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

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

Chaos, Creativity, and the Cool Kids

What an interesting year this has been so far, isn't it?

When we talked last time, back in April, I was skeptical of the things that would unfold over time. For some time I felt like the world is on the verge of the collapse and nothing will ever be the same again.

But here we are, trying to make life happen with whatever we have got. And I must say, I don't feel that disappointment anymore. Particularly in the case of the software testing community, I would say I feel more connected with it than I ever felt before.

This year I have interacted with so many testers, discussing so many topics via so many forums. And I can't stop mentioning that we are indeed the cool kids in the industry and an awesome bunch. Many testing conferences turned virtual yet they kept the spirit and experience of it as good as real. And it is worth the praise. Isn't it interesting how creativity flourishes when things get chaotic? Hats off to all of you, who allowed me to see it and believe it.

Communication being one of the important factors for effective testing, I for some time felt that our time is up. Remote working, limited access to the system and people so that one can observe them meaningfully, and breaking changes of such kind made me feel, testing won't be as effective as it could be otherwise. But no. The way I have seen things being handled, the workarounds found and how quickly and smartly we all have responded to this adversity is commendable.

This very issue of Tea-time with Testers is living proof of it. I am so very delighted to have read, and now publishing some of the finest articles written on software testing in this issue. Interesting thing is, these articles are written by experts and newbies in the field, by professional testers, and also by a non-tester. And it makes me feel proud to see so much wisdom, ideas, and thoughtfulness springing in amidst this chaos.

I hope you will enjoy reading it, as much as I did.

Stay safe, stay cool, and please continue with your creation paying no heed to this chaos.

Until next time, people!

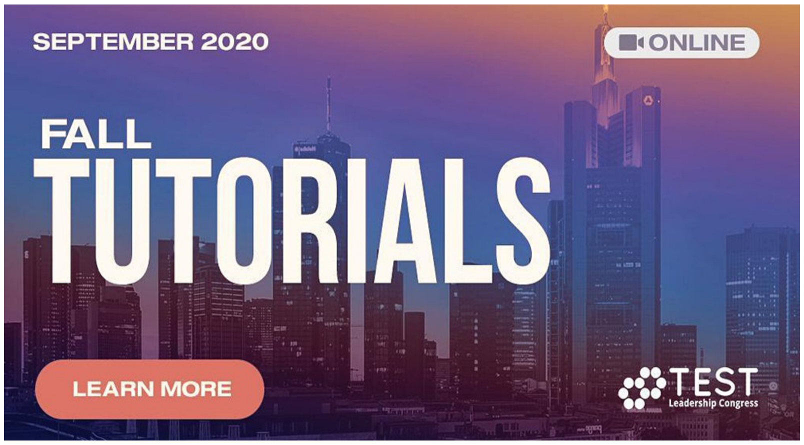
[illegible]

– **Lalitkumar Bhamare**
editor@teatimewithtesters.com
@Lalitbhamare / @TtimewidTesters

QUICK LOOK

SPOTLIGHT

NEED OF THE TIME



Speaking Tester's Mind

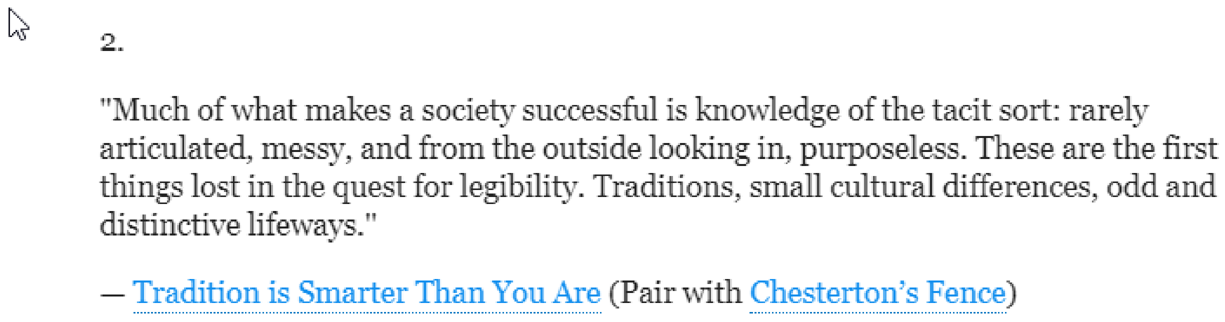
- straight from the author's desk

The “tacit”

- Rihab Loukil

Photo by - Jeff Nissen on Unsplash

In the RST slack, I liked a picture posted by Adam white.



2.

"Much of what makes a society successful is knowledge of the tacit sort: rarely articulated, messy, and from the outside looking in, purposeless. These are the first things lost in the quest for legibility. Traditions, small cultural differences, odd and distinctive lifeways."

— Tradition is Smarter Than You Are (Pair with Chesterton’s Fence)

Few questions came to my mind:

- How does this apply to software testing?
- Could we name 3 rarely articulated, messy, and "purposeless" behaviors related to testing?
- Why are they lost in the quest for legibility?
- What would the community reaction to this knowledge brought to light?
- Will any tester be able to take advantage of it?
- or does it require a special background/personality/environment?

Adam's answer went like this:

"That's a great question...

In your question, you missed the "From the outside" part of the quote which is important to give context to the answer. Here are 3 "testing" skills that can sometimes be seen as purposeless. One could argue that all team members need this, but, as you pointed out, we are mainly talking about software testing here.

1. Bug Investigation
2. Coverage reporting and storytelling about the product, the project, and testing
3. Risk identification and mitigation

These 3 are lost in the quest for legibility because they can't be summed up in a single number or word. They don't fit neatly on a dashboard and it's messy to figure them out. It takes time and we don't wind up with a pretty green bar at the end. You need **brain thinking** and **thoughtfulness** and perhaps most importantly **context**. These things are, oftentimes, tacit. The person who's doing the work has a **feeling**, an **inkling**, a **hunch**. They follow it and find the most beautiful issues ever thought of. (Bug investigation and storytelling - Ask me about the 'on the hour' bug sometimes).

This sense, *this tester sense,* can be taught but only to those that are willing and who are willing to try a different way. It's very Tao, Buddhist like as well as relating to the stoics.

This is why certain people can predict the future as it pertains to software and why heuristics are useful. The patterns are the same from product to product. There is no magic - you just need to know where to look, what patterns keep popping up. They start to form a model in your head. "On this new product we saw an issue that was the same as the old product - I wonder if there more that are similar".

At a certain point you don't even have to refer to the heuristics anymore they become as etched into your brain as sleeping, eating, and breathing. (Risk Identification and mitigation).

The community reaction to this, (if you mean community to be the company one works for) when done appropriately, is one of astonishment that mere mortal testers can shed such a profound light on the qualities (chose that word on purpose over quality) that the product is exhibiting (or sometimes not exhibiting).

There will be a shock that testers could add such value. Developers will talk about you for years to come. Old colleagues will remember things you and your team did that you have long forgotten. I was able to take advantage of this when I played the role of tester, test lead, test manager, test director, and all roles after that. *I don't think I have a special background, personality, or environment. I have a willingness to learn, to try new things, to teach what I know to others. I bring the lore of bugs from the past to the present. I do my best to explain my tacit knowledge and explain what goes on behind the curtain of software and my brain. I like to reveal the mystery behind the small cultural differences, the odd and the distinctive lifestyles (of skilled testers).*

So I thought:

What if we can train experienced testers to speak/teach/share their experiences in a way that decipher them? How could we do that? I believe testers do more/differently/deeply than they tell. And I would argue that not describing the "tacit" could lead/encourage some "talkative" people to minimize expertise to mathematical formulas, "best practices", and fancy measurements. My research aims to bring this knowledge to light, so I need to sense it myself first to be able to describe it to others.

Adam replied:

"I think we need to do more talking together as testers. We need to talk more to developers, product managers, executives, and then report back the challenges and the wins! (The wins are important - sometimes we forget that).

I used to go to conferences where we could discuss ideas, hash them out, explore, debate, argue. I've also attended conferences, or workshops, where people present experience reports.

Just being a part of the RST slack community is one thing we can do. Encourage more people to take any course from @satisfice @michaelbolton or @Robert Sabourin (those are the ones I'm most familiar with).

Thankfully there's been a lot of groundwork done already:

Explaining Testing To THEM and Low-Tech Testing Dashboard.

Rihab is a context-driven tester, a Peer Advisor on the RST (Rapid Software Testing) class, James Bach's student, and a Ph.D. student in software testing.

She is conducting qualitative research about what happens during testing (testers behavior and mental models), this may help in optimizing testing workflow, developing brand new testing tools or creating better courses/workshops for testers, just to name few.

One thing she is very good at is investigating. Asking questions, taking notes, reaching for outstanding people and possibilities, and reconstructing the puzzle to get the full image.

Here is a second thing she is good at meeting people, interviewing them, falling in love with those who are true to themselves, and seeing clearly what they have in special.

Yes, people like to be told about their strengths and talents, but she doesn't just tell, she investigates. What makes THIS person so? why did he do that? how can he manage things this way? And find patterns out of that. Expertise is not a ghost. Expertise is basically people-intrinsic.

What helps her get very easy into "authentic people"s close circle are her faith and principles. she doesn't try to please or to impress anyone. She is free. I just follow her instinct to answer her questions.

Otherwise, she is very bad at doing repetitive work or any stuff that doesn't require creativity, imagination, and thinking.

TEST Leadership Congress

SEPTEMBER – OCTOBER

ONLINE

UPGRADE YOUR CAREER THIS FALL

TUTORIALS AND CERTIFICATION

Our unique Quality Leadership Tutorial Curriculum program builds leadership skillset and expertise for Test and Quality Professionals.

The renowned industry experts teach courses on:

- Leadership Skills Development
- Test and Automation Strategies
- Remote Management
- Change Management
- Diversity and Inclusion
- AI Literacy
- Hiring and Team-building Practices

Jennifer Bonine

Ole S. Rasmussen

Ash Coleman

Thomas Haver

Lisa Crispin

Zhenya Rozinskiy

Kirk Walton

Janet Gregory

Bob Schatz

Dana Pylayeva

Anders Dinsen

Fiona Charles

Get access to 50+ sessions from Test Leadership Summer Season: watch recordings, meet the speakers, discuss with peers.

Learn more at: testleadershipcongress-ny.com/program/

How can I introduce change in my organisation?

- Patryk Oleksyk

Photo by Chris Lawton on Unsplash

Did you ever feel like being stuck in a hamster wheel? Doing the same work over and over again without any engagement? The job that you were happy to have and brought you joy, became… just a job that you had to do.

Burnout is a state where I would not wish anyone to be. For me, the worst period of it was 2 years ago, when I joined a project (still being in the old one) as a test lead, where the development process started 5 months prior. There where are a lot of issues with it:

1. CDD ("Conference driven development") - implementing new features before or on the conference day, and after it.
2. Unit/Integration checks - we had few and the ones we had brought neither value nor information.
3. Ticket/task handover - It was vaguely described and there were a lot of shallow agreements while working on the handover.
4. Business and domain knowledge - was not shared evenly among team members.
5. Requirements hadn't been explored - lack of time meant, that we lack some domain and business knowledge.
6. Our team depended on another team - communication between them was close to none.
7. Crunch periods - to deliver what was promised, overtime wasoverhours where required.
8. Dissatisfaction within the team - due to issues above.
9. And many more bottlenecks, that we don't have time to deal with right now

After a couple of months of me working there, the project was definitely not a good place to be in. I ended up waking up at 5 am on one Saturday just to create a presentation that would include all the things that were wrong with the project. This process let me vent out some frustrations and have the outline of all the project issues. I could just pat myself on the back and let it go, but I had further plans.

After being done with the presentation, I thought about possible solutions that we could implement in the short and long term. With that in my hand, I could start a conversation about the change with my manager.

Catalyst

Every transformation needs an event that will push someone to alter their behaviour. These events can be categorized into 2 types:

- First one, where you are self-driven to bring the change upfront.
- The second type is driven by external factors. Someone/something pushes you to adjust. That can be people base (demands from your boss, stakeholder, your work peers) or situation based (the technology used in the product is outdated, we lack funding or the world is about to end).

In this article, I want to focus on the first type. About the alterations coming from you, especially in the project context.

Why people want to make changes in the project. In my case, it was the burnout due to the work overload. Plus the tester inside me wanted to bring high business value to the customers and in my opinion, at that moment we did not deliver that. For you the trigger can be something like:

- Lack of information (e.g. you need to learn more about domain knowledge).
- Being dissatisfied with the status quo (e.g. the current testing process is more of a bottleneck than value).
- Yearning for stability in the project (e.g. we want to reduce the number of new things coming after sprint planning).
- The knowledge that things can be done better (e.g. if we formalize our development process, we won’t have a shallow agreement over the handover).

Plan

When you have your catalyst for change, the second thing you need is the proper communication of your change requests.

But to do that properly, in my opinion, we need to step back to the article title and break it into pieces.

"How can I introduce a change in the project?"

In this question, we have three key things to consider that will let you see things from different angles and help you achieve what you want.

These things are ***I, change*** and ***project.***

I , the change maker

You need to think about your reputation within the project and how it can increase or decrease the success during the meeting with others regarding the shift. For that I would consider these factors:

- Role - what are you doing and what are your responsibilities in the project? It's easier to introduce the change when you have some kind management or leadership role.
- Expertise knowledge - having a business, domain or technical knowledge will most likely allow you to push further.
- Relationship - When you have a more positive relationship with other people, they are more keen to support you with your ideas.

Change - the type of change and the way it will be introduced

The next to consider is our main topic itself. Think about its shape and form, the more you spend time with it, the easier it would be to explain the necessities of it to others.

The (or some) factors to consider:

- Scope - how large is it and can it be divided into smaller parts?
- Speed - how fast do you need to make it happen?
- Steps - what are the milestones for your change to happen?
- Essentials - what do we need before it can happen (tools, people)?
- Owner - who will be accountable and push it forward?
- Visibility - how can we display its results?

Project - everything is done within the project

Everything is happening in the project space. Things to think about:

- People - who will be affected by it (development team, users, stakeholders)?
- Affected - how will it affect others?
- Type - in what state project is in (in development, in production), what methodologies do we use?
- Cost - how much will it cost (effort, time, money)?
- External - what are the outer factors for your project?
- Mood - how will the overall emotions shift?

Having considered these things (and many others depending on your project context) we can start introducing the implementation plan. A clear vision and many other factors will help with the next step.

Evaluation and Inception

We have our plan and vision. Nothing else is left than crash test it.

In my opinion, you have two options to evaluate your change plan and move it through the project pipeline:

- Use agents of change - start planting a seed of a change ideas between your teammates and try to push it further up (Your peers can evaluate your initial idea and even bring more to the table).
- Present it to your higher-ups - show to your higher-ups the problem (they will be affected by them in the end) and balance them with your ideas (which can have a positive outcome for the project and the team itself).

What I deem the most successful way (and a little bit cliche one) is presenting your problems/ideas and the change proposals to your peers first. This way you can pretest your work and evaluate it. Then an improved plan can go to your higher-ups.

What is left is to set-up a meeting.

Management - patient zero

This step is a crucial one and you need to be prepared for it.

- Set a goal of the meeting - give context to the meeting.
- Prepare an agenda for the meeting - it will be easier for you to move from point to point and keep track of what you are doing.

- Have the facts and numbers ready - this will let the manager make proper decisions.
- Think about what matters should be prioritized - your improvement plan will be additional work for the project itself, depending on the project roadmap, not everything can be done as fast as you want.
- Have a can-do attitude.

Regarding the attitude, I would like to use my favourite lesson from the book “Lessons learned in software testing”.

Lesson 101: “When you decide to fight, decide to win!”. What I like about it is that it can be used not only in software testing but in other areas too. The main premise of it is:

“The principle that we urge you to adopt is that every appeal you make must be persuasive. Even if you don't win every appeal (you won't, of course), you should develop a reputation that every appeal of yours deserves to win.”

Getting back to the main point. The purpose of the meeting with management is to show your proposal for the change, if you don’t believe in your prior work that you did, then you just waste your and everybody's time.

From now on you may expect two outcomes.

Failure

“Shit happens”.

That does not mean that your ideas are wrong, there can be other reasons behind it:

- You can not have a full picture
  - Stakeholders or management plans for the project.
  - Other things can be more important (and bring more value) than your improvements.
- Humankind rejects changes - we love to have our conformity.
- It requires time - stuff that you presented can seed other concerns in your management mind, which can make alterations in the future.
- Sometimes the effort for advancement with the plan can cost more than you have imagined.

Bringing change is not an easy path to follow and don’t beat yourself down about it. Learn from your experience and try again in the future.

Success

Congratulations, you have introduced it. What is left is an equally hard part of cultivating the change culture in your team.

Find the person accountable for the process (like we had in the planning phase). The owner of the change process needs to take care of the initial push back (Humankind rejects change) and align the direction of the process with the team. Without that person, everything you did so far will go in vain.

Other key rules that I would suggest to follow for the process owner:

- Set up a deadline for the process
- Formalize the process. It will be easier for the team to stick to it.
- Have checkpoint meetings to evaluate the progress and decide what to do next.
- When you have a lot of improvements, introduce them in batches (don't overburden team with them).
- Keep communication clear between team members. Listen to their feedback and...
- Don't be afraid to make decisions.

The End game

Event > plan scope > inception > escalation > fail (learn, try again) / succ (owner, deadline, iterate)

The final question is, how did this whole process go for me?

Like I wrote before, I finished my presentation on Saturday. What was left for me was to have a meeting with my manager on Monday.

The meeting itself... Went well. I had built the rapport with the manager regarding the project, its issues and what path we should go. As a team, we introduced some short and long term repair mechanisms and other issues (which could not be solved by the team itself) were escalated further to C-level executives.

So was it a success? It depends…

The project itself finished 2 months later, due to a misconception of what the client wanted at the end (minimal viable product for conferences that would be also production ready). That way I moved to my previous project again.

On the plus side, I learnt a lot during that short time (from problem-solving to soft skills) and to this day on other projects I use the knowledge that I acquired during that time. The above change model itself also helped me a lot with other projects.

As a company, I think we gain a lot of expertise regarding these kinds of projects and how to deal with them in the future, but this could be another story for another day...

Annotations:

Cem Kaner, James Bach, Bret Pettichord (2002): Lessons Learned in Software Testing: A Context-Driven Approach

Rob Lambert (02.2020): Develop your Superpower at work with effective communication skills

Patryk Oleksyk is a test consultant and the first tester in software consulting company Consileon. A self-learning generalist, who uses Rapid Software Testing and Modern Testing methodologies in day to day work.

When not doing testing or automation in the project, he may be trying to improve or optimize anything that would benefit the client, project or team.

Some other testing thoughts https://consileon.pl/?s=oleksyk

Quality Talks with TLC Speakers
An interview series by Astrid Winkler
Click on the pictures to open
QUALITY TALKS

Do we need Batman if we have Testers?

- Astrid Winkler

Photo by Daniel Cheung on Unsplash

They say interesting things happen to you when you expect the least. I had such an interesting experience lately.

If you are active on Twitter, some of you might have noticed my reports on Test Leadership Congress 2020. For those who don't know, I am not a tester. When I was asked if I would like to write about my experience for the conference, I was unsure but excited. And with my official entry in this congress scene, I was testing the waters and trying to get the feel of the interesting world of software testing and tech-leadership which I was experiencing through this congress for the first time.

If someone would have told me, "Hey, you are going to attend a conference in NYC this summer ." I would have said yea, sure. We are in a global pandemic, what exactly you did not miss? I was 100% sure, this water was too cold for me.

But here I am after four weeks, writing an article about my understanding of the testing world. And I have built this understanding by attending a conference that should have been happening in New York but ultimately I attended from my cozy home office in Switzerland. I attended it on my computer, listening to and learning from great people from across different time zones. Thanks to the technology that made it happen. But, a HUGE thanks to the testers who made sure that this technology worked reliably. The more I reflect on my entire experience with technology, the more I realize that it is humans behind the technology who make sure that it works, against all odds.

And hence I wonder, do we need Batman if we have testers?

Attending this conference made me realize being a tester is not an easy job and for sure it takes one to be the master of the many trades. And here are some of the lessons I learned around what it takes to be a great tester (from the sessions that I could attend):

Continuous learning is crucial

This has been my biggest take away from this conference. If you are a tester, you must not stop learning new things. In four weeks of the conference, I was exposed to so many different topics, themes, areas of the tech industry and to be a great tester means to be able to cope with all these dynamic and vast things. I can't see any other way than a continuous learning attitude to make it happen.

The beautiful thing about learning is that nobody can take it away from you. - B.B. King

If you ask me, what I got from attending the sessions is beyond valuable. Over the weeks I gained knowledge, confidence, and the ability to understand a topic, I didn't even know existed before. Can I use all this knowledge in my current job? Maybe not. Will it push me forward in the future? Possible. Will all this have an impact on me? Most likely YES.

People matter (the most)

Softwares are made ultimately for people and by people. And hence, testing better be human-centric. By people, I do not only mean the end-user but also the people in the team that work together to deliver software.

And this fact kept coming up in some or the other form from different sessions that were delivered at the conference. For example -

An interesting story shared by Dawid Pacia in his talk, "Startup meets quality - a short story of QA Game changer" emphasized the importance of valuing the employees and the teams. And how much it is important to work together as a team. The quote from Richard Branson stuck with me for long -

"Clients do not come first. Employees come first. If you take care of your employees, they will take care of the clients."

The team-building workshop by Brittany Sherell was so much fun in a serious way which made me realize how important it is to grow as a private person also as a business person. And again, the importance of working as a team as she shared strategies to build a solid team:

- Discuss team personalities and preferences (survey, a team meeting, etc.)
- Collaborate as a team on how each member can use their strengths to contribute in the most powerful way.
- Create an environment that welcomes two-way, continuous feedback

Communication is the key and giving Feedback is a skill

Good testing is incomplete without communicating about the information found in a way that people would be able to relate to. And so is one's ability to give and receive feedback effectively.

Multiple sessions shared this key message in a way and I can't agree more.

"Why is there a Marble on my Nose?" by Angela Riggs made me realize how important it is to understand that even at work, we can not separate our emotions from ourselves as an individual. People need time and space to get through their emotions. Negative emotions reduce people's capacity for effective and productive work.

On communication skills, Angela said it is a way of transferring ideas, it happens whether or not we intend it to happen. Effective communication means adapting your communication style for different audiences. Communication makes us better testers because we have more awareness and control over the ways we share and receive information.

On feedback skills, Mike Lyles said in his session, “Constructive criticism is imminent, but we should remember to praise in public and criticize in private. Giving and accepting feedback, honestly, holds the team accountable for continuous training and education with open door policy.

There was another great session by Priyanka Halder where she shared the FBI Framework for communication and feedback:

- **F = Feeling:** What emotion did the action or behavior of the other cause you?

  (Are you angry, anxious, sad, disappointed, or happy, surprised, thankful?)

- **B = Behaviour:** What was the exact action that caused this emotion?

  (Note that you should not use the phrase “you always” or “you never”)

- **I = Impact:** What were the consequences of this action?

Data is the new fuel

Data and hence test-data plays a critical role in testing effectively.

Joshua Russell his session “Functional Test Data” emphasized this point. And honestly, this was one of my key sessions where I started to realize what is really happening behind the curtains. He recommended, to create the test data that tells a clear story, with believable details, and is carefully curated, including the use of personas. Well, this made me really happy actually. At that moment, I was convinced that the user-persona that I represent in my personal life is surely going to be part of someone’s test-data and thus making sure to consider my needs as a user.

Mindset is everything and Change is the only constant

Based on my learning about testing, by attending this conference for over a month, I learned that an important part of testing happens in the tester's mind and not really on their computer. Also, testers need to be flexible with changing contexts and changing their way of testing accordingly.

Amy Jo Esser made an everlasting impression on me with her outstanding talk around change and mindset.

In order to be the best version of ourselves, we need to change our habits which in turn can change our lives. Easier said than done, I know and this is why we need rules to guide us in the process of embracing the change. Amy Jo gave us three rules:

Rule One.

Make it a must. A Necessity. A non-negotiable. Create and write down a big why. It’s critical you master this skill/habit

Rule Two.

“Eat that Frog” first Brian Tracy has said, that your “frog” should be the most difficult task on your things-to-do list. The one you are most likely to procrastinate on.

Rule three.

Start tiny and atomic. Tiny habits and atomic habits. To change your habits and so your life seriously. On your journey always keep one thing in mind. “Every struggle s one step closer to greatness”

Christina Thalayasingam talked about Full Stack Testing which I believe also means that testers need to be flexible to cope with changes and work on their mindset to deal with those.

If you wonder “why should we go with full stack testing?”Christina answered that it is because we will have the ability to spot issues not just in functionally but any quality issue and that will enable us to narrow down bugs at an early stage. According to her, full-stack testing enables a quick feedback mechanism for every change. It is not required to rely on test teams working in silos and it creates business value by giving fast feedback loops on all layers of a technical process and always keep in mind, “You can't change your destination overnight, but you can change your direction overnight.”

Do not underestimate the power of Community

The community of Practice and Interests are powerful aids for testers to achieve desired culture change in the organization. At the same time, it matters that testers know how to contribute to communities and how to bring best out of those.

“How can we create links between different people?” you may ask. Emna Ayadi, in her session, answered this - we need to create a community of practice.

When groups are isolated (testers, developers, etc.) it’s easy to blame one particular group. Mixing the groups is effective for teamwork, which means to bring people together with the same interests but not the same skills. A Community of Practice is a combination of Community, Domain, and Practice.

In “Growing a Culture of Quality”, session Simon Prior talked about the first steps which need to be done to grow a culture of quality. Before changing the quality mindset in your organization, define where you are now and where do you want to be. Be clear on it and then you can define how to come from 1 to 2!

Automation helps but use it wisely

Paul Holland shared interesting stories with us. On automation in testing, he recommended asking - what is your biggest fear? Answer to this question gives us more input for our testing strategy and coverage of the required documentation. UI automation is often used too much and created by non-developers. When you take a tester away from testing and have them write automation you likely lose a good tester and gain a bad developer. Automation will likely not increase your coverage, decrease your costs, save your time, or allow you to reduce headcount. Unless you also increase risk. Automation can give you a decent sanity check of your product and execute in less time than a human performance the same checks.

I could go on and on and write more about the fascinating world of testing. But I guess I should take a break here. Over one month of interesting experiences, knowledge exchange, and inspiration, I couldn't be more grateful, that I had the chance to be a part of this world and community. The testing community is very engaging which reflects in the way you support and help each other.

The more I reflect on the key lessons I shared above, the more I start thinking of Batman who is my favorite superhero. I am very convinced that being great at testing requires practice, skills, courage, consistency, and a strong mindset. And Batman has it all.

Which makes me wonder, do we need Batman if we have testers?

A very special thanks to Anna Royzman and her team who did an amazing job organizing the conference and leading it in such difficult circumstances and never losing their smiles.

I am looking forward to meeting you all again. Until then, stay safe, take care, and love what you do.

Astrid Winkler is a budding freelance journalist and content writer from Switzerland. She is currently working as a Project Lead for a creative ad-firm based in Switzerland.

Creativity is Astrid's passion and writing is her lost-and-found love which she is willing to develop with more care.

Astrid recently covered Test Leadership Congress conference by Test Masters Academy and has done very detailed reporting about her impressions. Feel free to checkout her reports for the conference starting from here.

Connect with Astrid on Linkedin or follow her on twitter @AstridWinkler4

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

A Human's Guide to Testing AI

- Nilanjan Bhattacharya

Photo by Frank on Unsplash

What does it mean to test AI? If you look at the spam detection feature in your email, it sometimes works, less often doesn't. Even if you flag items as not spam, some again show up in spam after some time. You could categorize that as a machine learning FAIL?

Amazon recommends books related to your purchase. Some of them make sense. Others don't. Hopefully the algorithm learns and gets better over time. Are the incorrect recommendations a failure of the algorithm? Isn't it expected that recommendations will learn and improve over time? So, failures are in fact **not failures**?

What does it really mean to test AI?

Do you test the way the algorithm 'learns' and improves? You could test with real world data. Are you testing the math? Can you test the math?

What is 'AI'?

Providing a concise definition of AI (artificial intelligence) may be excruciating. The quote below is good enough context for this article (from a Human's Guide to Machine Intelligence).

AI involves enabling computers to do all the things that typically require human intelligence, including reasoning, understanding language, navigating the visual world, and manipulating objects. Machine learning is a subfield of AI that gives machines the ability to learn (progressively improve their performance on a specific task) from experience—the aptitude that underlies all other aspects of intelligence. If a robot is as good as humans at a variety of tasks but is unable to learn, it will soon fall behind. For that reason machine learning is, arguably, one of the most important aspects of AI.

In the rest of this article, I use the term, 'AI Software'. For the most part, the problems I describe are approached using **machine intelligence**. These algorithms use historical data to recognize patterns. Emails with text promising lottery wins are probably unsolicited spam.

Validate your hypothesis

Algorithms and the choices developers make have real world consequences. What is the purpose of a recommendation engine on Amazon? When you make purchases, recommendation engines will display related items. To test a recommendation engine, you may validate the relevance of the recommended items. That in itself is a very challenging problem. However, recommendation engines have a broader role.

An expectation of recommendation engines is they promise that obscure titles will become popular. As a tester, how will you validate that hypothesis? You can validate your hypothesis by using a control group which isn't exposed to recommendations. Then compare the actual users with the control group. Do users purchase more of the recommended items (compared to the control group)?

Can you consider other factors? How does the purchasing behaviour change across all products? Can you look at market share as well as absolute sales? Do recommendations get **new users** to purchase products? Or is it the same users who are purchasing the same type of products? How will recommendation engines work, when books don't have sufficient purchasing history? There are many more questions you can ask which can lead to further experiments.

Algorithm impact on user behavior

Algorithms can impact how people behave in the real world. That seems obvious with social media apps. How do you measure the influence of algorithms on social media apps?

You could tweak an algorithm to show a different set of choices. Then measure user behaviour. In a research study, Facebook users were shown more hard-hitting news. These voters had a higher voter turnout.

You could alter recommendations shown to users along with a control group. Then measure user behaviour. In another study, Facebook users were shown more positive and more negative posts. User's subsequent posts reflected the emotion of their news feed.

A Human's guide to Machine Intelligence

In the last few years, AI FAILS have been a novelty mixed with indignation. It's easy to find examples of failures in the news. In contrast, the book, 'A Human's guide to Machine Intelligence', is a very readable account of **why** algorithms fail. In this blog post, I've posed some of the questions, from the book, that you might ask if you were evaluating algorithms **as a software tester**. The book has detailed analysis and explanations of how algorithms fail and the background on some of these questions. The focus of the book is **not on the math**, but how algorithm design choices can cause problems in the real world.

As a software tester, it's better to focus on strategies to test AI, or **personal accounts** of how **you** have tested AI algorithms.

What is testing AI?

Testing starts with an observation or an idea. You then validate your idea by conducting an experiment.

- Facebook researchers have a hypothesis that social media posts influence user behavior.
- A software developer notices that businesses are not getting good SEO rankings.
- A user complains that his email suddenly flags messages as spam.

These observations are followed by an experiment.

The approach to testing AI is probably no different from testing other software. When testing a calculator, I wonder whether it assumes numbers are entered in a particular format. Does it accept '.03' without the leading zero (0)? If it doesn't, does it matter? I can test out different options to infer what the calculator does.

This does not mean testing AI is straightforward or **even possible** in your role as a tester. Testers or developers may not be part of the group conducting tests with users. On the other hand, in smaller teams any team member may be able to ask questions. They may be part of the group which reviews experiments with users.

Unanticipated consequences

When working with complex algorithms, especially those that use real world data, you need to think about side effects or unanticipated consequences.

In a, now, well-known problem, when using autocomplete on a search engine, users are prompted with common prejudices. Entering, 'women should', may display suggestions such as 'women should stay at home'. A search engine's auto-complete may not only offer harmful suggestions, but direct users to those sites. The unintended consequence of auto-complete is that users may be led to sites which give harmful messages. A seemingly harmless enhancement such as auto-complete on a search engine can influence people's attitudes and can impact society as a whole. (As an aside, how do you design systems to address this issue?)

When designing auto-complete or similar systems, a bigger challenge is how do you differentiate between concepts, such as porn and sexuality? Does your algorithm understand the difference between concepts, or are they just words?

On some social media sites, you are alerted about offensive language. How do you handle names or locations which may include offensive words? One way to handle the issue is to ignore proper nouns when alerting users – which itself may be a challenge. If you do allow proper nouns, how do you handle attempts to misuse the system?

Social media sites like Facebook and Linkedin create trending topics and feeds. How do you handle the presence of ‘fake news’ in the feed? Do you question the credibility of news sources? Do you question whether someone can tamper your data?

To be fair, many of these questions may not be in the purview of the development team or of software testers. However, this post should give ideas about the questions that you could ask if you can influence decisions.

Real World Data

Problems solved using AI often use a large amount of real-world data. Real-world data will have its quirks which are difficult to anticipate in the lab. You can only go so far in simulating a Facebook feed (which does not imply that you do nothing, or that there aren’t powerful alternatives).

Problems solved using AI often use social data – information related to people’s lives. Facebook and similar apps use friend details and activity and user interaction, information related to social groups. Other systems impact business, such as automated trading, or a social feed on a financial website, book recommendations or search engine rankings. Advertising systems affect consumer behavior.

In the case of auto-complete, in a search engine, you need to handle loaded topics, like race, gender, religion. You also need to consider people wanting to mislead gullible users. Image recognition is not only about pixels, but about people and their gender, race and location.

Not being able to use real world data is a major challenge for testing AI software.

The problem with testing AI

Some of the most important insights about testing AI from the book are:

We should look beyond the accuracy of algorithms.

> *But we generally don’t evaluate, let alone measure, algorithms holistically, considering all the ways in which they change their users’ decisions—individually or in aggregate.*

We should think about unintended consequences.

> *Most other algorithms continue to be evaluated narrowly using one or two technical criteria and without regard to the kinds of metrics that keep social scientists awake at night—issues such as fairness, accountability, safety, and privacy. Unintended consequences can arise even when every step of an algorithm is carefully designed by a programmer.*

These are my insights when I add an understanding of testing:

- The start of a test is the question we ask. A question may come up from exploration, experiments or from a problem faced by the user.
- In general terms, you will need to be creative, as opposed to following a process, to ask the right questions.
- **Whether** you ask the question is much more important than **who** asks or **when** you ask.
- Asking the question is more important than what led you to ask the question – whether it is the use of tools or a thought experiment.
- The examples I have described in this article are definitive, i.e., there is a clear hypothesis followed by an experiment. The actual process of testing (and thinking) is much more fluid and situational. The result of experiments lead to more questions and more experiments. You may also investigate multiple questions. You may use the software, like a user would, to discover more problems. The overarching purpose of testing is to keep learning about the software in order to find unknown risks.

For an article like this, there is no big reveal. I didn't focus on specific techniques. I also didn't promise a theory which explains everything. I won't be smug that testers or developers **can** test AI based software, if **only** we ask questions or are inquisitive. Good testing will require preparation along with aptitude. Finding problems will also require a strong understanding of AI and the related algorithms.

The problem with testing AI is not mistakes or oversights. The challenge is unintended consequences. The problem is with algorithm design choices and their impact on the real world and users. The broader problem with testing AI is not recognizing that **we need to ask questions and keep learning.** It's the same problem with testing any other software. Just as with software in general, it may not be apparent that there is a problem to be solved.

Notes

1. A Human's Guide to Machine Intelligence, Kartik Hosanagar, https://www.amazon.com.au/Humans-Guide-Machine-Intelligence-Algorithms/dp/0525560882

2. Kartik Hosanagar is a professor of Marketing at The Wharton School in the University of Pennsylvania. He lists research interests as: internet advertising, e-commerce, digital media. Links: Personal website, Twitter.

3. If you are not a tester, this is a great article to understand testing: https://www.linkedin.com/pulse/how-cem-kaner-testssoftware-nilanjan-bhattacharya/ (All credit goes to the original author – Cem Kaner.)

4. If you want to understand testing, without reading much, you can follow Michael Bolton on Twitter: https://twitter.com/michaelbolton/ He tweets frequently and more importantly, many of his tweets are profound. For more, you can read his blog – www.developsense.com/blog. Another thought leader in testing is James Bach – www.satisfice.com/blog. I recommend the Kindle version of the classic and easy to read book on testing: https://www.amazon.com/Lessons-Learned-Software-Testing-Context-Driven-ebook/dp/B07MGKRDBY

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/).

Secure your spot today.

Contact us at sales@teatimewithtesters.com

Photo by Kate on Unsplash

How To Talk About Software Testing

- James Bach

Photo by Frank on Unsplash

Managers, developers, and even testers often have questions about testing that need answers:

- “Why didn’t we find that bug before we released?”
- “Why don’t we do prevention instead of testing?”
- “Testing would be better if we used {X} practice, just like {successful company Y}!”
- “Why can’t we automate all the testing?”
- “Why does testing take so long? How much testing is enough?”
- “Why do we need dedicated testers? Why don’t we just get user feedback instead of testing?

Why can’t developers do the testing? Why don’t we just get everyone to test?”

Questions like these often arise when there is trouble in the organization that seems to come from testing or to surround the testing process. But to deal with these questions you first must understand what testing is and what it isn’t. Then you must understand the parts of testing, how they relate together, and what words describe them. Only then can you productively talk about testing without being crippled by unhelpful concepts that would otherwise be embedded in your speech.

This document has two sections:

- Testing Basics (stuff you need to know about testing to talk about it coherently)
- Testing Conversation Patterns (kinds of conversations you might have about testing)

Testing Basics

A powerful definition of testing is the process of evaluating a product by learning about it through exploration and experiment. That means it's not the same as review, inspection, or "quality assurance" although it plays a role in those things.

By "evaluate" I mean primarily identifying anything about the product that is potentially troublesome. Another word for potential trouble is risk. Testing is therefore a process of analyzing business risk associated with the product (from now on let's just call that "product risk").

1. The essence of testing is to believe in the existence of trouble and to eagerly seek it.

Testing is an *inherently skeptical* process. The essence of testing largely lies in how you think about what you see. A tester should be negatively biased (because it is usually better to think you see a problem and be wrong, than to think there is no problem and be wrong). When a competent tester sees a product that appears at first glance to work, his reaction is not "it works!" but rather that "it's possible that it's good enough to release, but it's also possible that there are serious problems that haven't yet been revealed." Only when sufficient testing has been done (meaning sufficient coverage with sufficiently sensitive oracles relative to business risk) can a tester offer a well-founded opinion of the status of the product.

Testing never "*proves*" that the product fulfills its requirements. This is part of a profound truth about science in general: you can disprove a theory about the state of the natural world based on one single fact, but you can never prove that the theory is true, because that would require collecting every possible fact. Consider a barrel of apples. You can pick one apple and see that it is rotten, and from that you could conclude that the barrel fails to fulfill the requirements of containing only fresh apples. But if that one apple is fresh, you could not conclude that all the other ones are also fresh.

The testing process does not determine that the product is "done" or ready to be released. In other words, there is no way to establish unambiguous, algorithmic, or mathematical criteria that can dictate a good and responsible business decision about releasing software. Instead testing uncovers the data needed for management (meaning whoever has responsibility to make the release decision) to make a sufficiently informed decision to release. Deploying software is always a complex business decision rather than a simple technical one.

The testing process is all about learning. Anyone who is not learning while testing is also not testing. We seek to learn the status of the product, of course, but we also seek to learn how we might be fooled into overlooking important problems. Good testing requires continual refreshment of our means of detecting trouble, based in part on studying the bugs that were found only after the product was released.

The mission of testing is to inform stakeholders. Specifically, testing exists to provide stakeholders with the right information to make sufficiently well-informed decisions about the product. The testing process itself does not and cannot determine if the product is "good enough to release," since that is strictly a stakeholder decision.

> ***Why this matters:*** *When we fail to understand and respect the essence of testing, management and other outsiders to the process will have inflated expectations about what testing can provide and may use testing as a scapegoat for problems that originate elsewhere.*

2. A test is an experiment performed by a person on a product.

Consider a “play” in the theatre. A script for a play is often called a play, but the real play is the performance itself, with the actors on the stage. People may even speak of the “play” as a set of performances (e.g. “the play ran for four weeks on Broadway”).

Think of a test in the same way. We can informally speak of a test specification as a “test,” but try to remember at all times that the test specification never fully specifies the actual test that is performed, because that involves human attention and judgment. We can also speak of the “same test” over time even though the test is evolving and perhaps never performed exactly the same way twice.

Just be aware that, in truth, a “test” isn’t really a test except in the moment it is being performed. That’s when it becomes real, for better (when a skilled and motivated tester is performing it) or for worse (when an incompetent or inattentive tester is at the wheel). Many tests begin as open questions and sketches and become more defined and systematic with time. Testing is inherently exploratory, just like the development process for the product itself. This is even true for tests that employ automated elements, since that automation must be prototyped and debugged and maintained.

> ***Why this matters:*** *This definition frames testing as an inherently human process that can be aided by tools but not fully automated. Defining it that way helps defend the testing process against attempts to oversimplify and dismiss it.*

3. The motivation for testing is risk.

If there is no product risk, then there is no need to test. Testing begins with a sort of faith that product risk exists; usually stemming from various risk indicators, such as the existence of complexity and the potential for harm if the code behaves badly. As testing proceeds, and finds problems or fails to find them, faith about risk becomes replaced with fact-based reasoning about risk, until that very process of reasoning leads testers to the conclusion that enough is known to allow management to make reasonably informed decisions about the product.

The ability to think systematically about risk is therefore a key to professional testing.

> ***Why this matters:*** *Although risk is fundamental to testing, few people are trained to think about risk. This leads to haphazard test strategy and wasted effort. Meanwhile, any time you design a test you must be able to answer this question: why does this test need to exist? If your answer is “because the product might not work” that’s not good enough. What specifically won’t work? In what way might it not work? Is that sort of failure likely? Is this the only test that covers the product? What specific value does this specific test bring?*

4. Anyone who does testing is, at that moment, a tester.

Doing testing well requires certain intellectual, social, and technical skills, but-- just as with cooking-- literally anyone can do testing to some degree, and literally anyone can contribute to an excellent testing process.

Some people specialize in testing and are full-time testers. But for the purposes of this document, the word “tester” applies to anyone-- developer, manager, etc.-- who is currently engaged in an attempt to test a product.

It’s worth distinguishing between two kinds of testers: responsible and supporting. A responsible tester is any tester who is accountable for the quality of his own work. In other words, responsible testers normally work without supervision. Responsible testers also decide on their own tactics and techniques, and make their own decisions about when to stop testing. Supporting testers are not expected to control their own test strategy. They include anyone who drops in to help the testing process or any other tester who works only under the direct supervision of a test lead. Often the task of writing formal bug reports is reserved for responsible testers.

Why this matters: *For many years, testing was handled mostly by specialists, who had the time and focus to learn deep truths about testing. But with the advent of “Agile,” many people now get involved with testing without making it their specialty and without having the time to devote to planning or self- improvement as a tester. That means it is especially important to discuss and review the essentials of testing explicitly, as an organization, rather than assuming that testing will automatically be performed to a professional standard by people who are steeped in their trade.*

5. Testing is not verification; testing includes verification.

You can only verify something that has a definite truth value: true or false. But the quality of a product does not have a definite truth value, partly because “quality” is a multidimensional concept that involves subjective tradeoffs. You can verify that a movie has received 46.7 on the “tomatometer,” but you cannot verify that the movie is actually worse than one that received 53.2 from the same website. Just as people can reasonably disagree about the quality of movies, people disagree about the quality of software. In that sense you can assess quality-- you can come to a fact-based judgment of it-- but not verify it. Quality cannot meaningfully be reduced to a single dimension. For instance, I can verify that, given “2+2” as input, just after startup, a particular calculator at a particular time displayed a “4” on its screen. But this is not the same as verifying that “addition works.” Another way of saying this is that verification establishes facts, but no set of set of facts with finite coverage is logically equivalent to a sweeping generalization.

In Rapid Software Testing methodology, we call verification “checking,” which is short for “fact checking.” But testing is bigger than checking. Testing does make use of verification as a tactic, but testing also involves critical thinking, tacit knowledge, and social competence. Testing involves hunches. Testing is a process of sensemaking and theory-building based on the facts that are observed. This is far more than simple verification.

Unlike verification, testing does attempt to make reasonable generalizations— leaps of judgment— grounded in facts, that management can use as one basis for decisions. Testing results in an assessment of the quality of a product.

Why this matters: *Verification is often easy. Testing is hard. It can be seductive to perform fact checks instead of tests because checks involve no judgment and therefore no possibility of anyone disputing your judgment. They are safe and objective. But that very quality also makes them systematically shallow and blinkered. A strategy focused solely on verification would fail to notice big product risks that any reasonable human tester should detect and investigate.*

6. Performing tests is just one part of testing.

Any time you are experimenting and exploring a product for the purpose of discovering bugs (which includes using automation to help you do that) you are performing tests.

But here are some things that are also testing: conferring with the team, designing tests, developing data sets, developing tools, using tools to create and perform checks, studying specifications, writing bug reports, tracking bugs, studying the technology used to create the product, learning about users, planning, etc. Any of these activities are testing when performed for the purposes of fulfilling the mission of testing.

In other words, testing is a bigger process than merely performing tests, and you can't just point to a set of test cases and honestly say "that's the testing, right there." It isn't. Testing is the entirety of your process that delivers the value of the testing mission.

Why this matters: *Good testing requires that testers have the time and resources to learn, model, design, record, collect, discuss, etc. The assumption that the process consists simply of writing test cases and then running them is not just wrong but terribly damaging. It forces testers to do rushed, bad work that results only in shallow testing that will miss important bugs.*

7. A responsible tester must think differently than a developer.

A developer must think of one good way to make things work; a tester strives to imagine 999 ways it could fail. This should not be characterized as "constructive vs. destructive" since the tester is in no way destroying anything (except maybe our illusions of confidence). The distinction is more "optimistic vs. pessimistic" or "imperative (do this!) vs. hypothetical (what if?)"

So, testers live in a world of many hypothetical failures, almost all of which don't happen, but many of which we are obliged to investigate-- because some happen. To a developer's eye, testing can look like an endless, fruitless loop. Where does it ever end? In fact, a major aspect of skilled testing is knowing how to make a case that it is time to stop testing; this is rarely a simple determination.

One way to put it is that development feels like a convergent task, moving toward completion, whereas testing seems to push in the opposite direction: opening up new possibilities and looking into each one, which leads to even more possibilities, and so on. For the same person to think like a developer and like a tester at the same time is quite difficult, even painful, requiring extraordinary discipline. This is not to say that developers should not include unit level checks in their code. Those are important. But unit "tests" are not really tests, they are low-level fact checks, and bear little resemblance to the skeptical and creative process of testing.

Testers and developers necessarily work by different incentives: the thrill of building something obviously good vs. the thrill of discovering hidden flaws. When they work together, they can build something that truly is good and doesn't just seem that way at first blush.

Coding is a useful skill for testers, but not all testers should be coders. Testers who are coders often focus on writing tools to help testing instead of directly and interactively testing the product. Testing benefit from diversity, including all forms of diversity, but especially cognitive diversity represented by some people who think more like developers and understand technology more deeply, as well as some people who focus on the behavior and look of the product and understand the users more deeply. It's good to have some people who want to "get things done" (even if the work is not the best it could be) and others who want to "do the job right" (even if it takes longer to get things done).

Why this matters: *The reason people specialize as testers is to allow them to focus on problem discovery without being hindered by biases that come from hoping that the product will work, or believing that it cannot fail, or being too tired from making it work to put the required energy in to detecting failure. For any non-specialist who occasionally does testing, these are dangerous biases that must be managed.*

8. The five parts of a test are: tester, procedure, coverage, oracles, and the motivating risk.

Tester. There cannot be a test without a tester. The tester is the human being who takes responsibility for the development, operation, and maintenance of the test. The tester is not just a steward for the test, the tester is part of the test. The tester's judgment and attentiveness are a vital part of any good test. A corollary to this is that when a tester gives a test to another tester, that changes the test. Two competent testers can follow the same formal test procedure, but they will not be performing exactly the same test, because each test depends on a variety of human factors to make it work. Another corollary to this principle is that giving a good test procedure to a low-skilled tester will diminish or even destroy the value of that test; whereas giving a poor test procedure to an excellent tester may result is good testing getting done, because that skilled tester will automatically correct the design or compensate in some other way.

Procedure. A procedure is the way that a test is performed. The procedure may or may not be written down, but it must exist, or there won't be a test. Even if a procedure is written down, the parts performed by a tester (as opposed to that done by tools) will always involve unwritten elements supplied automatically by the know-how of the tester. If tools are used to perform a test, then the tools are part of the test procedure.

Coverage. Coverage is what was tested. There are many kinds of coverage, but among them are these broad categories: structure, function, data, interfaces, platform, operations, and timing. You need to know, at least in broad terms, what your tests cover and what they don't cover. Code coverage is one kind of structural coverage, and there are tools which can measure that. Data coverage, by contrast, is not so easily measured, since there are so many kinds of data and they can be combined in so many ways. Measuring that kind of coverage would require keeping track of all the data you test with and comparing that with all the data you could have used.

All coverage is based on a model, by which I mean some particular way of describing, depicting, or conceiving of the product; a model is a representation of the thing it models. For instance, code coverage is based on a model of code, which is usually the human readable source code itself. The usual form that models take when discussing test coverage is a list or an outline. A model of browsers for the purposes of browser coverage would be a list of browsers and a list of relevant features of browsers.

You could say that a model provides a perspective on the product, and to find every important bug we need to test from different perspectives.

A test condition is defined as something about the product that could be examined during a test, or that could influence the outcome of a test. Basically, a test condition is something that could be tested. If you model the product in some way, such as listing all its error messages, then each error message is a test condition. If you have a list of buttons that can be pushed, then each button is a test condition. Every feature of the product is a test condition, and so is every line of code. We cannot test all possible conditions, but we do need to know how to identify them and talk about them.

Oracles. Oracles are how problems are detected. No matter what your coverage is, you can't find a bug without some oracle. An oracle is defined as a means to detect a bug once it has been encountered. There are many kinds of oracles, each of which are sensitive to different kinds of bugs, but every oracle ultimately comes down to some concept of consistency. We can detect a bug when the product behaves in a manner that is not consistent with something important, such as a specification, or userexpectation, or some state of the world that it's supposed to represent, etc. Testing can be characterized as the search for important inconsistencies between the state of the product as it is and as it ought to be.

Motivating Risk. The motivating risk for a test is the probability and the impact of a problem with the product that justifies designing and performing that specific test. All testing is both motivated by our beliefs about risk and all good testing leads to a better understanding of the actual risk posed by the product. Ultimately, the purpose of testing is to establish and maintain a good understanding of risk so that everyone on the team, including developers and management, can make the right decisions at the right time to develop the product.

> ***Why this matters:*** *Discussing, evaluating, and defending your tests begins with understanding these five elements that every test must have. When you explain your tests and show how they fit in the overall test strategy, you will have to speak to each of these essential elements.*

Conversation Patterns for Testing

"Why didn't we find that bug before we released?"

This may be interpreted as:

1. That is the sort of interesting bug we would have wanted to find before release.
2. We tried to find every interesting bug before release.
3. Yet, we did not find that bug, which is disappointing.
4. Something about our process must have been wrong, which led to that disappointment.
5. What went wrong?

But premise #4 is incorrect. It is not necessarily true that something must go wrong for a bug to remain undetected. That's because testing is necessarily a sampling process. We cannot test everything, and we don't even try to do so. We make educated guesses about risk, and while we can improve our education and make better guesses, we can't remove the element of guessing entirely. In other words, even the best test process that it is possible to perform may lead to some disappointment. Testing just isn't a sure thing.

However, it may very well be true that something did go wrong with our process, especially if the bug that escaped is a particularly bad one. Treat every escaped bug as an opportunity to ask healthy and constructive questions about what happened and why.

Suggested response: "Let's look into it and find out."

No need to feel defensive about this. No one can rationally expect perfection, but our clients can expect testers to learn from experience. But beware of reducing the situation to a simple one-dimensional explanation. When investigating and discussing the matter more deeply, keep these issues in mind:

1. Testability. Was there anything about the design the of the product, or organization of the project, which allowed this bug to hide from us. Is there some identifiable improvement in those things that would make it harder for future bugs to hide?

2. Opportunity cost. If you had done what was necessary to find that escaped bug, would other bugs have escaped instead?

3. **Hindsight bias.** When analyzing how to avoid future bugs, it is tempting to focus on the exact circumstances of the bug that escaped, because you know that bug actually occurred. But that is too narrow. The next bug to escape won't be exactly like that one, but may be similar, so think about the set of all bugs that are similar but not the same, and what you could do to find any future bug belonging to that set.

4. Automation. Is there an automatic way to catch bugs like that? Are such bugs important and prevalent enough to justify that automation?

5. Tester Agency. Does the reason the bug escaped imply anything about the focus, commitment, or skills of the tester? Perhaps the test strategy and tool set are fine, and the tester just had a bad day. Or perhaps no one has taken enough ownership of the testing process.

"Why don't we do prevention instead of testing?"

On its face, this question asserts a false choice. But the questioner probably meant to say "Perhaps it would be better to put more of our effort into prevention and less into testing."

Suggested response: Affirm the value of both activities, but turn the conversation to the central issue, which is risk: "We must do both. For instance, we do lots of things to prevent our building from burning down. We have circuit breakers to prevent electrical fires. But we also have smoke alarms and fire stations, just in case our prevention measures fail. The depth of our testing should relate to the potential risk we perceive, and as that risk falls, so might our investment in testing. Of course, the ultimate prevention is not to develop a product in the first place. Assuming we want to create something new in the marketplace, however, a certain amount of risk is inevitable."

"Testing would be better if we used {X} practice, just like {successful company Y}!"

No matter what field of craftsmanship you look at, there are a few things that you will always see. One of them is practitioners who are aware of a variety of practices (i.e. methods, tools, and any other heuristics that are available for use) and make choices about which ones to use in which situations. Methodology should be context-driven, not driven by blind pursuit of whatever is popular at the moment.

When someone suggests that there is a successful company (Facebook and Google are often cited) that gets its success through not doing certain things or always doing other things, this is probably not coming from a consideration of problems and how to solve them. It's based on incomplete rumors. We don't know what Facebook and Google actually do; nor why they chose to do it that way. And we aren't in a position to evaluate that. We may be hearing a self-congratulatory myth.

Furthermore, what makes companies successful is primarily their business models and intellectual properties, not their engineering excellence. Famous companies can generally afford to be shockingly wasteful and still make a profit. And anyone who has worked in a famously successful company can tell you it's a constant challenge to get people to believe in the possibility of failure. Success creates a haze of complacency which is toxic to process improvement.

Suggested Response: Affirm that good ideas can come from anywhere. Remind that responsible technical organizations probably do not thoughtlessly follow trends for the sake of trends. Then offer to seriously consider the practice. If a team wants to try it out, encourage them to make an experiement. But keep certain caveats in mind and be prepared to discuss them:

1. **Skills.** Does the new practice require special skills to use correctly? Is any outside training required?
2. **Opportunity cost.** If you deploy this practice what will be pushed aside? What will you not have time to do?
3. **Evaluation.** How will you know if it is going well? How will you detect new problems that are created?
4. **Progress horizon**. How much time do you need to give the experiment in order to declare it a success or failure? When is it reasonable to expect results?
5. **Required support**. How much cooperation and what infrastructure is needed to try it out? Can it be done on the cheap?

“Why can’t we automate all the testing?”

This is a reasonable question if you believe that testing is the same as fact checking. Fact checking can be automated, although it might be expensive and slow to create and maintain it. Nevertheless it is possible.

But testing is much more than fact checking. Testing encompasses the entire process of building mental models of the product and of risk that are required in order to make a compelling, credible report about the readiness of that product for release. Remember, you are not verifying facts when you test, you are making an assessment.

The short reason we can’t automate “all the testing” is that to make an assessment of a product is a non-algorithmic, exploratory, socially situated learning process. But here is a slightly longer version of that: Some bugs are easy to anticipate and easily triggered, and easy to spot when they manifest themselves (call those bugs “shallow”), but many bugs are not easy to anticipate, or not easily triggered, or not easy to detect unless you know just what to look for (call those bugs “deep”). Testers are not just looking for shallow bugs, but for all important bugs. To find deep bugs requires insight of a kind that often comes only after playing with the product for an extended period of time. It may require noticing something strange and following-up on it. It may require doing complex operations that are easy for a human to do, yet expensive to automate. No good tester has all the good testing ideas right at the beginning of a project. Meanwhile, to automate something requires that you have a specific formal procedure that will spot every kind of important bug. There is no such thing. Big bugs don’t follow any pre-ordained set of rules, and without rules we don’t know what code to write to find them.

One more reason we can’t just automate all the testing is that testing requires social competence. A test must make judgments about what kind of data matters, what kinds of problems matter, which testing ismore important or less important, etc. These decisions are based on an understanding of the shifting and evolving values of the stakeholders of the product. These decisions must be defensible. All that requires social competence.

Automated checking only succeeds as a substitute for testing where there are no deep bugs that matter. Wherever you find that, it’s usually in a very stable codebase or in a product that has customers who are tolerant about failure.

Suggested response: “Look. We can’t automate parenting, management, getting to know people, falling in love, grief counseling, medical care, government, and no one is wondering when we can fire all the programmers and replace them with automated programmers. So, why do you think a skilled activity like hunting for a massive and completely unexpected bug is an exception to that? We can automate lots of interesting things that are part of testing, however. Wherever we can do that in a cost-effective way, we should.”

“Why does testing take so long? How much testing is enough?”

These questions can only be answered in context. Some testing takes longer, and some testing can be done quite quickly. It depends on many factors, all of which can be called aspects of testability. See the Heuristics of Testability document for details.

Suggested response: “Which specific testing are you talking about? If we are talking about something specific, then we can reach a specific answer. Otherwise here is the general answer: testing takes however long it takes for the tester to build a compelling enough case that the product is tested well enough so that the stakeholders are able to make decisions based on a good enough knowledge of the risks associated with the product.”

“Why do we need dedicated testers? Why don’t we just get user feedback instead of testing? Why can’t developers do the testing? Why don’t we just get everyone to test?”

Often the foundational issue that gives rise to this question is a lack of understanding about what skilled testing is and what skilled testers do. The questioner may consider testing skills to be ubiquitous. The questioner may really be saying “since anyone can test as easily as anyone else, why bother with full-time testers?”

However, it may be that the lack of understanding is about roles: maybe the questioner is questioning the very idea of having people specialize or focus on any one activity, rather than fluidly moving from one activity to another over time. See the How to Think About Roles and Actors document for a list of dynamics that affect roles and the people who play them. For instance, one benefit of a having a person specialize in a role is readiness. If you only test once-in-a-while, you are probably not looking ahead and planning and preparing to perform testing. While a dedicated tester would be preparing, a casual tester if doing some entirely different job.

Suggested responses: “Dedicating people to a complex task improves their efficiency. This involves the improvement of competence, commitment, readiness, and coordination. Of course, in a low risk situation, we might not need such efficiency and effectiveness. Yes, we can get feedback from users, too, and we should, but you still need people who can process that information. Users are terrible bug reporters, and developers don’t have the time and usually not the patience to follow-up on all thosehalf-baked reports. Certainly, everyone on the team can help testing. But someone needs to be responsible for it, or it will turn into chaos.”

James Bach is the creator of Rapid Software Testing methodology, co-author of Lessons Learned in Software Testing and author of Secrets of a Buccaneer-Scholar.

He is an avid student of Jerry Weinberg’s work, and was also co-host of the first Amplifying Your Effectiveness conference, and lead editor of the Amplifying Your Effectiveness book, working with Jerry.

Visit https://www.satisfice.com/ to know more about James and his work.

Tribal Qonf 2020 Video Roundup

TRIBAL QONF
• A Testing Conference by The Test Tribe •
Virtual
27-28 JUNE
20 Speakers | 2 Days | Talks | Micro Workshop | Expert Panel | Fireside Chat

Sharing is caring! Don't be selfish ;)

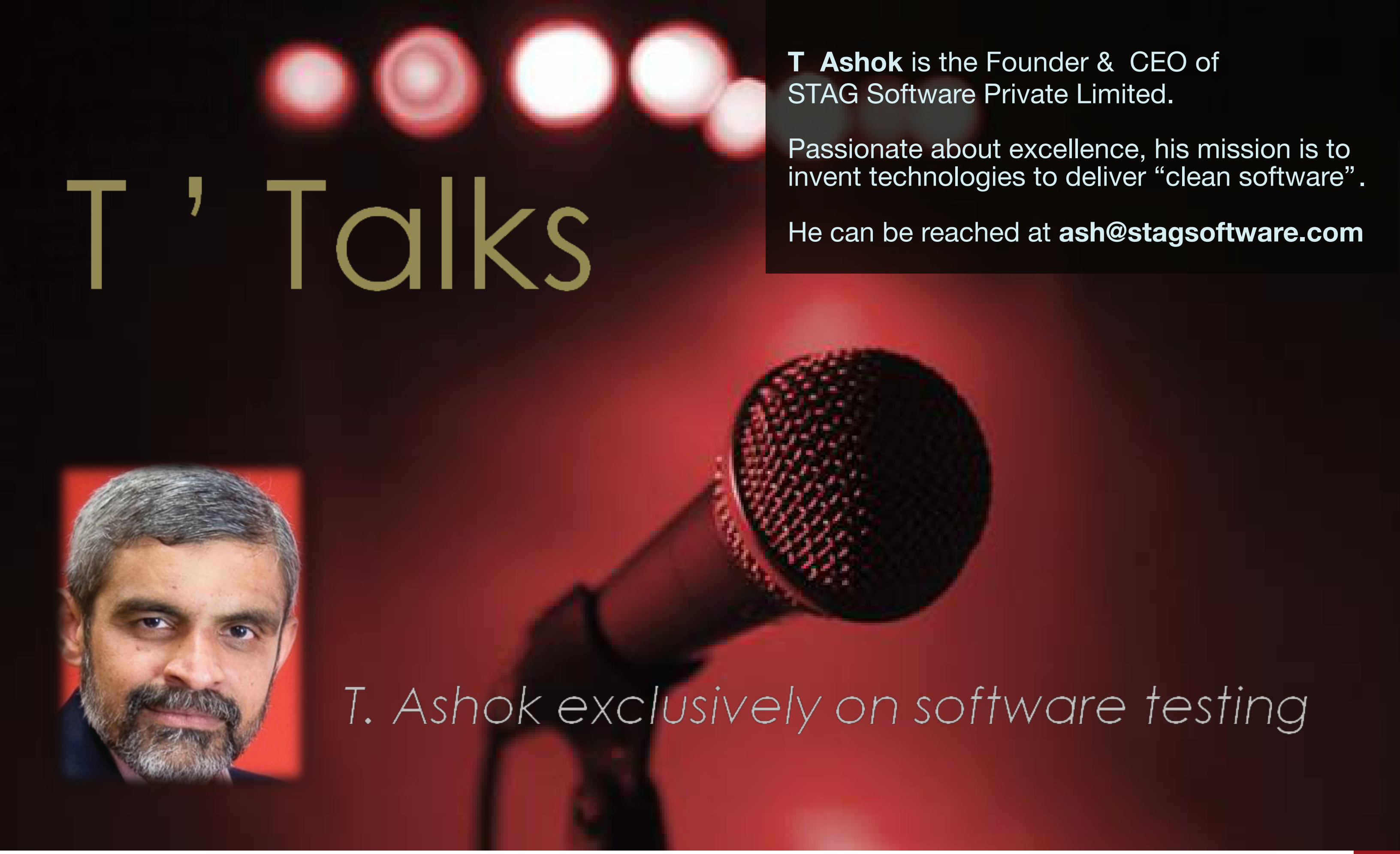
Share this issue with your friends and colleagues!

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**

Be in a Flow. Test Brilliantly.

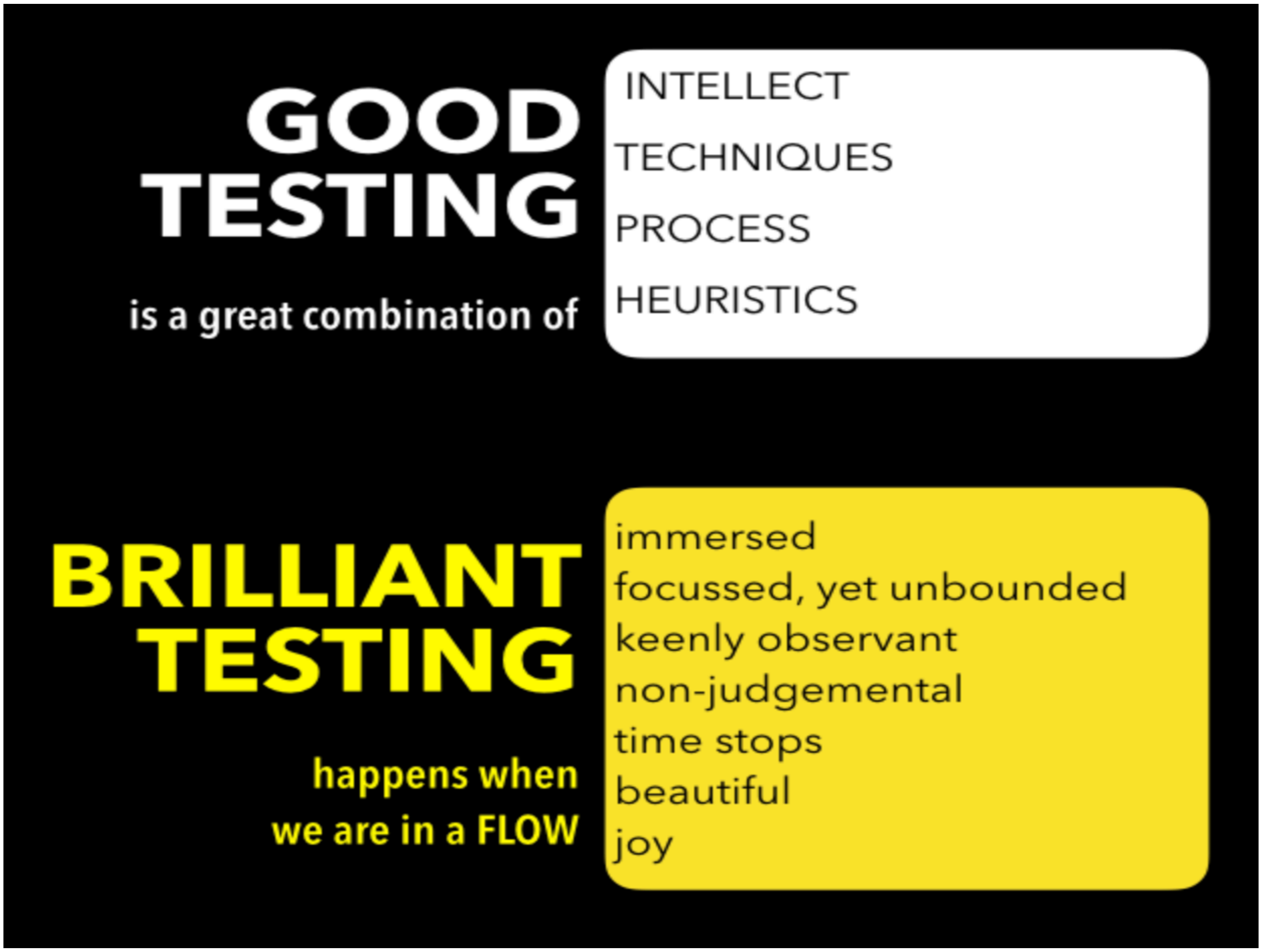
Summary

Good testing is a great combination of intellect, techniques, process and heuristics. What does it take to do brilliant testing ? It requires one to be immersed in the act, be focussed yet unbounded, be keenly observant but non-judgemental, with outcome that are beautiful, the activity so joyful that time stops. A state of flow. What does it take to get there?

Introduction

As engineers we use good techniques, tools, process and intellect to do things well. So how can we do far better, that is brilliantly? Having possibly exhausted all things "external", in terms of tools, techniques, process and intellect, it is time we considered the huge "internal" potential that we all possess.

It is about going beyond the intellectual mind, deeper into the sub-conscious to harness the infinite potential. A good start for this would be get into a state of 'flow'.



GOOD TESTING

is a great combination of

- INTELLECT
- TECHNIQUES
- PROCESS
- HEURISTICS

BRILLIANT TESTING

happens when we are in a FLOW

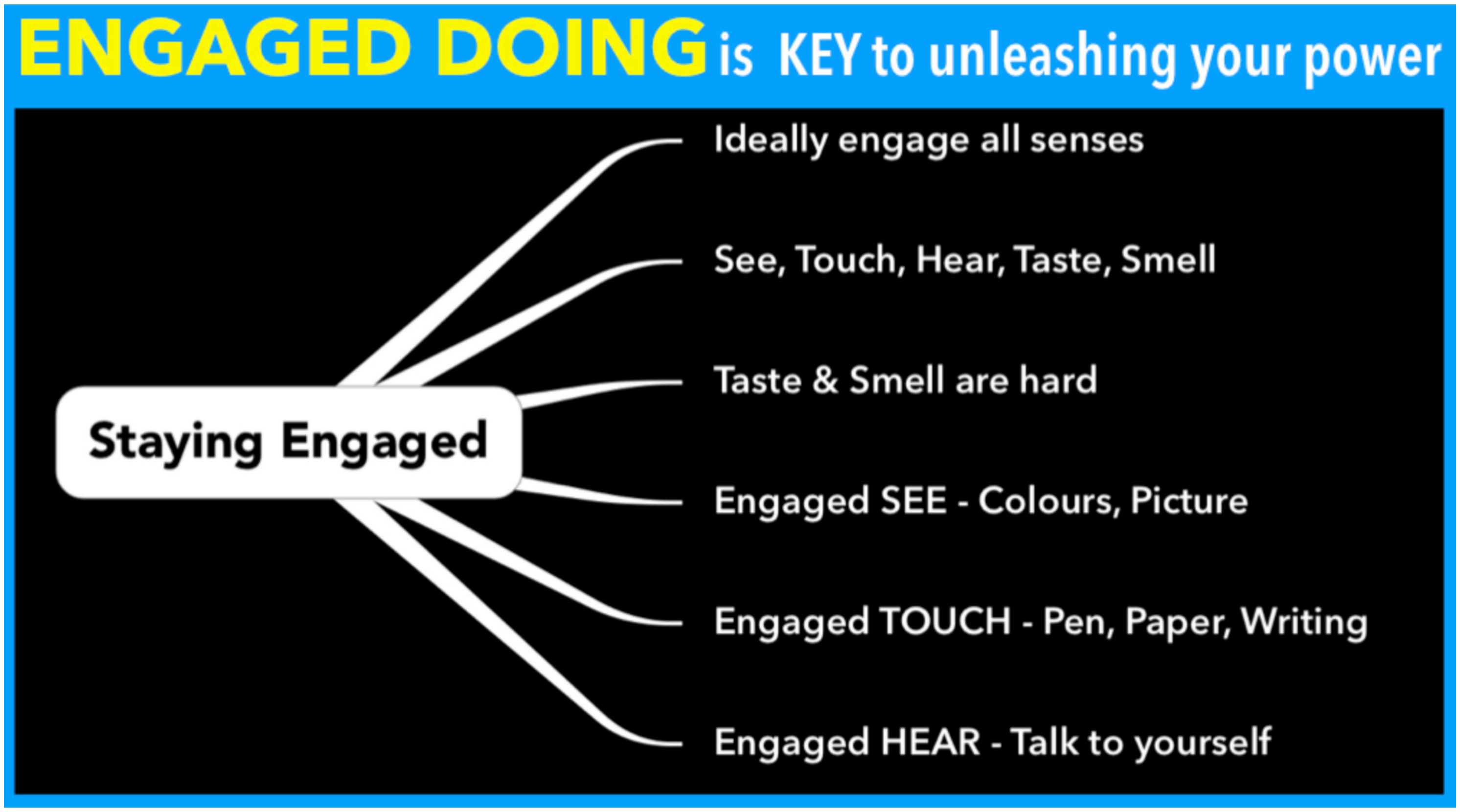
- immersed
- focussed, yet unbounded
- keenly observant
- non-judgemental
- time stops
- beautiful
- joy

So, what is FLOW?

Flow is the state when you are immersed in what you are doing, where you are totally mindful. It is when all energies are harnessed fluidly to do brilliant work without tiring or trying hard, becoming an observer, fine tuning actions with extreme agility, when time seems frozen. It is when you accomplish a lot, doing work effortlessly and experiencing absolute joy.

So how can we get into the state of flow?

When multiple sensory excitations converge harmoniously, there is a good chance of entering that state of flow. What does that mean? It is engaging the various senses well – colours, picture, visual test, mind maps for the eyes, using pen/pencil, paper for the touch, and maybe background music or talking to oneself for ‘engaged hearing’. Note the keyword is ‘harmonious’.


ENGAGED DOING is KEY to unleashing your power
Staying Engaged
Ideally engage all senses
See, Touch, Hear, Taste, Smell
Taste & Smell are hard
Engaged SEE - Colours, Picture
Engaged TOUCH - Pen, Paper, Writing
Engaged HEAR - Talk to yourself

When we stimulate our creative side with interesting visuals, tactile and possibly sound, it activates us to get into an ‘engaged state, a state of mindfulness.

Exploit visual thinking by using visual text , sketch maps, mind maps to : (1) enable deeper understand the system under test (2) to sketch test strategy (3) jot down test ideas (4) note down observations, questions, suggestions (5) list scenarios (6) record issues.

Exploit the tactile sense by using pen/pencil on paper or finger/stylus on tablet instead of keyboard. The objective is to be write/draw freely rather than be constrained by the keyboard.

If you want to engage your auditory sense, quietly talking to yourself, melodious quiet whistling or if you are music person, then a suitable background song played could enable you to get into the flow.

To ensure that we can perform in the state of peak performance being in a FLOW, it is imperative that we do testing in short sessions. A session may be anywhere between 45-90 minutes. The key is to stay focussed, setup an objective at the start of session and then engage as stated earlier to get into a state of FLOW.

How does this help?

When we are fully engaged, in a state of FLOW, it is no more about just left brain centred logical/deductive thinking, or the creative right. It is an expansive multi dimensional thinking brought about by the harmonious stimuli enabling one to become a keen observer and absorb deeply and rapidly. This is when we exploit the infinite power of what we possess, delving into the sub-conscious which is much larger that conscious mind. Interesting questions pop up, ideas germinate rapidly, actions are done quickly, smallest observations captured resulting in brilliant effective testing.

In closing…

Test automation allows us to do more, and machine intelligence to do better. It is now necessary for us to delve deeper so that we complement the machine rather than compete, enabling us to be super efficient and effective by being smarter. Get into the state of flow to harness the power of sub-conscious to do brilliant testing.

Mindful Testing – because testing should not be done on autopilot

I was asked to do a talk on Mindful testing for Swiss Testing Day.

I liked the idea. I have been practicing Mindful Testing personally (and mostly quietly) for the last couple of years, and this gave me an excuse to look back at the ways I have been improving it with my experience.

For reference, the first time I wrote about this topic was a couple of years ago, you can read the original here the original Mindful Testing post.

Mindfulness is obviously not related to testing

I have been practicing mindfulness for 3 or 4 years now. It started after I read about the practice and the apps circling around the Internet. I kept with it because it helps me to focus whenever my ADD tries to take over.

Today I do mindfulness exercises almost every day, and sometimes two or three of times a day. I have a couple of apps to help me with it, if you want recommendations leave me a comment and I will send you the names and why I like them.

In extremely simple terms, mindfulness is the practice of Focusing one's complete awareness on the present moment.

If it sounds simple and if you think you do that all the time, then you have not thought about it all the way through.

Focusing one's complete awareness means clearing your head from everything and anything else.

- You do not think about the groceries you need to buy, or the trip you plan to take, or the problems with your kids, or the fight you had with your partner earlier today.

- It means that you are only and completely interested in what you are doing now. Learning and capturing every detail. Imagine the feeling of sitting at the edge of your sit in anticipation, just like you did when watching the last Avengers film in the theater.

- It is being so immerse that time passes without you noticing, because time is less important when you are 100% interested in what you are doing right now.

It is not easy to be 100% mindful, but you also do not need to 100% mindful to gain all the benefits. Sometimes it is enough to be aware of the fact that our mind wanders around, in order to start getting value out of this practice.

Mindful Testing

As I wrote earlier, I have been exploring the practice of Mindful Testing for a couple of years, expanding on the practical aspects of it.

Today, I believe that the main ideas about this practice can be synthesised in the following 7 points:

1. Focus totally on your current testing objectives
2. Test only what needs to be tested now
3. Test as early as possible
4. Provide the important feedback first
5. Maintain live communications
6. Plan to have feedback sessions
7. Reduce interruptions

Let’s review them quickly.

One – Focus totally on your current testing objectives

Many times we start testing based on a vague sense of the feature that stands in front of us.

This is especially true if you are an experienced tester, working on the same project for some months or even years. It happens as we gain a fail sense of self-trust and when we start falling into a repetitive routine causing us to loose focus.

NO – Stop!

You need to understand not only what the feature is, but also what value it is supposed to bring to the user.

More importantly, you need to have a clear objective for what aspect of this feature you want to test in the present session.

If the answer is “I am going to test the complete feature” then make sure you break this down into smaller objectives you can focus on, and more importantly that you can verify you covered them.

Two – Test only what needs to be tested now

Do not try to test too much in one session. Get a clear understanding of what you want to test during the present session, and focus on this point only.

Sure, as you are testing the specific feature you will get ideas of other stuff that needs testing. Don’t fall into the trap of making a quick detour to explore this new path, write it down on your notebook and make it another objective for a separate session.

Three – Test as early as possible

start testing as early as possibleThe objective of testing is to provide feedback on the feature (and if possible on the process). Good feedback will help improve the feature, but you also need to make sure to provide this feedback when there is still time to make the improvement.

Many times you will find important things, only to be told that there is no time to do anything about it. This is why you should test functionality as early as you can.

Linking this with the previous point… If you break your testing into small chunks, then you will do good by scheduling these sessions as early as possible.

Four – Provide the important feedback first

Plan your tests within the session in a way that will seek the important feedback first, and then move to the other areas of interest.

If the main question of the team is around response time, then start there! If they are looking for a general validation, do a quick overview before you move into any specific areas. You get the point…

The important thing is to start from the most important areas, so that you can provide this information early in the process, and as part of the process.

Five – Maintain live communications

The previous point was about providing the important feedback first. This one is about providing it as you find it, instead of waiting to communicate any important news until the end of your session.

Small things can wait, but big things need to be know ASAP. It is OK if this will interrupt your session.

This is especially true in Agile and DevOps organizations, where release cycles can be measured in days (and sometimes in hours).

Six – Plan to have feedback sessions

feedback sessionMany times you will not have a big ticket item to communicate. No smoking guns, or "Stop the Presses" bugs, but you will have good feedback that can't always be translated into concrete bugs or test results.

Make sure you have feedback sessions with your development team planned regularly, to be able to explain what you are doing and finding, and get their inputs on these points.

I even recommend setting up debriefing sessions once you are done testing the User Story, to make sure you covered all the areas and that there are no unusual findings that may raise a flag to someone else in your team.

Seven – Reduce interruptions

This may be also the first point in the list, but for some reason it made more sense to me to place it at the end.

Make sure you can test without being interrupted unnecessarily. These interruptions may come from your team, but nowadays it may come from a whole set of different places (with our current Working from Home lives).

It is very hard to reach a state of concentration that can help you carry a Mindful Testing Session. Once you reach it, you will want to maintain it for at least 45 to 90 min straight.

Be Present & Work on the Highest Value Testing Task

I think this sums up the whole principle in 2 main ideas:

- Be Present means to be mindful. To put all your attention on what you are trying to achieve now, and not focus on 3 or 7 things at the same time.

- Highest Value Testing Task is the place where your work can add the most to the objective of your team. Helping everyone to release the product as quickly as we can, and with the target level of Quality and User Satisfaction.

All the rest is the icing on the cake.

Joel Montvelisky 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 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.

Association for Software Testing (AST) comments on Online Exams

- AST news

AST was recently asked for an opinion about online examinations for certification bodies, specifically online Bar Exams.

AST's conclusion is that to properly serve their technical and social purposes, online examinations must be administered in a fair and unbiased manner. They should not be difficult to undergo for people of modest means, and their administration should not create additional stress on already stressed examinees due to implementation or technology. Any gatekeeping these exams represent must be based strictly on merit.

No candidate should fail an exam other than on merit. Failure grades because of equipment barriers, power outages, the widely known unreliability of the Internet, or inherent racial and class biases in algorithms and examination methodologies are unfair to both the candidates and society as a whole.

If an examination as currently planned can't meet these requirements for all examinees, then it should not proceed. Even during COVID-19, there are methods to administer in-person exams. Decisions about how to administer exams should be made to best accommodate examiners AND examinees.

Read the Full Press Release here

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:

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 – Wasim Khan on Unsplash

Editorial Board:

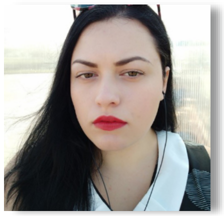
Dr.Meeta Prakash (India)

Dirk Meißner (Germany)

Klára Jánová (Czech Republic)

Dr. Meeta Prakash

Dirk Meißner


Klára Jánová

Online Collaboration:

Shweta Daiv (Germany)

Shweta

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

Give Feedback