AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

# Tea-time with Testers

DECEMBER 2020


# TEA-TIME WITH TESTERS

Created and Published by:

**Tea-time with Testers.**
Schloßstraße 41, Tremsbüttel

22967, Germany

Editorial and Advertising Enquiries:

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

Lalit: (+49) 15227220745

# Editorial

## Gratitude is worry's cure.

While I write this on the very last day of unforgettable 2020, my mind is clouded with so many thoughts and flashbacks of this year.

2020 has been hard. There is no denying that. But, it seems to have opened lots of new possibilities and it compelled us to take a pause and reflect on life. It occurred to me at least in my case. Though I greatly missed working together with my wonderful colleagues or missed meeting friends and spending quality time with loved ones that live far from here, I also got a chance to gather myself back, meet myself again, invest in myself and in people I dearly care for.

I do not know what all 2020 took away from me really but it has certainly given me lots of things I would forever be grateful for. The time I am got to spend with my dog and my little one has been precious. I could know about the beautiful birds that nested in my backyard during spring every year which I was clueless about in past.

I have closely seen the trees changing their colors as seasons changed. have seen those beautiful flowers blooming back to life in spring despite being almost dead during freezing winter.

I read books which I always wanted to but could not. I watched movies with loved ones and created memories to cherish forever. I learned new skills, I invested in researching topics that fascinated me. I got elected to the Board of Directors at the Association for Software Testing this year. I got to closely work with the amazing team of The Test Tribe to deliver Tribal Qonf 2020 and TestFlix. Together with my fantastic program committee, I could create an impressive program for ConTEST NYC 2021 as its Program Chair.

Oh, and how could I forget this? It has been 2020 that motivated me to restart and reboot Tea–time with Testers and meet all of you once again. We added two wonderful people to our team this year, Klára and Astrid. And having them in Tea–time with Testers family makes me feel immensely proud.

Despite the tough times we all have through this year, I choose to be grateful for what it has given me. If I may, I would beg of you to do the same. On this last day of the year, allow me to share this beautiful poem by Susan Frybort.

**In a world where rugs are pulled and tables turned, words fall flat and hearts get burned, there are three things I know for sure:**

**Life goes on,**

**love endures, and**

**gratitude is worry's cure.**

Until next time, people!


**– Lalitkumar Bhamare**

editor@teatimewithtesters.com

@Lalitbhamare / @TtimewidTesters

# QUICK LOOK

## SPEAKING TESTER'S MIND



## IN THE SCHOOL OF TESTING



## T' TALKS



## QA INTELLIGENCE



## SPOTLIGHT

WHO CARES?

*Should public care about software testing? Find out.../p15*

## NEED OF THE TIME

EXCELLENT SOFTWARE TESTING

*What is it really about? Alice wil tell you.../p26*

AST ASSOCIATION FOR SOFTWARE TESTING

AST- BBST 2021 SCHEDULE AND SIGN UP!


Speaking Tester's Mind
- straight from the author's desk

# Whose Quality Is It Anyway?

## - Djuka Selendic

Photo by - Emily Morter on Unsplash

I've always been a lover of nature. I grew up surrounded by nature and learned how to explore as a child in the woods close to our house and a forest only a half hour hike away. As soon as I was old enough to make reasonable decisions on my own and remember my home phone number, I was allowed to explore the natural world around me without much adult supervision. Thanks to my parents, I knew that littering was a problem and I grew up with an awareness of leaving the land better than I found it. In today's terms, I was what some would describe as a "free-range" child.

As an adult, my love of nature and exploration in all forms were the primary reasons I found my home in the Burning Man community. The community has an established Leave No Trace (LNT) ethos, one of ten guiding principles of Burning Man.


*Our temporary homestead in Black Rock City*

Burners refer to anything that is not "originally of the land" as Matter Out Of Place (MOOP) and any MOOP found or created is to be disposed of in an environmentally responsible manner. Burning Man takes LNT very seriously and in addition to every citizen doing their part to keep the playa clean, the event itself has volunteers dedicated to ensuring the desert is restored post-event to its original pristine state. Side note:

Any tester will find the MOOP Map *is a fascinating way to visualize Leave No Trace coverage in Black Rock City and measure progress from year to year.*

I am a LNT enthusiast and my standards for quality are very high when it comes to MOOP. As a tester, my standards for evaluating product quality are also quite high.

This brings me to some important questions:

## What exactly do I mean by "quality"? And how much do my standards matter?

In his book, *Quality Software Management: Systems Thinking,* Jerry Weinberg brings up a few foundational points about quality we should consider to answer these questions:

### *Quality is relative*

> *"…what is adequate quality to one person may be inadequate quality to another." (p. 5, Weinberg, 1992)*

During my adventures at Burning Man, the heuristic I apply is "If you see it, pick it up" – when I see MOOP, I stop and pick it up to dispose of it properly later on. I do not feel good if I see MOOP and do nothing about it. I feel the very presence of it threatens the value of the natural environment we all have the privilege to enjoy.

This heuristic does not apply well in the "default" (that's what we burners call the world outside of our community) because the larger society does not generally cultivate a common ethos around LNT, so there is a lot of MOOP everywhere. Many people treat the environment like their own personal trash can or ashtray, throwing trash and cigarette butts anywhere they please. In the default world, it is not possible to stop and pick up every piece of MOOP one sees.

***Spot the MOOP! If you see it, pick it up!***

The presence of trash in the natural environment negatively impacts my enjoyment of those spaces. The value of my experience is lessened when I see a water bottle gently floating down a stream. The presence of trash also proves that there are people out there – many people – who are not negatively impacted by the presence of trash at all. Or at least not enough to do anything meaningful about it in the moment.

Many people are content to just walk past that crushed beer can sitting on the side of the trail. Perhaps people do not feel it is their responsibility to clean up after others or maybe they do not want to touch trash on their afternoon walk when they do encounter it.

Whatever the reasons:

- people have different perceptions of value,
- and these perceptions affect how they perceive quality,
- which ultimately influences their decision on whether to do something about it.

In software development teams, testers will often find their notions of quality at odds with developers, product managers, designers, and other stakeholders. Much of the time, our standard for quality will be much higher and our level of scrutiny of the product much deeper. Our craft requires that we evaluate the product from multiple angles, so testers are in a great position to know what the product risks may be, what problems the team may run into, where the known bugs exist, and so on.

Over time, we become repositories of product knowledge as we build our credibility within our teams. When influencing decisions about quality, we can advocate for what we think makes sense or sound the alarm bells when something about the product isn't making sense (and hopefully have some evidence or data to support our arguments).

## There is a person behind every statement about quality

> *"Quality does not exist in a nonhuman vacuum. Every statement about quality is a statement about some person(s)." (p. 5, Weinberg, 1992)*

Burning Man's adherence to the Leave No Trace principle is a statement about quality. Defining anything not "originally of the land" as MOOP and setting the expectation that all MOOP be removed raises the collective bar for the LNT effort in the Burning Man community. Every responsible burner participates and does their part in keeping the playa clean.

In addition, the Burning Man organization must meet strict LNT requirements for use of the public land to be able to hold the event every year, so the community and the Burning Man organization work together to create accountability around LNT. Guidelines are established as to what constitutes MOOP and how to deal with it. The community has a clear vision of what our quality story looks like from year to year.

In software development organizations, we often hear the following statement about quality: "Quality is everybody's responsibility." Often this statement comes from somebody in a leadership position, or it may be heard among frustrated testers who feel like the burden of product quality falls only on them. The problem is that this statement is very vague and usually doesn't go into enough detail to be useful in practice. And if we do not know the specifics of what various individuals and departments do to improve quality and what it means to them in their contexts, then it is nearly impossible to instill the shared ethos of "Quality is everybody's responsibility."

Most organizations do not have a clear picture of what quality means to the various stakeholders in and outside the organization. Often leaders within the organization have disparate opinions on what's important. Without effective leadership to steer the quality ship, teams end up building products that are incongruent with the company mission or consumer expectations, departments fight for budgets and position leading to bad blood between teams, and the wrong projects are prioritized.

A clear vision of what quality means to an organization is key and being specific about it is even more important. What does the organization value? Where does it want to go? What value does the organization want to bring to its customers? What about all the other stakeholders? What do they value? What has to improve now?

### *Opinions and decisions about quality are political and emotional*

*"More quality for one person may mean less quality for another." (p.6, Weinberg, 1992)*

The existence of trash cans in parks and natural spaces is a perceived necessity because, as a society, we have not made any efforts to cultivate a culture of LNT. One of the tenets of LNT is to pack out your own trash. This means that once you're done drinking that soda down at the beach, you take the can home with you and dispose of it.

There are no trash cans or dumpsters for public use at Burning Man. By LNT standards, trash cans actually lower the value of the experience and disincentivize some participants from practicing LNT.

Parks officials make decisions to provide trash cans in certain places, yet MOOP still ends up thrown in the bushes, or maddeningly, next to the park trash can. The existence of trash cans does not necessarily improve the cleanliness of the park or the value of the park experience. Trash cans may alleviate the existing problem from becoming or looking worse but they do nothing to change the underlying offending behavior.

Worse than seeing trash scattered randomly around a park is seeing a park trash can overflowing with it.

Trash cans are a band aid; not a solution.

When evaluating software and assessing quality, it makes sense to pay attention to the trash cans and what's in them. I've been a part of many product builds over the course of my 22-year tech career. During every one of those builds, certain design or technical shortcuts in the product were made. Reasons for this vary. Often, the team is under a tight deadline and there is simply no time to dedicate to a better solution. It becomes problematic when every decision starts to feel like a shortcut and then the whole product experience starts to feel more like trash, and less like something a customer would value. It's good to stop once in a while and assess the landscape, as a team. Ask some questions:

How many of those trash cans exist in the product? How many are dumpsters? Are any of them an active dumpster fire? Are we dealing with the accumulated trash?

There are times when it may be ok to let trash pile up. For example, during the build stages of a feature, we may let some bugs sit around until we fix other more important stuff and then come back to those bugs later. Eventually we should strive to take out the trash or risk an even bigger mess.

### *A practical definition of quality*

*"Quality is value to some person" (p.7, Weinberg, 1992)*

Jerry Weinberg's definition resonates because it takes into account the political and emotional motivations behind decisions about quality. He goes on to say, "By 'value', I mean, What are people willing to pay (or do) to have their requirements met?"

I've talked at length about value in the above paragraphs. Our emotions and political motivations drive how we assess value, how we form opinions about quality, and ultimately what we do about it, if anything. I take a trash picker tool and a trash bag on most hikes and walks now, so that I can do something about it, knowing the job will never be complete and there will always be more trash to pick up.

*"Whose opinion of quality is to count when making decisions?" (p. 7, Weinberg, 1992)*

It matters whose opinion counts when making decisions about quality.

Park officials making decisions about placing trash cans in the parks may not care about the opinions of persons like me who practice Leave No Trace anyway. Their concern is with those who would not take their trash home with them and instead litter it about the park. Their decision to provide trash cans around the parks is a quality decision that makes sense, given the context.

Do I like that this decision has to be made? No. Do I accept it? Yes.

This happens in testing, as well. Testers are generally not the ones making the final decisions about product quality.

The stakeholders who do have the power to make those decisions are continually weighing whose influence and opinion matters as they make those calls. Realize that it may not be your opinion that matters, depending on the context. It's our job as testers to support these stakeholders with evidence and data about the product and the risks so that they are empowered to make informed decisions.

We get to influence those decisions, and that's good enough. Many times, the final decisions will mirror some aspects of our influence, but there are also times when no matter what we say, a decision we do not agree with will be made. Those times that the decisions made do not reflect our advocacy, it's time to accept it and move on.

There is so much more to test and it's not personal.

**References:**

Weinberg, Gerald. Quality Software Management: Systems Thinking. 1992. p. 5-7.

**Djuka** is a Test Engineering Manager at Northwestern Mutual where she serves a team that supports a consumer-facing product. Two years ago, Djuka became a student and practitioner of Rapid Software Testing. Djuka values RST as a discipline and methodology to guide the performance of testing in any project situation.

Djuka started her career in technology in 1998, when she attended her high school's first HTML class and landed a job as a student web designer. Since then, her career has taken many twists and turns as she tried her hand at a variety of disciplines: web developer, website manager, content manager, and finally a tester.

Out of all the paths she's walked in her 22-year career, testing has been the most interesting and the most challenging. Djuka is convinced that she'll never stop testing.

https://www.linkedin.com/in/thedjuke/

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.

Photo by - Woodendot on Unsplash

# The Future of Test Automation Must Be Intelligent

- Eran Kinsbruner

Photo by Bud Helisson on Unsplash

A decade of Agile, DevOps and Continuous Testing is behind us. I heard this week in a conference I was presenting the phrase “Agile is Dead” and other claims about testing being inefficient in many ways.

While many tools have evolved over the past years, and test automation practices and lessons learned have been communicated to practitioners, it is IMHO not enough to move the needle on the efficiency of test automation and continuous testing – especially in the demanding and ever changing digital space.

To have a better DevOps pipeline, and to minimize both the waste as well as the level of escaped defects, testing and especially test automation must become smarter.

In this post I would list the opportunities for test automation to be more intelligent.

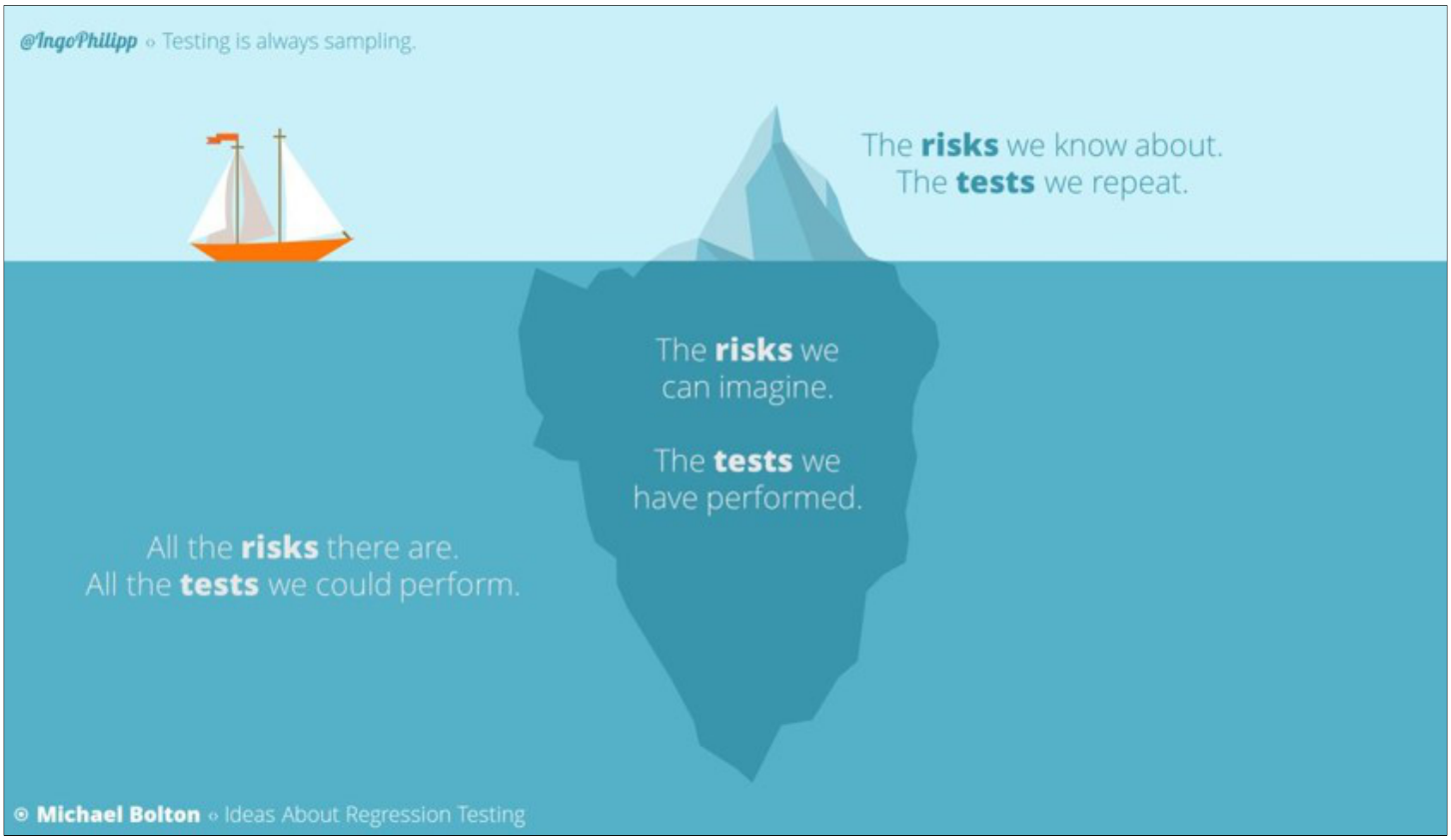
**WHAT?**

Test automation intelligence should guide engineers on which test automation scenarios to execute, automate if missing, and continuously maintain. This is one of the most complex debates within each automation team – WHAT TO AUTOMATE?

Answering this question is never easy, requires risk management, tradeoffs, and often guessing. In addition, it does not always align with the sprint time frames, hence, being thrown out of the software iteration.

With more intelligent solutions that are based on production data, business driven decision making, code analysis, teams will be in a better position to make such decisions.

I love the below visual that Ingo Philipp and originally, Michal Bolton drew around testing is always sampling. This shows that test automation usually addresses what we know about risks and the product out of the entire ocean of risks and features that we either don't know, cannot assess, or things that were addressed in the past and there isn't sufficient data to advise whether we should continue using.



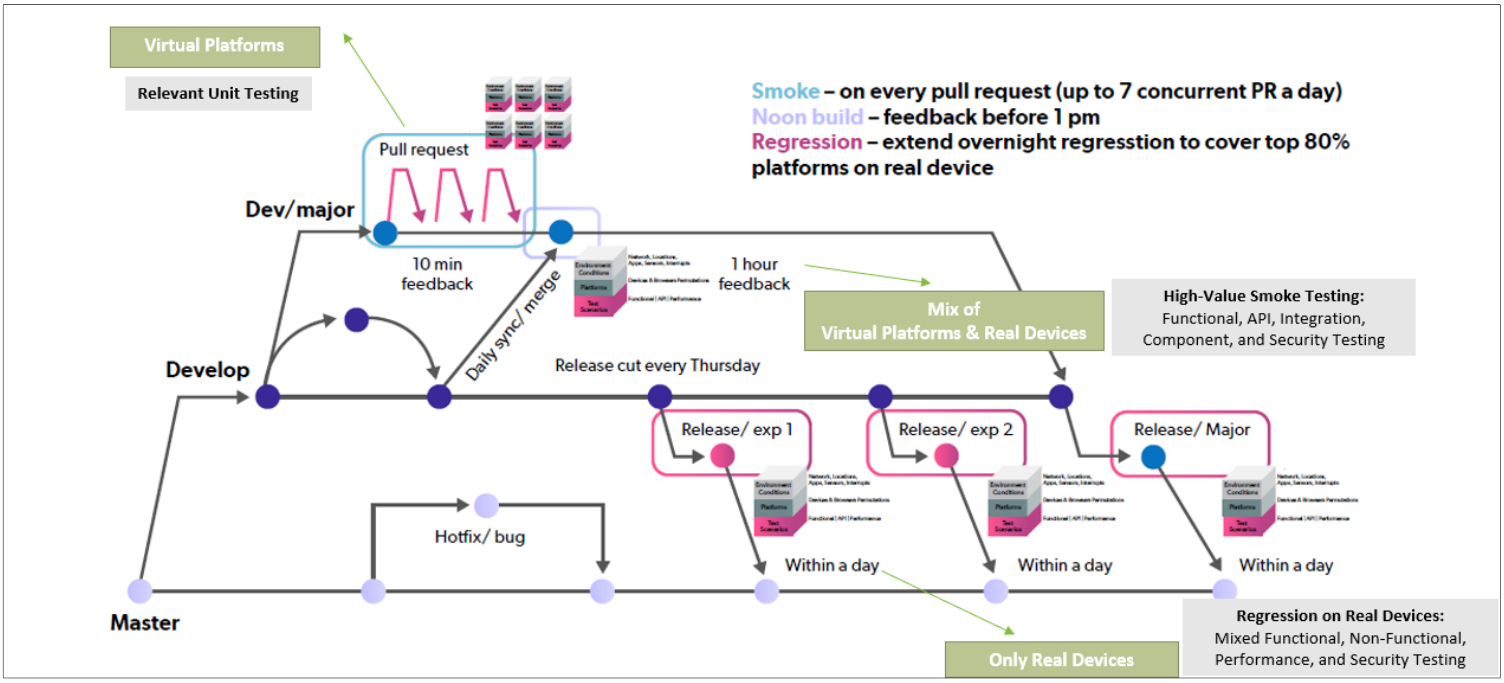

New tools that are rising such as Launchable, SeaLights and others aim to help answer some of the above questions, and to help form a continuously relevant regression suite that matches the most recent product code changes, history of defects and more.

**WHEN?**

The second question that needs to be answered in a more intelligent way is – When to automate and execute my test scenarios? Some say – everything , everyday, some say – everything shifts left, and some are breaking the tests based on personas, skills, and naïve objectives.

Scheduling and deciding which tests to run when must be smarter, and needs to be tied to the overall process, the feedback loops, and the ability of the developers to act upon such execution feedback. If testers throws on the developers to late or to early the results, it makes it unrealistic for them to prioritize, resolve and fix.

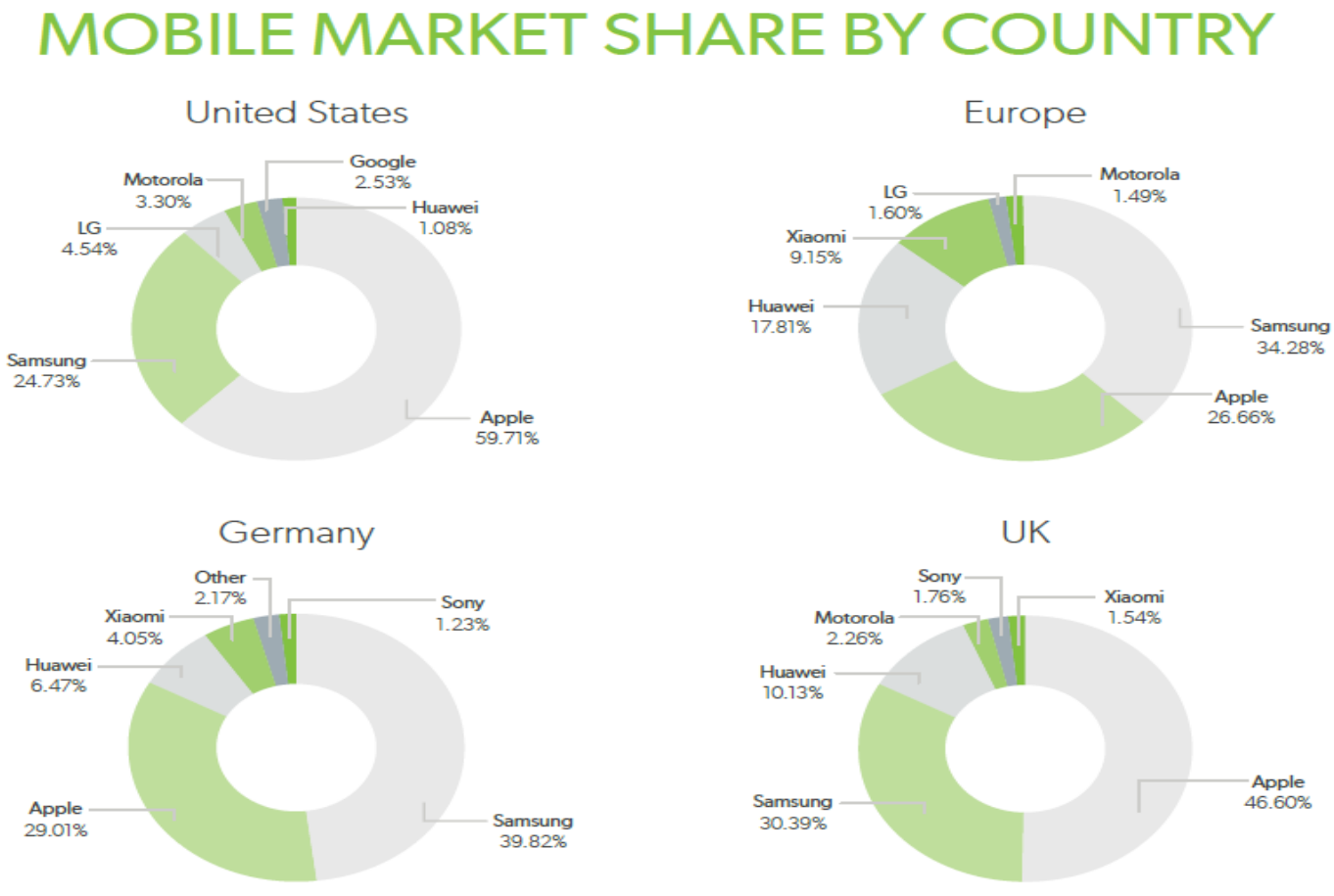
Intelligent systems, must be able to determine based on the above WHAT section, which test scenarios, types, and cadence, such should be executed. In an ideal world, an intelligent system would be able to split between CI, BAT (build acceptance testing), integration, regression, NFT and production synthetic monitoring the entire test suites.




As the above diagram suggest, the pipeline has a pre-defined set of quality gates, phases, and milestones to manage and mitigate risks. An intelligent testing system should effectively place all the right test scenarios –**CONTINUOUSLY**– in the right phases. Keep in mind, the right test cases change from one release to the next.

**WHICH?**

Now that **WHAT** and **WHEN** questions are addressed, it is important to have a smarter approach into test automation coverage – on **WHICH** platforms and environments should my test automation scenarios be running constantly? Which mobile devices, which desktop browsers? which backend configurations are important? which data-driven scenarios should be plugged into the test cases? Which load and KPIs should be applied so I get the proper user experience feedback from my testing? and more.



Determining the WHICH in test automation is an art, and require data analytics, market research, customers validation, production monitoring, AIOps systems and more, but it is an imperative for success.

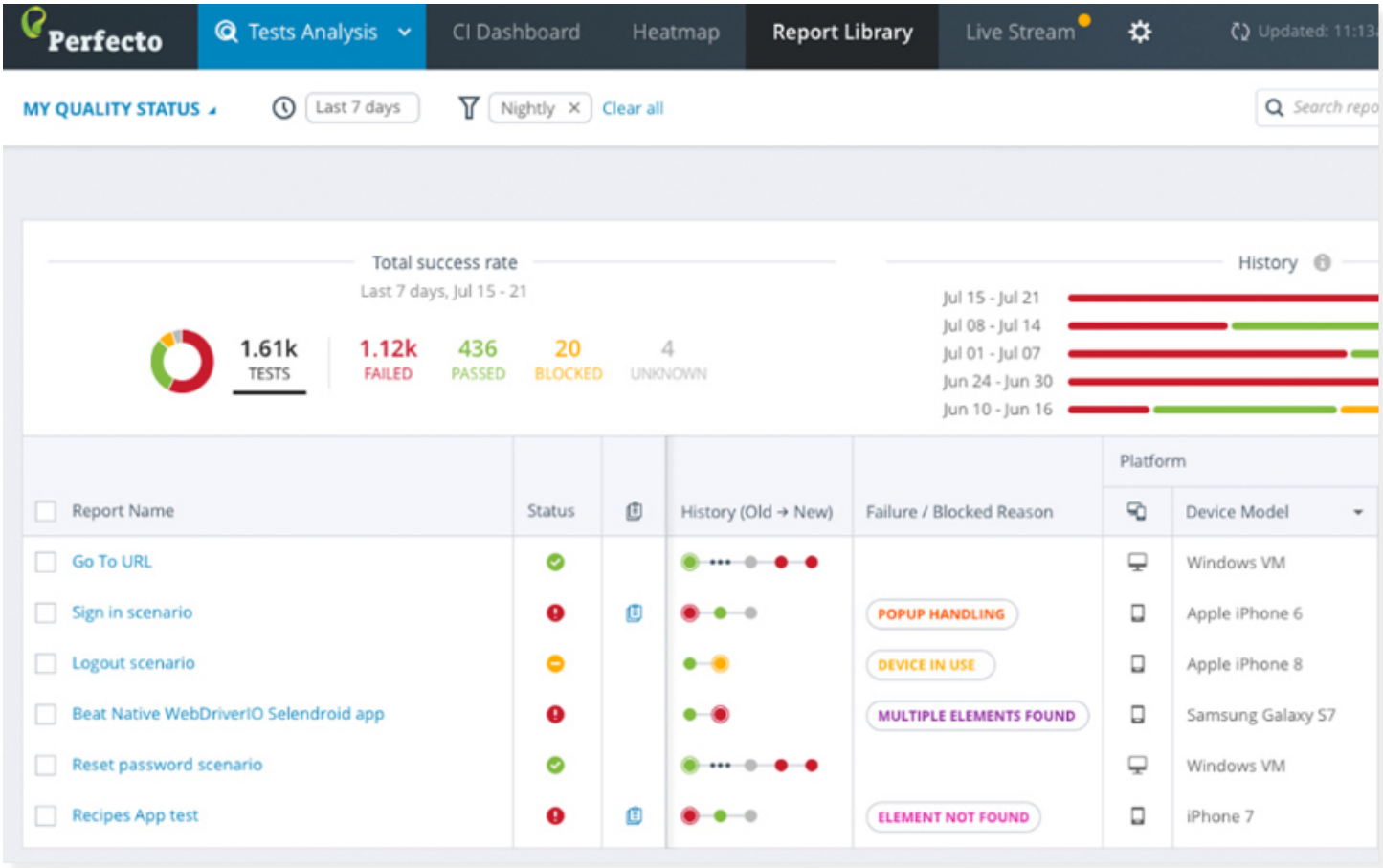
**WHY?**

After determining few **WHs** questions in test automation above, it is time for the analysis part of test automation intelligence – We defined what to automate, when to run, and on which platforms and environments, but once executing the tests, can we efficiently determine WHY tests are failing? What is the root cause per each failure so we lower the time for resolution (MTTR)?

Here, intelligent reporting platforms are critical in minimizing the feedback loop, the future decision making on which tests are of higher value, which platforms are more flaky then others and much more.

Without a solid test automation analysis platform, a great chunk of teams investment in testing can be of waste.

It is not that complicated to realize why tests are constantly failing, and put them into buckets of failure reasons. Perfecto is doing that in an automated fashion through its smart reporting SDK, and allows teams to filter a lot of the test noise post a regression cycle completion.



**HOW?**

Test creation has always been one of the hardest and most challenging task for SDETs and business testers. It includes great technical skills, meeting short time windows for automation, and requires deep understanding of the product requirements to really automate what's expected. While many test automation frameworks evolved over the past years, Including Selenium, Appium, Espresso, XCUITest, and Cypress, these are all code based frameworks that even when coming on top of a BDD platform, they are hard to maintain over time, and analyze when they result in a failure.

New and more intelligent test automation frameworks that are codeless or low-code have grown and are becoming a new reality of test engineers. Such tools can really help address few key challenges that are prominent. Time to automate, time to maintain, flakiness and skillset gap. Intelligent test creation that is plugged with a TIA can be the future of smarter test automation, and enable practitioners to create the most relevant test scenarios with minimal maintenance and shorter amount of time. As I am writing this blog, such tools are still growing and have a lot to prove to the world prior to being widely adopted, but this in my mind should be the future of test automation, and once this is in place, connecting these tools to the cloud for parallel and scaled testing will be the ultimate solution for continuous testing.

**BOTTOM LINE**

A lot of what is written in this blog already exists to some extent, some if being developed these days and will be available in 2021. It is important to start drawing within each DevOps team the future of test automation goals and objectives and learn how such solutions will fit the process from test automation creation, execution, maintenance, analysis, TIA, defect management, quality governance and more.

This year, I launched my 3rd book that focuses on "Accelerating Software Quality: AI and ML in the Age of DevOps". This book looks at the modern DevOps and explores how using AI and ML and smarter algorithms, teams can maximize their productivity and deliver more quality functionality faster.

There is a lot to get excited for in 2021, and especially around test automation in DevOps – Happy New Year and Good Luck in your Future of Test Automation.

**Eran Kinsbruner** is Chief Evangelist and Product Manager at Perfecto by Perforce. He is also the author of the 2016 Amazon bestseller, "The Digital Quality Handbook," "Continuous Testing for DevOps Professionals," which was named one of the Best New Software Testing Books by BookAuthority, and "Accelerating Software Quality: AI and ML in the age of DevOps". He is a development and testing professional with over 20 years of experience at companies such as Sun Microsystems, Neustar, Texas Instruments, General Electric, and more.

He holds various industry certifications such as ISTQB, CMMI, and others. Eran is a patent holding inventor (test exclusion automated mechanism for mobile J2ME testing).

He is active in the community and can be found all over social media (Facebook, Twitter @ek121268, LinkedIn), and on his professional blog: http://continuoustesting.blog

# Who Cares?

- James Thomas

Photo by Kai Pilger on Unsplash

Should the public care about software testing? That's the question that the Association for Software Testing and the BCS Software Testing Specialist Group asked at our first joint peer conference on 22nd November 2020. The remit was:

> *It's our contention that most people don't think very much about software development and software testing, despite software being deeply embedded in almost all aspects of our lives.*
>
> *When there's a publicised issue such as a national bank failing to process customer orders for several days, a government IT project overrunning for years and then being canned, or a self-driving car causing a fatal accident, then society might take notice for a while.*
>
> *However, even when that happens it's rare for the complexities and risks associated with the creation, integration, and maintenance of software systems to be front and centre in the discussion, and testing almost never makes it to the agenda at all.*
>
> *In this workshop we aim to explore why that is, and what testers, testing organisations, software development companies, and governments could do to persuade the public that software testing is worth understanding and caring about.*

As with so many events in these troubled times, we shifted an in-person experience online and were concerned that the buzz that comes from being in the same room as other people who are both knowledgeable and care deeply about the topic would be lost.

I'll blog about the way we set it up later, but I'm pleased to report that the buzz was still there. I'll also blog about the breadth and depth of the conversations we had around the presentations but here I'm just going to summarise what the speakers said.

--00--

We kicked things off with Fiona Charles using Qantas flight QF72 as an example of how well-intentioned automation, complexity, and fail-safe mechanisms can result in at best unwanted behaviour and, at worst, disastrous outcomes. In that incident, faulty sensors caused on-board systems to misinterpret the plane's attitude as dangerous and take unnecessary action to correct it. Flight envelope constraints prevented the pilot from overriding the manoeuvre, leaving him a helpless spectator as passengers were tossed violently around the cabin.

Fiona posed the question "what level of control should we keep for ourselves?" and suggested that we have to build products that keep people at the centre of operations, facilitating and enhancing skilled human decision-making and enabling the human operator to work in whatever way is best for them, not fight against a system built for its own convenience.

Many modern systems (particularly those labelled AI) rely heavily on data, but data is not knowledge, and it comes with biases. As an industry and a scociety we are increasingly recognising this, but we tend not to recognise that the way the data is collected is biased, the architecture of the systems is biased, the models underlying the design are biased, the decision processes are biased, and, in fact, anything humans create is likely biased in some way.

To help ourselves to see that bias and counteract it, Fiona said, we need to take seriously such questions as what could possibly go wrong, what possible outcomes there could be, and how does our product solve well-known concerns such as security or accessibility? Oh yes, and we should look to tune our bullshit detectors.

Harmful outcomes also concern Amit Wertheimer. In his presentation he took the position that software testing as an activity in its own right is just one way to help to create working products that do no harm. He described a separate team of testers distinct from developers as a crutch, providing a way for the builders of a product to disengage from the need for checking their work.

For Amit, in general, developers should be reviewing what they create and should be ready to "feel the pain" of it being found wanting by its users. That's not to say that the critical thinking skills that we testers like to claim have no value, but rather that other people, directly involved in development, can provide them too. He makes an exception for highly-specialised fields where domain knowledge may be instrumental in understanding the desired and observed behaviour of a product.

The public should not care about these details, though, he says. The public should care whether their product works and does no damage, not whether testers were involved in its production, or how.

Huib Schoots offered this perspective too, and also the opposite: the public should not care about software testing because that's the responsibility of the producers, yet the public should care about software testing because they need to be able to trust products, particularly in safety-critical situations.

He asked us to consider what happens when there's some kind of significant software failure. People tweet about it for a while, it blows up in the news perhaps, and then we all forget about it. Where is the forum in which the public can ask questions and request improvements? How can regulation and the law evolve with rapidly-changing technology?

He called for efforts to make an environment in which the public and software producers can get together to discuss how people's needs and concerns can be met, but then also warned about the risks of these being overrun by fringe views and conspiracy theories and noise.

Eric Proegler had no qualms about coming down on one side of the question: the public must care about software testing because of the increasing dependency of our lives and lifestyles on software.

Testers also have something to think about, he says. With current technology, it's easier than ever to create an "AI solution" and push it out to innumerable devices at a terrifying pace with huge numbers of poorly-understood and dynamic dependencies.

Software testing is under siege, in Eric's view.  One way forward is to find ways to incentivise better testing. Weak testing is motivated by short-termism, by getting to market, and by meeting quarterly targets. The business need trumps the societal need most times and Eric challenged us to wonder how we can change that.

The BCS are trying to change that, Adam Leon Smith claims, by engaging with government, the media, the public, and industry to highlight failures in IT and understand how they can be reduced both in frequency and impact. "We shouldn't hinder innovation" shouldn't be a popular opinion, in his opinion.

Regulation and certification is one avenue that can be explored. While this might seem like an anathema to some, Adam noted that software testing in some parts of some industries is already regulated, and cited the UK gaming machine testing strategy regulation as an example.

Acknowledging that testing is context-dependent, Adam speculated that there is scope for wider regulation of this kind. He would begin with high-risk scenarios where context could be restricted sufficiently that a non-generic standard for testing could plausibly be created. The high-level results of this kind of work could be communicated to consumers using a simple labelling scheme, similar to that used on food packaging to summarise the "healthiness" of the contents.

Take a step back, Janet Gregory urged us. We need to think about risks while producing our products and then find a way to communicate the risks to our users. A labelling scheme of the kind Adam suggested could certainly help, but Janet would like to see something like industry-wide checklists for things to consider on purchase, similar to the paperwork packaged with medicines that list potential side-effects.

She also asked for much tighter controls on the way in which products are advertised and cited self-driving cars as an example. For those kinds of products, adverts might claim "this model is an industry standard for safety" but consumers need to be aware that this doesn't mean getting in, pressing the take-me-to-mother's-house button and then going to sleep.

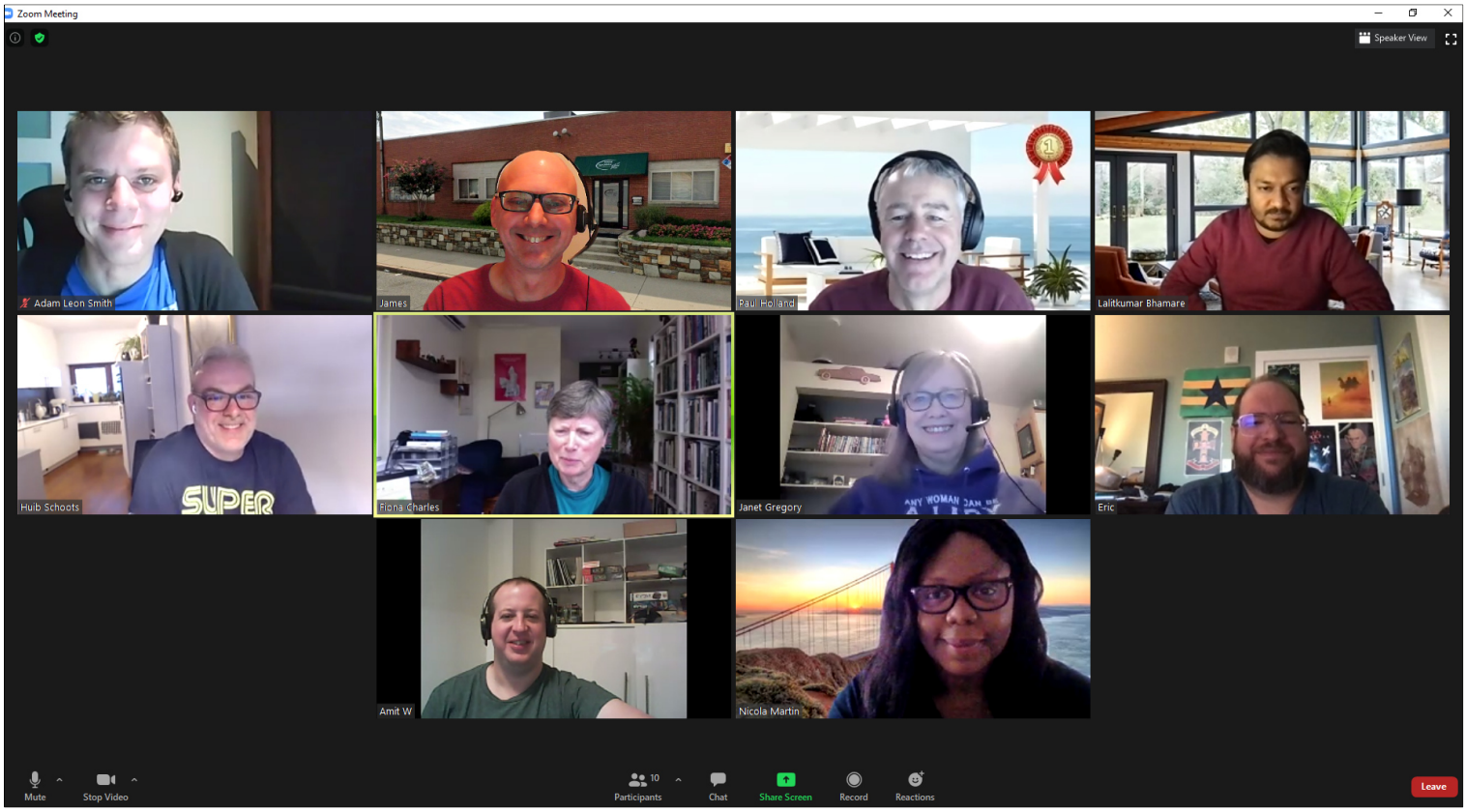
It's not enough to simply make this information available, of course. It needs to be made available in format that consumers can and, crucially, want to engage with. Techniques such as visualisations and analogy could be employed here. Bodies like AST and BCS can definitely be part of a dialogue around that.

Lalit Bhamare rounded things off for us with a call to action for consumers themselves. He talked about clean software — parallel to clean air — and said that unless consumers demand it, producers are unlikely to sacrifice convenience and profit to provide it.

What is clean software? Features that Lalit mentioned included reliability, quality, customer service, and a strong consideration of a product's impact on the world and not just the bottom line. Companies today largely take the public's acceptance of low-quality software for granted.

Testing is a crucial part of Lalit's idea. The public should care about software testing because testing helps them to have confidence that the software is clean, and if there's a public conversation about testing then manufacturers will be forced to pay attention to it.

*The material created at the conference is jointly owned by the participants: Lalitkumar Bhamare, Fiona Charles, Janet Gregory, Paul Holland, Nicola Martin, Eric Proegler, Huib Schoots, Adam Leon Smith, James Thomas, and Amit Wertheimer.*




*From Top Left:*

*Adam Leon Smith, James Thomas, Paul Holland, Lalit Bhamare, Huib Schoots, Fiona Charles, Janet Gregory, Eric Proegler, Amit Wertheimer, Nicola Martin*

**James Thomas** is one of the founders of Linguamatics, the world leader in innovative natural language-based text mining. Over the years he's had many roles in the company, including tech support where he built a team from scratch.

He is currently the test manager, a position in which he strives to provide an environment where his testers have an opportunity to do their best work.

James is on Twitter as @qahiccupps and blogs at Hiccupps.


In the
school of
Testing
for your better learning & sharing experience

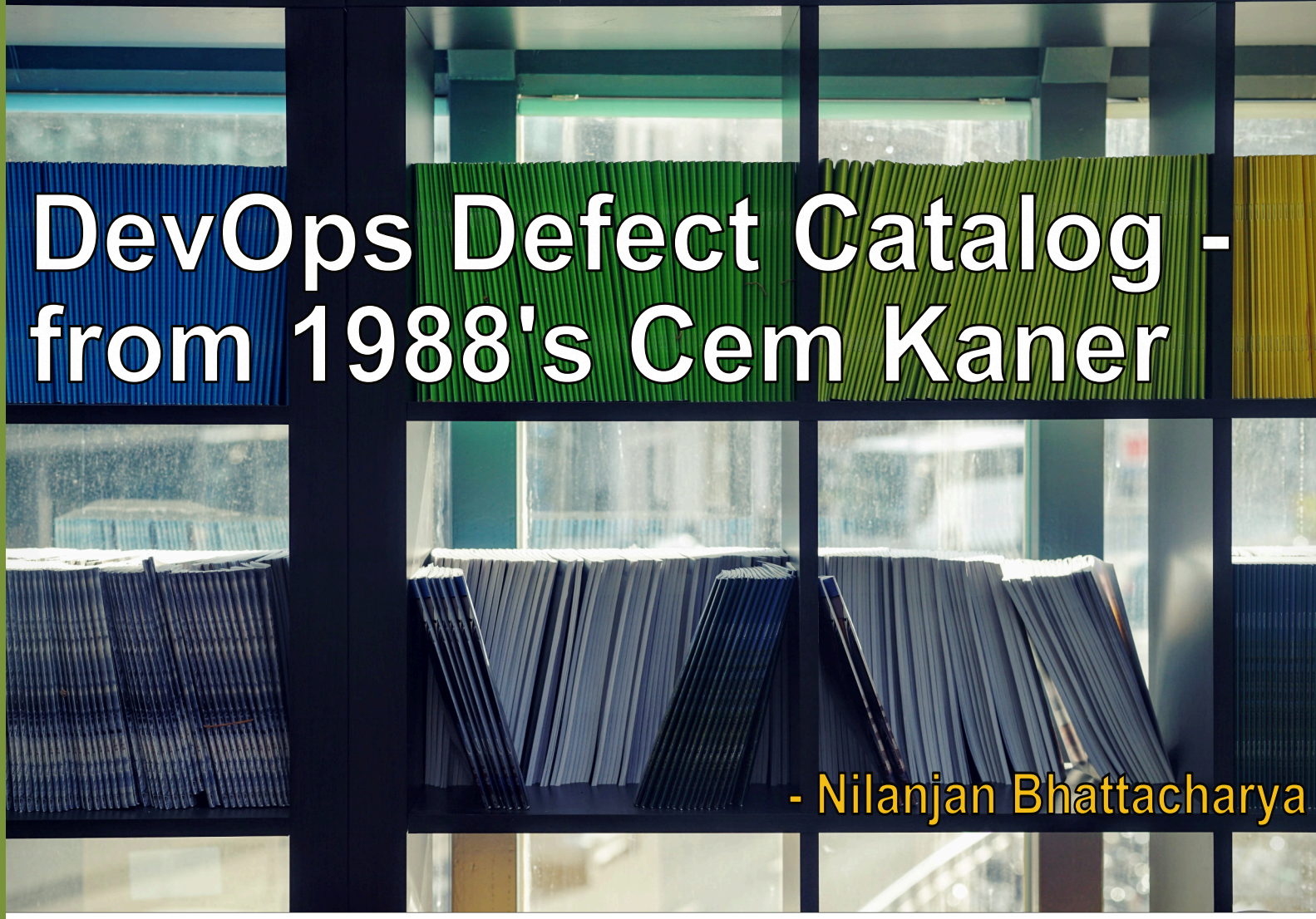



Photo by Maarten van den Heuvel on Unsplash

Good ideas in software testing endure, despite changes in technology and development approaches. In 1988, Cem Kaner first described a 'defect catalog' for software in his landmark book, Testing Computer Software. In this blog post I describe how to create a defect catalog for Infrastructure as Code as part of DevOps.

Kaner's defect catalog was a list of common errors in software. He listed the different ways you can use such a list:

1. Evaluate test materials developed for you by someone else.
2. Developing your own tests.
3. Generate hypotheses for bugs which are difficult to reproduce.
4. Generate related ideas for unexpected bugs.

Here are a few examples of defects in Kaner's defect catalog:

- In the category, 'Command Structure and Entry', the book lists the error, **'Inconsistent abbreviations'.** Without clear-cut abbreviation rules, abbreviations can't be easily remembered. Abbreviating delete to del but list to ls and grep to grep makes no sense. Each choice is fine individually, but the collection is an ill-conceived mess of special cases.

*Abbreviating delete to del but list to ls and grep to grep makes no sense.*
*-- Cem Kaner*

- In the category, 'Missing commands', the book lists the error, **'No undo'.** Undo lets you retract a command, typically any command, or a group of them. Undelete is a restricted case of undo that lets you recover data deleted in error. Undo is desirable. Undelete is essential.

*Undo is desirable. Undelete is essential.*
*-- Cem Kaner*

A defect in a defect catalog is more general than a defect logged for a particular software. A customer or a tester (scrum developer) might complain that when importing data into a spreadsheet, he can't quit midway. This would include **specific steps** to reproduce the **specific** problem and workflow. In a defect catalog, this would be a more **generic** learning – 'Can't stop mid-command'. This generic idea can be applied to many different situations, whether it's a wizard or other workflow.

A defect catalog **won't be used like a checklist**. You can use it to generate ideas on what might go wrong (#2 and #3 from the preceding list). You still need to do the hard work of determining **whether the defect is relevant to your context.**

Defect catalogs are dependent on the **context**. When you are creating a bar chart, you may want to sort the axis categories manually and not alphabetically, e.g., **H**igh, **M**edium, **L**ow (instead of **H**igh, **L**ow, **M**edium). This might translate to a defect in the catalog, 'Sorting depends on the type of data'. When you are looking at usernames in a password manager, sorting the list alphabetically is sufficient. The defect for sorting may not be of value for the password manager.

**Infrastructure as Code in DevOps Defect Catalog**

In an earlier article I had written about how you can anticipate problems in infrastructure as code as part of DevOps. Although, the word, DevOps, is focused on collaboration and organization change, there is significant technology used in DevOps. I have listed a few defects, and their descriptions, below, in the form of a defect catalog. The focus is on **potential problems** when creating Infrastructure as Code.

**Risk of using defaults**

When you work with a service, could you be using any defaults which pose a security risk? Is it possible that you are using some default components without thinking about them?

When you create new EC2 instances in AWS, a default security group is used. It's possible that you might be using the default groups, instead of creating groups with custom and specific permissions. Given the large number of services in a platform like AWS, you may be using defaults which can pose a risk.

This may be a bigger problem with service providers or teams working with multiple vendors, or those working on multiple projects. Each project may have a different setup and different set of services. In that case, you may use defaults without realizing it.

**Packaged components which are not up to date**

Are there plug-ins, libraries or other components which are out of date? If they are not actively maintained, do they pose a security risk?

When using AMIs if the AMI has not been updated for months, it could be a security risk. Can we detect AMIs which are older than a certain number of days/months?

In the case of AMIs they may be vulnerable to operating system patches. Containers may also be vulnerable if they have not been updated.

**Prevent resources from being inadvertently deleted**

How do we make sure important resources are not inadvertently deleted? We can prevent a database from being deleted by setting a flag. We can prevent storage from being deleted by required MFA access. Are there other resources which we want to prevent from being deleted? Are there resources for which it doesn't matter whether they are deleted?

Enabling MFA delete will prevent deletion of versioned S3 objects.

**Unused resources**

Are there any resources which are not used? Unused security credentials may be a security risk. Are there access keys which are not being used? Can we detect unused resources?

## How can you convert risks into a defect catalog?

I created these defects, as part of a defect catalog, from the rules used by a tool for Infrastructure security and compliance, Cloud Conformity (from Trend Micro). If you are unsure about what type of defects you can expect from IaC, Cloud Conformity's rules provide great insight. While security and performance are obvious risks for IaC, Cloud Conformity provides insights into the five pillars of AWS's well-architected framework - **Operational Excellence, Security, Reliability, Performance Efficiency and Cost Optimization.**

I analyzed each rule and converted it into a **generic version**. For example, while we want to prevent AWS S3 buckets from being accidentally deleted, we also want to make sure that other resources are not accidentally deleted.

Cem Kaner's original defect catalog provides many ideas on thinking about defects. Although, the defect catalog was published much before current software development practices and technology, there are many useful ideas on thinking about what can go wrong.

## Approach to anticipating risk

You could **automate** specific instances of the defects described. In fact, Cloud Conformity does automate the detection of these problems. However, in addition to automation, the use of a defect catalog is that you can **keep thinking about new risks.** Adding a defect to a defect catalog also generates new discussions and ideas related to risk:

- Is this catalog defect description sufficiently generic? How do you make it generic?
- Which resources might this apply to?
- How do different cloud platforms handle this?
- You create new models of thinking. In the second example from Kaner's catalog, you realize that 'Undelete' is a special case of 'Undo'. For your cloud based infrastructure, when do you need an 'Undelete' instead of an 'Undo'?

Once you discover new risks, you can create new rules which can then be automated. Agile retrospectives and incident postmortems can generate new defects in a defect catalog. Generating new defects for a catalog is really **not optional**, but is part of agile's feedback loop. The combination of thinking about defects in a defect catalog and automating defects, once you discover them pre-release or in postmortems, is a powerful approach to anticipating problems and addressing risk with Infrastructure as Code.

## Notes

Cem Kaner, Jack Falk, & Hung Quoc Nguyen, Testing Computer Software (2nd Ed.), International Thomson Computer Press, 1993. The first edition was published in 1988.

The defect catalog was published in the Appendix of Cem Kaner's book and is available for download.

Cem Kaner used the word 'taxonomy' instead of 'defect catalog' in later publications: Giri Vijayaraghavan & Cem Kaner, "Bug taxonomies: Use them to generate better tests." [SLIDES] Software Testing, Analysis & Review Conference (Star East), Orlando, FL, May 12-16, 2003

Anticipating problems with Infrastructure as code in DevOps: https://testingindevops.org/anticipating-problems-with-infrastructure-as-code-in-devops/

### Cloud Conformity Rules referenced

Unused resources: https://www.cloudconformity.com/conformity-rules/IAM/credentials-last-used.html

Redundant resources: https://www.cloudconformity.com/conformity-rules/IAM/unnecessary-access-keys.html

Logging access requests: https://www.cloudconformity.com/conformity-rules/S3/s3-bucket-logging-enabled.html

Exposure due to multiple functions: https://www.cloudconformity.com/conformity-rules/S3/buckets-with-website-configurations.html

Logging access requests for resources to audit unauthorized access: https://www.cloudconformity.com/conformity-rules/S3/s3-bucket-logging-enabled.html

Prevent resources from being inadvertently deleted: https://www.cloudconformity.com/conformity-rules/S3/s3-bucket-mfa-delete-enabled.html

Redundant resources: https://www.cloudconformity.com/conformity-rules/IAM/unnecessary-access-keys.html

Implications of automatically launching resources: https://www.cloudconformity.com/conformity-rules/AutoScaling/cooldown-period.html

## Some more rules converted to a catalog

### Detect configuration changes in resources

Can we detect configuration changes in resources? Can we detect changes which are unauthorized?

An example of this is changes to S3 (AWS Simple Storage Service) configuration.

### Components with multiple functions with a less used function creating an unnecessary exposure

When you have a single component with multiple functions, you might expose the component due to misconfiguration of the less used function.

AWS S3 allows you to configure a bucket to host a website. Enabling buckets to host a website might expose data. We should periodically review buckets which are configured to host a website.

The more common use of S3 is for secure storage. It is handy to host simple static websites on S3. It's possible to overlook buckets which host websites and create an exposure.

### Logging access requests for resources to audit unauthorized access

Are we logging all access requests for resources? Is access logging enabled for the service by default? AWS S3 Server access logging should be enabled to log access requests for access. AWS S3 Server access logging is disabled by default.

### Redundant resources

Resources may be created for special purposes. IAM access keys are created for key rotation. Once the keys are rotated, it is a good idea to delete them. Redundant resources have the same risk as unused resources. Can we detect such redundant resources?

### Implications of automatically launching resources

An important part of IaC is the ability to launch/re-launch resources automatically. Instances may be launched to handle increased load, as part of auto-scaling. There are additional considerations when launching resources. Do instances need a cooldown period to allow instances time to start handling traffic?

Nilanjan works in DevOps Delivery in Sydney.

Nilanjan has a background in software testing and project management. He has worked with a variety of software teams, including software product teams. His experience in analyzing customer support issues and working with successful software products with a multi-year lifecycle, provides a unique perspective on software quality and defects.

Nilanjan maintains a blog on Medium (https://medium.com/@revelutions) and can be reached on Twitter (https://twitter.com/nilanjanb) or Linkedin (https://www.linkedin.com/in/nilanjanswmanager/).

# Excellent Software Testing


Photo by Paul Skorupskas on Unsplash

This story is about Alice. Alice is an excellent tester. Excellent testing is hard to describe.

**You only find out what excellent testing is when you see it.**

Here's what I have seen.

## Alice acts like a sociologist.

@IngoPhilipp ⬨ Look for different problems for different people.

**Quality** is subjective; It's value to people who matter; Testing is social science

✸

**Bugs** bug people; Bugs threaten the value of the software

✸

Different **people** think differently; Quality can only be assessed, not measured

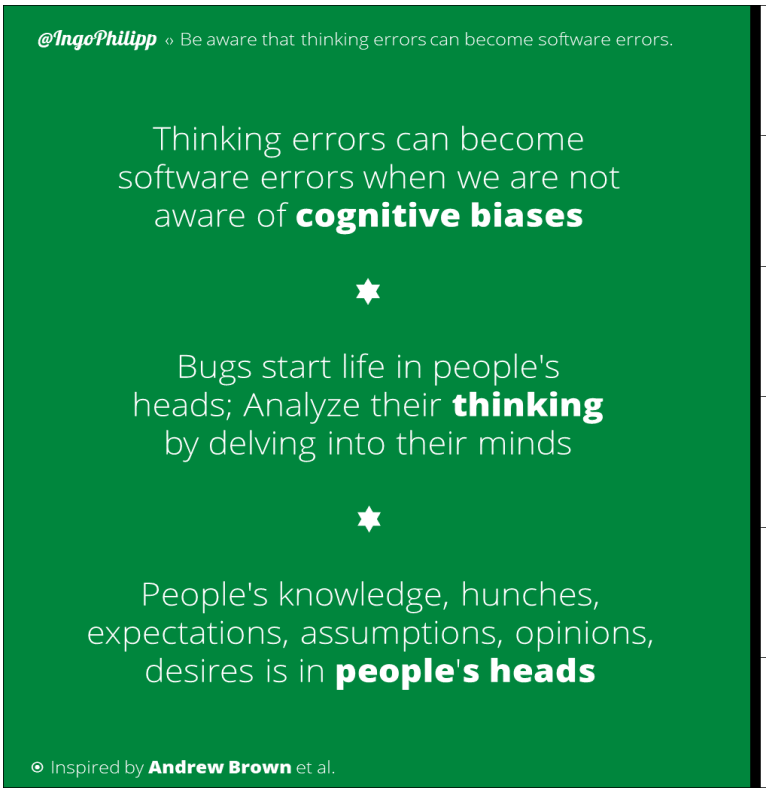
⊙ Inspired by **Cem Kaner** et al.

| | | |
|---|---|---|
| Sociologist | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that testing is a social science. She is aware that the quality of the software is inherently subjective. She understands that quality is value to some people who matter at some time (Gerald Weinberg). She knows that a bug is something that threatens this value. So, she draws the conclusion that a bug is something that bugs somebody (James Bach).

She understands that a bug isn't necessarily in the software but rather represents a relationship between a person and the software (Cem Kaner). For that reason, she accepts that quality cannot be measured but only assessed. So, she acts like a sociologist who looks for different problems (that matter) to different people (who matter), because she knows that different people will perceive one and the same software as having different levels of quality.

**Alice acts like a psychologist.**

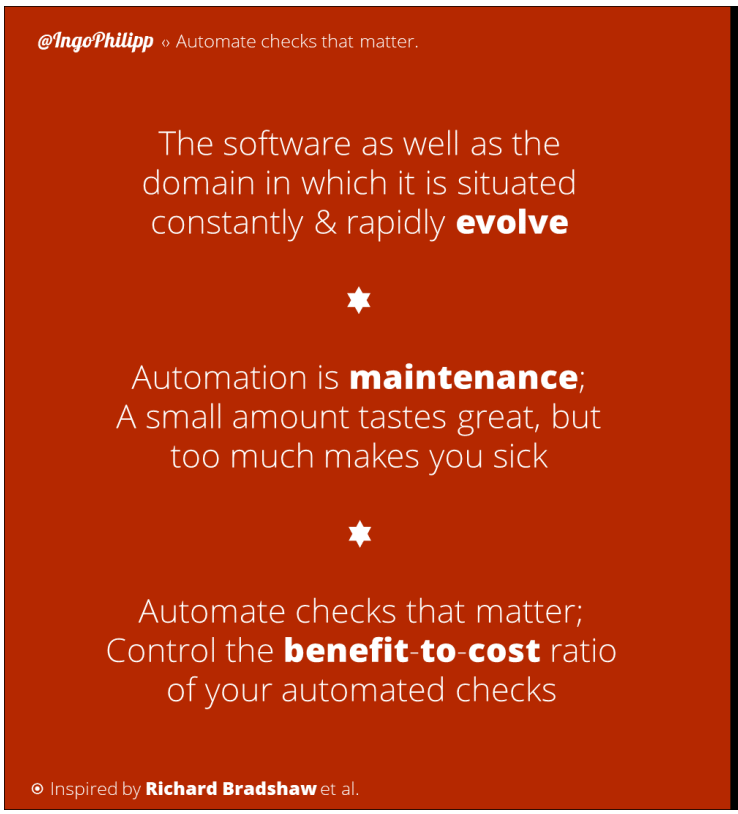



| | | |
|---|---|---|
| Sociologist | Psychologist | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that our thinking errors can become software errors when we aren't aware of our cognitive biases (Andrew Brown).

Therefore, she looks for bugs in the mind of people. So, she acts like a psychologist who delves into the mind of people to analyze their thinking, because she knows that most of their knowledge (including their opinions, assumptions, expectations, desires, hunches, feelings, experiences) about the software is in their heads.

**Alice acts like a controller.**




| Sociologist | Psychologist | Controller |
|---|---|---|
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

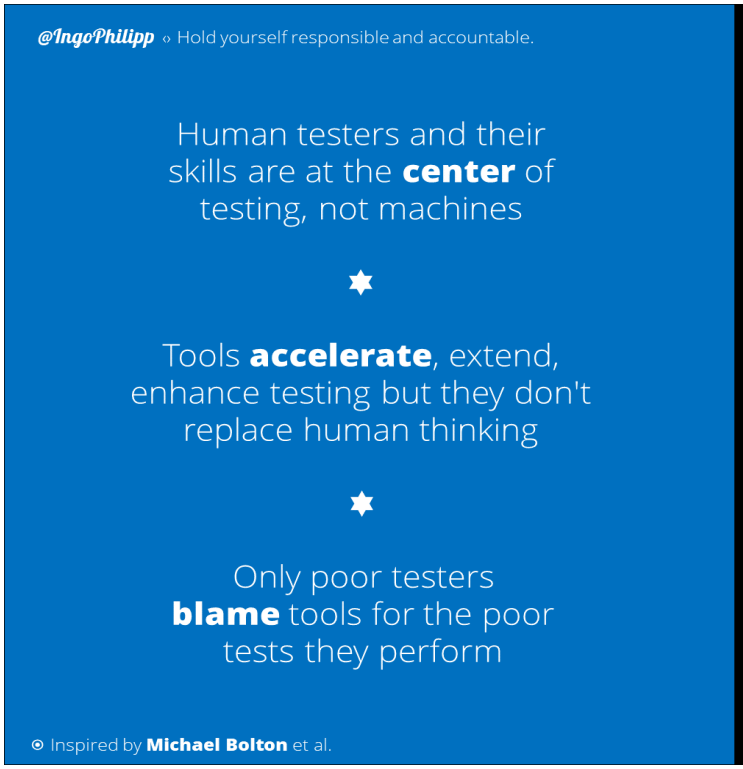
Alice knows that automated repetitive checking is important because she knows that she cannot trust the software. She doesn't trust the software because the software continually changes its "mind".

The software and the (business, technical) domain in which it is situated constantly and rapidly evolve. So, she automates checks to make the process of comparing the current state with some past state of the software faster and less error-prone. In doing so, she thinks of regression checking as an airbag, not as a substitute for good driving, because she understands that repeatability in software testing is not as big a deal as adaptability is (Michael Bolton). Ergo, she doesn't overfocus on confirmation in testing because she is aware that the risk lies in the part that hasn't been explored yet.

In short, she knows that checking lies in the domain of the known knowns, while testing lies in the domains of the known unknowns and unknown unknowns (Del Dewar).

She has also learned that there's no such thing as free lunch when it comes to automation. She knows that automation requires maintenance. So, she treats automation like cake, since she knows that a small amount can taste great but too much can make her sick. So, she doesn't just automate some checks, she automates checks that matter. She automates checks that repeatedly reveal useful information about the software. So, she acts like a controller who carefully controls the benefit-to-cost ratio of automated checks to not get drown in maintenance efforts.

**Alice acts like a leader.**



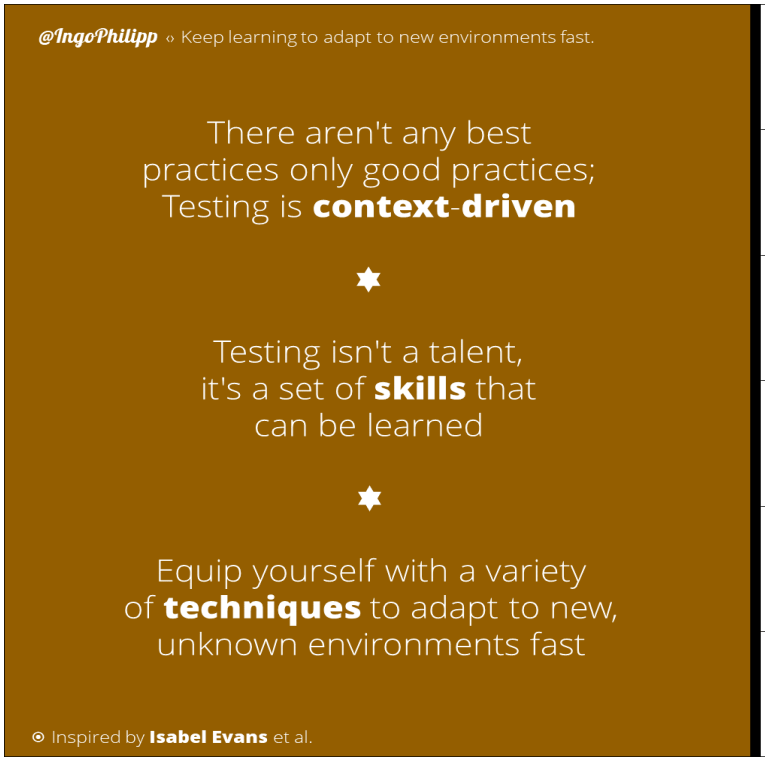

| Sociologist | Psychologist | Controller |
|---|---|---|
| Leader | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that software testing is not a mechanical, one-dimensional activity but a creative, multi-dimensional, and human-centric activity. She sees human testers and their skills at the center of testing, not machines. She is a master in programming but she doesn't feel the need to artificially elevate her role by inventing new names for it (e.g., SDET, automation tester, test automation developer).

She knows that saying tester is enough. She understands that she is actually being paid to test, not to code. She sees tools as something that can accelerate, extend, amplify, and simplify testing (Michael Bolton). She knows that testing is no more about "test tools" than mathematics is about rulers. She knows that she can test without tools. For that reason, she doesn't see tools as something that eliminates the need for human thinking in testing. She knows that it is silly to blame gravity for falling flat on the face.

So, she doesn't blame tools for the poor tests she performs. She knows that only poor testers blame tools for the poor tests they perform because she understands that blaming a "test tool" for her poor testing is like blaming Microsoft Word for a meaningless article. So, she acts like a leader who is leading by example by holding herself responsible for her testing actions and accountable for the results of her testing actions.

## Alice acts like a chameleon.



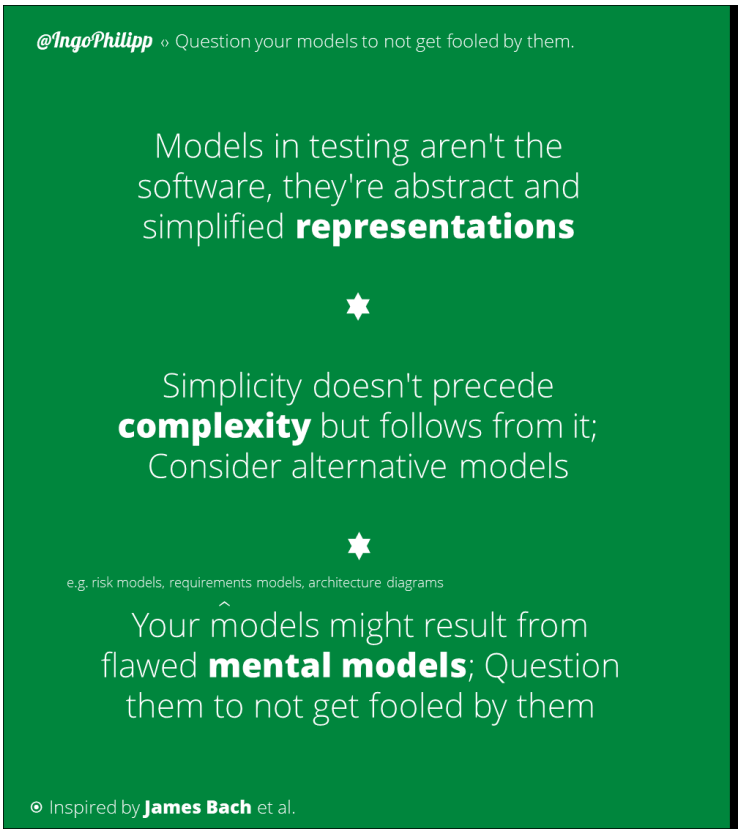

| Sociologist | Psychologist | Controller |
|---|---|---|
| Leader | Chameleon | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that testing is context-driven. She is aware that there aren't any best testing practices, only good practices in context (Cem Kaner). She knows that the value of any practice depends on its context (e.g., project, product, people, environment). She knows that she wouldn't test an autopilot of an airliner in the same way as a website (Iain McCowatt). Of course, sometimes, instead of finding and learning how to use a chisel, she turns to the hammer she is already holding (Matt Parker).

But, in hindsight, she becomes aware of that. She keeps equipping herself with a variety of testing techniques (e.g., heuristics) and has them ready in different forms (e.g., mindmaps, cheat sheets, checklists) to design, perform, prioritize, and analyze tests in different contexts. She doesn't shy away from making this effort because she considers herself a good tester who strives to become an excellent tester (Katrina Clokie).

She knows that testing is neither just a talent nor a process. She considers testing as a set of skills that can be learned (Isabel Evans). She also relates testing to other disciplines (e.g., physics, mathematics, history, philosophy, social science) to broaden her skill set (e.g. problem-solving, modeling, speaking, critical thinking). She understands that as technology advances so must her testing approaches (Angie Jones). So, she acts like a chameleon that has the ability to adapt to new, unknown, or unfamiliar environments fast.

## Alice acts like a critic.



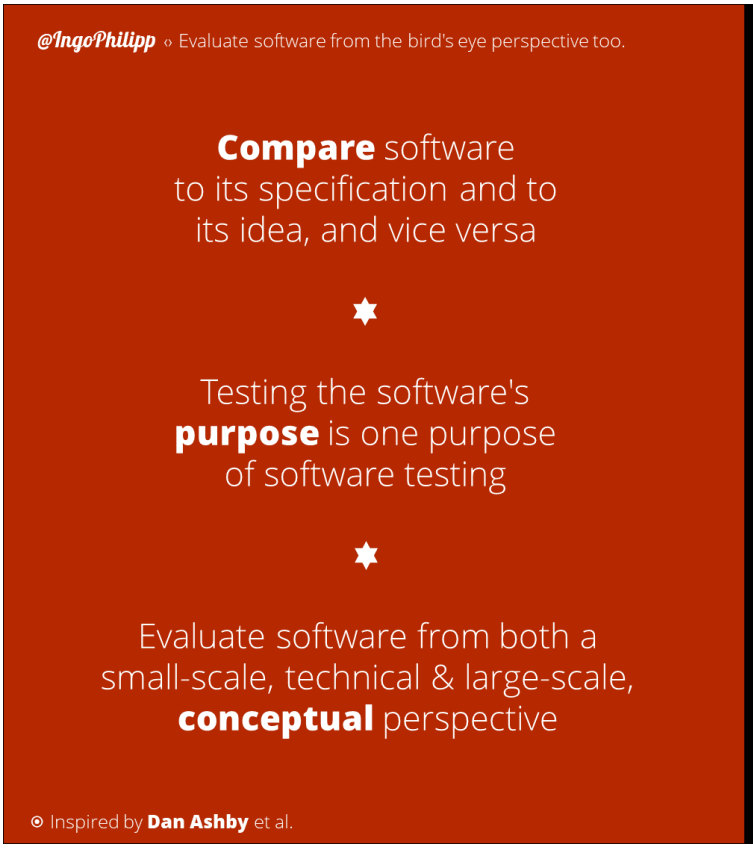

| Sociologist | Psychologist | Controller |
|---|---|---|
| Leader | Chameleon | Critic |
| 7 | 8 | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that the set of models (e.g., risk models, requirements models, architecture diagrams) she is using in testing are just a set of ideas about the software. She knows that these models aren't the software; they are abstract representations of the software.

She is using these models to understand certain aspects of complex software in simple terms. She is using these models to form conjectures to direct her testing, to make decisions, and to communicate her conclusions. She is aware of the fact that these models are simplifications of the software and understands that simplicity doesn't precede complexity but follows it (Alan Perlis).

So, she reflects on her conjectures, decisions, and conclusions by considering alternative models. She knows that these models are probably flawed representations of the software based on her flawed mental model of the software. So, she acts like her own best critic who questions her own models in testing to not get fooled by them.

## Alice acts like a astronomer.




| Sociologist | Psychologist | Controller |
|---|---|---|
| Leader | Chameleon | Critic |
| Astronomer | 8 | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that testing software is more than just checking the specification of the software. So, she doesn't just compare the actual software to its specification, she also compares the idea of the software to the actual software and to the specification of the software.

Alice knows that the purpose of the software is to solve someone's problem. So, she evaluates the purpose of the software too, since she knows that testing the purpose of the software is one purpose of software testing. She does so because she understands that conformance with the specification has little to no value if the software solves the wrong problem.

So, she acts like an astronomer who not only uses a microscope to evaluate the software from a small-scale (technical) perspective but also uses a telescope to judge the value of the software from a large-scale (conceptual) perspective.

## Alice acts like a historian.

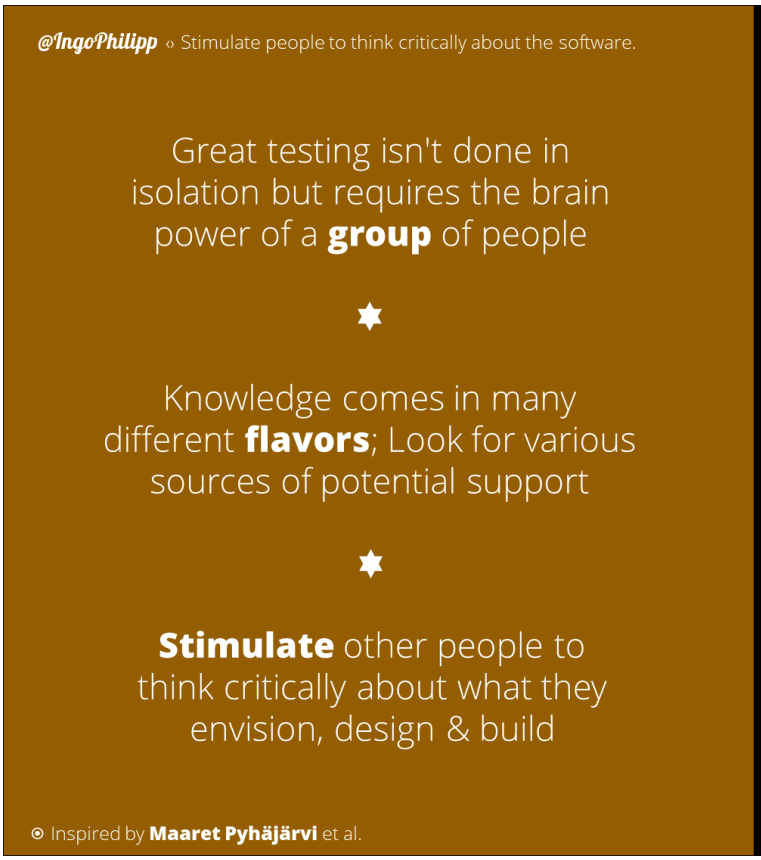

| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | 9 |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that those who cannot learn from history are doomed to repeat it (George Santayana). So, she looks back to the mistakes she has made, identifies patterns of failures, and extracts the lessons she has learned. She doesn't deny her mistakes and so she doesn't hide them.

She documents these mistakes and shares them with other people (e.g., testers, developers) by conducting regular retrospective sessions (e.g., monthly). In doing so, she learns from other people's mistakes and she enables other people to benefit from her lessons learned.

So, she reflects on mistakes before she takes on new challenges (e.g., project, release, sprint) to prepare herself to not repeat these mistakes again and again. She knows that prevention is better than cure. So, she acts like a historian who builds a learning culture by analyzing her past testing activities to continuously improve her future testing activities.

## Alice acts like a guide.



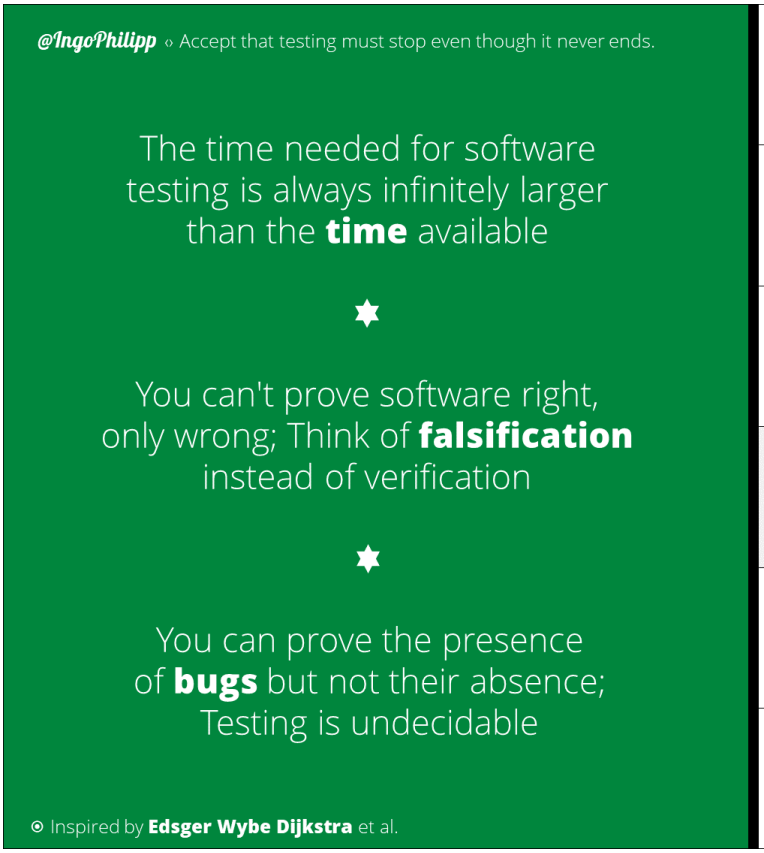

| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| 10 | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice understands that having a testing specialist on the team doesn't mean to restrict the responsibility of testing to that single person (Trish Khoo).

She is aware that excellent testing is never done by one single person in isolation but requires the collective brainpower of a group of people. She understands that great (testing) teams are teams of diverse talents cooperating. She knows that diversity makes testing powerful because she knows that the idea of one person having all the answers is deeply flawed. She understands that knowledge usually comes in many different flavors, and so she looks for sources of potential support through various collaborative activities (e.g., pair testing, mob testing).

So, she acts like a guide who actively involves, guides, and advises other people (e.g., developers, UI/UX experts, product owners, documentation experts) in software testing by stimulating them to continuously think critically about the software they envision, design, build, and sell.

**Alice acts like a realist.**




| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | 11 | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that the time needed for testing is always infinitely larger than the time available (Cem Kaner). So, she prioritizes her testing activities, because she knows that testing has to stop although it actually never ends (Michael Hunter). She is aware that she cannot prove the software right, only wrong. She knows that she has to think of falsification instead of verification. She accepts the fact that she cannot answer the question "Did we find all bugs?".

She understands that she cannot quantify the number of absent bugs. She knows that testing can only prove the presence of bugs, not their absence (Edsger Wybe Dijkstra). She is aware that only infinite testing can find every bug, but (most) testing budgets are not infinite (Dorothy Graham).

So, she acts like a realist who knows that testing is undecidable because she understands that the problem of finding all bugs in the software is undecidable.

## Alice acts like a experimentalist.


| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | Experimentalist | 12 |
| 13 | 14 | 15 |
| 16 | 17 | 18 |

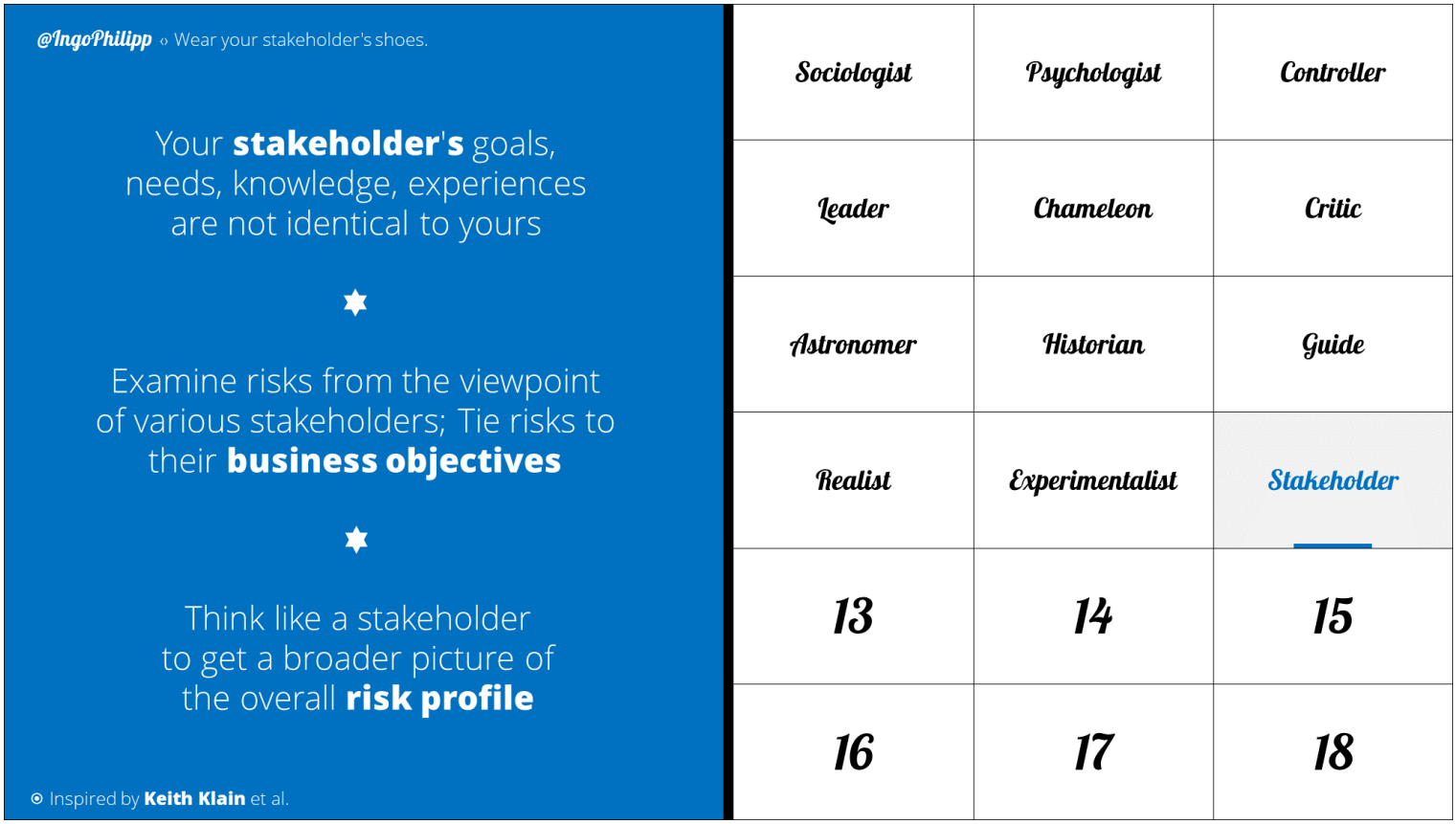
Alice knows that testing is learning. Whenever she runs out of test ideas, she suspects that she probably hasn't learned enough about the software.

She keeps exploring, investigating, and experimenting to constantly evaluate this conjecture. She never stops learning about the software and the (technical, business) domain in which the software is situated. She knows that testing is experimental science.

She sees testing as an adaptive investigation rather than a factory process of creating, automating, and executing test cases. She knows that testing is more than just checking things out (Karen Johnson).

She understands that her test success is not defined by the number of her test cases but by the quality of her test ideas. So, she acts like an experimentalist who doesn't confuse testing with test cases, since she considers tests as experiments to challenge hypotheses, assumptions, and opinions about the software.

**Alice acts like a stakeholder.**

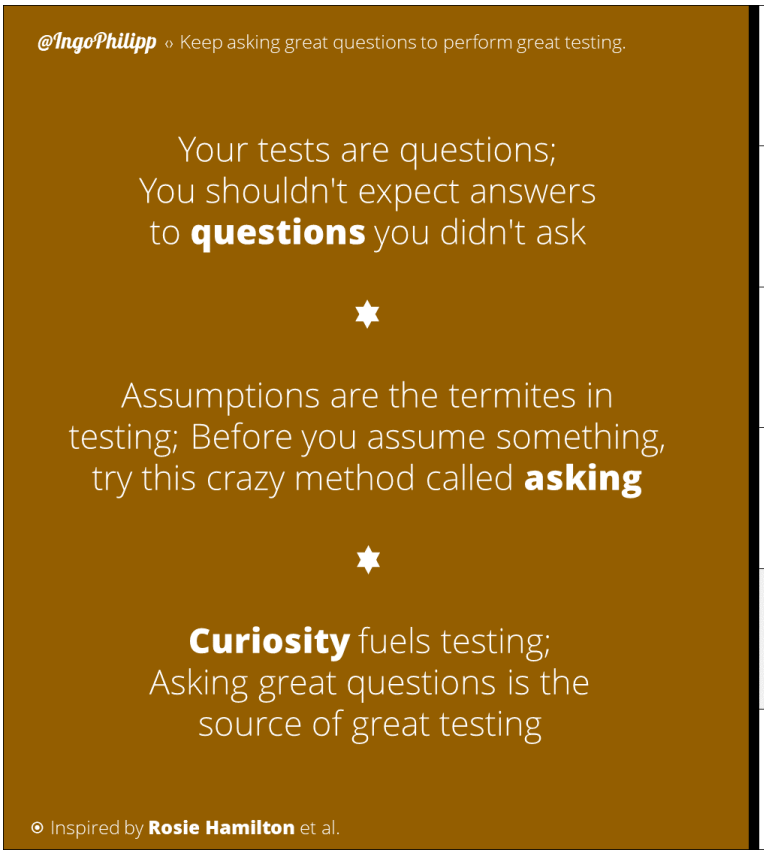



Alice knows that testing means comparing the invisible to the ambiguous (James Bach). She knows that the stakeholder's knowledge, goals, and experiences are not identical to hers. She knows that the universe of stakeholders is vast, but she doesn't get tired to examine risks for both internal and external stakeholders. She considers how internal stakeholders (e.g., sales, marketers, product owners, developers) would look at the risks she is uncovering.

She ties risks to their objectives, needs, and goals. She examines risk from the perspective of external stakeholders (e.g., customers, partners). In doing so, she avoids saying and even thinking things like "No one would ever do that!". She knows the more advanced she gets in the software, the harder it becomes to relate to true beginners (e.g., end users). So, she is trying to wear the end user's shoes to not forget what it's like to be one. She is aware that it isn't just the software but the product (e.g., software, people, environment) the end users depend on (Lisa Crispin, Janet Gregory).

So, she steps out of her role to take on the perspective of different stakeholders by looking at the risks from their vantage points, since she knows that these different viewpoints allow her to look at the value of the software, and the impact of a threat against the software, in a variety of ways. So, she thinks like a stakeholder to enrich the way she assesses and prioritizes risk to get a broader picture of the overall risk profile.

## Alice acts like a kid.



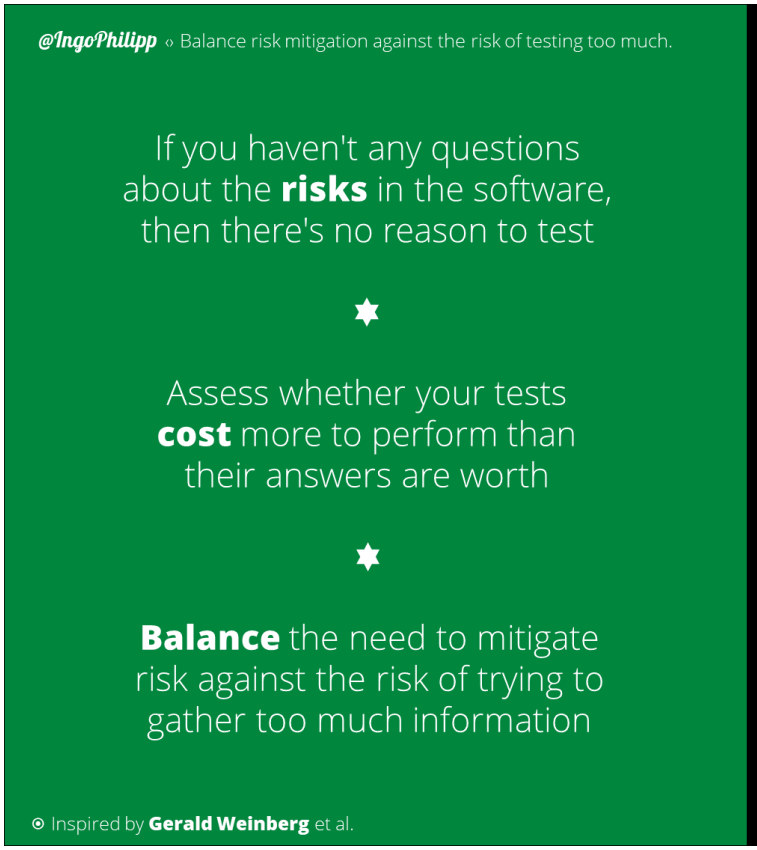

| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | Experimentalist | Stakeholder |
| Kid | 14 | 15 |
| 16 | 17 | 18 |

Alice knows that she can't expect answers to questions she didn't ask. So, she isn't afraid of looking "stupid" by asking simple questions. She understands that tests are questions she asks the software to reveal useful information about the software (Cem Kaner).

She knows that assumptions ("unexamined beliefs") are the termites in software testing. She suspects that she probably has answers that might be wrong and beliefs that might not be true, but she isn't afraid of not knowing things (Richard Feynman). So, she doesn't get tired of applying this crazy method called "asking" before she assumes something. For that reason, she judges herself not only by the answers she gives but also by the questions she asks (Simon Sinek).

She knows that curiosity fuels testing. So, she acts like a curious little kid who never stops asking, because she knows that asking great questions is the source of great testing.

**Alice acts like a accountant.**

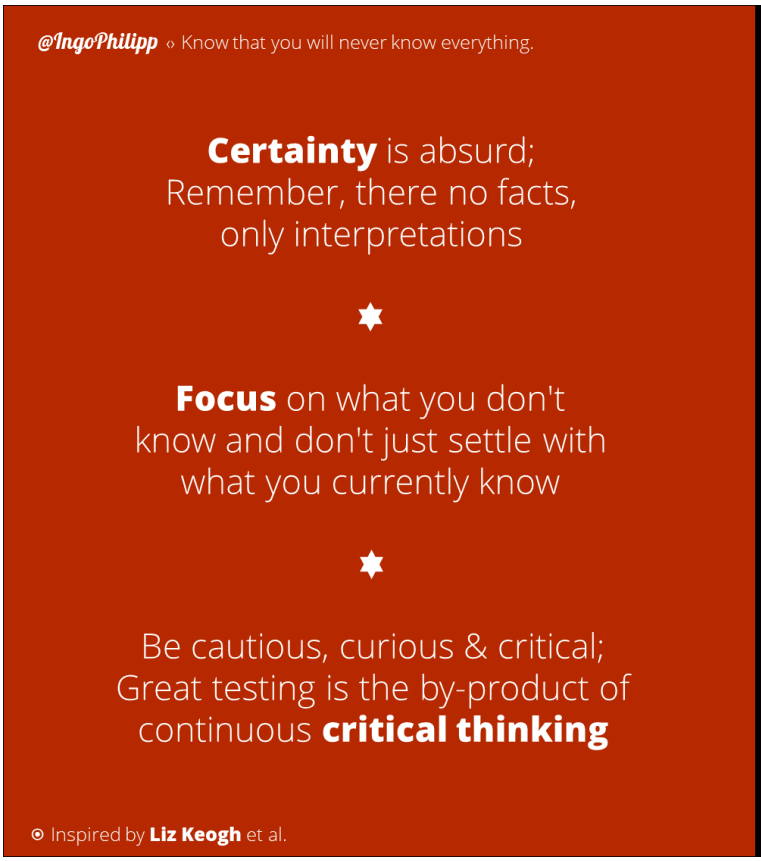



| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | Experimentalist | Stakeholder |
| Kid | Accountant | 15 |
| 16 | 17 | 18 |

Alice knows that if she doesn't have any questions about the risks in the software, then there's no reason to test at all. If she has at least one such question, then she will ask: Will these tests cost more to perform than their answers will be worth?

So, she acts like an accountant, since she understands that good testing involves balancing the need to mitigate risk against the risk of trying to gather too much information (Gerald Weinberg).

## Alice acts like a skeptic.




| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | Experimentalist | Stakeholder |
| Kid | Accountant | Skeptic |
| 16 | 17 | 18 |

Alice knows that she shouldn't believe everything she hears and only half of what she sees (Edgar Allan Poe). She knows that there no facts, only interpretations (Friedrich Nietzsche). She knows that certainty is absurd. So, she thinks critically about what she has been told about the software.

She is skeptical about what she knows about the software, and she knows that she will never know everything. She is aware of the fact that she isn't Laplace's Demon. She focuses on what she doesn't know about the software and doesn't just settle with what she currently knows.

She keeps thinking despite already knowing. Hence, she sees testing as a thinking activity and she treats it like one. She knows that she has to change her thinking to change her testing. She knows that software is a cunning deceiver that is always trying to fool her in one way or another. For that reason, she remains cautious, curious, and critical to not get fooled over and over again.

So, she acts like a cosmic skeptic, because she knows that excellent testing is just the by-product of continuous critical thinking.

## Alice acts like a friend.

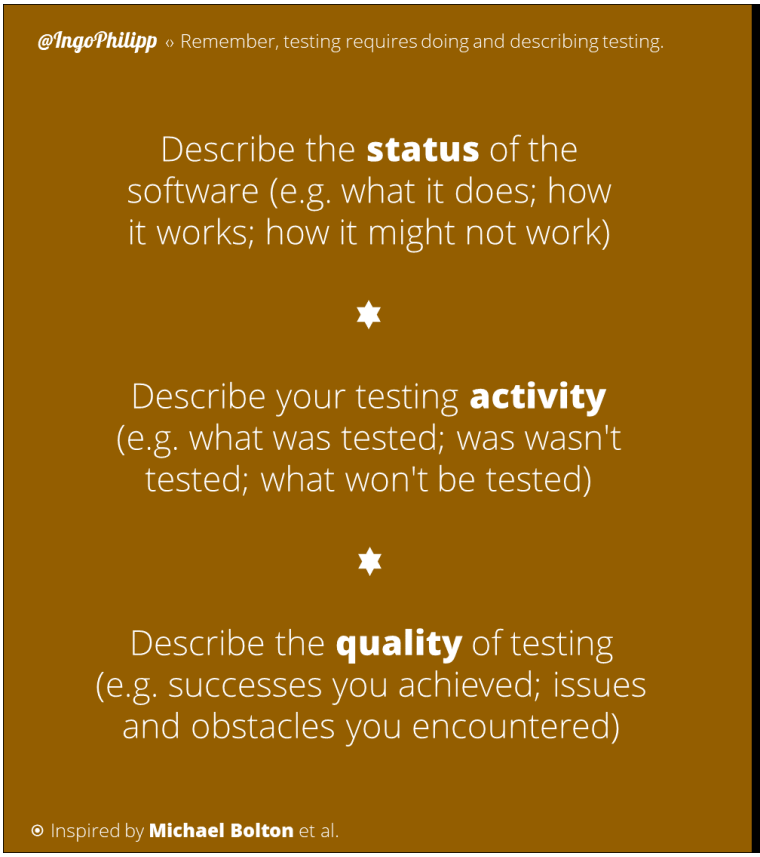

| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | Experimentalist | Stakeholder |
| Kid | Accountant | Skeptic |
| Friend | 17 | 18 |

Alice knows that the primary goal of testing is to make other people (e.g., product owners, developers) aware of problems in the software. In doing so, she doesn't break the software, she breaks people's illusions about the software (Maaret Pyhäjärvi). She knows that making people aware of these problems is a socially difficult task, because of people's natural desire to avoid trouble (Michael Bolton).

She understands that problems are something negative at a first glance. She considers a problem as a deviation from what is perceived and what is desired. The bigger this gap the bigger the problem. The bigger the problem the more it might threaten the software's value. She is rational enough to know that different people react differently, and sometimes irrationally, to potential threats. She knows that there are people who welcome critical feedback, and there are people who don't. She knows that there are people who value the perspective of others, and there are people who don't.

She knows that there are people who hold themselves responsible and accountable for their mistakes, and there are people who don't. So, she knows that the nature of humans is diverse. So, she studies people's personalities and their relationships among each other to make them aware of problems in a dispassionate, deliberate, and personal way. She understands that if she isn't making people aware of problems, these problems will eventually find them. So, she communicates these problems right away with modesty and empathy. She is direct, critical but not judgmental. She doesn't confuse being nice with being kind. She knows that just being nice doesn't help, serious feedback does (Alexandra Schladebeck). So, she talks about negative things in a positive way because she sees problems as an opportunity to improve. She knows that the software doesn’t have to behave the way she wants it to behave. She is aware that she doesn't design, code, or sell the software. She tries to help other people to understand the software they have got, so that they can decide whether it's the software they want (Michael Bolton). So, she acts like a friend who reduces people's ignorance about the software without embracing arrogance.

## Alice acts like a storyteller.



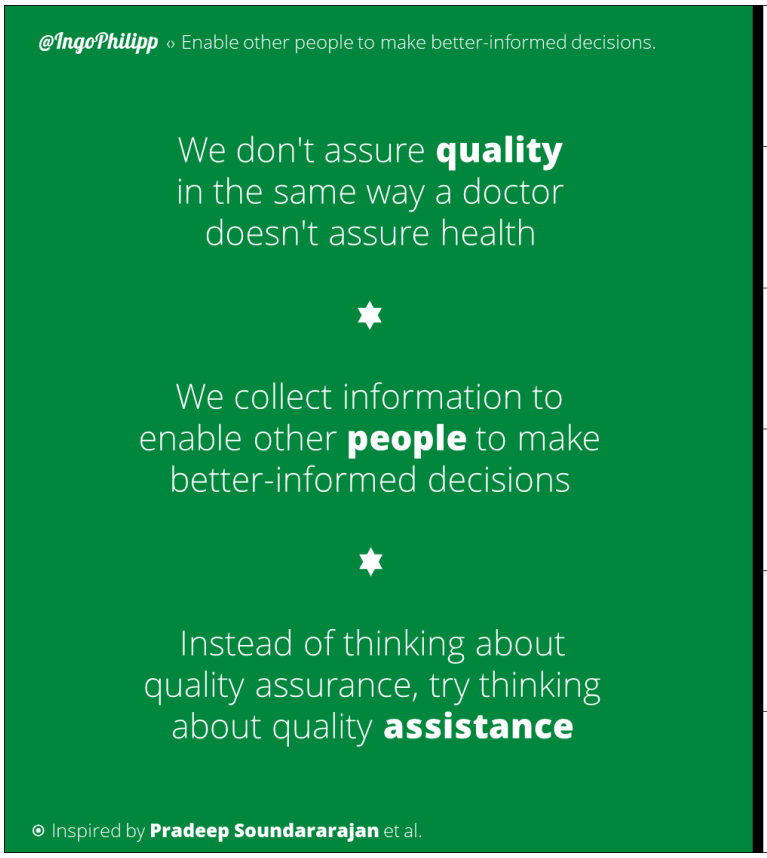

| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | Experimentalist | Stakeholder |
| Kid | Accountant | Skeptic |
| Friend | Storyteller | 18 |

Alice understands that software testing is the headlights, quality is the journey, and business outcomes are the destination (Anne-Marie Charrett). So, she lights the way for other people by describing the status of the software in terms of what it does, how it works, and how it might not work. She describes her testing activity in terms of what has been tested, what has not been tested yet, and what probably won't be tested due to several constraints (e.g., time, resource, budget).

She describes the quality of testing in terms of the successes she achieved and obstacles that impeded her testing. So, she not only looks for problems that threaten the value of the software, she also looks for problems that threaten testing. She understands that problems in the process of looking for problems in the software (aka testing) can prevent her from finding problems in the software.

In summary, she provides a 360° narrative on testing packed with useful and actionable information instead of bombarding her stakeholders with vanity metrics (e.g., test case counts) packed into shiny but useless dashboards. So, she acts like a storyteller who understands that testing requires doing and describing testing.

**Alice acts like a assistant.**




| | | |
|---|---|---|
| Sociologist | Psychologist | Controller |
| Leader | Chameleon | Critic |
| Astronomer | Historian | Guide |
| Realist | Experimentalist | Stakeholder |
| Kid | Accountant | Skeptic |
| Friend | Storyteller | Assistant |

Alice knows that she cannot assure quality in the same way a doctor cannot assure health (Michael Bolton). She understands that her primary objective is to collect quality-related information (e.g., risks) about the software to enable other people (e.g., product owners, developers) to make better-informed decisions (e.g., shipping decisions, fixing decisions) based on the information she provides. In short, she tests, others decide. She understands that testing is an information-based science (Keith Klain). So, she considers herself as an assistant that neither does quality assurance nor quality control but rather quality assistance.

This list could go on and on. It's not complete. It never will be. You get the point.

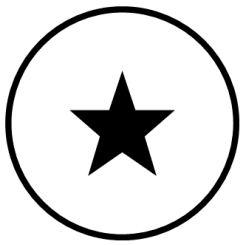
Alice is a cocktail of testing excellence.

Her testing is fast, inexpensive, credible, and accountable.

Here's a secret. Alice is real but only in a metaphorical sense. Alice is a fictional character that symbolizes the collective testing excellence of a large, inspiring, and diverse (e.g., age, gender, race, ethnicity) group of software testers I have worked with during the last years.


This story stands on the shoulders of the giants listed below. Don't follow me, follow them. They know the way, they go the way, and they show the way of excellent software testing.

@IngoPhilipp ‹› This story stands on the shoulders of these giants ▸



*People who know the way, go the way and show the way of excellent software testing*

| | |
|---|---|
| Michael Bolton | Dorothy Graham |
| Liz Keogh | Pradeep Soundararajan |
| Dan Ashby | Rob Lambert |
| Jeff Nyman | Marit van Dijk |
| James Bach | Cem Kaner |
| Angie Jones | Isabel Evans |
| Alan Richardson | Lena Pejgan Wiberg |
| Gerald Weinberg | Paul Grizzaffi |
| Maria Kedemo | Katrina Clokie |
| Alexandra Schladebeck | Trish Khoo |
| Lisa Crispin | Elisabeth Hendrickson |
| Rosie Hamilton | Richard Bradshaw |
| Anne-Marie Charrett | Karen Johnson |
| Maaret Pyhäjärvi | Fiona Charles |
| Andrew Brown | Keith Klain |
| Zeger Van Hese | Janet Gregory |
| Rich Rogers | Aaron Hodder |

⊙ Special thanks to all those others I need to thank but didn't.

**Ingo Philipp** is a product management expert in the information technology & services industry with strong product marketing and evangelism skills.

He holds a master's degree in theoretical astrophysics, has deep technical skills, and currently completes an MBA program at the Vienna University of Economics & Business.

## TestFlix 2020 Atomic talks

Sharing is caring! Don't be selfish ;)

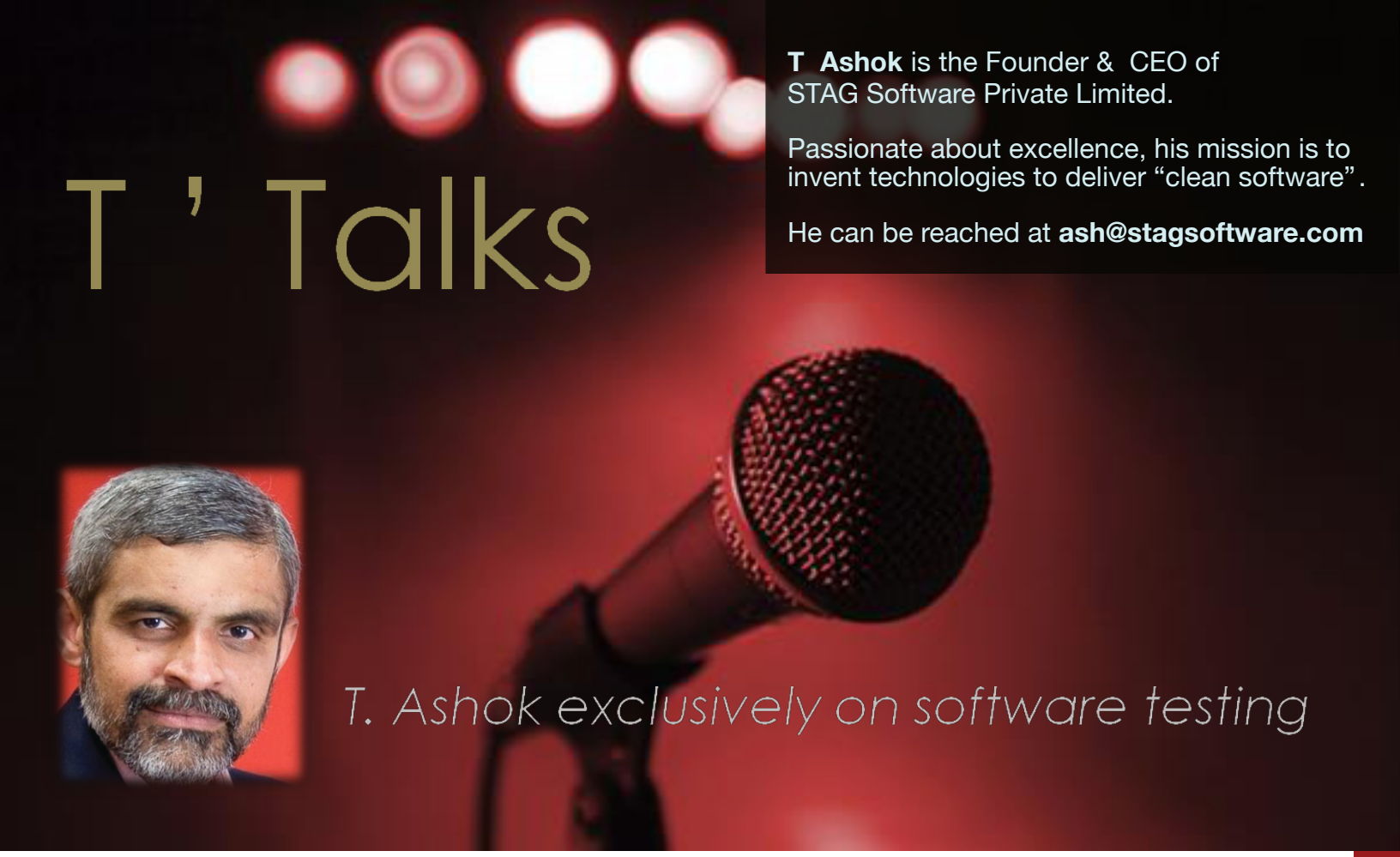
Share this issue with your friends and colleagues!

# Happiness is....

Making home-office better by reading about testing!!!

Like our FACEBOOK page for more of such happiness

https://www.facebook.com/TtimewidTesters




**T Ashok** is the Founder & CEO of STAG Software Private Limited.

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

He can be reached at **ash@stagsoftware.com**

# 10 things to be sensitive to deliver brilliant code

**Summary**

Great code is not result of mere unit/dev testing at the early stage. It is really a mindset that is key to producing brilliant code. This article outlines ten things that a developer should be very sensitive to enable the delivery of brilliant code.

**1.Be focussed**

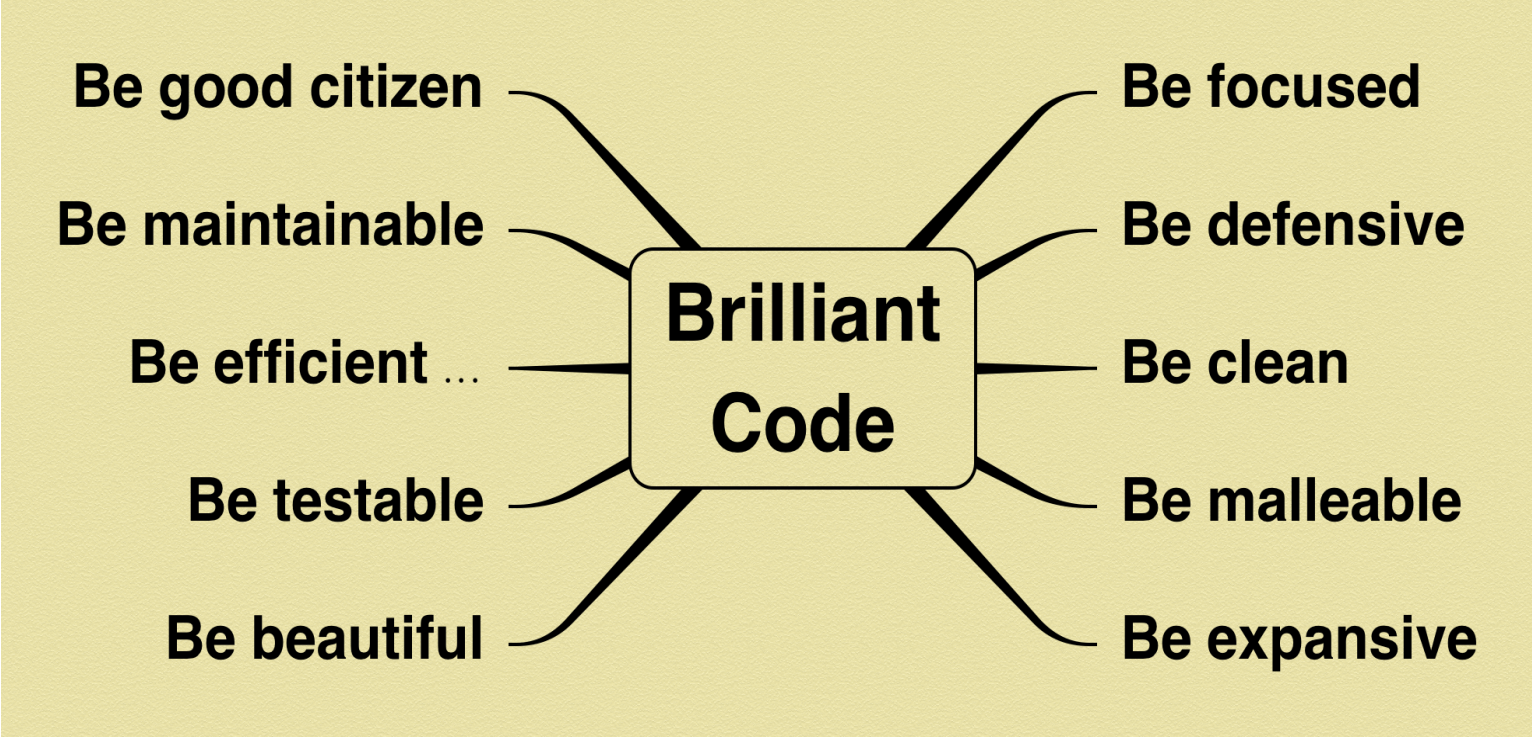
In each code fragment, do one thing well. Attempting to do many things can become messy! Stay single-minded in what you are solving with this code.

**2.Be defensive**

Accidents happen, inputs may be tainted. Prepare for eventualities, be defensive in your style of coding. After all, it is your responsibility to stay safe!

**3.Be clean**

Reduce clutter in code, it is just about somehow getting the code to do work. Organisation, structure matters.




**4.Be malleable**

Avoid magic numbers, hardcoding. Be soft and pliable so that you can modify, extend easily.

**5.Be expansive**

Strive to understand the larger context where your code will be used, so that your code delivers value. 'See the forest for the trees' too.

**6.Be a good citizen**

Respect the environment in which the code runs, consume resources only what you need and release them when you need don't them.

**7.Be maintainable**

Code begs changes to be done the minute you execute it. Make it maintenance friendly. Also, remember that it may just be you who will maintain the code. Well with time, one also forgets why some things are, the way it is.

**8.Be efficient**

It is not just about the functionality, it is about all other '-ities', like security, reliability, compatibility, performance etc. Be sensitive to these aspects while you are coding.

**9.Be testable**

The ability to ascertain if the code is behaving correctly is paramount. Putting it hooks to enable testability enables you to write great code.

**10.Be beautiful**

Finally great code is not just text that when executed, works correctly. Treat it as a work of art that you produce. Let the choice of names, the structure, the organisation have a sense of aesthetic appeal. After all brilliant engineering becomes art.

QA Intelligence
Testing & QA Management blog by PractiTest

# Perspective – the secret weapon we seldom use while testing

I am writing this post on that magical interval between Christmas and New Years Day when the World is a different and quieter place.

Yes, we have the Covid virus around us, and so we are not traveling or sharing with loved ones as much as we would like to. But hopefully you are enjoying some time off with a limited numbers of your closest friends and family.

As it is usually the case, and thankfully so, during this week or two of the year we will not be submerged in work. I even hope you are reading these words sometime during first weeks of January, as even blog posts can be relegated to "later", when you are not spending time solely focused on your family and yourself.

One of the unintended "presents" of this season, in addition to the pounds of cake around our waists, is due to arrive January 4th when we will get up on Monday morning, dress up and sit up straight for work. (In a normal year I would have written go to work, but as we are in Covid days many of us will be zooming to work from home!)

This "present" I am talking about is the feeling that we are now starting something new. A new year, a new project, even an old project with a new view on things. In reality not many things have changed other than one very important thing: You have a Fresh Perspective.

We all know what perspective is, but if you look it up in the dictionary (as I just obviously did) you will find there are a number of slightly different of definitions. Still, these two are the ones that I believe drive my point best:

- The interrelation in which a subject or its parts are mentally viewed
- The capacity to view things in their true relations or relative importance

And the three most important parts are (in case you have not realised this up to now): "mentally viewed", "true relations" and "relative importance".

## Some bad news and some good news

The bad news first: We are only human.

This means we have flaws, we get tired, become frustrated, and even start thinking things are better or worse than they really are.

The good news now: Just like MS-Windows we can reboot.

And when we do this we get to clean our registry, close unneeded processes and free up memory space.

BTW, interestingly enough we also have a couple of reboot options:

- Soft reboots – when we go home at the end of the day, or on the weekend.

- Shutdowns and reboots – when we take a week or two and go on vacations, or stay home for the holidays.

And while the former helps clean up some of the clutter in the system, we use the second when we also need to cool the system down and really take a fresh look on things.

## The three points of perspective: mind, relation and importance

Back to our three part definition of perspective that I wanted to review today:

1. **Mentally viewed:** You appreciate everything through the lens of your eyes and based on what is reflected on the screen of your mind. Yes, this is a reflection of reality but only as far as you are interpreting it, and it is influenced by all the other things you are doing at that same time. If you take a new look with fresh eyes and a clear mind you will be able to see more things and closer to the cold reality where they are taking place.

2. **True relations:** Our minds are great at coming up with links between things that we can't really see with our eyes. We call this intuition, hunches, gut feelings, etc. The problem is that our minds will create both links and relations that are real together with some that do not really exist. And so we need to find a way to evaluate these links and understand which of them are true and which of them are the noise generated by the process, and should be disregarded or discarded

3. Relative importance: Following up on the previous two points, when we have many things in front of us we need to find a way to sort them out so that we can decide where to start and what to do next. Many times the most difficult task is to decide what to focus on and what we should drop, just as much as we need to learn not to judge problems based on how loud the person in front of us shouting. Taking care of urgent things is import, but sometimes it is more urgent to take care of the important things ;)

## Perspective and testing

No one I know thinks they have ugly kids, but I know many ugly kids nonetheless… When we are personally and deeply invested in something it is close to impossible to have a clear perspective – we lack "the capacity to view things in their true relations or relative importance".

This is one of the reasons why I think developers should not test their own code – not the main reason, but yes one of them. Developers can surely test (when they are not being prima-donnas or lazy about it!) but no one should be the only person testing his or her own code, because this person will lack perspective.

I also think that a tester who has been testing the same feature for a long time should switch tasks with one of her peers in order to gain some perspective on the testing efforts applied to this feature. And when we do not have the luxury of having other testers on the team, then at least take some time off from the feature, concentrating on other tasks in order to distance oneself from it and approach the testing from a different angle.

This advice is also true regarding all other tasks in your work. Getting feedback to improve your work is a no-brainer, and when you do not have people to give you feedback then at least distance yourself from the task and come back to it once you've gained a little perspective…

## Wishing you more perspective in this new year

May you see your challenges in their real colors and proportions.

May you be able to differentiate between urgent and important.

May all your testing be true and find the relevant issues.

And most importantly, may you achieve the correct balance between your work and the rest of your life in this 2021.

**Joel Montvelisky i**s 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 co-founder 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.

## State of Testing Survey – 2021

# The State of Testing™ Survey – is NOW LIVE!


It's that time of year again. PractiTest and TeaTimeWithTesters are happy to present the 8th edition State of Testing™.

**Seize this window of opportunity to influence the global testing community and the trends we'll all be talking about in the coming year.**

### Help us identify –

- Most common successful testing practices and methodologies
- Professional skills to acquire to stay relevant
- The best sources of knowledge and professional information
- The challenges and trends to be aware of

  and so much more...

TAKE THE SURVEY NOW

Brought to you by:

Media Sponsors:

### What is the State of Testing™?

The State of Testing™ initiative seeks to identify the existing characteristics, practices, and challenges facing the testing community today in hopes to shed light and provoke a fruitful discussion towards improvement.

The final report is translated into several languages and shared globally, further expanding the reach and impact this report has on all of us in the QA world.

Each year the amount of participants has increased, and the final report becomes even more valuable as a culminated reflection of testing trends, challenges, and characteristics.

Blog Collaborators:

# Advertise with us

## Connect with the audience that MATTER!

Adverts help mostly when they are noticed by **decision makers** in the industry.

Along with thousands of awesome testers, Tea-time with Testers is read and contributed by Senior Test Managers, Delivery Heads, Program Managers, Global Heads, CEOs, CTOs, Solution Architects and Test Consultants.

Want to know what people holding above positions have to say about us?

Well, hear directly from them.

And the **Good News** is…

Now we have some more awesome offerings at pretty affordable prices.

Contact us at sales@teatimewithtesters.com to know more.


Every Tester
who reads Tea-time with Testers,
Recommends it to friends and colleagues .
What About You ?
Image : vernhart

# our family

## Founder & Editor:

Lalitkumar Bhamare ( Germany)

Pratikkumar Patel (The UK)


Lalitkumar


Pratikkumar

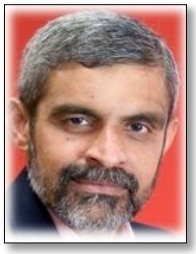
## Inspiration and Contribution:

Late. Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)


Jerry


T Ashok


Joel

## Editorial|Magazine Design |Logo Design |Web Design:

Lalitkumar Bhamare

Cover page image – Mukul Wadhwa on Unsplash

## Editorial Board:

Dr.Meeta Prakash (India)

Dirk Meißner (Germany)

Klára Jánová (Czech Republic)


Dr. Meeta Prakash


Dirk Meißner


Klára Jánová

## Operations and Sales:

Shweta Daiv (Germany)

Astrid Winkler (Switzerland)


Shweta Daiv


Astrid Winkler

## Community Partner :

Kiran Kumar (India)


Kiran Kumar

*|| Karmanye vadhikaraste ma phaleshu kadachna | Karmaphalehtur bhurma te sangostvakarmani ||*

To get a **FREE** copy,

Subscribe to mailing list.

SUBSCRIBE


Join our community on

facebook

Follow us on - @TtimewidTesters

www.teatimewithtesters.com