irritated tone:

A: "Testing is a critical process performed with the team, and not a matter of completing the test case and reporting whatever bugs you have found. If the developer does not understand that, we have to teach him."

Making choices

A test manager early in my testing career once asked me about my gut feeling about the system I was testing. I still vividly remember the panic I felt when she asked as I had not imagined she would have been interested in my feelings: Software should work. Software does not involve feelings. That's a different domain. That was my reasoning, the reasoning I had learnt at home by the dining table.

Her question however, shaped a learning path for me as I worked on the project for more than 3 years. The learning path involved learning what it takes to work on a big project in a big organization. It is very much about the choices we make.

Choices and decisions are made all the time: A developer decides to solve a problem in a certain way. A tester chooses to perform a specific test. A manager reports a specific metric. A senior manager decides to allocate funds.

Choices made and decisions taken by developers and testers are usually very concrete. The effects are often seen immediately. This immediateness mean that we often forget the choices we make and focus on the work carrying them out: Writing the code, performing the test, preparing the report.

Defining testing

I perform a test and find a bug. I perform another test, and things work as specified. I change an input value and repeat the test and the front-end crashes. Oops!

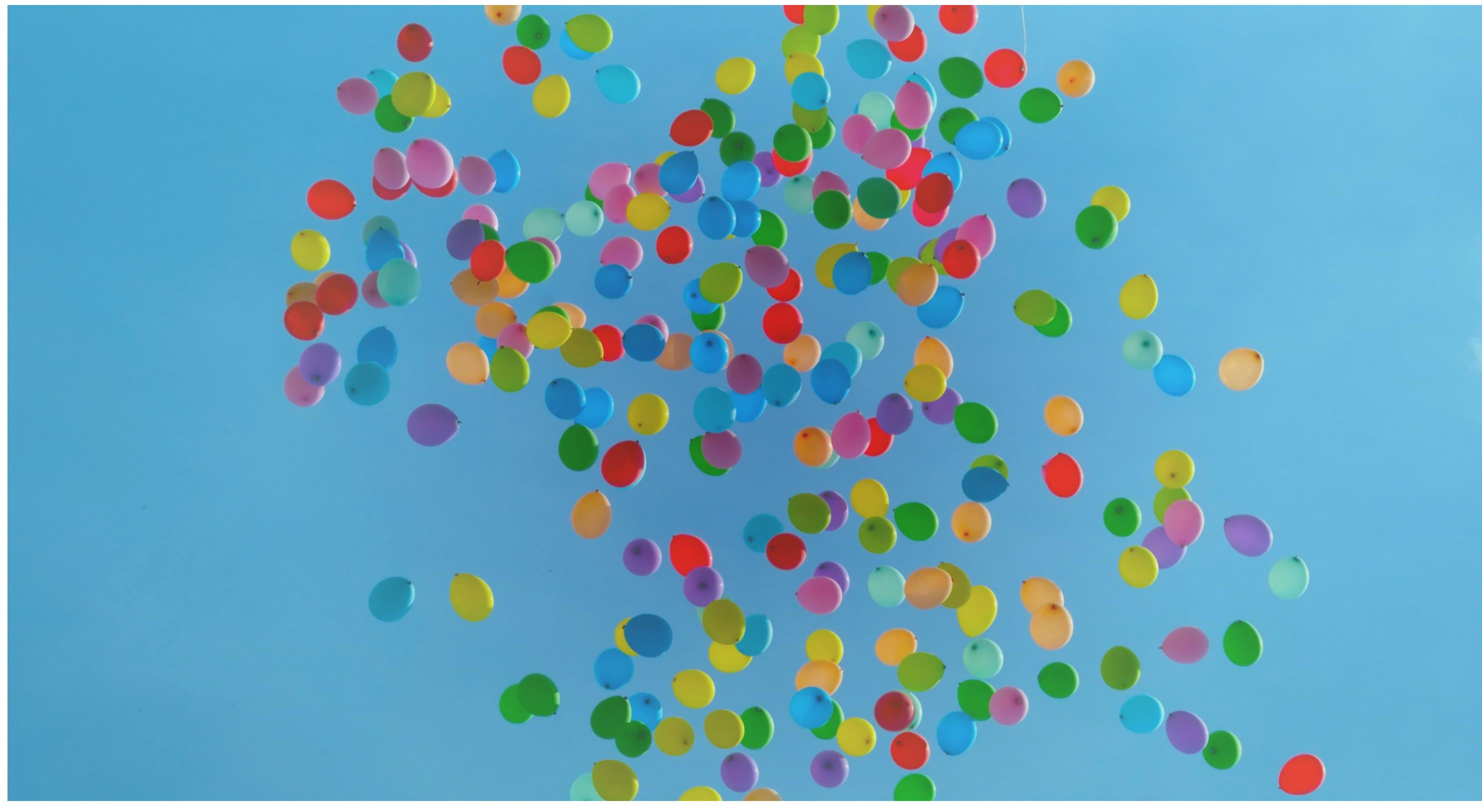
It is all happening right in front of me as a tester. It happens because I test. It happens because of the process of testing.

But let us talk about definitions: What defines the testing I? The defining thing about my testing is not the process I follow, but the choices I make in my testing.

Results matter, and the choices I make before getting these results matter more as there is always a direct causation between the choice and the outcome in testing. In the big project, for example, we were under pressure and I learnt I could sometimes make a difference by the way I was reporting my testing: Even though my testing could still only identify (prove!) bugs, there was a lot I could not report: Like the effects on users. So I focused on the people around me: and started reporting the absence of bugs as well because it helped me express appreciation of my developer colleagues' work. The social contract we had on the team helping each out had to be remembered.

Taking time to socialize with colleagues was a choice I made. It was not in my job description, but it helped me establish something else. It helped me establish a sense of meaning with what I was doing.

People do not often see that.



Should I let the tester go?

I felt I had to apologize to my tester to get the conversation back on track. I was a high-level project manager in a complex project, and I told my tester that I was ready to support him in his role. Testing was critical in this project.

"You need to approach the necessary people in the hierarchy when you find a bug. "

We agreed that I would e-mail the senior developer asking him to prioritize the bug reports.

At my next meeting with my manager, I raised the problem:

A: "It seems our testers are not allowed to address developers directly."

I got his support to change that, but not the support I expected:

M: "Let me know if I need to make a statement. We have had problems with bad testers before."

I felt worried. I did not want to let my tester go as he was good at his job. But he was not providing value to the project.

Enlightenment and gut feeling

Enlightenment is the concept of understanding things from a higher level. At almost 52 I feel more enlightened that I used to. A wise person once said that before enlightenment you chop wood and carry it

home so you can burn it and keep your family warm. So is there less work now? No, he also said, that after enlightenment, you still chop wood and carry it home to keep warm.

Testing involves lots of tedious work of managing test data, preparing testing, performing testing, noting results, gathering, and communicating details in reports, collaborating with colleagues, taking time to socialize. I still do the labour.

But as I mentioned above, I have realized something about the labour, namely the importance of the choices I make while labouring.

This is where "gut feeling" comes into play. While we often think our choices have a reason, and most of us think about ourselves as rational beings who prepare and make plans, and who mostly follow the plans we make, we must admit that that cannot be the whole truth: In the moment of actually making a choice, we often end up improvising - instead of sticking to the plan we made.

There is always "gut feeling" involved in decisions. Gut feeling is there but can be difficult or outright impossible to explain in a situation. And if we try afterwards, the question is if we are not only making up a story about our reasoning.

Facing a dilemma

So, are we as humans fundamentally irrational? I do not think so. I just think our rationality is bounded: We can think rationally, but in the moment of making a choice, our judgment is tied to the event and we cannot say if we are. Because of that, improvising to me also means taking responsibility.

I was in a dilemma with my tester: My boss saw the problem I did not want to see that my tester was not ready to make bold decisions and take responsibility. He could test according to his books, but he did not have the courage to do more.

Should I let him go? I thought about it and made a choice. Some people need support in the form of clear directions, and training before they are ready to escape the plans and best practices. I decided to teach him that it's a best practice to escape the best practices he had learned.

A new realization

Although the developer had assured me that he would attend to our bug reports, things were still not moving: Testing was slow, and bugs did not get fixed. Speed mattered: We had deadlines to meet and testing and debugging was becoming a bottleneck. I had to do something, but what? Everybody was working as fast as they could, it seemed.

One day I looked at one of the bug reports that my tester had raised. I could not make sense of it. No wonder the bugs did not get fixed! The bug reports were impossible to understand.

I felt I was at the root of the problem that had bugged me: The problem was about communication. It irritated me. I was especially irritated that I had not realized this before. I had pushed everyone to work faster and longer hours. Test more. I felt stupid.

I woke up early next morning frustrated as I realized I might have to let this guy go now. Had I wasted everyone's time?

In office I called him up and explained the problem. I then made a choice: I took time to explain and ensure he understood that the current practice did not work, and that he had to change. He was not comfortable, and neither was I, but I kept us on the path.

Afterwards I felt better. Was it not my moral obligation to make this choice? Explaining the problem as well as I could giving my tester a chance to improve. He listened, but it was difficult for him. He did not know how to make the change. He was not ready.

But he had listened and seen the problem. He probably knew his role in the project was at stake.

I offered him to rewrite the open bug reports. That was easy for me. I would also make a template to guide him and his colleagues: A new best practice in the project. He thanked me. I felt we had a process and could make progress.

We discussed perspectives for a moment: His view on bugs, and the view developers would have on bugs.

We are all different.

Epilogue

The story ends here. It turned out the bug reports was not the only problem we had in the project. There was lots of friction in the project where people were sticking to whatever best practices they had been trained at and forgetting the best practice of doubting your best practices.

But over a year or two, we succeeded improving our processes and deliver a working product. At the end people were asking themselves questions about the meaning of what they were doing. They started doubting their proofs. They started saying: "I think this is because...". And others chipped in in the discussions.

They started learning and because of that they started making more enlightened choices.

They still wrote code and tested it. Often the tests we did in my team still proved lots of bugs, but often they did not produce the obvious proof, so the bug reports had to become items of communication and discussion. To the outsider nothing changed: Development was still a job along the same lines as before as we went through ideas, concepts, designs, solutions, code, and back to testing ideas, scenarios, data, cases, scripts, and exploring. That was sad as we could not establish this way of working as a new model because what management saw was just a delivery and best practices followed.

But I knew what we did: We had more conversations about communication, collaboration, and making personal choices about things to do. And that helped us succeed.

The Germans have a beautiful word for enlightenment: Bildung. They use it in their word for education: Ausbildung. Ausbilding literally means taking your images and imaginations out of yourself, and into the social contexts in which they can make a difference.

Bildung is about having an image by your heart that you follow.

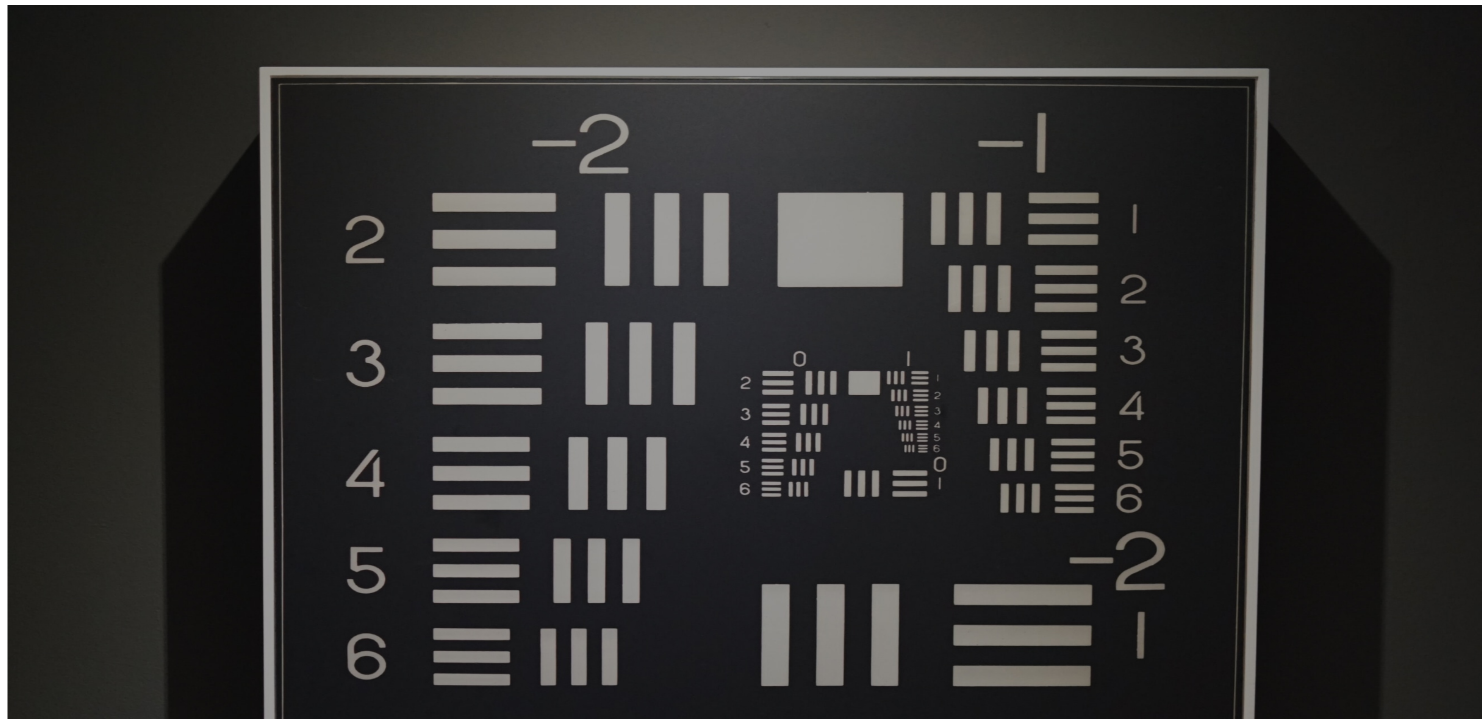
Testing has potential to teach people things about the software they are working on. Done right, testing is an education for everyone taking part. But let's use the German word: Ausbilding is about what you can do as a colleague: Help get the images they have out in the social context, make them items for exploration and discussion so they can make a difference to people in the real world

.

Your
Advertisement
Could Be Here

Your
Advertisement
Could Be Here

CONTACT SALES@TEATIMEWITHTESTERS.COM



WHAT DO WE REALLY MEAN WHEN WE SAY QUALITY?

How does your team or organization measure quality? People often equate testing to good quality or 'quality assurance', but if you have good testing practices, does that mean you have a good quality product? Many teams measure process quality and don't realize they forget about the product quality – which is what the customer cares about.

There are many things that go into a quality product, and testing is only one aspect. In this article, I explore the interaction between the development process (which includes testing) and different types of quality measures that organizations use.

There are many dimensions to thinking about quality, but I'll focus on product quality and process quality, and the correlation between the two.

How we develop our products, influences how we view our product quality. How we view our product, influences how we develop our product.

There are many contexts, and each may need a different way of looking at quality and in all cases, quality needs to be built into the product from the beginning and make the customers part of the process.

There are 3 sections to this article: 1. The Product Quality, 2. Process quality, and 3. how they might be measured.

Product Quality

Describing product quality is hard. There doesn't seem to be any easy way to do it. Gerry Weinberg has used the definition "Quality is value to some person". This seems to be the most popular definition because it's true, and it's easy. However, I think it might be too simplistic and understates some of the dimensions teams should be thinking about.

Consider your product. Is your product simple enough that you can say, I know what Sally likes in our product so if she says it's a quality product, it must be so? Most of us do not have that luxury. We have many different types of customers, and end-users, and they all look for different aspects. They have different perspectives. There are also internal customers such as product management who want fitness for use and user experience for the customer, the finance group who cares about profit, or the regulatory governance group who cares about legality. Teams have to satisfy all those needs.

There are many conflicting interests in how we define quality, and we should be looking at them, having conversations about them, and making decisions based on those needs. There are different lenses in how we view quality, including – are we getting value for our money? There are also generational differences in how we view quality. For example, younger people seem to care more about the user experience more than some older folk. I recently had a conversation with a group of people (between 20 – 30) about coffee and coffee shops. They told me they never thought about the money if they liked the experience. It's a different way of thinking about quality. There is no right or wrong – only different perspectives.

It's these differences in expectations that make the quality discussion difficult.

Process Quality

Process quality is much easier to talk about so most organizations concentrate on that – how well do they build their products. Testing activities are one way to contribute to product quality. Many people think about testing only as testing the software after it is built, but testing activities also happen throughout the delivery cycle. Testing activities:

- Provide feedback in many forms (defect reporting or code reviews are two examples).
- Identify hidden assumptions – many are because of different perspectives.
- Help identify and mitigate risks (product, business, technical are a few).
- Give information about the state of the product.
- Assess quality (assuming the team knows what that means)

Note: I do not believe that testing (investigating or evaluating) can assure (tell someone something positively or confidently to dispel any doubts) quality.

The agile testing quadrants (Figure 1), including all their variations, is a model to help teams talk about testing activities, and which ones are important to consider for their product. The testing activities in the diagram are examples of tests that a team could perform.

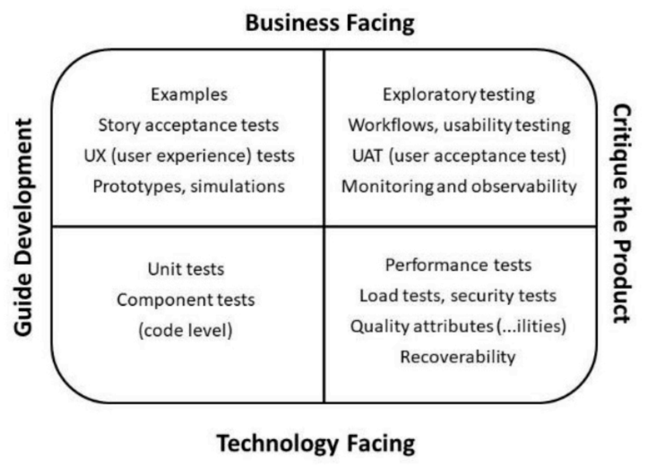


Figure 1: Agile Testing Quadrants

The left-hand side of the quadrants is about activities that happen before code is written – the focus is on preventing defects. The right-hand side is about finding defects in the code as quickly as possible. Both are needed but preventing defects in code is much more cost-effective. The top half is about tests written so that business can understand them, and the bottom half are tests that are written from a technical perspective. The business might care about the results but couldn't read the tests. If you are interested, I suggest you read more in Chapter 8: Using Models to Help Plan, from More Agile Testing.

The testing activities shown in the lower right quadrant are quality attributes that address product constraints and risks. Many quality attributes are "expected" by our customers. For example, if your team is working on a medical device, safety is probably extremely important. Another example might be data integrity – each of us carries a lot of personal data in our phones. We expect the apps we use to treat our data with caution and not to share it.

When teams don't think of different dimensions of product quality, they often overlook building it in. In every aspect of our delivery cycle, they need to consider the best way of building their product and how to build quality into their process.

As team members, we test ideas to make sure we are solving the right problem. We clarify our needs. Identify hidden assumptions and test our understanding of the problems by providing examples. We talk about the risks and which quality attributes are important and ask more questions to learn more. For example, do we need to think about the diversity of who is using our product? We may be building in limitations without even thinking about it. These types of testing are early in the cycle and are about preventing defects in code.

Testing activities during development include unit testing (TDD), code analysis, pair testing, exploratory testing (ET), test automation, even user acceptance testing (UAT). These activities help teams feel confident they are not introducing coding-level bugs. ET and UAT do address product quality by testing as different stakeholders and looking at the product holistically.

There are even testing activities that can happen after releasing to the customer – for example, testing in production (observability) and monitoring enables teams to get feedback from the customer's actions and reactions. The learning from these activities feeds back into building new features or addressing something that wasn't quite right.

So, with all these feedback loops, teams should have enough information to assess the quality, right? The question that still needs to be answered, is: "How do they know what to assess if they don't know what to measure."

Measuring Quality

Measuring quality is not easy, but we can agree that low quality is bad, right? But is that always the case? We sometimes accept lower quality because it comes at a cheaper price – that doesn't make it bad.

Teams often measure things because they are easy. For example, the number of bugs or severity of bugs tells me how bad the quality is, not how good it is. Even if absolute numbers aren't used, but watch the trend, it is not guaranteed that the product has good quality. It might be better than before, but still not great.

Accelerate has some good measures for process quality like cycle time, rework rates, etc. Other measures like test coverage tell us nothing about product quality but are also about process quality.

Organizations tend to use more qualitative measures for product quality like above based on surveys or feedback from customer support. Customer loyalty can be measured – how often do customers come back, or how long do they stay customers. For example, if a company offers a product for free, but gets income from add-on services, customer retention is extremely important.

As another tact is to ask questions about risk. Ask "What's the worst thing that can happen?" – that would be the biggest risk. Ask, "What's the best thing that could happen?". That might be something you want to measure. A good quality strategy should be based on mitigating risks.

This might be a good time to consider those quality attributes from the lower right quadrant of the agile testing quadrants. Those tests are about mitigating risk, and a step towards measuring product quality.

Margaret Dineen did a talk and wrote a blog post about using sliders for quality attributes to start the conversation within a team. Identify quality attributes that are important to your product based on the identified risks and prioritize using priorities sliders as in Figure 2. Compare, discuss, and then take it to other stakeholders until a shared understanding is reached. It can be very revealing and may open up new conversations. Once you have that shared understanding, you can decide how to measure.

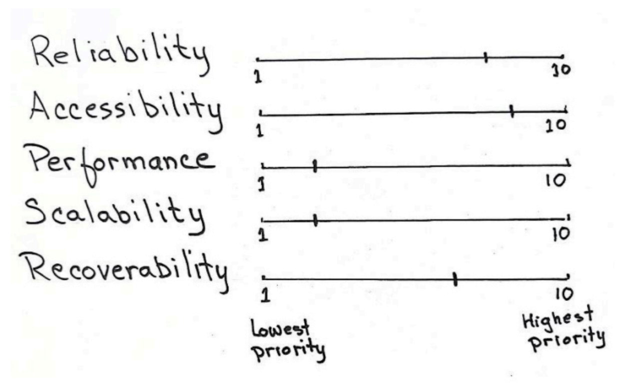


Figure 2 – Quality Attribute Sliders

Conclusion

Everyone in an organization plays a part in delivering a high-quality product:

- The **organization and senior management** provide a safe environment to enable questioning and learning.
- **Product management** provides clear priorities to enable the teams to work effectively.
- The **business** understands the 'what' and the 'why' and answers "Did we build the right thing"?
- The **delivery team** strives to build it right.
- The **individual** knows how they contribute to the quality of the product and works toward that end.

It is not enough to make sure the product works, it needs to satisfy all the needs of the customer. If your teams are not talking about quality first, you have the opportunity to start that conversation. If you don't have that conversation, how will you know what testing needs to be done? How will you know what risks to consider?

Testing supports good quality but does not assure good quality. Process quality helps but doesn't automatically make a product good.

The nature of quality is complex and diverse, so understand what you mean when you say 'quality'!

References:

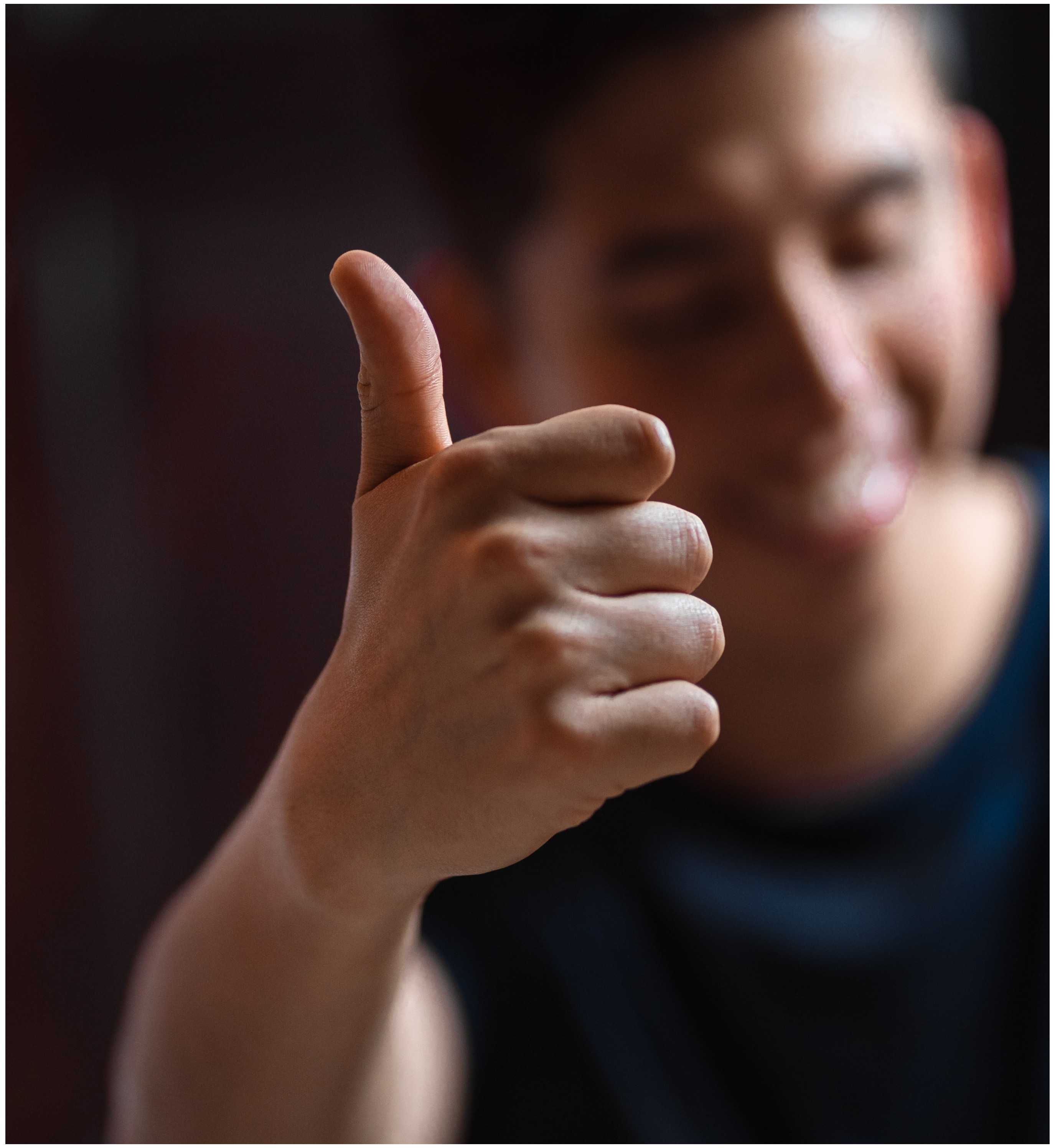
- Accelerate: The Science of Lean Software and DevOps, Nicole Forsgren PhD, Jez Humble, Gene Kim
- Agile testing quadrants, Janet Gregory, Lisa Crispin agiletester.ca/wp-content/uploads/sites/26/2014/09/Gregory_Chapter_8_Final.pdf
- Quality sliders, Margaret Dineen, https://3weststreet.com/using-priority-sliders-to-help-create-a-team-vision-of-quality/
- https://www.forbes.com/sites/nicolemartin1/2019/03/26/why-millennials-have-higher-expectations-for-customer-experience-than-older-generations/
- Gerry Weinberg's definition of quality, Quality Software Management: Volume 1, Systems Thinking, 1992

Janet Gregory *is an agile testing and process consultant with DragonFire Inc. She is the co-author with Lisa Crispin of Agile Testing Condensed: A Brief Introduction (LeanPub 2019), More Agile Testing: Learning Journeys for the Whole Team (Addison-Wesley 2014), and Agile Testing: A Practical Guide for Testers and Agile Teams (Addison-Wesley, 2009), the Live Lessons Agile Testing Essentials video course, and "Agile Testing for the Whole Team" 3-day training course.*

Janet specializes in showing agile teams how testing activities are necessary to develop good quality products. She works with teams to transition to agile development and teaches agile testing courses worldwide. She contributes articles to publications and enjoys sharing her experiences at conferences and user group meetings around the world. For more about Janet's work and her blog, visit https://janetgregory.ca or https://agiletester.ca You can also follow her on twitter @janetgregoryca or LinkedIn

Together with Lisa Crispin, she has founded the Agile Testing Fellowship to grow a community of practitioners who care about quality. Check out https://agiletestingfellow.com to find out more about courses and membership.

EXCELLENCE IN TESTING, TODAY MORE THAN EVER



What is excellence?

Most places will define excellence as outstanding, being extremely good, the quality of excelling at something, or being the best at what you do.

All these are good definitions, but when I look at testing and specifically at excellence in testing, I am referring to those special attributes we see in some testers that draw them apart from the rest of the pack.

They are not the fastest testers, nor are they the most technical ones, and usually they are not the ones finding the most bugs...

They are the testers that know how to test thoroughly but accurately, reviewing the system from the important angles and asking the questions to help the team deliver the features correctly but also quickly.

They shed light on the things that matter and leave aside those that are not relevant, helping to focus the conversation and not to confuse it with irrelevant information and noise.

They get the job done, effectively and efficiently.

Do you need to be technical?

To excel at testing you don't need to be the most technical tester, but you need to be technical enough in order to understand what you are testing.

You don't necessarily need to know how to write the complex scripts, but you will want to use technical tools to facilitate part of your tasks.

Most testers work on projects that include technical aspects, and so being technical will allow you to comprehend the system you are testing, and to understand the reasons behind the different behaviours of the systems (both the good as well as the bad behaviour).

There will also be testing projects that require advanced technical skills, and in order to excel at them you will need to acquire these skills either before or as part of your work. You should not shy away from this opportunity, instead use it as a reason to expand your virtual toolbox of skills and knowledge.

Can I excel at testing without intimate knowledge of the user?

Being an airplane pilot may help you test systems that run on an airplane's cockpit, and being a medical doctor may help you test tools used during brain surgeries. But you don't need to be a pilot or a brain surgeon or to excel at testing either of these systems.

You can use a number of approaches to understand enough about the users and their interactions with the system in order to test correctly. Yes, it helps if you have previous experience in the field, but most times this is not a mandatory requirement.

You can spend time with subject matter experts, reviewing what they do and what is important to them as they do it. You can then create personas or user profiles to help both you and your team.

When you don't have access to end users, there is the option of interviewing people within your organization who can tell you more about them, what is important to them and how they interact with the system. For example you can reach out to the Customer Support team, Sales people, or Professional Services engineers.

JOEL MONTVELISKY
–
Joel is a Co-Founder and Chief Solution Architect at PractiTest. He has been in testing and QA since 1997, working as a tester, QA Manager and Director, and a Consultant for companies in Israel, the US and the EU. Joel is also a blogger with the QA Intelligence Blog, and is constantly imparting webinars on a number of testing and Quality Related topics. Joel is also the founder and Chair of the OnlineTestConf (https://www.onlinetestconf.com/), and he is also the cofounder of the State of Testing survey and report (https://qablog.practitest.com/state-of-testing/). His latest project is the Testing 1on1 podcast with Rob Lambert, released earlier this year - https://qablog.practitest.com/podcast/
Joel is also a conference speaker, presenting in various conferences and forums world wide, among them the Star Conferences, STPCon, JaSST, TestLeadership Conf, CAST, QA&Test, and more.

You can also decide that for a specific project there is no substitute for real users, and choose to have phases of field testing to compliment the internal testing you do as part of your testing cycles.

In short, being a real user is an advantage, but not a requirement to being an excellent tester.

Is excelling at testing different in Agile or DevOps teams?

Yes and No.

Some of the skills needed in one organization will not be the same as those needed in others, but excellent testers tend to excel in most places they work. This is because testers who excel at their work are usually flexible in their approach and focused on the value of their work - not on the methodology used to achieve this value.

The value we provide can be in a number of different aspects of the testing and quality areas of the project.

We can be focused on the actual testing efforts, spending all the time interacting with the product and critically evaluating its behavior.

We can be more focused on leading and orchestrating the testing efforts of a number of testers. Breaking the tasks into smaller pieces, organizing them and assigning people to them. Reviewing the plans and findings of your team in order to ensure proper and successful testing, even when we are not the ones doing the actual testing efforts.

There will be times when you will find yourself coaching testing to players who are not usually testers in your team. This can happen if your organization adopts an “all team testing” approach, where you are tasked with empowering developers, POs, and other members of your group to plan and execute their tests, even when they have never done it before and sometimes they believe it is beyond them to do it properly.

Most times you find yourself doing a mix of all the above, while constantly performing risk-based reviews of your project in order to adapt your approach to the ever-changing constraints and goals of your team, forever focused on providing business value to the organization.

How do I learn all I need in order to excel at testing?

This is the most important but also the most gratifying part of excelling at testing (or at any other task for that matter), the approach that will make you stand out from the pack. It is the realization that learning is not a task or a step along the way, it is a way of life.

Those who excel at what they do, in this case testing, are those who never stop learning and are looking for new lessons everywhere they work and from everyone they interact with.

Those who are passionate about something can never get enough of it, and will seek new knowledge and experiences whenever they have a chance to find them

There are all the theoretical testing-related sources you can and should read, but there are also more diverse skills that are no less important such as critical thinking, story telling, observation, experimentation, and more.

Today knowledge is right in front of you, there is no real problem in finding sources, even free sources, with high quality information for you to learn. Sometimes the problem is how to select from all the different sources to review - but this falls under my list of good problems to handle.

If you are reading these words, it means you already found an excellent source. I am writing this article on the occasion of the 10th anniversary of Teatime with Testers, one of the most amazing sources of knowledge and information about real-world testing available today. Lalit and his team have gathered countless articles from many authors and testers around the world, all of them sharing their knowledge, their experience, and their passion for testing with the rest of the world.

I want to congratulate the Team on a job well done all these years, but most of all I want to thank them for their efforts and for making the testing world a better place, one edition at a time! May we all enjoy many more editions of Teatime with Testers in the years to come!!

A PANDEMIC ISSUE

ILEANA BELFIORE

Quality obsessed and Agile enthusiast professional. Software tester by day, wine taster by night.

Multilingual writer. Compulsive questioner. Slow and deliberate thinker.

Do you know what pandemic and software testing have in common? If not, taking into account that the likelihood of you never having heard about the former is pretty low, for the purpose of this piece of writing I will assume that I need to go deeper into the latter.

This is weird because, if you are reading this magazine, I guess you are already familiar with software testing.

Well, I hope I don't bore you too much...

So, if you have been working in this industry for a decent amount of time, I bet you have had to deal with **counting disease** at some point in your working life.

As brilliantly explained by Michael Bolton throughout his enlightening course on **Rapid Software Testing**,

“Counting tests (and requirements, bugs, and other measures derived from these counts) is an endemic means of deception in the testing business. Some well-known testing experts promote this form of deception; testers then practice it, and project communities have learned to ask for it. [...] in the testing business, we are infected with counting disease - we are constantly counting test cases, requirements, lines of code, and bugs.

Yes, it happens all the time.

Most managers want to count things. Which is quite understandable. The problem is that they usually tend not to spend enough time thinking about the right things to count. Or fail to realize that some things should better be assessed, rather than measured(1).

By the way, do you want to see a situation start going off the rails? Make your metrics become goals.

Yes, I believe Goodhart was absolutely right: when a measure becomes a target, it ceases to be a good measure.

Do you want your project or product go even worse? Just add money to the equation.

Yet again, it happens all the time.

Some managers may decide to start counting test cases, for example. And they usually know pretty well that they don't need to reinvent the wheel: just look at the percentage of test cases executed, the percentage of test cases passed, and, at most, a few variations around the same theme.

Now, if the fact there's usually no too much value in these metrics might be not self-evident to some people, the moment such (questionable) information starts being confused with a goal, it turns into something completely meaningless. Do you want your test cases to become automatically irrelevant? Make sure everybody understands the goal is to make them pass. Your test suite will magically start returning perfect green reports and will be quickly converted into a deceitful, unreliable, meaningless tool. (2)

Similarly, some managers may decide to count bugs too.

In doing so, chances are they will fall into the trap of choosing between two different (yet equally toxic) approaches either rewarding people based on the number of bugs they find - hence fostering the intentional injection (and consequent exposure) of (usually shallow) bugs into the product(3) or blaming them for the excessive amount of bugs they have uncovered - so discouraging them from doing their job.

I'm not even sure what's worse...

Other managers may decide to make people put a lot of effort into automating test cases at the GUI level, for instance.

After a while, they may even celebrate that, at last, there are fewer bugs in their software product.

They may never realize that the reason behind that apparently good news might be that, on one hand, their (poorly) automated (yet really difficult to automate) test cases are not able to find any bugs, and, on the other hand, nobody is performing genuine exploratory testing - that is, nobody is really trying to uncover all those bugs that, in spite of not being exposed by an ineffective tool/strategy, are still there.

I wonder why opportunity cost(4) is so important and so underrated a concept...

After a longer while, those same managers may start acknowledging that a lot of new bugs are now surprisingly being reported (hence exposed) by customers or final users. Their reaction to this bad news will likely result in more ineffective (yet expensive) automated test cases.

They actually seem to believe that more is better [].The idea can never be wrong, it is just that people are not doing it with sufficient vigor.[5]

In other words, they repeatedly fail to understand that when something doesn'tt work, the answer is not to keep doing it with even greater fervor. The real answer is to stop doing it and try something else instead.[6]

But how is this related to the current pandemic? - I'm hearing you asking.

Well, if you have had a look at the last couple of footnotes, you might be guessing already...

Anyway, *let's start talking about a PCR test*, for example. Something not that different from a test case after all.

The main issue I have with it is that, even though it's nothing more than an indicator, it is being treated, trusted, probably even venerated as though it was an accurate diagnostic tool.

Yet, as far as I understand, as an indicator of a potential infection, it should rather be used to help physicians make better decisions.

Even the World Health Organization seems to officially agree with me7.

Does this patient presents symptoms compatible with coronavirus? [8] Let's do a PCR test. Is the result positive? Let's do some other tests to confirm the cause of the symptoms. Is the result negative? Let's do some different tests to try to figure out what's going on here.

Otherwise, have this patient's symptoms (e.g. a joint pain) nothing to do with coronavirus? Don't even consider ordering a PCR test. Do whatever makes more sense to discover the reason behind the symptoms instead. Isn't this just common sense after all?

Finally, also taking into account the high cost involved, what's the point of performing a PCR test when there are no symptoms at all?[9]

The problem is that, as I said before, the infamous PCR test is being improperly used as though it was an accurate diagnostic tool. It is not(10). As a matter of fact, like practically all tests, it might be affected by false positives(11) or false negatives .

Anyway, even if it were more reliable, I want to stress once more that it is just an indicator.

As such, its value alone cannot (and should not) be misused to declare the healthy or unhealthy condition of a patient.

Think about a standard blood test.

It usually includes a lot of indicators.

An out-of-range value in one of them is never enough to declare a patient ill.

What an out-of-range value usually does is triggering some questions, an investigation and probably the scheduling of some other tests.

I wonder why an indicator alone is now being methodically (yet inappropriately) used as a diagnostic tool, actually like a weapon of mass destruction[12] to close shops, restaurants, gyms and schools; to overlook other (often more serious) diseases; to shut down the economy; ultimately, to scare people.

Meanwhile, *language* as well is worryingly changing.

I don't really understand why for so many people testing positive to a PCR test has become a perfect synonym of *infected* with coronavirus and *dangerously contagious*. It isn't. It shouldn't.

Or why nowadays mass media are so fervently tarring heterogeneous groups of people with the same brush.

I mean, should someone dare to question any of the methods applied to counter the effects of the current pandemic, they would be immediately and without mercy labelled as negationist, conspiracy theorist, anti-vax, etc.

How come? Is this the end of critical thinking?

Also why some words/expressions (such as lethality, mortality rate, morbidity[13]) are hardly uttered by news reporters (especially in comparison with previous years' data).

Or why a powerful and reliable tool like EuroMOMO[14] seems not to be among their favorite official sources of information.

By the way, why aren't they asking questions instead of taking data for granted? It seems to me that they are now officially in the business of spreading fear by means of incomplete or misleading figures.

Oddly enough, they don't call this fake news, though.

To make things worse, similarly to what I have mentioned before about rewarding testers based on the number of bugs they find, in this case too, the moment *money* was added to the equation that is, the moment public administrations or health systems start receiving funding in direct proportion to the number of positive results[15], PCR tests became automatically/practically irrelevant. Goodhart's ears must have been burning at that point...

But let's talk about lock-downs now.

I'm going to quote Dr. Malcolm Kendrick again here(16).

"no study is ever done to find out if the idea works, or not. It is just conceived to be so obviously beneficial, such common sense, that there would be no point in wasting time and resources trying to prove it works. [...]

The most expensive, invasive, and potentially destructive medical intervention ever attempted by humanity. Was there any evidence from anywhere, in history, that lockdowns would work? No, there was none."

Nevertheless, ***"The idea has become the truth. Its proponents now demand that those who doubt the efficacy of lockdowns prove that they dont work. However, I dont believe its up to those who dont believe that lockdowns work, to prove that case.***

The starting point, for any scientific hypothesis, is for the proponents to disprove the null hypothesis. Demanding that those who believe something may not work, to prove that it doesn't, is to turn the scientific method upside down. You can never prove a negative."

No, to turn the scientific method upside down doesn't seem a good idea indeed...

Speaking of negatives, as pointed out by Jose Gefaell[17] within one of his reports about the current pandemic(18), since lock-downs have always - at least in the UK and in Spain - been applied when R0[19] was already negative, it would be extremely difficult to demonstrate any positive impact of this kind of measures (20).

Still on negatives and back to software testing, if you agree and if you are reading this magazine, I hope you do that nobody can demonstrate the absence of bugs, only their presence, you should also agree that requiring a negative PCR test to enter a country, to attend an event or to "safely" perform an activity does not make sense at all(21).

Now, don't get me wrong: I do believe the virus exists, of course. Yet, I don't think it's a good idea to get obsessed with it, or to misuse questionable indicators in order to justify that obsession(22) , to such an extent that we are not thinking about anything else.

.That's not life. That's ***infodemic-driven mayhem***(23).

All in all, I guess my answer to the leading question of this article should be pretty obvious now.

Meaningless metrics: definitely a pandemic issue...

References:

[1] For a deep discussion about this topic, please find Assess Quality, Don't Measure It. https://www.satisfice.com/blog/archives/, by James Bach.

[2] I find this issue so worrisome that it even triggered a series and a hashtag of mine about software testing https://www.ileanabelfiore.me/whats-the-goal-of-your-bloody-test-suite/

[4] Opportunity cost means, basically, that we won't be able to do that potentially valuable thing because we're doing this potentially valuable thing. - excerpt from The Sock Puppets of Fomal Testing (https://www.developsense.com/blog///sock-puppets-of-formal-testing/),by Michael Bolton. Also in Value and Cost in Automated Checking or *Don't Fall into GeMPuB* (https://www.linkedin.com/pulse/value-cost-automated-checking-dont-fall-gempub-michael-bolton/): "Opportunity cost: the degree to which stuff you're doing displaces your opportunity to do other stuff that you might value more."

[5] Excerpt from *Does Lockdown work, or not? (https://drmalcolmkendrick.org/2021/01/27/does-lockdown-work-or-not/)*, by Dr. Malcolm Kendrick.

[6] Excerpt from What is left to say? (https://drmalcolmkendrick.org/2020/12/30/what-is-left-to-say/), by Dr. Malcolm Kendrick. Yes, again. A rare source of common sense nowadays.

[7] "Most PCR assays are indicated as *an aid for diagnosis*, therefore, health care providers must consider any result in combination with timing of sampling, specimen type, assay specifics, clinical observations, patient history, confirmed status of any contacts, and epidemiological information." emphasis added. – excerpt from WHO Information Notice for IVD Users 2020/05 (https://www.who.int/news/item/20-01-2021-who-information-notice-for-ivd-users-2020-05), issued on January the 20th of .2021

[8] After all, even according to the Centers for Disease Control and Prevention (https://www.cdc.gov/coronavirus/2019-ncov/downloads/Factsheet-for-Healthcare-Providers-2019-nCoV.pdf), "The CDC -nCoV Real-Time RT-PCR Diagnostic Panel should be ordered for the detection of COVID-19 in individuals *suspected of* COVID-19 by their healthcare provider." [emphasis added.]

By the way, I would say that considering everybody suspicious instead doesn't usually correlate with effective detection, does it?

[9] "Mass testing is simply causing mass panic and achieves absolutely nothing." - excerpt from Excerpt from What is left to say? (https://drmalcolmkendrick.org/2020/12/30/what-is-left-to-say/), by Dr. Malcolm Kendrick. Once more, I definitely agree with him.

[10] By the way, also the Centers for Disease Control and Prevention seem to agree with me (https://www.cdc.gov/coronavirus/2019-ncov/downloads/Factsheet-for-Healthcare-Providers-2019-nCoV.pdf) "Laboratory test results should always be considered in the context of clinical observations and epidemiological data … in making a final diagnosis and patient management decisions."

[11] "In previous epidemics, health authorities voiced concerns that false positive results from PCR-based tests could harm both the individuals tested and the ability of government agencies to assess the outbreak, and they adopted measures to limit the occurrence of false positives. For example, the World Health Organization and the U.S. Centers for Disease Control and Prevention limited PCR-based testing to individuals with a high probability of infection those with symptoms and/or significant exposure and usually required confirmation of positive results by a second, independent test …. These warnings and requirements are absent from the same organizations guidance on SARS-CoV- testing." – excerpt from Diagnosing COVID-19 infection: the danger of over-reliance on positive test result. (https://www.medrxiv.org/content/10.1101/2020.04.26.20080911v3.full.pdf)

[12] Here you can find a post of mine inspired by this concern (https://www.linkedin.com/posts/ileanabelfiore_ihelppeoplestopbuyinglies-activity-6752156557372338176-VCIQ/)

[13] Some other factors starting with the number of tests carried out that, in my opinion, should be uttered more often can be found within the previously mentioned Dr. Malcolm Kendricks article titled "Does lockdown work, or not?"

[14] https://www.euromomo.eu/

[15] As explained within the Spanish Official State Gazette BOE published on June the 17th of 2020 https://boe.es/boe/dias/2020/06/17/pdfs/BOE-A-2020-6232.pdf for example.

[16] Excerpt from Does lockdown work, or not?

[17] https://www.linkedin.com/in/josegefaell/

18 https://www.dropbox.com/s/s9muiwsixqy9b7e/Excess%20Deaths-1st-2nd-3rd-Waves-Spain-29Jan2021.pdf?dl=0 , p. 26 and 19 https://en.wikipedia.org/wiki/Basic_reproduction_number

[20] Or, in Dr. Malcolm Kendricks words: "Unfortunately, once you introduce a medical intervention that affects everyone, everywhere, you have lost the possibility of carrying out a controlled experiment of any sort." - excerpt from Does lockdown work, or not?

[21] As a matter of fact, according to the Centers for Disease Control and Prevention (https://www.cdc.gov/coronavirus/2019-ncov/downloads/Factsheet-for-Healthcare-Providers-2019-nCoV.pdf) "a negative result does not rule out COVID- and should not be used as the sole basis for treatment or patient management decisions."

22 "Paradoxically, human beings, when compelled to act, learn to justify a chosen course with an assurance unwarranted by the evidence for the course chosen. " - excerpt from A Chai he Rece (https://bernardlown.wordpress.com/2011/02/03/a-chair-to-the-rescue/) by Dr. Bernard Lown.

[23] And I haven't even mentioned masks! Well, I would need another article just to talk about this topic...

INTERVIEW

Is the craft of testing dying? What does it mean to test regulated software? Did testing lose it all to Agile? Hear from Griffin Jones.

Hi Griffin, it has been an honor to interview for this special issue of Tea-time with Testers. Please tell us how have you been and what are you up to?

It is nice to talk with you again - I've been very busy.

Since we last spoke, I have continued as a fly-to-the-client, consultant - working in my venn-diagram combination of testing, agile, and regulated.

Six years ago, as an agile coach began working thru BigVisible Solutions, SolutionsIQ, and eventually Accenture on business agility transformations.

Currently, my typical client work is eighteen months, individuals and teams, and up to the C-level in their agile journey. Testing is a part of that ecosystem.

My specialty is helping individuals and teams unfold and move toward becoming the best versions of themselves in their situations and role. I look for the people that have that glow about them - that are ready and willing to level up. I find what they often need is a person to give them some time, attention, and a bit of direction.

It is a bit like being the Wizard of Oz, I help people rediscover the things that already have hidden inside themselves.

When I think of Weinbergians and regulated software, Griffin Jones is one name that my brain never skips. Would you like to tell us more about Griffin the Weinbergian and Griffin the expert on regulated software?

After college in 1987, I started working at Eastman Kodak as a tester - on a giant project that would digitize, index, store, and retrieve all the historical physical paper records of gigantic organizations. I stayed at Kodak till 2007, focusing on testing the initial imaging digitization of different business verticals. Computed radiography is an example of my regulated work, while movie special effects software is an example of 'cool', but unregulated work. The big point is that I sought out broad technical and line-of-business experiences while developing a deep capability in medical devices.

That broad experience led to my ideas about testing evolving (described with some exaggeration) from a Boris Beizer function point proof, with elaborated and traceable requirements, a Capers Jones metrics program, and a Quality Assurance gatekeeper's veto mindset - toward an Exploratory Testing mindset as described by Cem Kaner and others. Bret Petticord's [Four Schools of Software Testing] is a good take.

In the early 1990s, I discovered Jerry's books, and his ideas fit for me. But I could never fully implement my insights at Kodak.

In 2007 I quit Kodak to work at the startup iCardiac Technologies (which designed and delivered cardiac safety analysis for pharmaceutical clinical trials) where I led their software testing effort and eventually became the Director of Quality and Regulatory Compliance. During my time at iCardiac, Jerry and Cem Kaner were both presenting at CAST 2008 in Toronto - and I decided to attend the conference and Jerry's workshop.

GRIFFIN JONES
–
Griffin is Sr. Manager: Agile Consultant / Coach - Specializing in regulated industries (FDA and Financial Services)

Jerry and the many people from our community attending the conference - made the event magical for me. I was hooked. I left iCardiac and became an independent consultant, binging on Jerry and our community - trying to make up for a lost time.

I directly interacted with Jerry of the following years by attending multiple AYE conferences [https://www.ayeconference.com/] , attending the Problem-Solving Leadership workshop (PSL) [https://www.estherderby.com/workshops/psl/] , and the Change Artistry workshop [associated book: https://leanpub.com/changeartistry]. My recent work with Jean McLendon and the work of Virginia Satir, and the Organization and Relationship Systems Coaching (ORSC) is just me following Jerry's footsteps.

Jerry has been the most influential person in my life that I did not share a home with.

On the day I learned of Jerry's death, I was working with a new client in Chicago. Throughout the day, I noticed each time I did a Jerry-like thing - and would remember with sadness-of-loss the past interaction where I learned that specific consulting move from him. I was struck by how often that happened. By the end of the day, those memories were still tainted with loss. But, there was also deep soulful gratitude to him for what he gave me: for what I had learned, and for the gift of his presence with me during that time in my life.

He profoundly changed me.

My regulated industry experiences are more straightforward.

First, I think I inherited some of it from my father who was deeply involved in the nuclear power industry at the local power plant as a trainer, safety officer, problem investigator, and designated company interface to the regulator.

Half of my career at Kodak was in Health Imaging. Plus I was often given a side-mission to create the Software Development LifeCycle for the program. I happened to be in Health Imaging in the middle 1990s and helped adapt the FDA Quality System Regulations (QRS) when they became effective.

I left Kodak to become employee #11 at iCardiac, where I was part of the team that created and implemented the entire FDA-compliant QRS. Multiple employees of the company were also Special Government Employees of the FDA which was our first customer. We built the QRS to allow us to implement agile practices in a compliant way. During this time I joined the Regulatory Affairs Professional Society (RAPS) and became the Director of Quality and Regulatory Compliance. iCardiac was incredibly successful, and I was hosting pharmaceutical onsite audits monthly. I started speaking at conferences and became a co-host of Workshop on Regulated Software Testing (WREST). My story of blending agile with regulatory compliance attracted people's attention, so I left iCardiac and became an independent consultant.

I worked as a consultant in multiple roles with medical device, pharmaceutical, and financial services companies in a blended testing and regulatory affairs role.

In my agile coach role at SolutionsIQ / Accenture, I am often the liaison between the regulatory and compliance organization and the larger business agile transformation. I am fluent in "regulatory".

How does testing differ when it comes to regulated software? Do testers assure quality in the regulated software domain?

Well, when it goes well, testing in that context has a four-part mission:

a. Gather evidence to support a "you built what you said you built".

b. Gather evidence to support that "it produces the desired outcome".

c. Actively gather evidence about risks to the testing effort and the business.

d. Weave all that evidence into a coherent story that your management and regulators will understand.

Elaborating on each of those:

a. Gather evidence to support a "you built what you said you built"

a.1 Which is like 'verified' to use the FDA legal meaning.

a.2 Which is like an attempt to understand in a very Cynefin ordered domain way.

a.3 Which is like to behave very ISO 9001-like.

a.4 Which is like to adopt a Phillip Crosy 'conforms to requirements' definition of quality.

[Cynefin: https://www.youtube.com/watch?v=N7oz366X0-8]

b. Gather evidence to support that "it produces the desired outcome".

b.1 Which is like 'validate' to use the FDA legal meaning.

b.2 This is like an attempt to understand using a very Cynefin complex domain meaning.

b.3 Which is like to adopt a Joseph Juran 'fitness for use' definition of quality.

c. Actively gather evidence about risks to the testing effort and the business.

c.1 Gathering evidence of the active exploration of risks to the testing effort.

c.2 Gathering evidence of the active exploration of producer's risks to their business (using ever harsher Karl Popperian assumptions).

c.3 Gathering evidence to help answer the question 'can we operationally execute all of this in an acceptable way as an ongoing business?".

d. Weave all that evidence into a coherent story that your management and regulators will understand.

d.1 Construct a coherent story of what it all means, in a very Michael Bolton Braiding the Stories way.

https://www.developsense.com/blog/2012/02/braiding-the-stories/

These missions work for non-regulated just as well. They are just good engineering.

But when it goes bad, one (or more) of those four missions has failed:

a. We didn't build what we say we built.

b. It doesn't work.

c. We can't function as a business doing this.

d. We can't explain the story of what happened in a way that our regulators will let us stay in business.

Regarding "assuring quality", well it depends. Some individuals will take on that mantle, but it is an overreach - as Brian Marik explained in The Testing Team's Motto. http://www.exampler.com/testing-com/writings/purpose-of-testing.htm

But, sometimes the organization is constructed around that belief, and the QA organization acts like an uber-program-manager. Brett Petticord described it in The Four Schools of Testing as The Quality Assurance School. That pattern is tending to be becoming rarer, except in the case where the organization had a near-death encounter with their regulator.

Would you say that in a regulated software context, there is very little room for creativity and free-thinking for testers?

[Laughs out loud] Yes and no.

Magic happens when you can be creative in a compliant way. The important problems will require creative compliance to solve.

Yes, because there is the desire to gain more understanding and reduce risk. It often takes creativity and free-thinking to make that happen.

No, because working in the "compliant" way tends to be slower and more expensive. And if you are going to share this work as evidence, you are not prepared to share the unedited versions of your attempts with all the mistakes and failures transparently included.

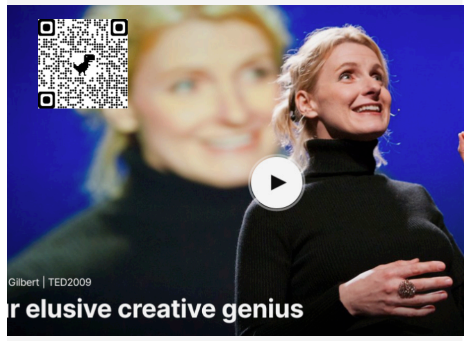
And frankly, some of the work is grunt work that you just have to grind through. It is part of the Cynefin Simple domain.

Part of the problem is the language and zero-sum thinking around control. There is a wonderful TED-talk, Lead Like a Great Conductor, by Italy Talgam - that illustrates control and creativity operating at different levels. See 14:50 to 16:47 of the video. [https://www.ted.com/talks/itay_talgam_lead_like_the_great_conductors]

Look for opportunities to put down the baton, and give others the freedom to contribute in their own way.

It is also expressed in *Turn the Ship Around*, when the Captain "vowed to never give another order ever again". Clarity of Intent, Competence - and then you can give people Freedom. [https://www.youtube.com/watch?v=psAXMqxwol8]

Creativity: My creative muse (or genius) is a trickster and waits for me to not have pen and paper at hand: See 11:50 to 14:20 of Elizabeth Gilbert's TED Talk.



"

A twisted but enlightening story about regulated industries would be to imagine a Cargo Cult where the desired outcomes seem to always happen. Or a magician that can only perform a trick successfully if there is no audience present.

How does the strategy for automation in testing get affected in the case of testing software in a regulated environment?

There are a few key points about using automation in this context.

The speed, precision, and repetition that tools can offer if used well can be an enormous benefit.

But before they can be used, every tool in the entire software development life cycle and development stack needs to go through basic processes of: vendor qualification and product selection, installation qualification, operational qualification, and production qualification. These answer the questions:

a. Who and what are you making/buying, and why are they a good choice to choose them?

b. Can you install and configure it in your larger system?

c. Does it operate (for the aspects we care about) as they described it in their documentation?

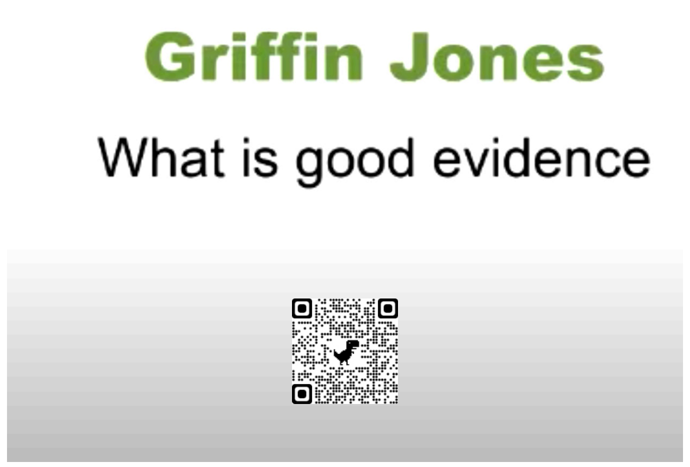
d. If you run it in your actual production environment, with production data, and production standard operating procedures - do you get acceptable results?

e. Keep documentation describing what you did for all the previous steps and what decisions were made, and by whom; have it at hand and reviewable by a third party.

All that work is before you use the tool for its intended purpose. Using the automation tools in these ways and preserving an authoritative and complete record of results is part of the significant work of being compliant.

The best use of automated tools that I have seen are hybrid-systems that combine the strengths of tools with human judgment, where the result is better than either alone. The tools don't fully replace the humans, rather they augment the judgment and evaluation capabilities of the people.

[This is elaborated in my talk What is Good Evidence?]


Griffin Jones
What is good evidence

You have great experience in the testing field as well as in coaching teams for Agile. How do you think Agile has affected the testing profession and the industry perception of it?

Testing won but didn't realize it.

Left-shift is a win for testing. Continuous integration and continuous delivery are a win for testing. TDD and BDD are a win for testing. People with T-shaped skills on teams is a win. Story acceptance criteria is a win. Frequent demonstration of valuable working software (to your customers) is a win. The whole team-ness is a win. Development and management have taken two big steps in the direction of "getting better incremental information via experimentation about what is really happening", aka testing, is a win. But management will frame and explain it from their own points of view, so Testing doesn't notice the win.

The best people and teams I have worked with were T-shaped with one or two deep specialties, multiple secondary skills, and always a desire to learn, teach, share, and pair. They tend to identify themselves to outsiders as "team members". The formerly-known-as-testers on these teams all bringing something extra to the party: facilitation skills, technical skills, UX-design, analysis skills, the ability to influence management, understanding of the business, relationships to others in the broad organization, new-eyes -- but they are all not one-trick-testing-only-ponies.

The best situations seem to disperse the testers into delivery teams, while also gathering them back together in testing communities of practice - where they can hone their craft. It is described as Business Agility in Scaled Agile Framework (SAFe) - a responsive value delivery structure along with an organizational hierarchy to give some permanence. [https://www.scaledagileframework.com/business-agility/]

I remember having a discussion with you around feedback loops and the system collapse (How Software Is Built - Jerry Weinberg reference). In the regulated software industry, have you seen the system collapsing? If yes, what were the reasons, and if no what protected them?

Run-away systems either collapse or explode. In Managing the Risks of Organizational Accidents, the author James Reasons cites multiple examples of critical systems failing - planes crash, banks fail, chemical plants explode, etc. So catastrophe still happens, just infrequently or the consequences are contained (and the public don't notice the near-miss).

Risk equals probability times impact. Regulated industries are regulated because by the nature of the work, they generate high enough potential risk of unacceptable high costs (especially to innocent and uninvolved bystanders), that society has imposed special conditions on the producers of those risks.

The people responsible for designing and running these systems have done a good job preventing single-points-of-failure. They recognize the critical nature of human communication and have instituted Crew Resource Management procedures and training. Since failures of the system are so infrequent, they actively search for and investigate "near misses". Often, they discover that the action by the human in the loop was what prevented a catastrophe. Or not. [https://www.youtube.com/watch?v=kjLrZ2SDDaU]

[Beyond The Black Box: The Forensics of Airplane Crashes by George Bibel; Organizational Accidents Revisited, by James Reasons; Normal Accidents: Living with High-Risk Technologies, by Perrow; Aircraft Safety: Accident Investigations, Analysis & Applications, by Krause; Crew Resource Management: Principles and Practices, by LeSage, Dyar, and Evans.]

I assert the recent Boeing 787 Max catastrophe was just the most visible outcome of a series of little failures with a root cause of the collapse of Boeing's engineering culture. [See the Swiss Cheese Model: http://blog.enterprisetraining.com/swiss-cheese-accident-causation-model/]

As our system grows bigger and older, a growing risk is the combination of: "your system and all its' emergent properties in the present is infinitely larger than your head was in the past, or is now - so you can't be absolutely sure what is going to happen"; and we are reluctant to retire anything - maintenance on and extension of existing systems can be very risky. Especially the low-risk changes.

A tiny little close-to-zero technical enhancement to a text editor on the Therac-25 was a technical cause of the device unexpectedly entering calibration-mode, massively irradiating six people and killing them.

Reading the IEEE account of what happened and why it happened should make you weep. https://ieeexplore.ieee.org/document/274940

Do you think software testers by nature of their work and skills are best equipped to help prevent systems from running into collapse mode or not?

Yes, in the sense that they are at the intersection of where the questions are first asked, and where the resources and inclination are to investigate.

Tester's super power is a belief that "things can be different" than they were described, and a strong inclination to run an empirical experiment to support or refute a theoretical analysis.

Testers need to be able to observe without preconceptions the produced system, the production system, and the human system doing the production.

Management giving license to testing to comment on all of this is a political problem for the organization. Often the testing group's mission is shriveled because the testers lack the technical skills to investigate and the political skills to present the results. Or, management fears that they lack the skills to effect a change in the organization in response to what testing would find. So the questions go unasked.

Jim Bullock's Big Book of Testing elaborates on this idea. [https://www.linkedin.com/in/rarebirdenterprises/]

As someone who has closely worked in an industry where a small failure in software can risk the patient's life, what is your opinion about the "Good enough quality" notion with which tech companies are trying to operate with, more and more (apparently)?

I have a few takes about "Good enough quality":

First, context matters. Second, search for the palmed-cards. Good enough for whom? For what values of 'good'? For what purpose? For what costs? Compared to what? Who gets to decide? Who is impacted?

I think Jerry's definition of Quality helps untangle the question: "Quality is value to some person that matters, (at some point in time)". Who matters, or doesn't matter? When do they matter? What does each of them value? Why do they matter?

Using that analysis, it seems to me that the general use of "Good enough quality" is about producers deploying what they think is a minimal viable product in a marketplace sooner versus later, and getting faster feedback. They have a gold rush model of how the marketplace works. They believe that consumers will tolerate a minimum viable product.

The extreme version of "good enough quality" is not really a viable commercial option in the regulated spaces because the regulator has the capability to block sales, confiscate your goods, freeze your business with investigations, fine you or just take your money, and put you in jail.

Regulators are incepted to search for bad actors and expand their regulatory power and scope.

Do you think our industry is suffering from a Cargo Cult problem especially when it comes to deciding about testing culture? (E.g. Google, Microsoft, Facebook test their product so and so way, let's do that because it must be the best thing to do.)

Success begets imitation, and imitation is the sincerest form of flattery.

The fast follower is a strategy that can work, but to work it has to be more than buying a template, or adopting the processes of others. They are not magical items that sit on our shelf.

In agile coaching, this is the "they are doing agile, rather than being agile" problem. It starts with mindset, accepting "this is my problem to solve", and engaging with the problem, including emotionally. It is like the scene from Bruce Lee's Enter the Dragon, the actions need emotional content. https://www.youtube.com/watch?v=sDW6vkuqGLg

If they are Cargo Culting their testing culture, they are focusing all their attention on the tangible artifacts, aka, the finger - versus the moon.

Choosing a high stakes strategy of half-forgotten, poorly understood, and badly executed rituals for channeling dangerous forces rarely leads to successful long term outcomes. But maybe we'll get lucky this time.

I get a lot of 'you must do test cases in a medical / life-critical environment because of auditors/FDA etc' - is that true? Were you able to use other forms of testing documentation other than test cases which the auditors were satisfied with?

The short answer is, "No, the regulation does not explicitly call out for a thing called 'test cases'".But the longer answer is, "maybe".

Maybe your company has painted their policies and procedures into a corner, and "test cases" is the only black-letter-of-the-law way to comply with your organization's policies and procedures.

Maybe people can't conceive of a different way, or can't convince an authority in the company to document a limited waiver to allow a different way. Maybe your regulatory posture and relationship with customers and the regulator demands that you use test cases.

Maybe you can't explain why a different way could be acceptable and address the concerns of those that matter that has concerns.

The most elegant solution I have implemented that survived audit was to embed inline a short test charter, description, and acceptance criteria with each requirement/story so they were tightly coupled. All testing was video recorded, both the tester and the system. All electronic test data, output, and logs were saved, analyzed, and preserved. Ditto with physical results.

The key in my opinion was having the tester confidently explaining real time what they were doing, why they were doing it, what system results they saw including anomalies, and what the meaning of the results was.

Pre-written text cases are wishes for what I hope will happen. What actually happened is bigger than a confirmation-bias "passed" test case checkbox. I want to see and hear a recording of what really happened. I want to watch the tester synthesizing meaning during the performance from tacit knowledge, observations, and pre-analysis and preparation. And express it so I can record it and share it with a regulator.

I have also seen traditional agile scrum stories with acceptance criteria pass audit if underneath the Jira ticket was enough evidence. Again, did your performance generate enough evidence, can you explain the evidence, does the story make sense?

Is it permissible to attempt, do you understand the different stakeholders and their concerns, can you design a test and make preparations, can you honestly and skillfully execute it, do you address the concerns raised in question #3, do you collect enough evidence of what was done during the execution, and can you construct a credible story with all of that information that a third party could examine and arrive at a similar conclusion? If all the answers are yes, then consider it. Like all things, consider asking for help.

You are an inspiration, Griffin. We would like to know more about things that made you who you are today.

I remember a summer weekend in 1975 when I was about ten. My father left me in the new truck alone while he ran into the store - the truck was off, and dad had the keys. I decided to investigate all the buttons and controls. I remember puzzling out that if I turned on the hazard lights, and turned on the turn signal - I could briefly energize the circuit to the fan or the radio. I always wondered if I had found a secret behavior that even the designers were unaware existed.

Activities: Science fiction and military history stories, chess, coin collecting, dogs, skiing, and the stock market, hopefully running again soon.

People-wise: Mom and Dad. Going to a Jesuit high school. Jerry and his work with Virginia Satir. My wife.

And Cindy Halstand, who delivered a very personal message to me, "Don't be afraid." I have pondered and tried to heed that message since.

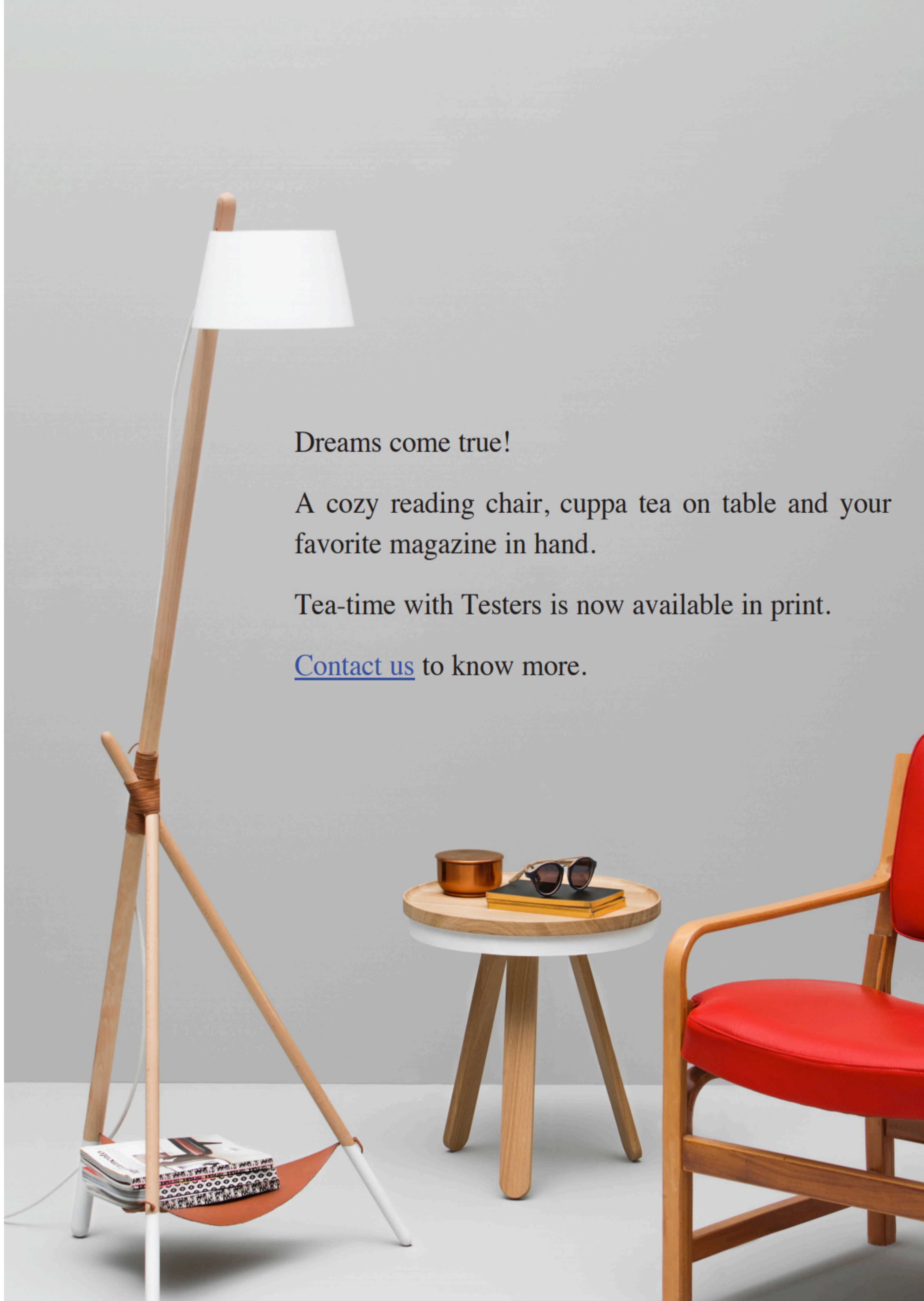
It is the interesting people I get to interact with that is both thrilling and exhausting.

What would be your advice for an experienced tester today and also for someone who is willing to make a career in testing?

Spend the time to work on yourself. Search out and make time to spend time with the best people you can. Partner with others, find a coach. Keep a journal, exercise, practice mediation, get enough sleep, and eat correctly. Read Becoming a Technical Leader, by Jerry. Read and explore other fields - how might that be similar to testing? Be kind to yourself - life is about making choices. Practice and get really good at something other than testing. Consider reading the best and most useful book I have ever read: More Secrets of Consulting, by Jerry.

Work on your Emotional Quotient, and the associated soft-but-difficult-skills. Work on being Congruent, in a Virginia Satir way.

May you grow into the best version of yourself possible, and may you help someone else you care about to do the same.


Dreams come true!
A cozy reading chair, cuppa tea on table and your favorite magazine in hand.
Tea-time with Testers is now available in print.
Contact us to know more.

~Processes

UNTANGLING TESTABILITY

In this article, I'll share the story of how my understanding of testability has evolved over the last fifteen years. Hopefully, once you've read the post you'll share my enthusiasm for all things testability. I've discovered that a whole team focus on testability is one of few levers in software delivery that can provide a positive impact on a team's productivity.

Discovering that developers can make software easier to test

Like many of the lessons I've learned in my career I discovered the value of testability by chance rather than design.

As a junior tester, I found myself working on an R&D project building the world's first camera safety system. It was a fascinating project that presented me with a host of different testing challenges. It was a complex system consisting of visual sensors, a physical control unit and a desktop application for configuring and controlling the safety system in an industrial environment.

Of all the challenges I faced the single most frustrating problem was trying to reproduce crashes in the application. The application allowed safety engineers to monitor industrial environments so that when workers approached dangerous machinery it would detect their presence and slow down or stop. The application required a sophisticated graphical user interface that allowed safety engineers to interact with representations of the environment.

An unfortunate side effect of this sophistication was a persistent stream of applications crashes triggered by even subtle interactions. The safety-critical nature of the system meant I had to investigate and replicate every single crash that we observed. Each crash took hours or even days to replicate as often both the order and location of user clicks were critical.

One day after encountering yet another crash in the application I decided to pair with one of the developers in order to replicate the issue. As usual, we followed the steps I recalled completing before the application crashed. We repeated the steps time and time again introducing slight variations each time. Eventually after hours back and forth we replicated the issue… success at last. Having replicated the crash the developer quickly isolated the cause and fixed the issue. That's where the story gets interesting.

Irked by having wasted hours trying to replicate this issue the developer suggested that he add a flag to the application that would record all interactions and write them out to a file. Two hours later his changes were in place and it was ready for use.

From that point forward those hard to replicate issues became a thing of the past. Whenever I experienced a crash I retrieved the contents of the log output into a bug report and the developers could replicate easily.

This experience taught me two valuable lessons.:

· The first being that we need to share the challenges we each face in each of our roles as a team. In this case, the developers had not felt the pain of trying to replicate crashes themselves and therefore did not realise this was such a huge problem. From my perspective I had no idea that this problem could be solved so easily. Being problem solvers when we expose the developers to a problem they naturally look for a solution.

· The second lesson I learned was that a relatively meagre investment in Testability can yield a hugely disproportionate impact on testing effort. In this case, a couple of hours of coding saved me countless hours of trying to replicate crashes.

Deliberately designing software with testing in mind

This set a seed in my mind that lay dormant for a few years until I again found myself in a situation where doing great testing seemed almost an impossibility. I started working as a test automation engineer in a newly established software engineering department for a hardware company. This was the companies first time building a software product and after positive feedback from the market, we were tasked with building upon their initial proof of concept.

As part of onboarding, all 35 people in the new engineering department were tasked with regression testing the software for a forthcoming release. Everyone in the engineering department got involved. After seven weeks of tortuous regression testing the release finally limped out the door.

This regression testing exposed the team to testing challenges that made the whole experience exhausting. The test automation in place was so unreliable it had to be disregarded. We had to test everything manually including checking thousands of product attributes via a REST API. When we did find and fix issues it often introduced new bugs in unrelated areas of the product. The highly coupled nature of the code meant that we had to run regression cycles for every change we made. This resulted in people working late nights and weekends. Within a week of the release numerous issues were reported by customers and morale in the engineering department plummeted.

Once the dust had settled, the software architect came to my desk and asked me a profound question "How can we make this system easier to test?". We talked through the testing challenges we had encountered. We formulated an approach to design our first component with testability as a primary concern. I use the mnemonic CODS to describe the design approach we adopted. Each letter represents a design attribute to consider when designing a system with testability in mind.

ROB MEANEY
–

Rob Meaney is an engineer that loves tough software delivery problems. He works with people to cultivate an environment where quality software emerges via happy, productive teams. When it comes to testing Rob has a deep and passionate interest in Quality Engineering Test Coaching, Testability, and Testing in Production.

Currently, he's working as Director of Engineering for Glofox in Cork, Ireland. Rob is co-author of "A team guide to testability" with Ash Winter, a keynote speaker and co-founder of Ministry of Test Cork.

Previously he has held positions as Head of Testing, Test Manager, Automation Architect and Test Engineer with companies of varying sizes, from large multinationals like Intel, Ericsson & EMC to early-stage startups like Trustev.

1. **Controllability -** The ability to control the system in order to visit each important state.
2. **Observability** - The ability to observe everything important in the system.
3. **Decomposability** - The ability to decompose the system into independently testable components.
4. **Simplicity** - How easy the system is to understand.

We decomposed the new component into a single, independently testable service. We added control points that allowed us to manipulate state and added observation points that allowed us to see the inner workings of the service. Within twelve months of introducing this new approach we revolutionised the way the whole software organisation delivered software. Painful regression testing cycles were a thing of the past as we now had robust reliable automation that provided almost immediate feedback on every change. This freed up our testers to do deep, valuable testing. Teams were working at a sustainable pace and delivering better software faster.

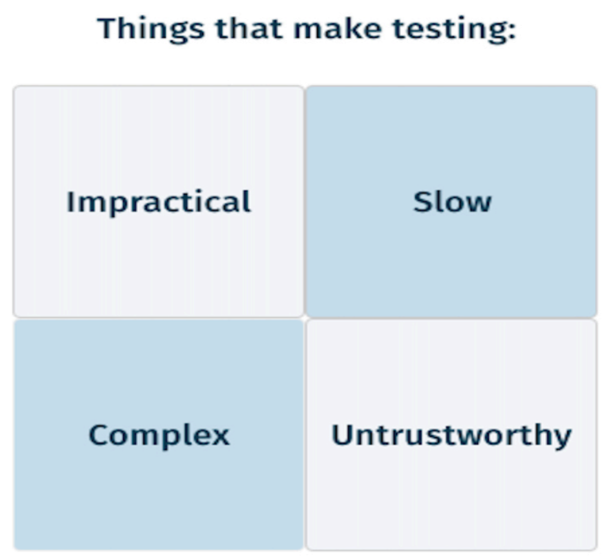
The transformation was staggering and convinced me that testability was the key to moving at speed without compromising quality. I also realised that the huge quality improvements weren't driven by testing. They were driven by building relationships and influencing the right people at the right time to build quality in.

I began consuming everything I could find on the subject of testability. I studied the content and models of James Bach, Michael Bolton, Anne-Marie Charret, Maria Kedemo, Ash Winter and Bob Binder to mention but a few. My CODS mnemonic helped influence the testability of the product itself but not the environment in which the testing was performed.

Helping teams capture their testing debt

Around this time I began coaching development teams to improve their testing practices. I created a simple model called the testing debt quadrant to help teams capture their testing debt. Testing debt occurs when a team chooses an option that yields benefit in the short term but results in accrued testing cost in terms of time, effort or risk in the longer term. In other words testing debt is the manifestation of a lack of testability.

The exercise requires the whole team to capture details of anything that makes their testing activities impractical, slow, complex or untrustworthy.



This exercise is a simple way of understanding the unique testing challenges each team faces. It provides a great starting point for discussing and addressing testability.

The 10 P's of testability

The more I worked with teams the more the sources of testing debt began to appear. It was apparent there was a common set of factors that influence a team's testing experience. I called this model the 10 Ps of Testability. The 10 Ps provide a lens through which teams can self assess their testing experience. By viewing the team testing experience through each of these lens teams can better determine where they need to invest their improvement efforts.

People The **people** in our team possess the mindset, skill set & knowledge to do great testing and are aligned in their pursuit of quality.	Philosophy The **philosophy** of our team encourages whole team responsibility for quality and collaboration across team roles, the business and with our customers.
Product The **product** is designed to facilitate great exploratory testing and automation at every layer of the product.	Process The team's **process** helps decompose work into small testable chunks and discourages the accumulation of testing debt.
Problem The team has a deep understanding of the **problem** the product solves for their customer and actively identifies and mitigates risk.	Project The team is provided with the time, resources, space(mental/physical) and autonomy to do great testing.
Pipeline The team's deployment **pipeline** provides fast, reliable, accessible and comprehensive feedback on every change as it progressed into production.	Productivity The team considers and applies the appropriate blend of testing. This facilitates continuous feedback and helps unearth important problems as quickly as possible.
Production Issues The team has very few customer impacting **production** issues. When they do occur the team can very quickly detect, debug and remediate the issue.	Proactivity The team **proactively** and continuously seeks to improve their test approach. They learn from mistakes and experiment with new tools and techniques.

The 10 Ps has proven to be a hugely versatile model. I've used it as a quarterly testability health check for development teams. I have used it to create career development plans for testers. Whenever I have to think about a testing related problem I find myself inevitably thinking through the 10 Ps.

Why do cars have brakes?

I'll finish up with an analogy. Consider for a moment why cars have brakes? Is it to slow us down? Is it to keep us safe? These answers are correct but ultimately cars have breaks so that they can go fast. Today's software development teams are moving faster than ever. Our teams need effective and efficient brakes, we need great testability. Without reliable testing to slow us down when we're in danger we're likely to crash head first into a wall.

So to summarise encourage the whole team to talk about their testing experience. Use the models described to identify and understand your testing challenges and work together to overcome those challenges as a team.

SEBTE: A SIMPLE EFFECTIVE EXPERIENCE-BASED TEST ESTIMATION - PART 1

The American Heritage Dictionary defines estimation to be:

A tentative evaluation or rough calculation

A preliminary calculation of the cost of a project

A judgment based upon one's impressions, opinion.

As a software engineer, I have a professional track record that includes the ability to consistently make and keep commitments. To consistently make and keep commitments I have developed the ability to judge the effort required to complete my work.

Over the years I have used many different estimation techniques, including individual and team approaches, analytic models, and even Monte Carlo methods.

The estimation method that sticks the most with me is something I call SEBTE, Simple Experience-Based Test Estimation.

Past Data

When using SEBTE I base my estimate on past data.

I try to make it remarkably simple to collect past data. I invest a few seconds per task to collect the information. I find that team members actively participate in the data collection process when the data can be collected, in a seamless, natural way, as part of the normal team workflow.

The data I collect is relevant to the team working on the project. If the team changes, or if the organization changes, or if the business changes, or if the technology being used changes, or if the culture changes, or if the lifecycle model changes, then I will need to recalibrate the estimation from scratch.

There are two points at which I collect data. The planning step and the task completion step.

During test planning activities I collect data associated with each task identified. I collect the SIZE and COMPLEXITY of each task.

About SIZE

I usually break SIZE into three levels, based on the number of factors, variables, or conditions the tester will need to consider when implementing the task. I define size as SMALL, MEDIUM, and LARGE. One way to define SIZING a testing task could be:

SMALL: The task will consider fewer than 5 factors

MEDIUM: Task will consider between 5 and 10 factors

LARGE: The task will consider more than 10 factors

Teams must have a simple clear definition of SIZE. I urge my customers to collect a series of examples of recently completed tasks that illustrate different SIZES. That way when trying to SIZE a new task the team can look at similar recent tasks for inspiration. Although the number and range of SIZE are up to you, I try to keep it to three levels. To me, this is just a simple rule of thumb that has worked well over the past 30 years or so.

About COMPLEXITY

During the planning session, I like to consider the complexity of the testing based on the nature of the changes to the software under test. As with SIZE, I try to define three simple levels of COMPLEXITY named something like SIMPLE, TYPICAL, and COMPLEX.

I might define these complexity levels as follows:

SIMPLE: There has been no change to the program logic

TYPICAL: There has been some change to the program logic or new logic has been added.

COMPLEX: The underlying code is new or has been redesigned or refactored.

It is important that the team have a simple and clear definition of COMPLEXITY. As with size, I urge my customers to collect examples of recently completed tasks that represent examples of SIMPLE, TYPICAL, and COMPLEX tasks.

About TASKS

I have been using the term task quite a bit so far. I call this estimation technique task-oriented because it applies to testing activities which can be divided into a series of tasks.

I consider testing tasks as a chunk of work done by a single tester to learn something about the software being tested. In my experience testing is about learning. I define testing tasks as activities in which the tester is endeavoring to learn something about the software under test.

I identify testing tasks when planning and as I am testing. The task list of a testing project is dynamic and evolves. As I learn more about the system under test, I identify more risks that may be important to explore. Indeed, the best testing ideas I come up with seem to be generated when I am actually testing.

As a rule, I use short phrases to describe testing tasks. The task description is an explicit statement of what I am interested in learning about. I also am incredibly careful never to use the word test as a verb in a description of a testing task. I will never have a task that says, "test feature ABC", instead I may have one or more tasks explicitly describing what I want to learn about feature ABC. "Confirm feature ABC can processes foreign and domestic transactions" is an example of a task description I might use. I try to keep task descriptions to under 160 characters of text. This is a rule of thumb I derive from my experience over the years using 80 column CRT terminals. I could not always get my testing task descriptions to fit on one line (80 characters) but I could almost always get my testing task descriptions to fit on two lines (160 characters). Today testers would probably prefer the metaphor of a testing task is tweetable.

I generally encourage a work break down granularity to the level of tasks that will require anywhere from 1 hour to a couple of days if the feature works well!

Some Popular Types of Testing Tasks

Here are some of the types of tasks I identify:

Capabilities

Test tasks based on what the application is supposed to do. Capability-based tasks focus on confirming that an application does what it is supposed to do. Requirement and functional specifications can be used as a source of capability-based testing tasks.

Failure Modes

Failure mode test tasks are "what if" questions. I ask "what if" something breaks. Failure mode test tasks are often inspired by how a system is designed. I look at all the objects, components and interfaces in a system and ask what if they break or exhibit some sort of unanticipated failure. Failure modes can be the result of harshly constrained system resources or forced error conditions.

Quality Factors

Quality factors are characteristics of a system that must be present for the project to be successful. Quality factors are the "ilities" including usability, reliability, availability, scalability and maintainability, Quality factor test tasks often involve experiments to determine if a quality factor is present. Examples include performance, load, and stress testing.

Usage Scenarios

A usage scenario test task challenges whether a user can achieve their tasks with the software under test. To paraphrase the Kennedy inaugural address – we ask not what the software does for the user but rather we ask what the user does with the software. Usage scenario-based test tasks involve identifying who is using the system, what they are trying to achieve, and in what context.

Business Rules

Business Rules are an asset. Business rules can be an excellent source of testing tasks for transaction-oriented information technology systems. Decision tables and process flow models can be a basis for powerful test tasks which can make or break a company. Testing business rules is not just about finding bugs, it is about validating systems to implement core values.

Combinations

When running transactions there are often many factors that influence the processing approach taken by the application. Each factor, or condition, may have several different values and they interact with each other to impact the behavior of underlining algorithms and processing in the system under test. Varying combinations and permutations of values of variables can help identify bugs related to how multiple factors are processed with a reasonable number of test cases.

States

When testing a stateful application a state model helps me come up with test tasks. For example, a transaction goes through many states of existence from creation through approval, payment, and delivery. I use state models to identify test tasks such as getting to states, exercising state transitions, and navigating paths through the system.

Data

Data is a rich source of testing tasks. Data flow paths can be exercised. Different data sets can be used. Data can be cooked up and build from combinations of different data types. Stored procedures can be verified. Test tasks can be developed to create, update, and remove any persistent data.

Environments

Exploring how the application behaves in different operating environments is a rich source of testing tasks. Environment test tasks can relate to varying the platform, hardware, software, operating system, locale, co-resident third-party software, and locales.

Internationalization and Localization

When products are developed for the international markets a lot of risks come up as part of internationalization and localization.

Internationalization enables products to support multiple languages and cultures.

Localization is creating a locale-specific version of the product.

There are enormous risks along the path to addressing the global market related to processing data, business rules, and literally the complete computer-human interface.

Unit Test

Unit Test tasks are derived from the technical work programmers do to build the software being tested. Unit test tasks can look at the structure of the software and the interaction between components, modules, objects. Methods, classes, stored procedures processes and other elements used to construct software.

Test Oracles

Test Oracles are strategies to assess correctness. Many different tools and techniques exist to help assess correctness. Some oracles are documents such as classic IEEE 830 style requirement documents. Other oracles are problem-solving heuristics or rules of thumb that guide domain experts. The method of assessing correctness may suggest testing tasks that can confirm or contradict the hypothesis that the application is working correctly.

Creative Tasks

Creative test tasks come from many sources. I often use deliberate lateral thinking techniques (For example Six Thinking Hats from Edward De Bono) to come up with cool and effective tests. I also use metaphors to come up with testing tasks. I wonder what would happen if the Tasmanian devil used the system? Perhaps a Dr. Seuss story inspires some testing tasks? Perhaps Great detectives? Movies? Pretty much anything goes here.

Path Test Tasks

Many applications have a series of capabilities that can be used in a wide variety of patterns, the pathways users use to navigate through a system are a great source of test tasks. There are also workflow paths, data flow paths, control flow paths, and many others. Paths can be a powerful source of test tasks to help identify problems in which one feature interferes with another.

Boundary Test Tasks

Boundary bugs continue to show up in software projects at points where extreme values for variables are selected or at the extreme minimum and maximum points of structures, Boundary test tasks can be suggested from written requirements or any objects in which two or more people must interpret the same technical descriptions.

Automation Test Tasks

Automation is not only a toolset to help in the creative execution and organization of tests. Having tools available to support the testing effort in and of themselves can be a source of test tasks. Imagine what testing you could do with the right tool. This section explores generating test tasks from test automation tools.

Regression Test Tasks

Regression testing dominates continuous integration servers in many agile projects. So much energy is dedicated to making sure we didn't accidentally break something when we change our code base for whatever reasons. How our codebase behaves in response to change is the topic of regression testing. How can regression testing be implemented, and what should be included in regression testing? This remains an open question and a remarkably interesting source of test tasks.

Collecting Task Data

Whenever a task is completed, I capture the amount of effort which was required to implement the task. I try to make this remarkably simple and done in the same system used to manage work. If I am in an agile team, I ask the Scrum Master to collect the data during the stand-up meeting.

So, the data I have collected for every completed task includes only three simple elements:

SIZE collected when the task is identified.

COMPLEXITY collected when the task is identified.

ACTUAL EFFORT when the task is completed.

I place this data in a spreadsheet. One row per data set. The unit of effort I choose to use in the project from that this example is derived is the number of sessions of testing required to complete the task. I defined a session of testing as 90 minutes, plus or minus 10 percent, of uninterrupted work on a specific task. During a session, the tester is not distracted by email, social media, or any other type of interruptions.

Task Effort Historic Data

Size	***Complexity***	***Effort***
Small	Simple	1
Small	Complex	3
Large	Typical	8
Large	Typical	7
Large	Complex	12
Large	Typical	7
Small	Typical	2
Large	Simple	5
Small	Simple	1
Small	Complex	4
Large	Simple	4
Large	Complex	8
Small	Simple	3
Medium	Complex	6
Small	Typical	4

You will note that there are exactly nine types of tasks categorized by SIZE and COMPLEXITY.

Task Effort Estimate with MODE

Size	Complexity	Estimate		Mode	Min	Max
Small	Simple	1.7		1.0	1.0	5.0
Small	Typical	3.0		3.0	2.0	4.0
Small	Complex	3.2		3.0	2.0	5.0
Medium	Simple	3.0		3.0	3.0	3.0
Medium	Typical	3.3		3.0	3.0	5.0
Medium	Complex	6.3		6.0	6.0	8.0
Large	Simple	3.8		4.0	2.0	5.0
Large	Typical	7.0		7.0	6.0	8.0
Large	Complex	8.8		8.0	6.0	15.0

To compute the estimate for each task type I use the Pert Estimation Formula. I learned the Pert Formula in my earliest days as a professional software engineer in the 1980s. The Pert Estimation Formula is taught as part of project management in general and in included in the Project Management Institute body of knowledge PMIBOK. Pert Formula is simple and practical. It makes use of what are known as three-point estimates. In Pert the estimated effort of a task is computed based on the optimistic (minimum) effort required, the pessimistic (maximum) effort required, and the typical (mode which is the most common) effort required. I rely on spreadsheet table look ups to derive this information from the collection of past data.

$$Pert\ Estimate = \frac{Optimistic+Pessimistic+4\times Typical}{6}$$

Estimating a new task

When a new task is identified I can look up the estimated effort required if I know what the SIZE and COMPLEXITY values are. In the above example if a new task is identified with a SIZE of LARGE and a COMPLEXITY of TYPICAL then I would estimate the effort required to implement the task as 7 sessions of testing.

Using Simple Experience-Based Test Estimation

A couple of years ago I was honored to receive an email from a client who shared their experiences using SEBTE for over 10 years of projects at a major financial services company. The method stood the test of time being applicable across dramatic changes in lifecycle models and technologies. The method was trivial to implement and was easily recalibrated whenever context factors changed, or teams were restructured or restaffed.

When Not to Use Simple Experience-Based Test Estimation

I have only been able to apply SEBTE to work which can be broken down into tasks related to what the tester is being asked to learn about. If I cannot break the project down to tasks, then I cannot use SEBTE.

What Simple Experience-Based Test Estimation does not do

SEBTE does not estimate elapsed time. SEBTE does not estimate capital cost. SEBTE does not estimate test coverage. SEBTE does not estimate bug counts. SEBTE does not estimate team size.

Can Simple Experience-Based Test Estimation work with more than two factors?

In most projects I use two factors for estimating tasks, namely SIZE and COMPLEXITY. I have on some projects used additional factors when it made sense and found SEBTE to be quite extensible. A specific example was on an internationalization project where the factors influencing the effort included SIZE, COMPLEXITY and MULTILINGUAL DATA HANDLING. I estimated and tracked over 430 tasks with this method and was able to accurately estimate to go effort as the project evolved.

How can a team use Simple Experience-Based Test Estimation?

In my experience the best way to get great test estimates is to consult a group of people who are involved in the project but who have varied roles and diverse perspectives.

I will usually start alone and build a list of testing objectives which will become tasks. I invest about ninety minutes in this activity and derive the testing objectives from sources available to me from the product requirements, design, past similar projects and even the source code.

I then share the list with different individuals from diverse roles such as programmers, testers, product managers, project managers, support staff, system administrators, customers, and users. I ask them to read through the list and come up with similar testing ideas based on their own personal experience and knowledge. Generally, I get a great set of ideas which can easily be ranked and sorted and turned into testing tasks.

Using Simple Experience-Based Test Estimation in turbulent projects

I consider a project turbulent when there are many and frequent changes to context factors. When Business, Technology, Organizational and Cultural factors change frequently it is important to revise estimates.

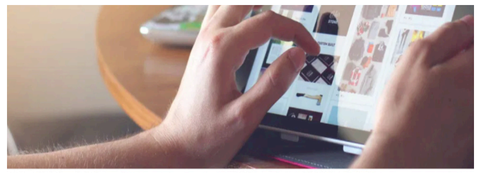
In turbulent projects you will need to update the tasks. I urge you to estimate testing effort on project elements in a short time horizon. For example, in one sprint you can estimate all the testing tasks for the stories being implemented but you should probably not estimate beyond the sprint boundaries since that is a moving target.

I have used SEBTE to estimate tasks combined with risk-based prioritization. For risk based prioritization the impact of the testing task on business and the likelihood of failure are used to prioritize testing activities. I sort activities by decreasing priority. I can then sum the effort required for all tasks sorted by priority to see how much testing we can complete on a fixed budget. This provides me with useful information to help advocate testing to project stakeholders and also shows different alternatives as to how the testing budget can be spread across product risks.

TO BE CONTINUED IN NEXT ISSUE

ROB SABOURIN
–

Rob has more than thirty-nine years of management experience leading teams of software development professionals.
A highly-respected member of the software engineering community, Rob has managed, trained, mentored, and coached thousands of top professionals in the field. He frequently speaks at conferences and writes on software engineering, SQA, testing, management, and internationalization.
Rob authored I am a Bug! the popular software testing children's book. He works as an adjunct professor of software engineering at McGill University; and serves as the principal consultant (and president/janitor) of AmiBug.Com, Inc.
Contact Rob atrsabourin@amibug.com


ras Shypka on Unsplash

HEURISTICS, WEB DESIGN

VIP BOA – A Heuristic for Testing Responsive Web Apps

EDUCATION, SPEAKING TESTER'S MIND

ISTQB, Fever Dreams and Testing

EDUCATION, WOMEN IN TESTING

How To Read A Difficult Book

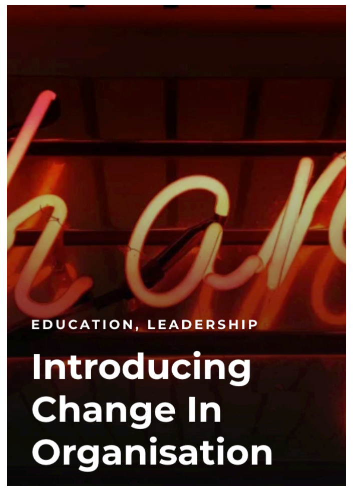
INTERVIEWS, OLD IS GOLD

Over A Cup Of Tea With Jerry Weinberg

mon Migaj on Unsplash

SPEAKING TESTER'S MIND

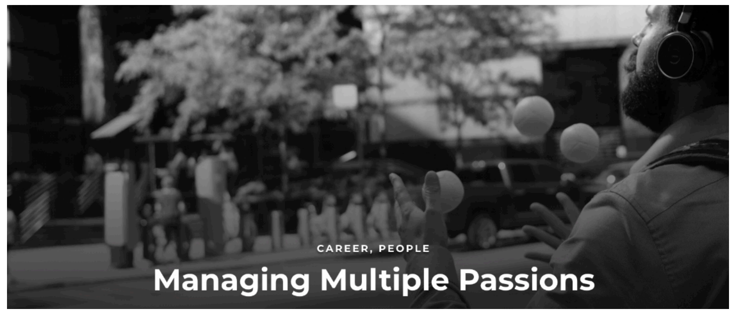
Software Testing And The Art Of Staying In Present


EDUCATION, LEADERSHIP

Introducing Change In Organisation

PEOPLE AND PROCESSES, WOMEN IN TESTING

Addressing The Risk Of Exploratory Testing Part 2


CAREER, PEOPLE

Managing Multiple Passions

LEADERSHIP, OLD IS GOLD

Leading Beyond The Scramble:

After nearly twenty years of working in software, I

many companies. One of them is what I call the sc

INFOCUS

Do you know all these amazing articles?

Great things survive the test of time.

Over the last ten years, Tea-time with Testers has published articles that did not only serve the purpose back then but are pretty much relevant even today.

With the launch of our brand new website, our team is working hard to bring all such articles back to surface and make them easily accessible for everyone.

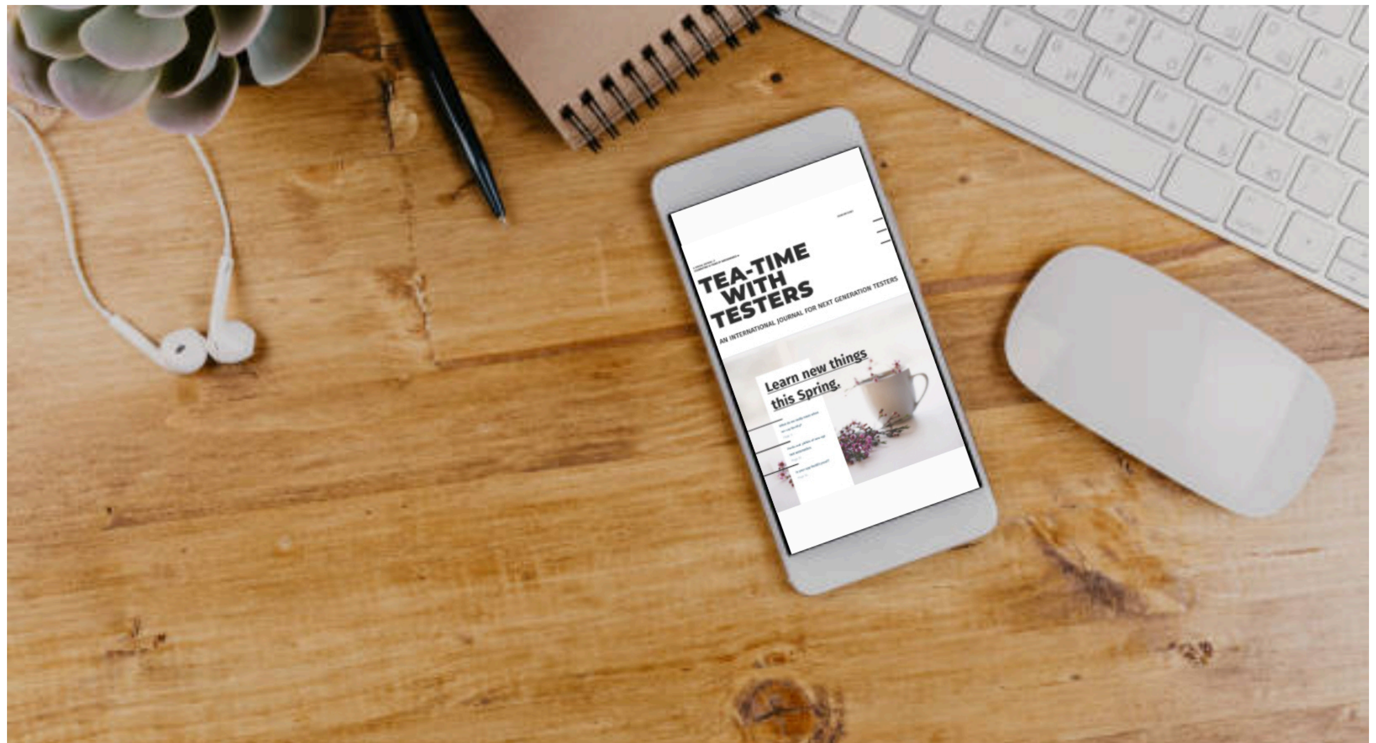
We plan to continue doing that for more articles, interviews and also for the recent issues we have published.

Visit our website www.teatimewithtesters.com and read these articles.

Let us know how are they helping you and even share with your friends and colleagues.

If you think we could add more articles from our previous editions, do not hesitate to let us know.

Enjoy the feast!


TEA-TIME WITH TESTERS
AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS
Learn new things this Spring.

FEELING GOOD ABOUT YOUR PERFORMANCE TEST COVERAGE?

Performance matters

Do you feel confident your Mobile App performs well enough to not only meet your company's expectations but also your customers' expectations?

Not all Mobile Apps have the same architectural design despite what some believe. Then, does it make sense to provide the same test coverage to all types of Mobile Apps? If you think it does make sense, ask yourself, "am I providing maximum test coverage efficiently?" For example, would you test a communication Mobile App like LinkedIn, Twitter, or Facebook the same way as you would an eCommerce app like a food delivery service? Would the same set of tests for a video streaming app be applicable to a game app or an eCommerce app? To be efficient in your test coverage, I suspect you will need to understand what types of performances are important to your company and to your customers/users. Each Mobile App type I refer to is architecturally defined as a hybrid app, which means usage of a Web server as well as the device's features like a camera, audio own storage.

Performance testing is an umbrella type of testing. I break down Performance Testing into 4 concepts:

1. Speed/timing
2. Load
3. Stress
4. Endurance

These concepts can be defined in different ways depending on the application under test. Define your needs of performance based on these concepts. Your Mobile App may display behavior concerns for your company but may not be seen as a problem by your users. Testers need to decide to balance out the type of tests applied to successfully launch the app for usage and meet your company's mission.

Let's examine each type of Mobile App and then discuss the types of most appropriate tests which provide efficient and maximum test coverage.

Speed/Timing or Response Tests

Communications apps rely on transactions that primarily fall on the responsibility of the webserver. When Mobile Apps utilize the device's features like a camera, audio, notifications, storage, then speed becomes a factor. Depending on the criticality of speed/timing is to the user response of these features existing on the device to communicate with the app, a speed test becomes a vital test where device features are utilized interacting with the communication app. For example, being able to take a photo you took is not as timing-dependent to post on LinkedIn or Twitter as it would be to loading that photo to a medical app where you are sharing the photo which will be sent to your doctor's office. How fast can the app respond to loading a photo into the app to be able to be sent would be the performance test? Setting up the appropriate condition to test that response time may or may not be important to your app's usage. Testers need to make decisions based on what is most relevant to the company's mission for the Mobile App's intended use as well as how relevant performance is to customers using the app. If the Mobile App is communicating statuses in real-time where users rely on those statuses, timing then becomes time-dependent therefore, the speed or timing tests will become more significant. Speed/Timing tests materialize based on context which is why testers need to recognize both the company's goals and the users' goals for the app and prioritize their testing activities accordingly.

E-commerce apps like a food delivery service have a few areas which require speed/timing tests. In addition, there might be some areas that require fast communicating, the speed at which search features exhibit results, calculating the total cost, or verifying payment method are all accessing a webserver performing those functions.

JEAN ANN HARRISON
–
Jean Ann has been formally involved in testing software applications and systems for 20 years and been a mobile systems tester for 14 years. The mobile systems testing involve law enforcement system, medical device systems, and commercial apps including healthcare apps, games, financial apps, weather apps, business administration apps. Mobile apps were to be used on proprietary devices, smartphones, and tablets. Jean Ann has been a consistent speaker at various testing conferences for 9 years, a contributing author in published books, articles, and webinars. Jean Ann is an active mentor to many testers and leaders.

Therefore, the Mobile App tester will rely on the web server tester to test those functionalities. A Mobile App tester can also conduct web server performance testing although performance testers typically concentrate on webserver testing only. If the Mobile App takes advantage of the audio on the device, does the Mobile App respond to the audio commands within an expected rate? Does the Mobile App utilize Notifications? A food delivery service can include notifications like expected delivery time, the shopper is communicating with the user within the app itself, confirmation of the order placed which are all important to the customer. Is the notification timely or is that notification lagging behind? What is the expectation by the customer and by the Mobile App company?

Gaming apps pose a lesser requirement when talking about speed/timing. Game apps are typically played by users accessing a web server and are not reliant on the mobile device to provide transactional data. Speed tests will be centered on the transactional speed of the webserver which means testing the webserver and not the Mobile App. Typically, web server testing is done by a different type of tester possessing certain skills while a performance tester has other specific skills to conduct performance testing on a webserver very similar to performance engineering. To learn more about webserver testing, I suggest following Performance Test Academy on Twitter and PerfBytes podcasts site are great places to start.

Video streaming apps are a combination of E-Commerce and Game Apps when talking speed and/or timing testing depending on the app. All of the content available to stream originates from the webserver. Does the video streaming app send notifications about available or changed content based on the user signing into an account? Does the app intend to have a functionality where downloading content is a primary or even a secondary function for the app? Paying close attention to the app's intended usage as well as meeting the company's mission for the app can dictate the types of tests prior to release.

Load Testing

Communications apps and testing the load can have different meanings depending on the Mobile App's objective. When performance testers talk about "load," they are concentrating on the load or number of concurrent transactions accessing the webserver. But the load on a mobile device can refer to concurrent transactions the mobile device's CPU is accessed. If the user has too many Mobile Apps in use at the same time and running in the background, the load on the CPU can create annoying problems or worse, crashes. Knowing the limitations of your app's thresholds is a vital test. Developers can provide the limitation of the CPU load on a mobile device but they also may not know what that threshold is, due to the many different types of devices. However, with a typical communication-type app, the load tends to be on the webserver and not on the device CPU. If the app is a safety-critical app, the tester should try to explore the threshold value to avoid a disastrous event. Checking with stakeholders might be a solid approach to assist in the planning of appropriate test decisions. To be efficient, project teams need to communicate early in the project regarding the intended design of the app and testers need to be involved as part of the project team.

E-commerce apps and Gaming Apps: Load Testing puts a very little load on the device's CPU. Unless the Mobile App requires downloading data, images, video directly saved onto the device, this type of test is not applicable to Mobile App testing.

Video Streaming apps pose an interesting challenge for Mobile App testers. Many apps we currently use for streaming require connectivity to a webserver. However, some video streaming apps allow the user to download the video content, making use of the device's CPU. If the video content infringes on the CPU to the point of not allowing other apps to run in the background, the functionality of that Mobile App will prove to be unsatisfactory to the user. Does the Mobile App allow notifications from other apps while a video is playing? Can the video be paused and not place an unforeseen load on the CPU? These tests might be considered important information affecting the use of the Mobile App.

Stress Testing

Communications app: Stress testing is testing the reliability and stability under normal and abnormal conditions. Having a known set of thresholds after setting a baseline of values although developers can set thresholds and communicate those values through documentation or verbal communication. Because communication apps themselves access data to and from a web server, there is little stress testing required. However, some conditions may be required to be tested. Can the app send only a certain-sized image taken by the device's camera? If the image is too large, how does the app communicate the problem? Does the app crash or recover? Is there a limitation of notifications it can display either from within the app or from other apps? Does the Mobile App's notifications step on other Mobile Apps? (Note: this was an actual bug I found with one communication Mobile App stepping on other apps which is why this question needs to be asked.) Depending on the criticality of the purpose of the Mobile App, these tests might all be vital testing.

E-commerce and Gaming apps: Stress testing can involve downloading the app to be saved onto the device. If the app is too large to be stored on the device, how does the app recover? Was there a warning to let the user know they need more free space on the device? Gaming apps in particular can be large with heavy graphics so the app should have a built-in check during the installation of the Mobile App and warn the user if the app is too large. If downloading is started but cannot complete, how does the Mobile App respond to the adverse condition? Will it crash the device with no explanation of the circumstances of the crash? Another type of test to consider, when the connection to the webserver has been lost, how does the Mobile App react? These types of apps tend to have advertisements pop up during use especially the game apps. But those transactions, along with storing the place where the user is while playing the game (or placing an order), are done through the webserver. When there is a loss of connectivity, does the app maintain the location where the user is, in using the app? Example: While playing the game, is the score saved locally on the device, and then when connectivity is returned, does the app update the data? Or does the app become unusable? With games, stoppage of game playing might anger gamers not being able to continue. These tests should be a standard stress test for any Mobile App and can be a fairly quick test to set up.

Video streaming apps can only allow downloading the content, allowing the user to view it completely locally on the device. This is the only type of video content viewing app that would require stress testing. Streaming content would involve access to the webserver and not have any impact on the device. Stress testing for the video content storage and playing type apps requires pushing the limits of the storage capacity, the type of image files to be played, and testing if that content file type is supported or not within the app. Will the app recognize if the image file type is playable, provide an error message if it's not compatible, or simply crash the app? Evaluate whether the impact of these tests would be important for the intended use of the app.

“

Do you test only to meet the company's intended design or do you concentrate on tests meeting the user's expectation?

Endurance Testing

Endurance performance testing of Mobile Apps doesn't come much into play for hybrid apps and definitely not with mobile web apps. This is because these types of Mobile Apps rely on transactions between the device to a web server. Endurance type tests need to be done on the webserver and not on the downloaded Mobile App.

E-commerce apps do not force the user to download inventory on their devices as this would be considered an outdated design type. Endurance tests for E-commerce testing will be done on the webserver and not on the Mobile App on the device itself.

Gaming apps have the ability to sustain without connectivity and then once connectivity is reestablished. Did the gameplay data endure reconnecting to the webserver where the gamer could continue to play the game with no interruption is apparent? An example of this type of game is a casino-type app where the saved coins earned from play when not connected are updated appropriately to the web server once reconnected. Losing their winnings would not make the gamer happy once there is a reestablished connection with the webserver. Specifically, can the user play the casino app for hours, stop play, exit the app and then start the app again without connectivity? Can the game then be updated once connectivity is reestablished? Would there be any loss of data on either the app itself or once the data is transferred?

Video streaming apps also rely on webservers to provide content. If the app allows the downloading of the content, testers should then test the ability to download content and set up some endurance-type tests. One test could be: play content, replay content, maybe on repeat to ensure there are no crashes. If the app stops playing after one completed content but is not available for further playing, it could create a negative reaction to the company's brand. Testers and stakeholders need to examine the risk of a poor result the conditional test is not tested. How extensive would the impact of not testing the conditions of endurance to the Mobile App be?

Mobile App performance testing requires testers to discern risk for a poor performance observed by the user. Testers need to understand what types of performance tests are appropriate for their type of Mobile App, the purpose of that Mobile App, and the Impact of not conducting certain tests. Do you test only to meet the company's intended design or do you concentrate on tests meeting the user's expectation? Testers and stakeholders need to prioritize what is the greatest impact overall, but testers need to evaluate a user impact as well as their company's brand. Each impact is a vital discussion point with stakeholders within the project.

A GUIDE TO TEST AUTOMATION PORTFOLIOS

TL : DR

BETH MARSHALL
–
A passionate QA advocate, Beth has been working in QA for the last 13 years, most recently as Senior Test Lead at Edtech company Smoothwall, based in Leeds, UK. Recent projects have seen her co-host a software testing bootcamp, and she also loves to collaborate with folk in the wider testing community.
Many moons ago, a colleague handed Beth a well thumbed issue of TTwT, and she remembers it vividly as the first time she was made aware that her "job" was actually a "career". She is both delighted and honoured to be part of the 10th anniversary edition.

You can check out Beth's blog at bethhetester.com, she Tweets @Beth_AskHer, or connect with her on LinkedIn.

Automation is considered an essential skill in many software testing positions. Yet it remains very hard to judge someone's automation experience by conversation alone. If you think it is important to demonstrate your automation skills, why not invest some time into creating a test automation portfolio. Use it as a chance to show off your existing skills, or grow your technical prowess by conquering something new. It is rare that something which could have such a positive effect on your learning, self development and career progression can be entirely free, and the payoff could be well worth the time investment.

What is a Test Automation Portfolio?

In 2019, Angie Jones authored an article titled "10 portfolio projects for aspiring automation engineers". Angie notes that front end developers, UX professionals and others in technology are asked to provide examples of their work when trying to secure a new job, but as yet this isn't commonplace for testers. A test automation portfolio fills that gap, by giving you an opportunity to demonstrate your skills, and have some fun in the process.

Why are they a good idea?

Have you ever been in an interview where people ask "can you tell us about your automation experience?" or "Tell us how you would automate a login page"? Wouldn't it be great to have a ready made response, to point them to some actual code you've written that demonstrates your point? Think of how much more powerful a persuader this would be to potential employers than a verbal answer. As someone who has done a lot of interviews over the years I know it would blow my socks off.

Future career success aside, it is a widely held view that the best way to learn is by doing. Getting your nails dirty by coding an automation framework, and working your way through those gnarly issues (and there always are gnarly issues) could really help you to move from beginner to intermediate to advanced level at a much faster pace than watching an online tutorial can. Plus you may be able to quickly transfer what you learn back into the frameworks you use in your day job, or indeed use your new code as a working Proof Of Concept (PoC) to encourage adoption of a new tool or technique at work. Nothing speaks to people like seeing something in operation - theory alone simply does not have the same powers of persuasion.

Most importantly, if you leverage the incredible resources online such as Test Automation University, Ministry of Testing or YouTube, you can do this without spending a single penny. Certifications can cost thousands these days, and I'd put a well written test automation portfolio on a par with some qualifications, and above others, when it comes to their potential impact on your future software testing career.

Once you feel like you're on a roll you can tackle the more challenging things.

OK, that sounds pretty cool. Where should I start?

It's tempting to dive straight in and start creating that GitHub Repo, but my advice would be to plan plan plan. Question why you want a particular project in your portfolio, let those principles drive everything that you select to go in it and set an achievable goal for yourself.

Ask yourself:

What is the driver for creating this portfolio?

Is it to demonstrate your incredible technical prowess to future employers?*

Is it to provide a Proof of Concept for a new tool to use where you work?*

Do you want a suite of examples you can use when creating online demo's or tutorials on YouTube?

Do you want to compare different versions of the same library / language side by side to assess new functionality?

The possibilities are infinite - what level of breadth and depth do you want to go to?

Perhaps you want to deep dive, and go into a great level of detail for a single tool/language/layer of the stack.

Perhaps you prefer to be shallow and broad, and give examples of a variety of different things to show you aren't just a one trick pony

Are you motivated by deadlines? If so, add one of those in too - just make sure it's realistic. My portfolio took me 4 months and I'm still tinkering with it.

*if so, bonus points for creating a project based on their product, or website. And if anyone lands a job/gets a tool adopted because of following this advice, 1000% reach out and let me know, because that's a win that needs to be celebrated.

From my experience, I'd also suggest starting with something easy, that you feel pretty confident achieving. Once you feel like you're on a roll you can tackle the more challenging things. I deliberately left Selenium Webdriver 4.0 until last for this very reason - that way I was in way too deep to quit by the time I got lost installing Chromium browser driver and had to push through that pain barrier.

Can you give me some examples?

In my blog series, I set about creating a goal which included frameworks throughout the stack. I wanted to try different tools, and use different sites to practice against.

Beth's Test Automation Portfolio Outline:

- C#/Java solution for web browser automation for Opencart using Selenium WebDriver
- JavaScript solution for web browser automation for Opencart using Cypress
- API testing for Restful Booker using Postman
- CI integration testing using TAIKO
- Uipath to do RPA testing reading from a sql server express database
- BONUS: NUnit Testing of Restful Booker

Tips

1. Set a goal. Decide in advance what you want to do and do your best to stick to it.

2. Use Github to host your completed code - use a private repo or share with the world

3. Take advantage of free online resources such as Test Automation University, they often provide example code to get you started.

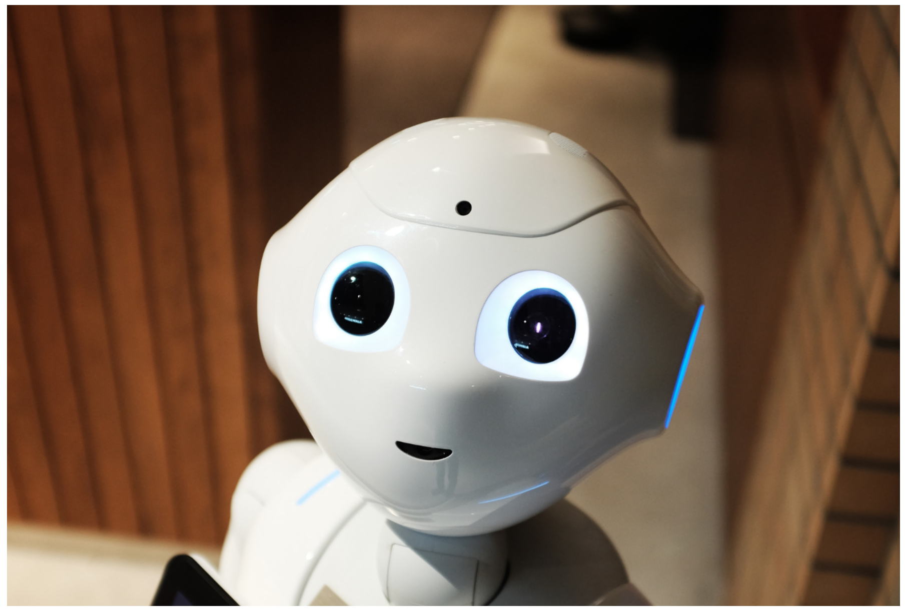
4. Ask for help if you get stuck - reach out to the wider testing community who are always happy to support.

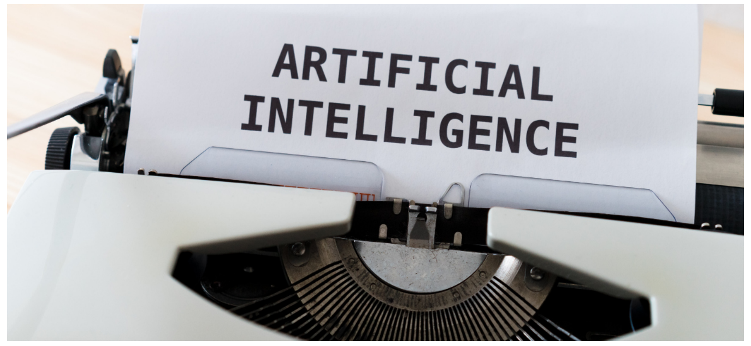
5. Brag about it! Tell others about what you are doing, maybe even try to collaborate on a project, the world is your oyster.

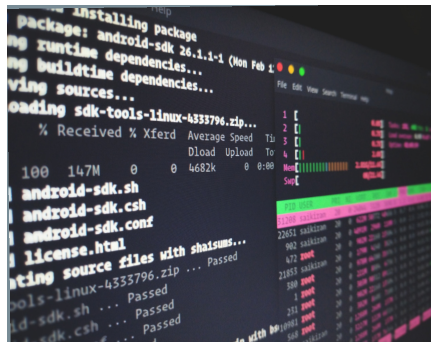
6. Enjoy it! If this starts to feel monotonous and like a chore, switch to a different project or leave it alone for a while. Automation can be a process of trial and error, and immeasurably frustrating, but getting over those humps is a great feeling and worth the perseverance.

7. Reward yourself! Every time you feel like you've made a breakthrough with the portfolio, give yourself a pat on the back, or treat yourself to something nice - the carrot is, after all, infinitely better than the stick

~Products







```
  installing package
Package: android-sdk 26.1.1-1 (Mon Feb 1
ng runtime dependencies...
ng buildtime dependencies...
ving sources...
oading sdk-tools-linux-4333796.zip...
  % Received % Xferd  Average Speed   Ti
                      Dload  Upload   To
 100  147M    0     0  4682k      0  0:00
 android-sdk.sh
 android-sdk.csh
 android-sdk.conf
 license.html
ating source files with sha1sums...
ools-linux-4333796.zip ... Passed
id-sdk.sh ... Passed
id-sdk.csh ... Passed
```

PERILS AND PITFALLS OF A NEW AGE TEST AUTOMATION

What lies beneath the cool tools?

Iryna Suprun shares her analysis of automation tools in the market that is bursting with buzzwords.

I just spent some time googling. I was trying to find who and where created the first-ever automated test. No luck. The same goes for the year when it happened. Most of us just agree that automation has been there in one or another form for a long time. I would say more than twenty years. Mass adoption of agile methodology triggered mass adoption of automation so I would say that we have been actively automating for the last ten-fifteen years. It is like ages in the software industry.

Some would expect that by this time we have mastered test automation but the reality is different. Many organizations are still struggling even with unit testing. Flaky functional and end-to-end tests are the constant topics of discussions between test engineers and the leadership teams, test maintenance takes time and dedication at the same time causing a lot of pain. There is always not enough coverage and while there are builds that take hours because test execution lasts hours.

Part of this is caused by the growing complexity of software under test. Many new technologies emerged in a very short time. First, we got cell phones, then wearables and IoT devices. We have containers, we have clouds, AI and Big Data. All of this needs to be tested and testing tools always stayed a little bit behind, to be honest. Many companies built their in-house automation tools and frameworks to compensate for it.

Something changed a couple years ago. I believe now we are reaching a new age of automation where automation tools are popping up on the market almost every month. I'd better say they are not exactly new, but newer. Most of them have been there for a few years and just recently became mature enough to be considered as a replacement to old technologies. I would not use a tool that has two customers and is six months old because automation is a big commitment. I don't know about you but I don't want to invest in something that might be gone in another year. Even if it looks very good.

These new(er) tools are all "industry-leading", "game-changing", "trusted by many", "AI-based", "future of testing", "fully autonomous" and there are many of them. I think we will see a lot of mergers and acquisitions in this industry soon (it already started to happen). We will end up with a few solid products, but now it's hard to see who will win because so many things can go wrong and right.

Today I would like to talk about tools that use the most popular technology nowadays - AI (Artificial Intelligence). There are many AI testing tools. The maturity of these tools, the usability, and the promises delivered vary. They use Supervised Learning, Unsupervised Learning, Reinforcement Learning, and Deep Learning algorithms to automatically generate tests, make test creation easy, and decrease maintenance time. Almost every test automation tool on the market nowadays uses AI in one or another form or claims it does.

There are two main categories of these tools. The first category uses AI/ML in a supporting role. AI algorithms in these tools are designed to ease tasks that are hard to do manually due to the physical limitations of human beings. They also help with testing where before some subjective measurements were used (audio/video quality for example) and turning these subjective ratings into objective numbers. Test authoring in this category usually is done with heavy user involvement. Users either need to write code or record the test. The second category of testing tools uses AI for test generation. They use the software under tests to create tests, it can be done through analyzing code or production logs, gathering clickstream of actual users, traversing links in the app.

I think that AI/ML pays the most in the following testing tools features:

1. **Test recording.** Although many traditional testing tools also provide the ability to record tests they do not collect data during the recording process. The number of data points for a single UI element can reach 30-50 entries. This info is used later to improve the stability and maintainability of the tests.
2. **Autonomous creation of automated test cases** based on usage traffic from real users, logs, and application functionality or code analysis.
3. **Self-healing.** Automation maintenance is the most time-consuming, never-ending task. AI tools help reduce maintenance time by making automatic adjustments to the tests if software changes. Although in most cases, it is still the job of the QA Engineer to decide if this change is acceptable or not.
4. **Converting test documentation to automation.** It is a very useful feature if you already have existing documentation and it is created in some structural, formal way.
5. **Visual testing.** Some test tools tried to achieve this without involving AI and ML algorithms by using pixel by pixel comparison. This proved to be not a very effective approach because usually it leads to many false positives. ML algorithms help to increase the robustness and stability of automated tests by identifying changes that do not impact user experience and ignoring them.
6. **Audio/Video Quality Testing.** AI can collect multiple data points about audio/video at any given point of time and learn how their variations impact audio/video perception by humans. Testing is done faster and is based on data, which makes it more objective.

This list looks impressive and promising but there are major drawbacks, at least for now. When considering the usage of newer tools that have not been on the market for a long time we should remember that AI tools are not mature or widely used compared to traditional testing tools. If their users encounter a problem, they most likely need to go to the tool vendors' support team which can increase test creation and troubleshooting time. There is less available information about AI tools in general. Comparison charts, use cases, real reviews, and case studies are hard to find. If you are considering adopting an AI tool, be ready to do a lot of leg work by yourself.

Most of these tools are also not cheap. The open-source AI-based test tools, as well as free tools, are rare and very young (and in this case you don't have wide community or company provided technical support, so you are truly on your own with your issues)

Last but not least - there are plenty of traditional testing tools for every type of application, most of the AI tools only support automation of the Web applications. There are just a couple of AI tools that provide automation support for mobile apps.

I was lucky to land my hands on a few test tools that use AI and ML. I used some of them (Mabl, Functionize, Appvance IQ) to go through the full-scale proof of concept. I played with a free version of others (Testim, TestCraft) and participated in some hackathons (Applitools Eyes). This gave me a pretty good understanding of where these tools are now and what works and what does not.

Let's start with positive and review what worked well:

- **Quick Start.** The pleasant surprise was how easy to start to use most of the tools. The documentation, the setup, the intuitive interface of Mabl, Testim, and Applitools Eyes are very user friendly. It took me just a couple of hours from opening a tool for the first time in my life to having my first automated test running. Of course, one should invest way more time to use any of these tools to their full potential, but the quick start is super important during the initial rollout.
- **Codeless Script Generation (recording).** Honestly, I was a little bit skeptical about this feature, but test recording went a long way in recent years. I found that Mabl is the most mature test recording feature for web applications, with the most stable and intuitive interface. Another tool I would like to mention is Testim. The test recording feature is also implemented pretty well there. TestCraft, Functionize, and Appvance also provide test recording but it is a much more challenging task and requires a lot of additional actions, not just pressing the "Record" button and going with the flow.
- **Decrease of maintenance time**. Tests automated by me using different AI-based tools were executed on multiple builds and releases. Every tool I tried handled insignificant changes in the UI, such as text, size, or other attributes well. If the change was something that cannot be handled by a tool (introduction of new elements, a major redesign of UI) it was much easier and faster to fix automated test cases by re-recording steps than introducing the same change using code.
- bUnfortunately, I was able to try this feature only during Applitools Hackaton using Applitools Eyes. I was impressed by how easy it was to automate challenging but very common use case (selection specific color and model of the product in an online store).

All these features are great but there are still some challenges everyone who wants to use an AI-based automation tool should be aware of:

Codeless script generation. Although the creation of tests using recording features is much easier and faster than using traditional tools, it rarely goes smoothly. It is not just "press the button and go with the flow". The automation Engineer still needs to add some Javascript snippets for complex cases, take care of reusable flows, setup variables, and test data. Hence, do not expect that it will be done with lightning speed and will not require any technical and test automation skills.

The self-healing feature Self-healing is a great feature, but it does not mean that all your tests will be magically fixed every time. Different tools handle self-healing in different ways. Some of them require the approval of every change, at least at the beginning while the system learns, so it is still a time investment. Others just make changes and proceed without notification, which I think is dangerous. Fortunately, most of the AI-based tools are starting to add the ability to review auto-fixed tests, accept or decline these changes, track history, and rollback to the previous version.

Every big change in the application, like introducing a new element, removing tabs, or anything else, will still result in the test updates that should be done by QA engineer.

Autonomous creation of automated test cases. The AI-based automation tools can generate tests in different ways. Some of them generate tests by collecting information on how real users use the software in production. This information can be extracted from logs, clickstream, or both. As you might guess this requires that the application should have quite a few users to collect enough data for ML algorithms. It also should be already running in the production environment. That said, test generation based on the usage of application can only be used to add missing regression cases.

Another type of self-generated tests are tests generated by link crawlers. These tests check that every link in the app works. This is a useful tool, but a working link does not necessarily mean it's the right link, or that it's a functioning application.

The auto-generated tests only cover what can be easily tested. They find shallow bugs, like not working buttons, particular values, broken links. They will never help you find missing requirements, logic flaws, or spot usability issues.

As we can see there are plenty of use cases where the usage of AI and ML can improve the everyday life of QA Engineers and help us solve the problems that are usually solved with traditional test automation tools more efficiently. AI-based tools are good in decreasing or in some cases fully eliminating the time spent by QA Engineers on mechanical, boring tasks. They also allow us to automate tests much faster. Although we should not expect that automation will be effortless. It will only be easy when software development is also an easy mechanical task. Another thing to remember that with the new technologies new issues and new testing needs emerge all the time, and tools... tools usually stay a little bit behind.

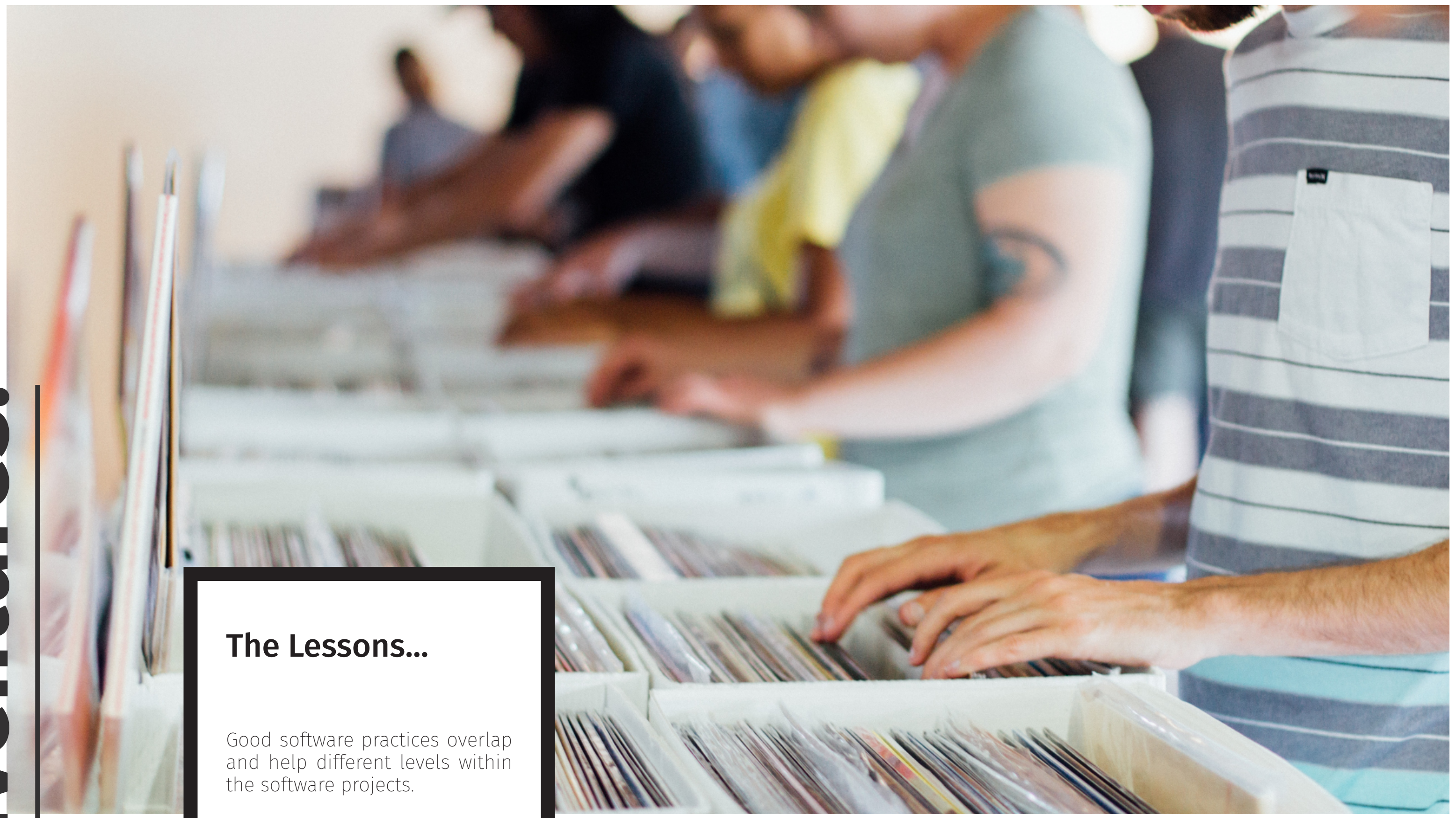
IRYANA SUPRUN
–
Iryna started her career as a software engineer in 2004 in Ukraine, where she was born. She received her master's degree in computer science in 2006, and in 2007 she began her first position as a quality analyst. She remained in QA, focusing on the telecom industry and testing real-time communication systems and products such as the audio platform for the GoToMeeting application. Three years ago, she decided it was time to try something new and moved to AdTech.

She is presently an Engineering Manager at Xandr where she concentrates on automation frameworks and implementing testing processes from scratch

SEARCH ENGINE OPTIMIZATION TESTING

An Experience Report

Blessed are the curious for they shall have adventures.

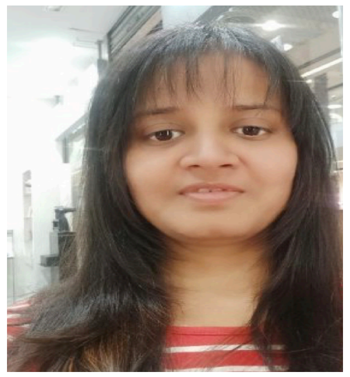


The Lessons...

Good software practices overlap and help different levels within the software projects.

A good software practice is not to take users or the user's perspectives for granted.

Challenge yourself by actively listening to all project stakeholders, talking with developers, and researching on your own to learn more perspectives.



TRISHA CHETANI

–

Trisha is software testers and automation enthusiast. She has been helping the team to follow testing processes and support that enable teams to deliver high-quality software in DevOps Environment. She is always enthusiastic to attend conferences, meet-ups for professional development as well as she is an active community member. Trisha is looking for opportunities to thrive in a management role. .

The Journey...

Back in 2014, I was assigned to the projects to test the search engine optimization along with the projects which were focused on increasing traffic on the website. During the initial days, developers used to tell me what I have to test since I do not understand the impact of these changes. I use to test the same, and report the bugs.

Let me take an opportunity to explain to you in brief, what happened in the past release, and why I was there, as I have learned from the existing team members. I had understood from the team members about the previous release, including what went wrong. The company had to pay a massive penalty to Google because of the issue which was introduced in the past release, since they did not want to lose the existing customers. Specifically, the issue was while fixing and improving so many things from SEO aspects, nobody looked that we are duplicating the pages one with HTTP and another one HTTPS. Since all the web pages were duplicated in Google search, our website and all the existing web page rank started lowering down rather than increasing.

Let's come back to what I was doing in the team, now I was trying to develop a deeper understanding of what the developer wants me to test because I was seeing a pattern. From the aspects of the team, I was lucky because I found myself in a situation that a senior tester was guiding me also. But this was for starting because soon I found myself in a situation where I was working alone since the senior test went on paternal leave. I was left alone during the critical period of the release. This senior tester was handling the whole Search Engine optimization POD(similar to SQUAD) alone along with my help for the past few months.

Since the senior tester was not present, I got the opportunity to talk to developer leads and business people. At times the conversation in the meeting was hard since I did not know what to answer when the developer lead and business people were asking me, "Did you test well"? "How confident you are with the release"? I was honest with them because I had not had much experience in this space at that particular point in time, and I was doing the best I could.

The developer lead and I used to share the same cabs while leaving the office. This is because we were living in the same locality. I had the opportunity to socialize with him, and ask him more questions, or even understand his previous experiences, and other stuff in this domain. One of the most interesting conversations was that he had explained to me that "usually, some companies have an internal SEO team and anti SEO team". This is because many people in a company do not understand how Google algorithms work, Companies hire a dedicated team who can look from a different perspective to help the company's SEO.

I thought this is important and he said, he is looking for our company to have something similar. To add to this, I further learned involving myself in the SEO conferences, that there are whitehat SEO experts and blackhat SEO experts.

Wait! Now more surprises are coming for me.

1. I learned, the senior tester has cleared the interview in the U.S.A in the same company, and he will move soon post his paternal leave. My manager was also leaving the company. I will have a new manager and new senior tester who will help me while learning about other ongoing projects and new projects. The whole situation was becoming challenging although, I was confident about a little experience, which I have received while working on the domain along with the senior tester.

2. While we were in process of releasing and adapting to many changes happening internally in the team. Google declared they would be making massive changes in their search algorithm which caused our release to be moved further out to incorporate and align with these changes. From a tester's perspective, being able to see how existing pages were impacted by the new changes allowed me to continue to learn more about testing SEO.

If I remember correctly, in those days mobile apps were becoming more popular, and Google came up with the recommendation that website and apps should not have major differences .

I pulled my socks. I started looking at internal documents which were across teams, I found many resources such as written documentation, videos, websites, conferences et Cetra. I also found there were existing test plans, test cases, bugs reported by another team on similar aspects of what we were improving. Now, I was developing and gained more confidence because I could talk to more people and learn about Search Engine Optimization while having an active conversation with them.

Here comes my learning while tackling all the difficult situation around me:

Terminology:

I began to realize certain common terms were very useful in day-to-day contexts, like organic(natural way of searching), inorganic search(paid search). These were further divided into ON-page SEO and off-page SEO. As far as I remember, there were more categories of an SEO search, which we were tracking, like direct, paid, social, referral, display, affiliates. I found it more interesting to learn and how to track these sources of traffic. This was one aspect in which I was able to learn by working closely with developers.

Crawling

Crawling(also known as spiders) of the website is a process, so the search engine can read the pages. But there are many challenges too. Crawling is the act of snaking the homepage to learn all the connected links which exist under that home page. This process goes on until they come out from the website. It is also called the internal mesh. There are different kinds of crawlers like bots(such as Google bots or Bing), sent by a human or someone who is stealing the content from the website. So the website owner has to always monitor these aspects else they will lose business. Crawlers keep track of information whether the page needs to be indexed. Some of the information specifically it keep tracks of is the title, HTTP status code of web page, header and meta tags, internal and external no-follow links, paginations[rel= next], hRef tags, canonical URLs, crawl depth, empty or duplicated pages, URL filters, page category, comparing crawls et Cetra. The crawler will have information about the user agent, crawl speed, crawl limits etc.

Indexing

Indexing is a process where it stores and organizes the page in the index after understanding the page content. Google search operators are useful to find whether the webpage is indexed or not: https://support.google.com/websearch/answer/2466433. Oh, how can I forget about Robot.txt, which also specifies the crawler which pages are open to crawl and not? Header and text can help a lot to improve indexing

.

Ranking

The ranking is the process where SEO is targeted to provide the best answer available from the keyword the user is searching for. Staying up to date improves the rank of the page. Not

the least, there are many websites which will suggest how to improve the rank of the webpage.

Sitelink

Sitelink, which was launched to improve Google UX, it is also called SERP. Sitelinks also appear with a search bar which the user can use directly to start his search on the website. It improves brand awareness, click-through rates, allows users to find the page they are interested in.

Rich Content

Another aspect to improve SEO on the webpage is looking to answer the below questions. Does the website have good and rich content which makes users come back on the website? For example, " Is there duplicate content"? "automatically generated content"? "little content"? "no content"? "hidden content"? "too many contents"? "no single focus, or motive of the webpage"?

Search Keywords

Keyword optimization is done to attract more users. We must start researching which keywords are more searched by users. Understanding the target audience is key here. It can also be done with the help of related keywords and comparison which is a bit of a time-consuming process. By taking part in stakeholder meetings and listening to developers' conversations, I was able to utilize tools to help me specifically test the keyword searches a target audience might conduct. This helped me to provide information to the stakeholders about how we could improve our keywords on the website page.

Header & Meta and other Tags

Most of the header and meta tags are used for improving SEO, such as title, description, URLs to improve the SEO. This is the reason it is good to keep them clear and relevant. Usually, alt tags are used for describing the image and it should be done properly and should have uniqueness.

Backlinking

I have also learned that backlinking from different web pages were helpful to increase the SEO of the webpage. It is also useful from a usability perspective(sometimes) because the user can understand where they are coming from.

Performance

A completely new perspective, if a webpage is loading fast then also it improves the SEO of a particular webpage.

Social Influence

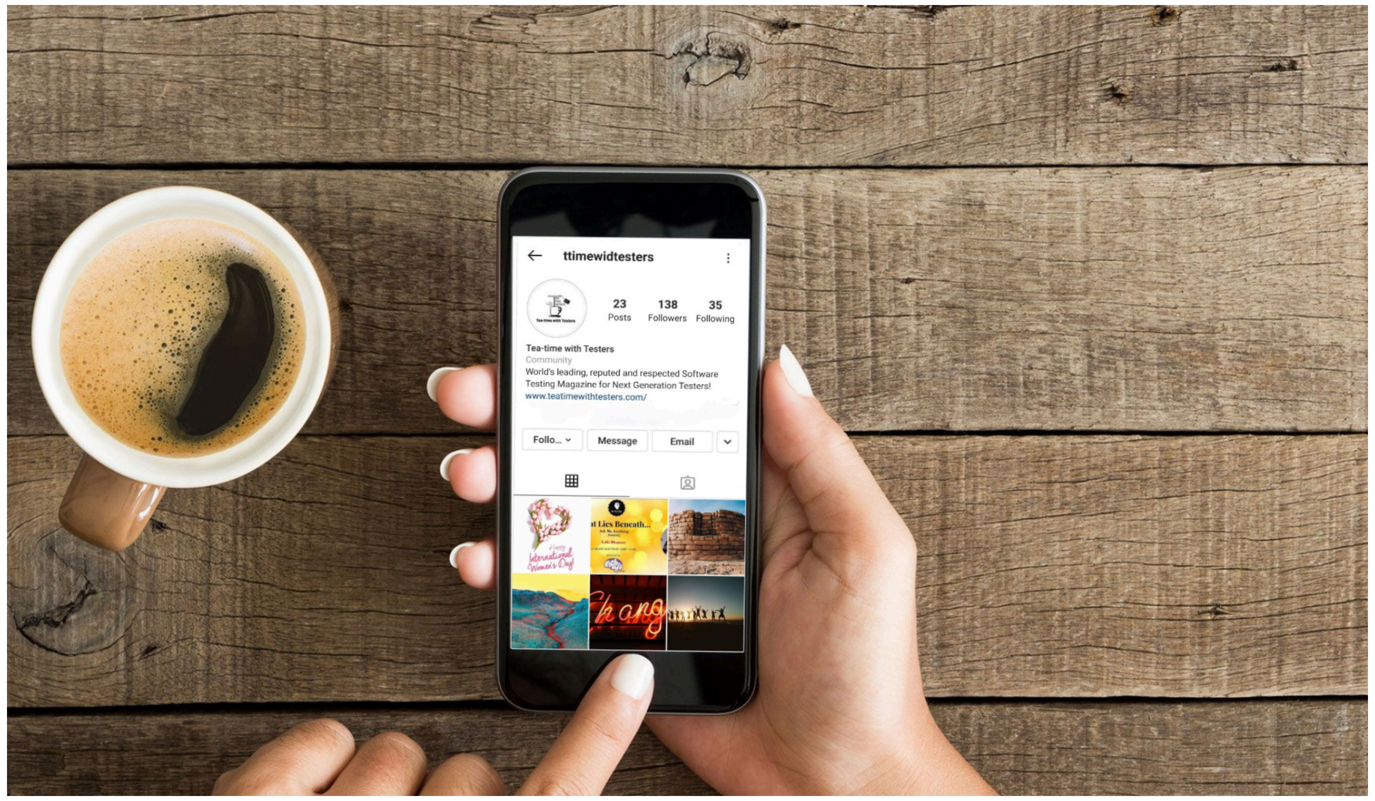
Another reason where a webpage can increase the SEO is having citations or social influence on the web page. By this I mean increase SEO, be sure to plan on adding links within the website, and not the links which are moving the user from the websites, such as, affiliates.

There were many more, but I think these were the most important concept from my learning journey.

My journey of SEO testing has taught me some meaningful lessons:

1. Good software practices overlap and help different levels within the software projects.
2. A good software practice is not to take users or the user's perspectives for granted.
3. Challenge yourself by actively listening to all project stakeholders, talking with developers, and researching on your own to learn more perspectives.
4. In foreseeing the future, we can create an automated crawler to improve further test coverage.

I grew as a tester from my journey and I hope this article will not only improve your learning beyond testing SEO but also to improve you as a tester in any domain.



“Robinhood and WallstreetBets” are warning signs.

Did you hear about the battle over GameStop between Reddit-based stock traders and big hedge funds during January? You better know how the vulnerabilities in your app can hurt you and cost lives for some...

An eye-opening piece by: Paul Maxwell-Walters

Did you hear about the battle over GameStop between Reddit-based stock traders and big hedge funds during January? This is about how the vulnerabilities and limitations of a popular stock trading app, Robinhood, were exploited by a group of stock gamblers on the subreddit r/wallstreetbets to not only gain profits for themselves but hurt Robinhood financially and eventually strike a blow at the heart of Wall Street finance itself. It is a lesson to devs, testers and company owners that in the ages of social media, financial vigilantism and doing things just to troll, the costs of lack of vigilance and underestimating your user base are far more costly than ever before. Robinhood’s experience also shows that the costs associated with bugs and limitations are amplified infinitely when social media and highly visible trolling are taken into account.

The Medium - GameStop

Over January 2021 the 4 million members of the retail stock trading subreddit r/wallstreetbets did something utterly incredible that caused major ripples throughout the US financial system.

The US bricks and mortar computer games retailer Gamestop had been struggling for several years. The COVID-19 lockdowns then hurt it badly such that it closed 300 stores at a loss of $165 million, its prospects looked bleak and its share price was in the doldrums.

Some US hedge funds, notably Melvin Capital and Citron Research decided to take advantage of Gamestop's continuing decline. They did this by short selling its stock - borrowing stock from other financial services institutions to sell and push down the share price, to repurchase later and pocket the difference. The short selling was enormous, with at one point more than 140% of the total shares issued being shorted.

Instigated by the update posts of a member of r/wallstreetbets, Keith Gill aka Reddit user u/DeepFuckingValue, members of the subreddit and associated Discord decided that this situation could be reversed by buying cheaper Gamestop (known as $GME) shares or put options (options to buy the stock at a specific price at a later date) to force up the price. Since shares borrowed for short selling have to be bought back and returned by a certain date, the buying by r/wallstreetbets users put pressure on the funds shorting the stock to buy it back (thus forcing up the price further) at a loss. This is a common technique known as a short squeeze. Similar short squeezes were done against the similarly short sold cinema chain AMC and phone companies Nokia and BlackBerry.

WallStreetBets users had various motivations. The subreddit, calling itself "Like 4chan found a bloomberg terminal." has a great irreverence of usual risk based trading and investing approaches favoured by Wall Street and an antipathy and meme-based mockery of Wall Street institutions. It also has a culture of heavily leveraged options trading, taking large gambles to make profits (known as "tendies") quickly (and in this case there were great profits to be had), with heavy wins and losses frequently shown off in posts. Members often see themselves as the vanguard of the common trader against the powerful elites of Wall Street, a democratizing

IS YOUR APP REDDIT PROOF?

force in finance. However other commentators like Bloomberg's Matt Levine put the motivation down to boredom and trolling - just "utter nihilism, a story perhaps best told with rocket emojis".

The Robinhood Mobile App and Brokerage

To explain how it provides a warning for testers and tech teams it is necessary to provide some history. The most popular brokerage and trading platform for retail and small traders was the mobile app startup Robinhood. Charging no brokerage fees and allowing small purchases of even fractional shares, it was an extremely easy way for young people with time on their hands and only small amounts in their pockets to get started in the world of share investment and trading.

For this reason it was the preferred app for the US-based of the WallStreetBets crowd and heavily used for the short squeeze.

WallStreetBets uses Robinhood to Attack the Hedge Funds

With the sheer numbers of retail traders piling in (especially after a supportive tweet by Elon Musk) the share price of $GME rocketed from $19.95 per share to $347 per share in just over two weeks. Melvin Capital was forced to close their short position at a 30% loss to their entire portfolio, requiring a $2.7 billion investment by other companies to keep Melvin Capital afloat. Overall an estimated $6 billion was lost by investment firms and hedge funds who were shorting Gamestop and the S&P 500 fell by about 5% over the course of about three days as nerves spread through the market. Meanwhile Keith Hill's investment of $52 000 in options was worth $42 million by the share price's peak.

Robinhood, Brought to its Knees, stops all trading in GameStop

The sheer numbers of Reddit traders taking part in the short squeeze also brought Robinhood to its knees. Clearing houses used by the trading platforms started asking for higher amounts of collateral for the trades than the platform could afford. This is important as trades are not instantaneous (usually taking a few days) and usually backed by collateral. Robinhood had to raise $1 billion from its backers and debt facilities to maintain collateral for its trades. Also Robinhood makes 40% of its revenue from a data selling arrangement with the hedge fund Citadel LLC, which part-owns the attacked hedge fund Melvin Capital, and the short squeeze was a conflict of interest that was starting to unravel that agreement.

On January 29th the Robinhood app along with other online trading platforms such as WeBull and IMC Markets took the unprecedented decision to ban or limit trading of $GME and other heavily shorted stocks. Retail investors subsequently turned viciously on Robinhood. Over a hundred thousand poor reviews were given against the Robinhood app on Google Play Store lowering its overall star rating to 1 star, requiring Google to remove them.

Criticism of the attacks on r/WallStreetBets and Robinhood's decision came from politicians, media and entrepreneurs across the political divide - Alexandria Ocasio-Cortez tweeting -

"Gotta admit it’s really something to see Wall Streeters with a long history of treating our economy as a casino complain about a message board of posters also treating the market as a casino."

PAUL MAXWELL-WALTERS
–
A British software tester based in Sydney, Australia with about 10 years of experience testing in agriculture, financial services, digital media and energy consultancy. Paul is a co-chair and social media officer at the Sydney Testers Meetup Group, along with having spoken at several conferences in Australia.

Paul blogs on issues in IT and testing at http://testingrants.blogspot.com.au and tweets on testing and IT matters at @TestingRants.

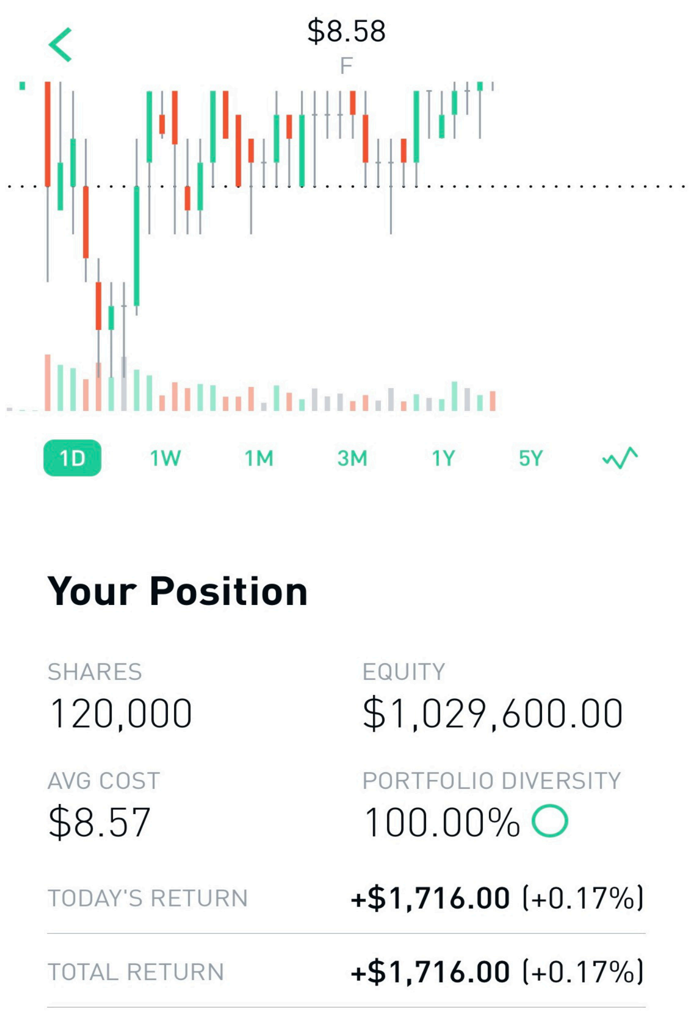
along with Donald Trump Jr. tweeting

"It took less than a day for big tech, big government and the corporate media to spring into action and begin colluding to protect their hedge fund buddies on Wall Street. This is what a rigged system looks like, folks! "

What does this have to do with software and quality?

The answer is a great deal. This is far from the first time that Robinhood and its app have been put to the test and found wanting by members of WallStreetBets and other new retail traders. The risk of large scale attacks and exploitation via social media opens up a new frontier in what business experts and testers have to watch out for.

The "Infinite Leverage" Bug



Your Position

SHARES	EQUITY
120,000	$1,029,600.00
AVG COST	PORTFOLIO DIVERSITY
$8.57	100.00%
TODAY'S RETURN	**+$1,716.00** (+0.17%)
TOTAL RETURN	**+$1,716.00** (+0.17%)

Check this post by u/Moonyachs showing a $1 million equity position leveraged from just $4000 deposit.

A much worse problem with the Premium Gold service of the Robinhood app was exploited later by around twenty r/WallStreetBets members in November 2019. As described by Business Insider it involved the following exploit -

· "Users who pay a premium for Robinhood Gold sell call options with money borrowed in the app (a loan know as a margin or leverage).

· Robinhood incorrectly adds the value of the options sold to the user's cash pile.

· This gives the user more capital to trade with, and the more a user borrows, the more the app adds to their buying power.

· There seems to be no limit to how much a user can exploit the trick."

Call options (in the above case "covered" call options) are contracts that allow the buyer to purchase a stock at a set price at a future expiry date. A seller (or "writer") sells for a fee the right to buy the stock (which they must sell if the buyer asks for it), the hope being that the underlying stock will always remain below the agreed purchase price (known as the exercise price) and thus the option will expire unused - the seller pocketing the cash made from selling the option and retaining the stock.

The bug in this case was that the more the user borrowed to sell call options, the more the app added to their balance and thus the more the app allowed them to borrow. The original discoverer of the bug, u/ControlTheNarrative, used the flaw to write $50000 worth of Apple put options from a $2000 deposit. One user, u/MoonYachts, was able to borrow a margin of at least $1 million for an original sum of $4000! The user u/Cal_Warrior went even further, turning a $3000 deposit into a position of $1.7 million! They wrote "After seeing people on the almighty wallstreetbets wager a timid 50k or so on average with this new feature available, I thought it was only a clear choice to raise the average for the good of all."

Overall about twenty members used the bug to borrow larger sums than were allowed, getting the cue from posts in the subforum. A user u/SocioButt even posted a "Hall of Fame" of users exploiting the bug. It took days for Robinhood to find out and release a patch to fix the bug and communicate with customers and there was no guarantee that it could claim losses from people who used the exploit and lost money. Robinhood also ran the risk of falling foul of regulators such as the SEC and FINRA along with the costs required to take legal action to claim back the funds.

Badly Displayed Losses Resulting in the Suicide of Alex Kearns

In June 2020 the student and budding retail trader Alex Kearns tragically committed suicide after seeing a negative cash balance of $730 000 in his Robinhood Margin (i.e. loan) account. According to his family, later that night the company sent an automated email demanding Alex take "immediate action," requesting a payment of more than $170,000 in just a few days.

A note left by Kearns to his family stated the following - *"How was a 20 year old with no income able to get assigned almost a million dollars worth of leverage? There was no intention to be assigned this much and take this much risk, and I only thought that I was risking the money that I actually owned. If you check the app, the margin investing option isn't even 'turned on' for me. A painful lesson."*

Bill Brewster, a relative and analyst at Sullimar Capital, publicly criticised how the app displayed temporary debt exposure, stating "I'd like them to fix the way that they're showing exposure — I want them to act like a financial platform should act. When you're dealing with retail money and actively soliciting traders under 30 years old to have errors like this is inexcusable and at the minimum negligence."

Robinhood responded by offering to make changes to their in app messages and history page to make the mechanics of trading options clearer, along with providing more stringent eligibility requirements and better educational resources for new investors. However William Galvin, chief financial regulator in the state of Massachusetts, found over 600 instances of people in the state who should never have been approved for options trading by Robinhood's own standards but were. CBS News confirmed how easy it was to get around Robinhood's eligibility checks by simply "upgrading your experience".

Alex Kearn's family have since filed a lawsuit for wrongful death against Robinhood.

Implications for Testers, Quality and Risk Management

The badly displayed temporary debt in the UI and poorly written automated messaging created a tragedy for a brand new trader like Alex Kearns. Robinhood app created the situation where easy access to risky options trading resulted in the tragic consequences as well as permanently damaging the company's reputation. That such a thing was allowed to happen and not flagged up by Robinhood's internal processes is nothing short of disgraceful and a moral failure.

One way that could have improved the interface such as to prevent the above would have been to apply persona based tests - testers creating personas to study the app interface, emails and warning messages from the perspective of new retail traders lacking experience and financial expertise.

The "Infinite Leverage" flaw in particular highlighted the speed in which bugs are made public and exploited in online forums along with the motivations in which anonymous exploiters use the bug to one up each other online. Suddenly issues that may carry one risk if an individual does it are much graver when social media is taken into account and lots jump on the bandwagon. They also carry new reputational and regulatory risks when forum posts go viral and are reported in the press.

”

The "Infinite Leverage" flaw in particular highlighted the speed in which bugs are made public and exploited in online fo-rums along with the motivations in which anonymous exploiters use the bug to one up each other online.

In effect, brokerages and companies reliant on traders in groups like r/wallstreetbets need to be aware that the spotlight is always on them and mistakes and errors will be found out and the word spread quickly. The costs of failure are thus potentially enormous and testers and developers working on these apps have to always be "on the ball". They also need real understanding of the users coming to their apps, along their levels of experience, and the social media worlds they inhabit and are influenced by.

The lesson gained from r/wallstreetbets and other groups of small retail traders in their Gamestop short squeeze is that they are realising their immense latent power and acting in ways that institutions on Wall Street would never have predicted. This includes using apps and brokerage tools to make incredible purchases together which makes collusion difficult to prove and police. This does not just affect shorting hedge funds but the tools they use - online brokerage apps now need to allow groups of small retail traders to make large moves en masse at individual stocks and always have the collateral to manage it, otherwise be punished by these same users.

For the rest of us, this is a parable about the power of social media to allow groups of ordinary individuals to troll and exploit - whether it be as anger against the elites, for financial gain or simply because they were bored and it is a funny thing to do. It is a lesson in that just because ordinary people take part in an activity or use your service doesn't mean you control them, predict what they will do or think they will act (in your definition of) rationally. We have to think again about what we expect of users and the online communities they dwell in. For those of us making and testing products to be used by the masses, this is a wakeup call to all.

*Thanks to the great editing work and support of JeanAnn Harrison, without whom this article would have been a poor shadow of itself.

Heuristics for identifying corner cases for testing

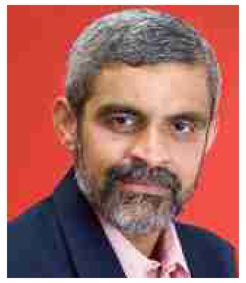
T. talks testing

Summary

Corners are interesting as they are subtle, invisible really. They could be complex with many things that intersect and therefore display an unique behaviour. They may not necessarily be symmetrical at ends, nor be similar to behaviour in the middle.

As a developer focused on solving a problem for typical or generic cases one may not see the interesting extremities. For example we do everything right for a system in normal state, but miss out what happens when it is brought up the first time. This article outlines eight heuristics I discovered when testing a product that we were building, a SaaS platform.

The heuristics outlined are based these aspects : Time, Lifecycle, Transformation, Position,Space, and Size.



T ASHOK

–

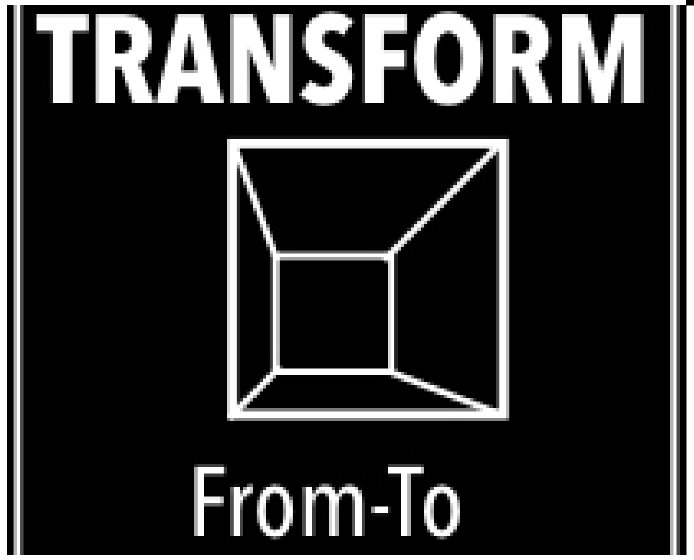
Ashok is a test Professional. Founder & CEO STAG Software.Ultra Cyclist. Ultra Runner. Wordsmith.

Passionate about excellence, his mission is to invent technologies to deliver "clean software".

He can be reached at ash@stagsoftware.com

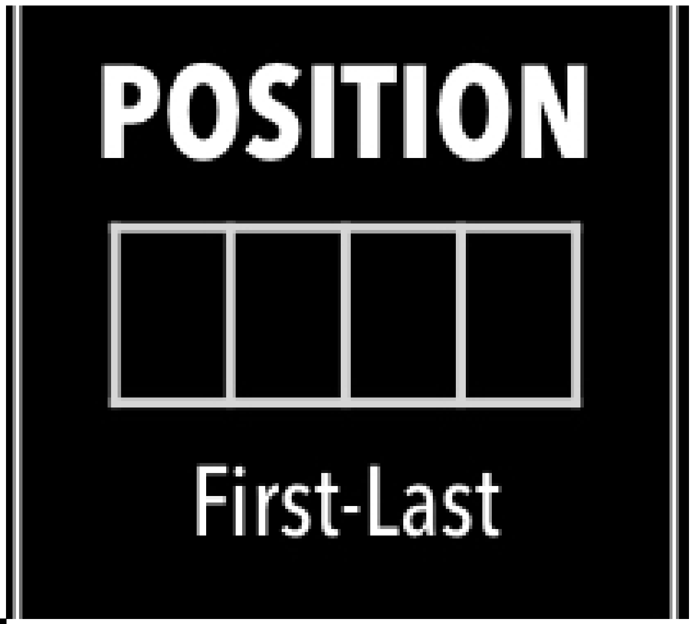
The first time when something is used. Thinking from the perspective of time. Doing something on an absolutely fresh system. Creating the first project, registering the first user.Doing the final action, removing an user. First time when a transaction is done on a empty system. Purging a system of content, signifying removal, the end.

Repetition of system states in terms of cycling through. Starting off, then doing activities and coming to an end. Then restarting and continuing. A workflow that is half done, suspended and then continued to finish. Finish by abandoning it or ending with to a logical closure.



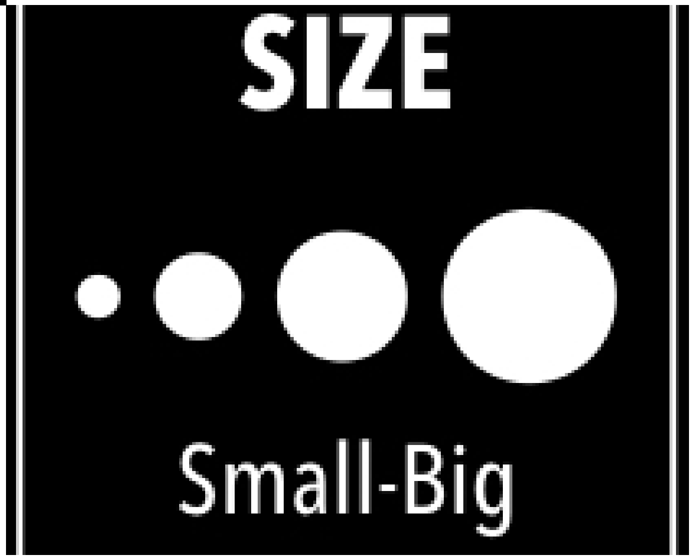
Changing something like say formats, views an act of transformation. In the case of UI, this could be relate to responsiveness like reaching the extremities of views? In case of content transformation, reaching the extremities of too large or too small or null content being transformed.

Looking for interesting behavior in case of the elements that are right at start or end. What happens when elements in the middle move to either ends?



The nation of space as when contents close are far, shrunk or expanded, especially when at extremities of too close or distant, too small or large. An example of responsive UI, when screen is shrunk or expanded.

The notion of volume, size when some is really small or extraordinary huge. Say uploading super large files, or really nothing or rally small ones. In case of display, showing tiny/large content/diagram, maybe via zoom?



RISE.

Synapse QA celebrates International Women's Day 2021.

*Synapse QA, a community-driven co-writing space specializing in Software Testing and QA space, celebrated an International Women's Day 2021 with their noteworthy **RISE** event.*

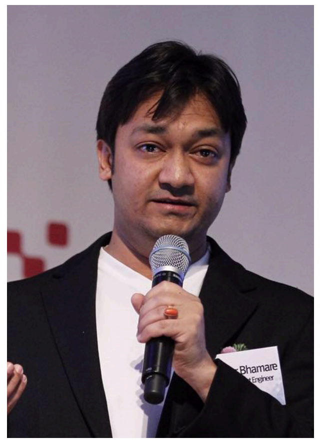
*A number of great women in tech field shared their wisdom and message on this occasion as part of **RISE**.*

We highly recommend our readers to check out their all posts published on this occasion and even to share it further.

Thank you Synapse QA for organizing such a lovely event. You won our heart!

COMMUNITY

A Synapse QA Anthology



LALITKUMAR BHAMARE
Chief Editor "Tea-time with Testers"
–
Passionate tester @XING_de
Director @AST_News
International Keynote speaker.
Software Testing/Quality Coach.

Connect with Lalit on Twitter [@Lalitbhamare](https://twitter.com/Lalitbhamare) or on [LinkedIn]() and [Xing]().

Put the Craft back in Testing

Ten years ago when we first launched Tea-time with Testers, the community of testers needed a platform, a medium to express their thoughts, ideas, concerns, and sometimes even their disappointments.

One after another we kept launching new issues that consisted of thought-provoking articles by bright minds in testing field. We published articles based on experience that helped testers solve problems at their work.

We published articles around technical testing, automation in testing, upskilling tips, and techniques to become an all-round tester.

The pleasure of finding things out on a tester's face was as alive and visible as it was when the magic wand of automation made them (and their higher-ups) look amused and awestruck.

Much to my surprise and amusement, that pleasure seems to have been lost somewhere today.

The discussions that inspire critical thinking, deep analysis, mindset building, and challenging ideas are seen sporadically. On the contrary, an unreasonably high focus seems to be on ~~over~~ engineering solutions to make testing(?) faster and easier.

I still do not understand our industry's obsession with "automate everything" nonsense. It was very much of conquest for some a decade ago. I am sure it was there even decades ago before that and here we are today, still selling and buying that snake oil.

No doubt we need automation, we need engineering solutions to assist testing and expand it further but it does not have to be done at the cost of killing and sabotaging the craftsmanship. Testing as a whole is an intellectual discipline and simply focusing on making machines do something is never enough.

What we need is the balance, the support of engineering solutions to make more room for nurturing the software craftsmanship.

While satisfying millions of customers can be a challenging task and therefore providing a good-enough quality software sounds a reasonable thing to do, let's please not forget that a glitch in software [can cost somebody a life](). And no amount of engineering innovation can repair that damage.

As long as we don't wait until millions of our customers to get affected that badly, we shall not live in vain.

Let's please put the craft back in testing!

See you next time.

TEA-TIME WITH TESTERS

JOURNAL FOR NEXT GENERATION TESTERS

CONTENT PREVIEW : ISSUE 02/2021

MORE AWESOMENESS IS ON YOUR WAY THIS JUNE

01

AN EXPERIENCE REPORT ON RST

In early 2019 one of the Engineering Managers in my organization encouraged me to attend the Rapid Software Testing Applied online course. Initially, I was skeptical about how beneficial a remote course on testing methodology might be for me. I had several years of experience working in testing roles and I was quite comfortable with the way we were approaching testing in my current product team...

02

SHIFT-LEFT MET EARLY WITH E-MBT

Nowadays more and more test activities are being automated, especially when it comes to executing test cases. But what about the development of test cases? Are the test cases still being developed in a structured way with certain test coverage and based on a well-considered risk?

03

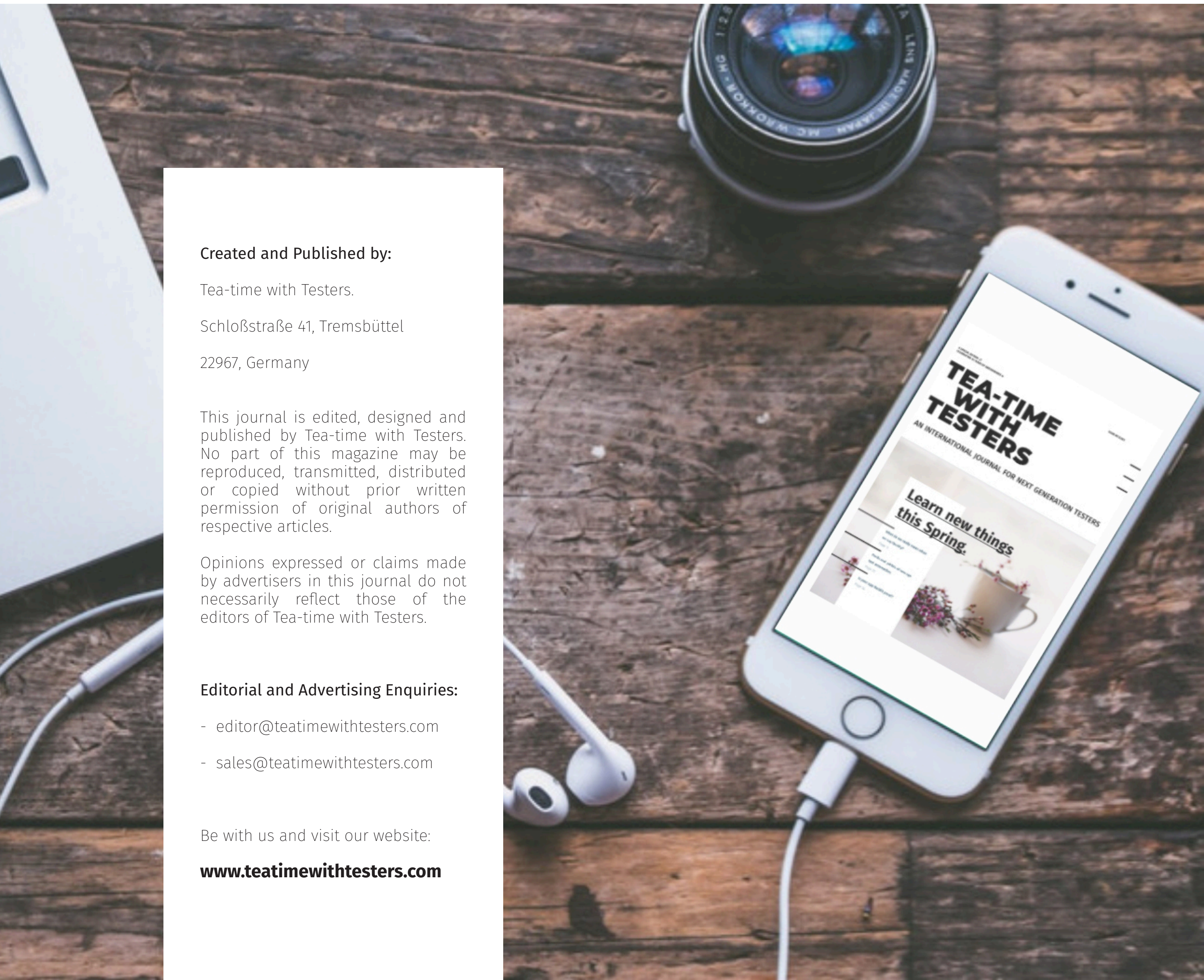
21ST CENTURY SKILLS FOR TESTERS

In this fast-changing world, you can ask yourself, what are the skills that I should have to be able to make a difference with the colleagues around me? Is it being an expert in test automation? Or able to read all kinds of different code languages? At least you know that the technology you learn today can be gone tomorrow. For example; how much time will it take to learn a new programming language?

INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

TEA-TIME WITH TESTERS

THE SOFTWARE TESTING AND QUALITY MAGAZINE


TEA-TIME WITH TESTERS
AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS
Learn new things this Spring.

Created and Published by:

Tea-time with Testers.

Schloßstraße 41, Tremsbüttel

22967, Germany

Editorial and Advertising Enquiries:

- editor@teatimewithtesters.com
- sales@teatimewithtesters.com

Be with us and visit our website:

www.teatimewithtesters.com