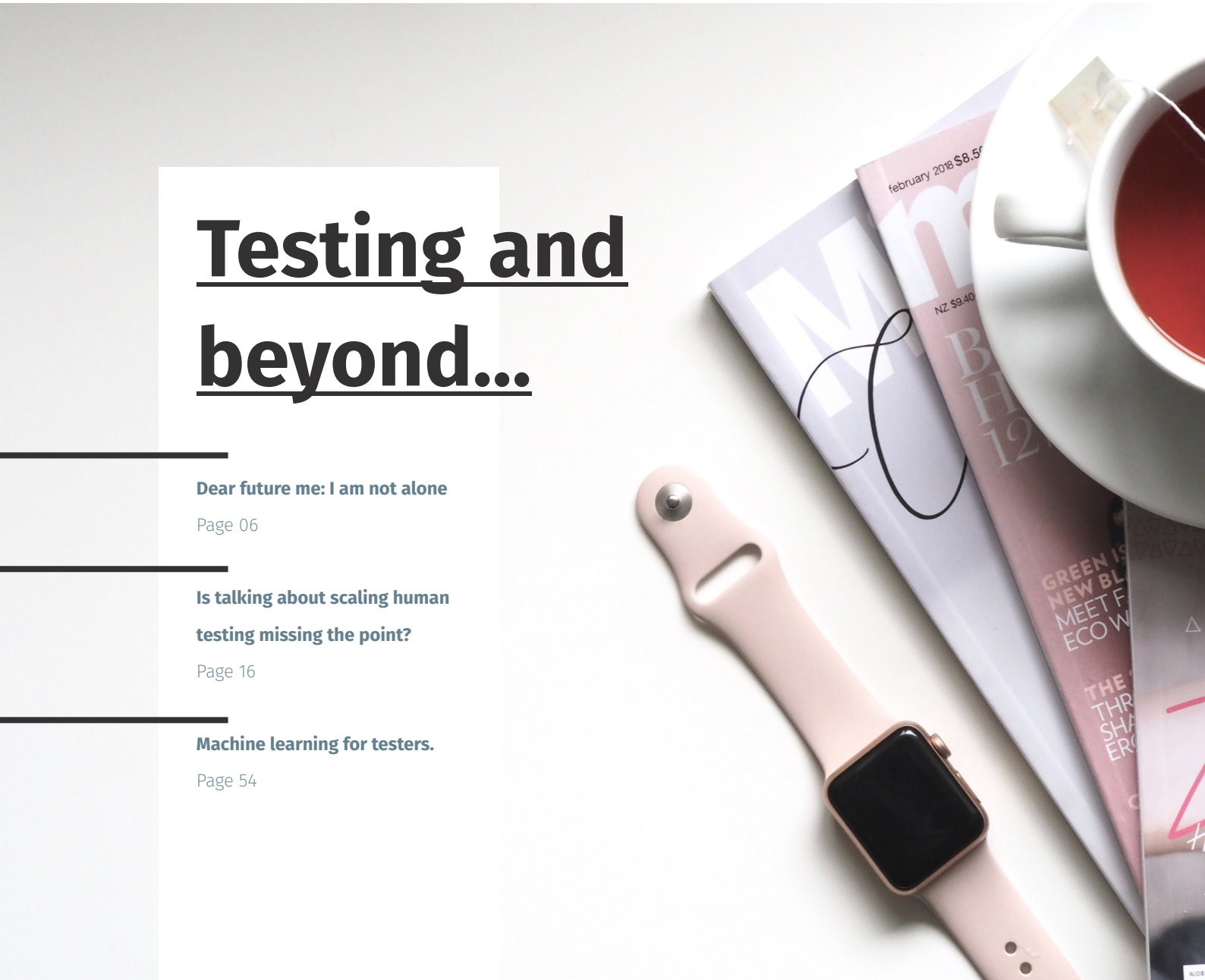
WAKING TESTERS UP SINCE 2011

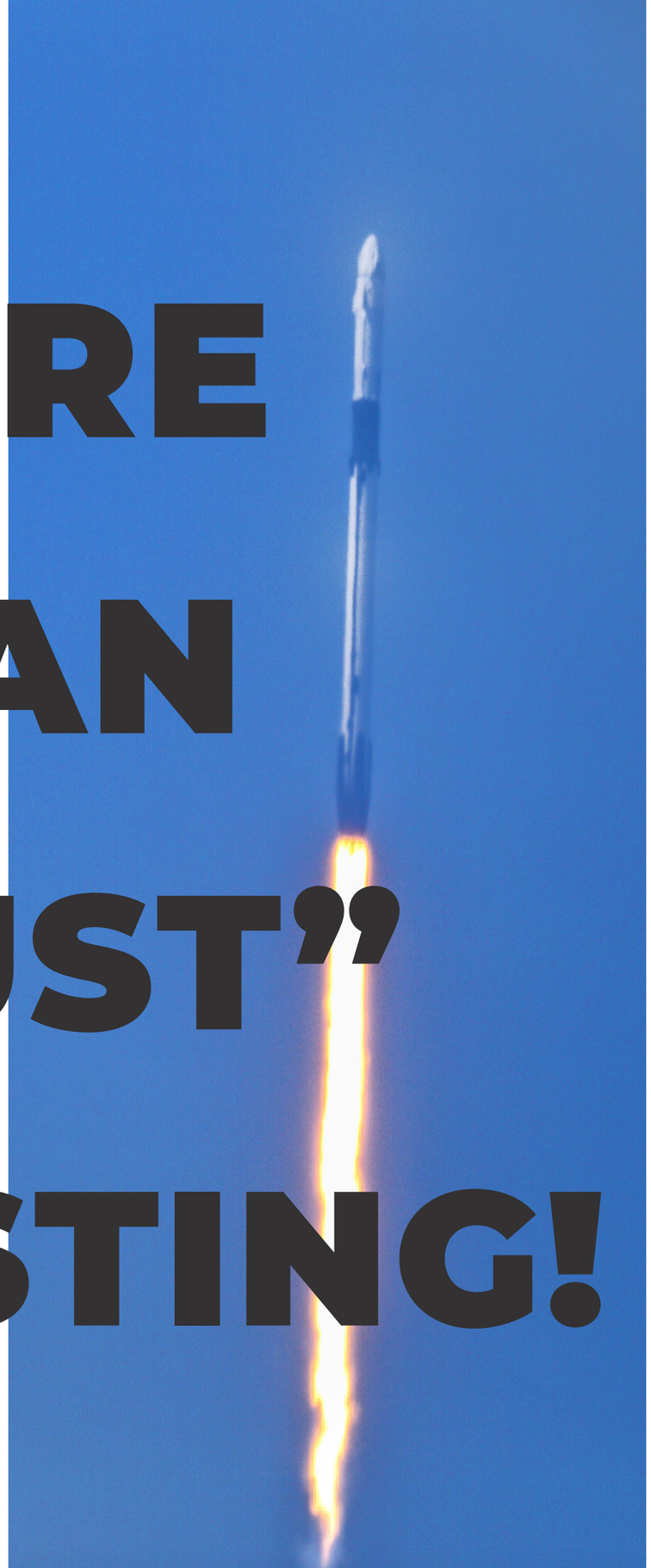
ISSUE #02/2021

TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS



Testing and beyond...



MORE THAN “JUST” TESTING!

TEA-TIME WITH TESTERS

TEA-TIME WITH TESTERS

A NEXT GENERATION MAGAZINE

FULL OF CONTENT AND TIPS FOR TESTERS

It is *just a tester* thing!

I still remember that day when one of the senior backend engineer I was working with proclaimed this in one of the retrospectives, "After working with Lalit, I am convinced for the first time that software tester is an important part of the team. Until now I never felt testers were much needed."

It was indeed flattering to hear that but as I thought deeper about it later, I wondered why do I never hear programmers making such patronizing statements about other job functions in software teams? Like, "Good job front end dev. Tell your parents they can be proud of you." or "This PO really knows his stuff. I am no longer worried about the future of our product."

Does absence of such patronizing comments about other job functions mean only superheroes get to do it all? Of course not. I could probably write a book on "How to be a smart programmer so you build products that people like to use." And I am willing to bet, it can be the best-selling non-fiction in non-technical, no-code category. I have only thirteen years of experience in the software field and by now I have already met enough programmers who were only good for writing software which nobody wanted to use. And some of them should have been prohibited from even coming close to the computers.

Does that mean we generalize those experiences and put certain job-roles in a fixed box-of-perceptions? No! Generalization is bad and typecasting of negative sort is horrible.

In fact, I would rather talk about great programmers that I met on my journey. I learned a lot from them. They helped me see things beyond the obvious which in turn helped me become better at testing. I honestly can not imagine claiming mastery over certain skills if I would not meet those wonderful colleagues.

The whole point of writing this all is that, in last six months I met at least three testers who have been let down, discouraged and profiled by their Dev colleagues. Not only that but in the State of Testing survey, every single year we notice testers reporting about the lack of respect and value they get in the organization. And this is not okay. In fact it will only disadvantage the project team, not only the tester.

Yes, I acknowledge that people can have poor perception about software testing as a profession for the experiences they made with testers. And I also acknowledge that fair part of that typical "checking" work can be easily automated. But I want to mention that testing is not just about automating the asserts or clicking here and there and saying it broke.

There are many testers who have been contributing to software projects way beyond "just testing". As a matter of fact, I know great programmers who switched to testing because they found testing to be more challenging.

I see no reason to explain how hard good testing can be and that it is not something anybody can "just do" it. Because if you need to be explained it all, you have been probably living in a cave and I urge you to come out, at least now. Or do me a favor and read what a legendary computer scientist had to say about software testing.

In one of his interviews Jerry Weinberg said,

"Testing is harder than developing. If you want to have good testing you need to put your best people in testing. Your smartest people and maybe a little different type of person, someone as we said who listens better, talks better, so it's a very exceptional kind of person that makes a great tester.

If you believe in that kind of thing then you should reward them better than you reward the developers. Instead, I go around and I find that people habitually pay their testers less than they pay their developers. That's the number one thing that is not understood."

Mind you, Jerry said that about "testing" and it is not about testers being more important or superior than programmers. If you enjoy doing difficult things, challenging things, be our guest and join the club! The global testing community is very welcoming, supportive and you will never feel like an outsider, that's for sure.

If you are one of those people who think very low of testing and testers, I forgive your ignorance but let me humbly request you, treat the testers (and everyone else) in your team well. Teach them if you feel they don't know something and also learn from them what you might not know. And if nothing else, at least be kind and let them do their job with focus. Because the most important part of tester's work happens far away from the IDE, deep inside their brains. And there is no AI driven co-pilot yet which auto-completes tester's next move when they test things as they think.

But no complaints, it is *just a tester* thing.!

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

Connect on Twitter @Lalitbhamare or on LinkedIn and Xing.

DEAR FUTURE ME: I AM NOT ALONE

Dear future me.

I'm writing this while being very tired. I am still spending my time and energy on this deliberately, hoping to remind you of a very stressful time I'm still recovering from: the last six months.

Over the past couple of years, I've started to think a lot more about my energy levels and capacity. Especially when the pandemic became obvious beginning of 2020 and life changed, I felt I needed to cut down on what I do and focus on a few things at a time, working at a sustainable pace. Thinking I had achieved that, I promised myself never to get back into a situation with high stress levels over a long period of time, feeling completely overwhelmed. Little did I know, I did not have everything in hand to prevent what happened since beginning of the year. So here's a reminder to myself and anyone who relates to this situation: if you encounter circumstances again where everything ends up on your desk, be reminded of what happened this time, what consequences it had and which strategies helped to get through it.

So what happened? In short, my work load exploded, I went with it and the energy spent left me depleted.

Team size exploded.

End of last year, my product team consisted of six full-time employees and a working student. With only three full-time developers we were looking for more people to join. Then one of the three decided to leave and the situation became more urgent to solve. As a blessing in disguise, four developers moved internally and one developer came from outside the company to join our team - all within one month. Yay, problem solved, right? Well, this meant we suddenly were eleven full-time employees plus one working student. I guess most people who have worked in cross-functional product teams can understand what that meant.

Our communication pathways multiplied with every new person on the team. With me as the dedicated tester on the team still being involved in all stories, the team mostly working in solo mode at that time, and people starting new stories when waiting for feedback, meant that I had at least twelve stories at the same time on my desk. Imagine the context switching effort and waste coming along with that alone. Feedback loops slowed down immensely and our cycle time increased. Everything took long. Working modes that were okay-ish before did not work at all anymore.

Onboarding effort multiplied.

So overall, in January and February five new developers joined nearly at the same time. With me being one of two persons having been the longest in the company, in the team and on the product, plus having a unique holistic view on everything that's involved with developing it, this meant a lot of the onboarding and knowledge sharing effort ended up on my desk. While I really enjoy onboarding new people - and these five were lovely people to join our team - this really took a toll on me. Yet still, we need to set people up for success and give them the knowledge they need to have impact themselves. I simply can't leave my teammates hanging.

Old conflicts reached the melting point.

My team from last year was together for quite a long time and it started to dissolve more and more. We had wonderful times together, and also times we did not manage well. There's a lot to learn from that alone, yet the sad fact is that a long and slowly growing conflict took a toll on each and every one who had been part of that old team constellation. In the end, the two other former developers decided to leave the company for new opportunities as well. All these ups and downs took a lot of emotional energy from everyone of us. It took up a lot of cognitive capacity as well and made any interaction way harder than it had to be.

Building a new team, remotely.

Since beginning of May, we're now finally our new team constellation and starting to shape this team to the one we want to be on, where everyone is welcome, included, safe to speak their mind, encouraged to experiment and collaborate and learn together and everything. Exciting times, yet we need to put in lots of effort. Also, this is the first team for all of us, where nearly everyone on the team only met each other remotely. We need to learn how to grant ourselves social time, get to know each other, evolve our culture, and more - all virtually. We are distributed across four locations, so the remote setup is amazing in leveling the playing field and providing the same access for everyone. I'm curious where this journey leads us, and already very happy to be part of this new team.

Upskilling people to enable them to take over activities that usually ended up with me.

No matter how often I reached a point where testing was indeed a whole team activity, with the former team constellation it ended up again mostly with me. Especially exploratory testing or testing for any kind of other quality aspect than the core functionality. The same with operations and infrastructure tasks, responding to alerts, user support, writing release news, and so on. Scheduling and facilitating meetings. Cross-team communication. All kinds of glue work to keep the balls from falling to the ground. I strongly believe in the whole team approach and creating a base of knowledge for everyone..

LISI HOCKE

–

Having graduated in sinology, Lisi fell into agile and testing in 2009 and has been infected with the agile bug ever since. She's especially passionate about the whole-team approach to testing and quality as well as the continuous learning mindset behind it. Building great products which deliver value together with great people is what motivates her and keeps her going. She received a lot from the community; now she's giving back by sharing her stories and experience. She tweets as @lisihocke and blogs at www.lisihocke.com.

In her free time you can either find her in the gym running after a volleyball, having a good time with her friends or delving into games and stories of any kind.

No need to become the expert in one area, yet we should be able to help each other out, reduce bottlenecks and waiting time, unblock each other, being able to go on sick leave and also vacation without things piling up for us in the meantime or worrying they won't get done. So with new people on board, this task could only be done by me, naturally. Super thankful that my new teammates are very open and supportive and not hesitating to see beyond their own nose. They stepped up and took over responsibility even if things were outside their usual comfort zone.

Taking up product work and sharing its responsibility in the team.

Our product owner had great news: his family got a new member! I really appreciate him going on a long parental leave and also preparing the team for it. He is still working one day per week, yet we all agreed to spread the product responsibility across the team and see that we all carry a bit of the load so it's not too much for anyone. Still, with me being the one longest on the team and product, I ended up as natural contact person for most people outside the team, even though it was communicated differently. So many requests coming in! While I am only seeing a fraction of it, I'm in awe of product owner work.

Also, we all in the team are now learning how to tackle user experience in a better fashion, how to spread UX knowledge in the team and how to support our researcher better with his work, and how to fill the gap of other UX roles like design or writing. A lot more to learn on this path!

Our team's domain got lots of fresh people.

Since beginning of the year, a lot more new people joined our domain, including two new persons on the domain leadership level. New people bringing new energy and lots of ideas! Naturally, onboarding needed to be done on domain level as well. Again, as being one of the persons longest in the domain and also having the "Principal" seniority level, sharing a lot of knowledge and experience ended up on my desk without the possibility of delegating this work. Giving feedback to new initiatives, doing my share helping to drive them forward, new sync meetings, participating in domain workshops, and more.

Seeds planted over the last years in my colleagues' minds finally began to sprout.

I really don't know what exactly happened, yet since beginning of the year a lot of people reached out to me. Suddenly they were taking me up on my continuous offers to give workshops or talks, to listen and give advice where wanted, to support their own experiments and initiatives, and more. I love seeing people this energized and acting on their ideas and I'm happy to support. "People first" as a principle does not only apply to my own team, so I didn't turn them down. Yet all these requests added up for me.

Mentoring, coaching and an accountability partnership.

Over the last years, I had about one mentee at a time, sometimes one or two more without the formal relationship. This year I got a new mentee to nudge further on her journey - which is great! Also, this part of the job comes with the seniority as well: growing more senior people. With my mentee from last year we had agreed to continue the relationship as accountability partners - on the topic of saying "no" (imagine).

In addition, I took on my very first coachee as an experiment for both our growth as well. Each and every of these relationships have clear boundaries and don't take up much time - yet overall they do add quite a bit. Yet again: people first.

Co-creating and running a series of six leadership workshops.

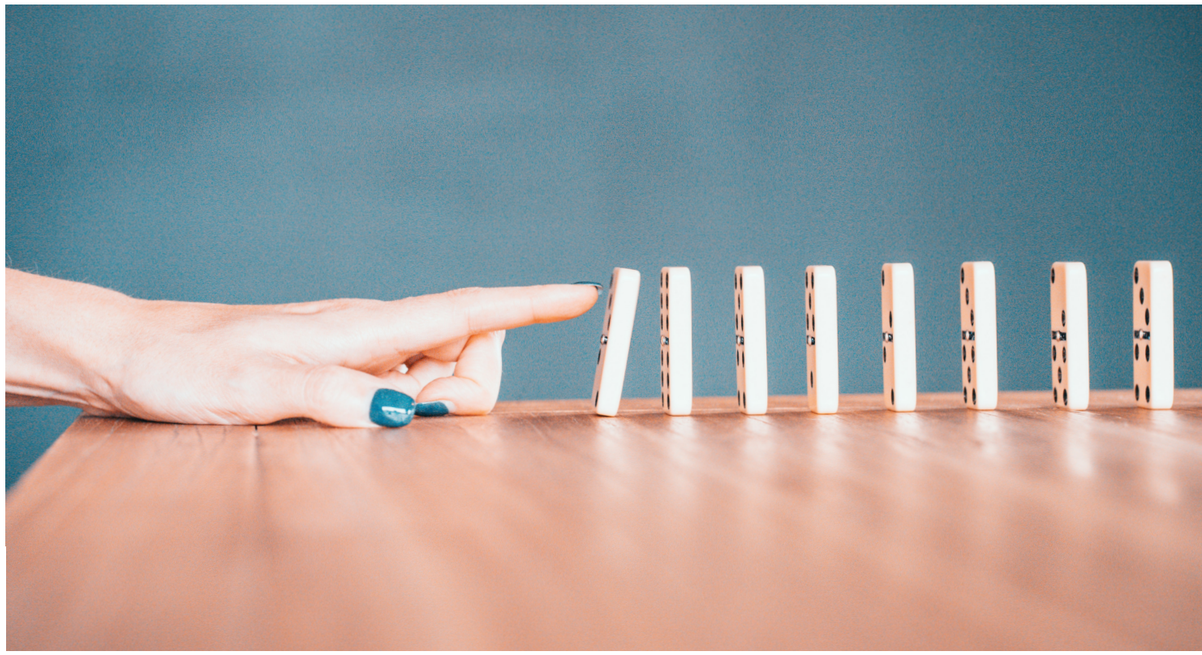
I'm in a hybrid role as a principal engineer who's embedded on a cross-functional team. This means that I spend part of my capacity on cross-team initiatives to drive change on a different impact level. Over the last years I've found a rough rule that worked nicely for me: one third of my time I spend on everyday work on my team to evolve our product, one third for thinking ahead and driving innovation within said team and product, one third for initiatives outside my team, usually on a global scale. Worked pretty well last year, helping me to focus on less work in progress and also keep a sustainable pace. For this year, the main initiative I chose to do outside my team was to pair facilitate a series of six leadership workshops together with our coach Shiva Krishnan. I had participated in these workshops in the previous year and found their content to be very relevant and valuable to spread further. My experiment was to build quality on yet another level here, setting the base line and culture for good things to emerge. According to my experience, driving specific testing and quality initiatives mostly failed when they clashed with the existing culture.

This year, I wanted to work on the mindset part from yet another angle and also build awareness on diversity, equity and inclusion topics as part of these leadership workshops. Well - long story short, Shiva and I ended up reworking each and every one of the six workshops, pulling them to a higher level and building yet a better framework by doing so. The magic of pairing! Any one of us alone would not even have imagined the end result. Together we put in a lot of effort yet also had a way better outcome in the end. I don't regret any moment working on these workshops - even though it was way more work than anticipated, and we had put ourselves up to keep hard absolute dates with each workshop.

There might have been more things that I've forgotten to list here. Yet you can already see that there's no way that all of this would ever fit into a 40 hours work week when everything needs to take place at the same time and the goal is to achieve all that in five to six months. Some colleagues reached out with lots of requests, and when I explained my situation they shared they can really relate and things can wait; while giving me yet three more tasks. Sigh. My task list of additional "small things" to work on grew to 40 tasks that all would take at least 15 minutes to 1 hour and I just didn't know when to ever do them, while me being the only person who could work at them. I felt I was set up for failure and adding to that myself. Believe me, on each thing landing on my desk I pushed back way harder than at any time in my career before, challenging every bit of work if it really needed to be done, be done this way, be done by me, be done by me alone, be done by time x, be needing my attendance - and yet way too many things ended up with me anyway.

Way too often I was in back to back calls the whole day (with follow-up tasks coming from them of course), while in the meantime I received so many chat messages with way more tasks waiting for me. Yes I can pull through this, and yet it will drain my energy levels completely. Way too often I felt I'm playing a game of "Whac-A-Mole" without being ever able to win - at least not on my own.

> *"I am not alone". This reframing allowed me to break out of my solo overwork behavior (that helped no one), reach out to others earlier, and accepting their help better.*



The consequences of all this?

Being the bottleneck

It's been a long time since I've enjoyed being the bottleneck. A long time since I said goodbye to that mindset and never wanted to look back. From time to time I still ended up being the bottleneck (e.g. being the only one knowing how to do a certain thing), yet I usually took this as an indicator to change the situation. Having bottlenecks and knowledge silos is neither resilient for the system nor fun for the individuals. It just increases waiting times, triggers unhelpful behavior (like taking on a new task while waiting) and more. Believe me, I'm totally happy with not being the bottleneck and I'm way better able to contribute then.

My cognitive load exploded.

With all the points above I had to keep way too many things in mind. Way too many context switches. Way too much balancing and juggling. If one thing dropped, way too often it cascaded into other things. Once more I realized what 100% (or more) utilization really means: a catastrophe. So many times I simply could not think anymore; you know, deeply think, really think something through. Staying too superficial just to cope with the situation led me to make less fortunate decisions. I lost focus on what's actually important and what can wait. Trying to make all these switches slowed my thinking to a halt. And yet I tried to pull through instead of taking breaks.

The more stressed I got, the more I fell back to bad habits.

Like solo work, trying to solve everything myself (so it's "faster" and just "done") without pulling others in (so more people could help out in the future). Especially in the first months this was a missed chance. Another bad habit: being unable to say no. I know I'm a people pleaser, a learned behavior from childhood years, so I'm aware I need to work on this. I do have an accountability partnership specifically for that reason, so we can keep each other accountable on keeping an eye on our load. Practicing saying "no" to opportunities, or "not yet". Trying to delegate things, sponsor other people instead of taking on more things that are not in our focus, outside the area where we want to make the biggest impact. End of last year this worked well, yet this year I learned that the higher my stress levels, the lower my boundaries to accept new work load. This is something to keep in close check.

Getting angry with myself and the example I set.

Angry with myself that I let this happen to me (which was not helpful at all but just added another layer of energy spent and capacity used up). Angry with myself that I felt the need to pull through all this and cope with the situation - despite me never wanting to take on as much anymore. The contradiction alone. Also, I was well aware being in a position of leadership, and leading by example taking on that much and not being able to delegate or take breaks, this is setting a really bad example I did not want for our culture. Working late or on weekends? I was always the first one telling my teammates that this is not the way to go (unless where really needed to balance with life), and that it's not good to set this bar for the rest of the team who might start thinking it's expected from them. All this while I was doing exactly that, just trying to hide this from my team. While I am also the one who advocates for transparency! Oh my.

My body alerted me of the elevated stress levels.

I mean, more than usual. I started developing new physical responses to stress to a level that my body actually made me notice - a first timer for me. This was something I could not push away yet really got me thinking. It made me realize I really need to stop this. Get out of this situation as soon as possible.

Feeling overwhelmed, helpless, anxious.

Way too many times I broke into tears or screamed in frustration or wanted to throw something. Anything. All this took up again time and energy that I felt I needed to spend to resolve this work situation and get out of it as soon as I can - while still being aware that this would mean months. Several colleagues and friends repeatedly reached out to me sharing their concerns that I'm stretched too thin and that I really need to get things off my table. Yes, I know. I don't know how though, and I'm sad it shows. I really appreciate you all for reaching out - I still couldn't see any other way out than pulling through.

Of course that's not everything. There's more to life than work.

We're still living in a pandemic.

I learned that I am totally happy with continuous change at work and drive improvements actively, yet especially in private settings I need a constant to hold on to and give structure, like my schedule. Any change in my daily routine takes a long time to establish new automatisms around it - costing lots of energy. The constant change of rules what's allowed and what not in the pandemic really drained me. Rather give me a more restricted set of rules and keep it for longer, I can live with that way better than having it change every few days or weeks.

Family and friends having a hard time.

People were needing me in many different ways. We went through lots of ups and downs. Having to solve a lot of things remotely with people who are not used to work that way is a challenge in itself. Solving problems I never had to solve before while conveying that knowledge at the same time took a lot of energy from my side.

My personal projects and endeavors came to a full stop.

At first I tried to make time for them nonetheless, then I realized I had to stop whatever I could. The last years I took on a lot and I'm aware of that. At the same time, my personal projects served as a sort of boundary for work, any hours outside working hours were simply reserved for other things already which helped me to keep these boundaries. They also gave me a lot of energy and allowed me to learn a lot of different things. Yet they had to go - during the past months I simply didn't know anymore how to ever manage that load otherwise. Slowly, I am now taking up a bit of public speaking again, yet mostly around existing sessions with the least effort possible.

Crossing my own boundaries for self-care way too many times.

I failed to keep up my personal goals to do things only for myself (like games, sports or reading). Feeling guilty here as well and trying not to.

Way too many private messages and communication.

While I heard many people struggling with the reduced connections during the pandemic, I so often wished for a lot less. I am usually receiving around 40 to 50 private emails every day, with around 10 I really need to respond to. Usually I'd be fine with that, I learned to get them out of the way quickly whenever I can. Also I'm happy when people are reaching out! Yet in above situation over the last half year these messages were way too many! Hence, seeing any kind of new email or social media notification immediately made me cringe and increased my stress levels.

If I've ever felt burned out or getting real close to it, it was over the last months. A really scary place to be that I wanted to get out of as soon as possible. So I did work on finding my own way out. Oh yeah, this came on top - yet I felt this would be my saver. I tried a lot of things, yet here are strategies that indeed helped me on the long run in my specific context and situation. They mostly resolve around setting boundaries and spreading the load by enabling people to help out.

Take breaks. Really

I need to take breaks even if I feel I don't have the time for them. Sometimes just getting away from the table helps, like when making myself a new tea even though I didn't need one yet.

Take time to reflect and think.

Sit down and reflect on what worked in times that resembled the current one, like back when I was working on big teams. The past months I continued reflecting and taking note of my thoughts in a journal. In hindsight this helped me a lot to unload myself of emotions or thoughts as well as to clear up my thinking. I had discovered over the last years that writing helps my thinking, so journaling is a great way for me to get my thoughts straight. If you wonder, I mostly just take bullet points, as little as needed, and sometimes thoughts are just flowing and filling up the white space.

Reduce the load in progress.

Making principles like "stop starting, start finishing" very explicit again so people start looking around if they could help out someone else before starting something new, or just not start something new not to increase our work in progress even further. Cut down what you're working on, and then cut down even more.

Remember you are not alone.

As shared, I'm a people pleaser and this often drives my behavior. I am using an "allower message" as antidote, so whenever I perceive being a failure (meaning I cannot please all people), I remind myself of a specific message to ground myself again. "Please yourself first" worked well for me last year. This year I changed it to "I am not alone". This reframing allowed me to break out of my solo overwork behavior (that helped no one), reach out to others earlier, and accepting their help better.

Refrain from solo work.

My current team is not yet familiar with ensembling, yet open for pairing. So I paired a LOT. To the point where I committed to testing only together with other people - if we didn't find time together, well this thing did not get done. Giving myself permission to use that time to finish other things instead, and not to use off hours to test something. I deliberately went slow here so we all could go fast in the future, together. When things are valuable to us, we need to own them together. Sharing knowledge is one thing, sharing activities is key.

Focus on unblocking people.

Teach people how to help themselves and the team and then let them do it. Let them be responsible of follow-up tasks instead of grabbing any little further todo in addition, like setting up meetings or taking notes or checking in with another person.

Invest in upskilling people, continuously.

Pairing added to that goal just as much as sharing knowledge in a fun and realistic way. For example, I gave two operations and support trainings that put my team in the actual situation of an incoming user request or an alert by our product, while also enabling them to investigate the situation and go through it. This helped a lot with sharing why this work is important, why we need to bring our pieces of knowledge together, and how to do it without making it a tedious burden yet allowing us to learn from it.

Explain your situation and manage expectations.

Share your context in any interaction. Being open and transparent with people helped a lot with their understanding and us seeing the full picture. It also allowed us to find alternative ways to accomplish things faster or with less effort.

Find sympathetic ears.

What really helped me was talking with a lot of people - people who listened and I'm ever grateful for that. Sharing your situation and speaking out loud helps reflection and becoming clear that this is not a situation to stay in. It sometimes triggered other ideas what to try, or re-established my confidence in doing my job; yet even if it only helped make the other aware of my situation, talking already helped.

Refrain from taking on more on top.

Saying "no", "not yet", "not me". A lot. If you can't (as priorities change, right?) communicate what goes instead - you can't do everything at once.

Connect people.

Instead of taking things on your own desk, empower others to do so. Sometimes all there's needed is to make people visible and connect them.

Reduce your cognitive load and singletask.

Wherever possible. Do one thing at a time and complete that one thing at a time before tackling the next one. Sometimes more stuff comes in on which you can't decide right away, yet instead of keeping this in the head just park it in a todo list or similar. Anything that freed my thinking capacity to focus on the current task at hand helped. Sometimes it also meant getting rid of a few smaller and less important things just to free my mind again for the big important impactful one. Anything to get calmer or stay calm enough and maintain thinking capacity. If I'm drowning, my biggest value is gone: being there for people, going deep thinking in different perspectives and creating bridges and connecting people, driving experiments and inspiring change. So I'd rather should help 10 people not 100. Spreading myself too thin does not work.

Consider unplanned work.

No matter what we do, we will discover new things as we go. Whether it's the tooling that suddenly does not want to work with us anymore, or incidents taking over priority, or a personal crisis. Some things will happen and shift previous plans. We need to keep this in mind and enable us to act on new insights quickly.

Maintain your own space.

I need space to drive testing and quality topics in my team. I need space to contribute pro-actively. I need space to help other teams and people in the company as well. I need space to be helpful. The safer I feel and the more space within constraints I have, the more ideas I get and think positively about my work and the impact we can have together. Sometimes I just needed space to tackle a few things on my todo list - so I'm really thankful for my team giving me this space. Yet remember: filling the day with back to back calls is the opposite of maintaining space.

Make space for others to step up.

My new team achieved a lot of things I wanted to drive forward just by getting the space to do so. Together, we finally tackled some long waiting improvement points. We introduced integration tests for consuming Kafka messages, added template testing for our frontend, increased the level of observability of our backend, introduced actual feature flags to decouple deploying from releasing, integrated first accessibility testing to tackle this increasingly important quality aspect, and introduced user tracking with Hotjar to get even more data and make more informed product decisions. Yes, I nudged a bit on all these topics, yet most of this was achieved by my teammates with me getting out of the way.

Reclaim your calendar.

Setting my work and private calendars to tentative helped me a lot. This can serve as a signal to others looking for free slots and having them reach out to me directly, yet it mostly served as a signal to myself when looking at my calendar. Screaming in my face: "no more extra meetings during this time, don't schedule or accept if not absolutely needed".

Reclaim your inboxes.

Responding to private emails only on weekends; just because this way no further response could come in between and add to my load.

Prioritize sleep.

This grew very important, especially after my body made clear my stress levels are too high. Cutting on sleeping hours is never the way to go.

Make space for "me time", no matter what.

Canceling private appointments to get a little me time here and then. Even if I just used it to watch another episode of my current series. Anything to distract my mind from work.

Accept what you cannot change.

We don't have everything in our hands. Accepting this can be hard, yet many times I can only change how I cope with what comes.

Be okay with being not okay and focus on finding a way back to okay.

As soon as possible. In the end it boils down to that. I cannot be of any help to others, especially on the long run, if I don't take care of myself. Remember the oxygen mask and why we need to put it on ourselves first.

Take time to socialize and have a good time with people.

With my new team starting to grow together, I drew a lot of energy from any social moment we had, great conversations as well as having fun playing a game together. I desperately need the bonding and building these relationships on more than just work topics.

Celebrate achievements.

I sometimes really need to acknowledge what I managed to do and allow myself to feel not only overwhelmed but also take in good energy from these achievements. The trick is: even if I don't feel like it, celebrate nonetheless. With the responses the good feelings come along. Looking back I'm thankful I did and can now feel more proud than at that point in time.

With all that, after half a year, I finally feel I'm in a better place again. Most of the topics on my desk that had absolute dates attached are done, people around me are finally enabled to help out, and I have capacity again to deal with what's still there while remembering not to take on too many new ones at a time. The rest can wait, there's a time coming for that.

Huge kudos and gratitude to my new team - I really appreciate you for stepping up big time, feeling responsible for the whole product and team, for more than your own area of expertise. For being ready to take over unfamiliar things outside your comfort zone. For sharing the load and helping each other. For learning every day with each other. For experimenting. For really having my back when I went on vacation or spoke at a conference. For listening. For sending me on vacation early. I know some of you feel you're just doing your work - yet let me make this clear again: yes, you are doing your work, and I appreciate you for doing it well.

Huge kudos to so many people of our lovely communities - thank you all for listening to me in this time or for bearing with me canceling appointments, not accepting pairing requests anymore, and not being there for a lot of things. I am coming back to all the lovely community stuff, yet I need to remember to use my energies wisely and look for synergies where I can.

Huge kudos to my friends for reaching out, and especially to my best friend and sister Marlene Guggenberger - thank you for honoring me as first listener of your first novels (which are amazing, so people who understand German should definitely check them out!). I loved the live reading as the story progressed, it was the time off my head that I needed on many of these days and hence a real life saver for me! Not taken for granted.

The one most helpful thing I've learned to prevent a situation as described above worked also as the way out of it.

Dear future me. I am not alone.

DO YOU NEED A TECH DEGREE TO SUCCEED AS A TESTER?

In the last three months, I have been interviewing multiple QA Leads and Managers about whether it is necessary to have a tech degree to have a successful QA career.

The reason behind my question was the fact that I have changed my career to tech in my 30s from being a lawyer. I did a coding bootcamp and then easily found a job as a front-end developer/QA. I found out that I like the QA profession better so have been following that path for now. But since there are other people who might want to change their career to tech as well here are the results of my QA interviews:

Attitude is more than education

All of the interviewed QA Leads/Managers agreed that QA candidates' attitude towards the job is much more important than having a tech education. However, this might be slightly different based on the country. For example, in Canada you might need to have a tech degree to get a QA job. The reason behind it is the fact that QAs in Canada are more of QE – quality engineers expected to be able to program as well. In contrast, in Europe it is rather easy to get a QA job even if you don't have tech degree. The reason behind it is the lack of skilled people in Europe (now mainly due to Covid and limited immigration programs) and the vast amount of available work.

Technical skills can be learned

Yes, everybody agreed that technical skills can be learned. People just need to be willing to learn new things. That means, if you say at the interview for a QA position that you don't like learning or feel like you have learned everything yet, you will probably not get the job. You will need to have some aptitude for tech as well as curiosity. Being curious will help you in your QA (and really any other) career immensely.

Interviewers can tell whether you will be able to learn technical skills

Most interviewers will test your technical skills. This can include hypothetical question about how to test apps you use daily or actual software testing. Either way, the interviewer wants to know your thought process more than whether you can solve a certain problem. It helps if you practice this skill in advance so that you are not surprised.

Interviewers can tell whether you will be able to learn technical skills

Most interviewers will test your technical skills. This can include hypothetical question about how to test apps you use daily or actual software testing. Either way, the interviewer wants to know your thought process more than whether you can solve a certain problem. It helps if you practice this skill in advance so that you are not surprised.

No need for software testing certification

Most interviewers agreed that having a software testing certification does not make you a great software tester. However, having a professional certification for example can show your determination to become a QA. It was also interesting to read that QA Leads/Managers who came from rather conservative countries recommended to get a certification. On the other hand, QA Leads/Managers from less conservative countries or with more of an open-minded background (for example psychology) believed certifications are rather useless.

Things not to say at the interview

Last, but not least, it was interesting to find out what the QA Leads/Managers thought as a no-no at an interview. As expected, they did not like when candidates were cocky and felt that they do not need to learn anything new. Some of the QA Leads/Managers were also of the opinion that not asking questions at an interview or thinking of software testing only as a tech entry job made a bad impression on them.

In conclusion it was great to find out that tech degree is not necessary for entering the tech industry as a QA Engineer/Software Tester. It is much more important to love learning new things and be curious than having diplomas and certifications. And that is a great news for people changing careers in their 30s to tech.

HELENA H.

–
Helena is a lawyer and fire acrobatic performer turned QA who enjoys learning new things. She also loves helping other people to achieve their dreams through her blog youintechnology.com.

IS TALKING ABOUT SCALING HUMAN TESTING MISSING THE POINT?

I recently came across an article from Adam Piskorek about the way Google tests its software.

While I was already familiar with the book How Google Tests Software (by James Whittaker, Jason Arbon et al, 2012), Adam's article introduced another newer book about how Google approaches software engineering more generally, Software Engineering at Google: Lessons Learned from Programming Over Time (by Titus Winters, Tom Manshreck & Hyrum Wright, 2020).

The following quote in Adam's article is lifted from this newer book and made me want to dive deeper into the book's broader content around testing*:

Attempting to assess product quality by asking humans to manually interact with every feature just doesn't scale. When it comes to testing, there is one clean answer: automation.

Chapter 11 (Testing Overview), p210 (Adam Bender)

I was stunned by this quote from the book. It felt like they were saying that development simply goes too quickly for adequate testing to be performed and also that automation is seen as the silver bullet to moving as fast as they desire while maintaining quality, without those pesky slow humans interacting with the software they're pushing out.

But, in the interests of fairness, I decided to study the four main chapters of the book devoted to testing to more fully understand how they arrived at the conclusion in this quote – Chapter 11 which offers an overview of the testing approach at Google, chapter 12 devoted to unit testing, chapter 13 on test doubles and chapter 14 on "Larger Testing". The book is, perhaps unsurprisingly, available to read freely on Google Books.

I didn't find anything too controversial in chapter 12, rather mostly sensible advice around unit testing. The following quote from this chapter is worth noting, though, as it highlights that "testing" generally means automated checks in their world view:

After preventing bugs, the most important purpose of a test is to improve engineers' productivity. Compared to broader-scoped tests, unit tests have many properties that make them an excellent way to optimize productivity.

Chapter 13 on test doubles was similarly straightforward, covering the challenges of mocking and giving decent advice around when to opt for faking, stubbing and interaction testing as approaches in this area. Chapter 14 dealt with the challenges of authoring tests of greater scope and I again wasn't too surprised by what I read there.

It is chapter 11 of this book, Testing Overview (written by Adam Bender), that contains the most interesting content in my opinion and the remainder of this blog post looks in detail at this chapter.

The author says:

since the early 2000s, the software industry's approach to testing has evolved dramatically to cope with the size and complexity of modern software systems. Central to that evolution has been the practice of developer-driven, automated testing.

I agree that the general industry approach to testing has changed a great deal in the last twenty years. These changes have been driven in part by changes in technology and the ways in which software is delivered to users. They've also been driven to some extent by the desire to cut cost and it seems to me that focusing more on automation has been seen (misguidedly) as a way to reduce the overall cost of delivering software solutions. This focus has led to a reduction in the investment in humans to assess what we're building and I think we all too often experience the results of that reduced level of investment.

Automated testing can prevent bugs from escaping into the wild and affecting your users. The later in the development cycle a bug is caught, the more expensive it is; exponentially so in many cases.

Given the perception of Google as a leader in IT, I was very surprised to see this nonsense about the cost of defects being regurgitated here. This idea is "almost entirely anecdotal" according to Laurent Bossavit in his excellent The Leprechauns of Software Engineering book and he has an entire chapter devoted to this particular mythology. I would imagine that fixing bugs in production for Google is actually inexpensive given the ease with which they can go from code change to delivery into the customer's hands.

LEE HAWKINS

–
In the IT industry since 1996 in both development and testing roles, Lee has spent most of his career helping Quest Software teams across the world to improve the way they build, test and deliver software. He currently helps teams and organizations to improve their testing and quality practices through his software testing consultancy, Dr Lee Consulting.

Lee considers that his testing career really started in 2007 after attending Rapid Software Testing with Michael Bolton. Lee was the co-founder of the TEAM meetup group in Melbourne and co-organized the Australian Testing Days 2016 conference. He was the Program Chair for the CASTx18 testing conference in Melbourne and also co-organized Testing in Context Conference Australia 2019.

He is a co-founder of the EPIC TestAbility Academy, a software testing training programme for young adults on the autism spectrum. Lee is the author of An Exploration of Testers, a book formed of contributions from the worldwide testing community in which testers share their testing, career and life lessons.
He is a frequent speaker at international testing conferences and blogs on testing at Rockin' And Testing All Over The World. When not testing, Lee is an avid follower of the UK rock band, Status Quo; hence his Twitter handle @therockertester.

LinkedIn: https://www.linkedin.com/in/lee-hawkins-3574148

Much ink has been spilled about the subject of testing software, and for good reason: for such an important practice, doing it well still seems to be a mysterious craft to many.

I find the choice of words here particularly interesting, describing testing as "a mysterious craft". While I think of software testing as a craft, I don't think it's mysterious although my experience suggests that it's very difficult to perform well. I'm not sure whether the wording is a subtle dig at parts of the testing industry in which testing is discussed in terms of it being a craft (e.g. the context-driven testing community) or whether they are genuinely trying to clear up some of the perceived mystery by explaining in some detail how Google approaches testing in this book.

The ability for humans to manually validate every behavior in a system has been unable to keep pace with the explosion of features and platforms in most software. Imagine what it would take to manually test all of the functionality of Google Search, like finding flights, movie times, relevant images, and of course web search results… Even if you can determine how to solve that problem, you then need to multiply that workload by every language, country, and device Google Search must support, and don't forget to check for things like accessibility and security. ***Attempting to assess product quality by asking humans to manually interact with every feature just doesn't scale. When it comes to testing, there is one clear answer: automation.***

(note: bold emphasis is mine)

We then come to the source of the quote that first piqued my interest. I find it interesting that they seem to be suggesting the need to "test everything" and using that as a justification for saying that using humans to interact with "everything" isn't scalable. I'd have liked to see some acknowledgement here that the intent is not to attempt to test everything, but rather to make skilled, risk-based judgements about what's important to test in a particular context for a particular mission (i.e. what are we trying to find out about the system?). The subset of the entire problem space that's important to us is something we can potentially still ask humans to interact with in valuable ways. The "one clear answer" for testing being "automation" makes little sense to me, given the well-documented shortcomings of automated checks (some of which are acknowledged in this same book) and the different information we should be looking to gather from human interactions with the software compared to that from algorithmic automated checks.

Unlike the QA processes of yore, in which rooms of dedicated software testers pored over new versions of a system, exercising every possible behavior, the engineers who build systems today play an active and integral role in writing and running automated tests for their own code. Even in companies where QA is a prominent organization, developer-written tests are commonplace. At the speed and scale that today's systems are being developed, the only way to keep up is by sharing the development of tests around the entire engineering staff.

Of course, writing tests is different from writing good tests. It can be quite difficult to train tens of thousands of engineers to write good tests. We will discuss what we have learned about writing good tests in the chapters that follow.

I think it's great that developers are more involved in testing than they were in the days of yore. Well-written automated checks provide some safety around changing product code and help to prevent a skilled tester from wasting their time on known "broken" builds. But, again, the only discussion that follows in this particular book (as promised in the last sentence above) is about automation and not skilled human testing.

Fast, high-quality releases

With a healthy automated test suite, teams can release new versions of their application with confidence. Many projects at Google release a new version to production every day—even large projects with hundreds of engineers and thousands of code changes submitted every day. This would not be possible without automated testing.

The ability to get code changes to production safely and quickly is appealing and having good automated checks in place can certainly help to increase the safety of doing so. "Confidence" is an interesting choice of word to use around this (and is used frequently in this book), though – the Oxford dictionary definition of "confidence" is "a feeling or belief that one can have faith in or rely on someone or something", so the "healthy automated test suite" referred to here appears to be one that these engineers feel comfortable to rely on enough to say whether new code should go to production or not.

The other interesting point here is about the need to release new versions so frequently. While it makes sense to have deployment pipelines and systems in place that enable releasing to production to be smooth and uneventful, the desire to push out changes to customers very frequently seems like an end in itself these days. For most testers in most organizations, there is probably no need or desire for such frequent production changes so deciding testing strategy on the perceived need for these frequent changes could lead to goal displacement – and potentially take an important aspect of assessing those changes (viz. human testers) out of the picture altogether.

If test flakiness continues to grows you will experience something much worse than lost productivity: a loss of confidence in the tests. It doesn't take needing to investigate many flakes before a team loses trust in the test suite, After that happens, engineers will stop reacting to test failures, eliminating any value the test suite provided. Our experience suggests that as you approach 1% flakiness, the tests begin to lose value. At Google, our flaky rate hovers around 0.15%, which implies thousands of flakes every day. We fight hard to keep flakes in check, including actively investing engineering hours to fix them.

It's good to see this acknowledgement of the issues around automated check stability and the propensity for unstable checks to lead to a collapse in trust in the entire suite. I'm interested to know how they go about categorizing failing checks as "flaky" to be included in their overall 0.15% "flaky rate", no doubt there's some additional human effort involved there too.

Just as we encourage tests of smaller size, at Google, we also encourage engineers to write tests of narrower scope. As a very rough guideline, we tend to aim to have a mix of around 80% of our tests being narrow-scoped unit tests that validate the majority of our business logic; 15% medium-scoped integration tests that validate the interactions between two or more components; and 5% end-to-end tests that validate the entire system. Figure 11-3 depicts how we can visualize this as a pyramid.

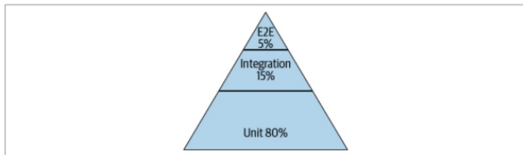


Figure 11-3. Google's version of Mike Cohn's test pyramid;[6] percentages are by test case count, and every team's mix will be a little different

It was inevitable during coverage of automation that some kind of "test pyramid" would make an appearance! In this case, they use the classic Mike Cohn automated test pyramid but I was shocked to see them labelling the three different layers with percentages based on test case count. By their own reasoning, the tests in the different layers are of different scope (that's why they're in different layers, right?!) so counting them against each other really makes no sense at all.

> This description of what exploratory testing is and what it's best suited to are completely unfamiliar to me, as a practitioner of exploratory testing for fifteen years or so.

I don't treat the software "as a puzzle to be broken" and I'm not even sure what it would mean to do so.

Our recommended mix of tests is determined by our two primary goals: engineering productivity and product confidence. Favoring unit tests gives us high confidence quickly, and early in the development process. Larger tests act as sanity checks as the product develops; they should not be viewed as a primary method for catching bugs.

The concept of "confidence" being afforded by particular kinds of checks arises again and it's also clear that automated checks are viewed as enablers of productivity.

Trying to answer the question "do we have enough tests?" with a single number ignores a lot of context and is unlikely to be useful. Code coverage can provide some insight into untested code, but it is not a substitute for thinking critically about how well your system is tested.

It's good to see context being mentioned and also the shortcomings of focusing on coverage numbers alone. What I didn't really find anywhere in what I read in this book was the critical thinking that would lead to an understanding that humans interacting with what's been built is also a necessary part of assessing whether we've got what we wanted. The closest they get to talking about humans experiencing the software in earnest comes from their thoughts around "exploratory testing":

Exploratory Testing is a fundamentally creative endeavor in which someone treats the application under test as a puzzle to be broken, maybe by executing an unexpected set of steps or by inserting unexpected data. When conducting an exploratory test, the specific problems to be found are unknown at the start. They are gradually uncovered by probing commonly overlooked code paths or unusual responses from the application. As with the detection of security vulnerabilities, as soon as an exploratory test discovers an issue, an automated test should be added to prevent future regressions.

Using automated testing to cover well-understood behaviors enables the expensive and qualitative efforts of human testers to focus on the parts of your products for which they can provide the most value – and avoid boring them to tears in the process.

This description of what exploratory testing is and what it's best suited to are completely unfamiliar to me, as a practitioner of exploratory testing for fifteen years or so. I don't treat the software "as a puzzle to be broken" and I'm not even sure what it would mean to do so. It also doesn't make sense to me to say "the specific problems to be found are unknown at the start", surely this applies to any type of testing? If we already know what the problems are, we wouldn't need to test to discover them. My exploratory testing efforts are not focused on "commonly overlooked code paths" either, in fact I'm rarely interested in the code but rather the behaviour of the software experienced by the end user. Given that "exploratory testing" as an approach has been formally defined for such a long time (and refined over that time), it concerns me to see such a different notion being labelled as "exploratory testing" in this book.

TL;DRs

Automated testing is foundational to enabling software to change.

For tests to scale, they must be automated.

A balanced test suite is necessary for maintaining healthy test coverage.

"If you liked it, you should have put a test on it."

Changing the testing culture in organizations takes time.

In wrapping up chapter 11 of the book, the focus is again on automated checks with essentially no mention of human testing. The scaling issue is highlighted here also, but thinking solely in terms of scale is missing the point, I think.

The chapters of this book devoted to 'testing" in some way cover a lot of ground, but the vast majority of that journey is devoted to automated checks of various kinds. Given Google's reputation and perceived leadership status in IT, I was really surprised to see mention of the "cost of change curve" and the test automation pyramid, but not surprised by the lack of focus on human exploratory testing.

Circling back to that triggering quote I saw in Adam's blog ("Attempting to assess product quality by asking humans to manually interact with every feature just doesn't scale"), I didn't find an explanation of how they do in fact assess product quality – at least in the chapters I read. I was encouraged that they used the term "assess" rather than "measure" when talking about quality (on which James Bach wrote the excellent blog post, Assess Quality, Don't Measure It), but I only read about their various approaches to using automated checks to build "confidence", etc. rather than how they actually assess the quality of what they're building.

I think it's also important to consider your own context before taking Google's ideas as a model for your own organization. The vast majority of testers don't operate in organizations of Google's scale and so don't need to copy their solutions to these scaling problems. It seems we're very fond of taking models, processes, methodologies, etc. from one organization and trying to copy the practices in an entirely different one (the widespread adoption of the so-called "Spotify model" is a perfect example of this problem).

Context is incredibly important and, in this particular case, I'd encourage anyone reading about Google's approach to testing to be mindful of how different their scale is and not use the argument from the original quote that inspired this post to argue against the need for humans to assess the quality of the software we build.

* It would be remiss of me not to mention a brilliant response to this same quote from Michael Bolton – in the form of his 47-part Twitter thread (yes, 47!).

MARIA KEDEMO
"Black Koi Consulting"
–
Maria is a generalist with a deep knowledge in software testing and agile development.

She has been working in software development for over twenty years in many different roles and industries. Since 2018 she has her own company Black Koi Consulting.

What she appreciates the most in each and every assignment she's had is the learning opportunities. Maria loves sharing her experiences and helping others to improve. She has spoken and organized workshops at many national and international conferences. Occasionally she shares her thoughts on mkedemo.wordpress.com

For many years we have seen different kinds of coaches appearing within software development. As more and more companies strive to become agile, various types of roles are becoming obsolete or transformed into something different. The most prominent one seems to be Agile Coach. In the software testing domain it is the Test Coach or the Quality Coach.

The transformation and changes in expectations of a role have in my experience caused some identity crisis within the testing profession. Even though there is a need for testing many companies choose to remove the tester as a role. (This post is however not about testers so I will not continue down that road).

As for myself I've been struggling to put a label on the work that I do. For those who know me I am not a big fan of titles and labels, although they can be helpful in some contexts. My work for the last few years have focused on transformation and how testing needs to be interlaced with development. Many of the companies I've worked with do not even have testers.

Are you really a Test Coach?

What title you carry is not as important as the approach you choose.

Coaching

Last year I went through a nine day training to become an ICF Coach. During that course I had many great insights. One of them relates to the Coach in the context of Software Development. A few times I've labeled myself **Test Coach** or **Quality Coach** but I struggled a bit with those titles as well. I just felt they didn't really do justice to my work. In the context of coaching a coach is an expert on the process of coaching and is someone who facilitates learning.

"A coach is an expert on the process of coaching and is someone who facilitates learning."

The client is the expert but the coach helps the client to unlock their potential to maximize their own performance. A skilled coach knows that the individual has the answer to their own problems.

A coaching approach

Why I struggled with titles like Test Coach became very obvious during my coach training. I was presented with the following model, "The flower", created by **Polhage & Lundberg**, who also run the training (the model is originally described in Swedish and this one has been visually modified by me). They differentiate between the coach as a profession and having a coaching approach. We can always apply a **coaching approach** whether it's in our daily life or at work.

> I am a subject matter expert in testing trying to help an organization, a team or an individual to improve their testing by guiding them and showing what to do, how they can do it and why.

The flower petals represents several roles which we might step into during our daily life or at work. The **Coach** is one of these roles (and the one I was in training for).

You can move between these roles and decide who to be in different situations. As an example, sometimes you need to take decisions based on your responsibilities which makes you the **Decision maker**. The empty petal is left for you to decide what to put in there. Your flower might have many more petals.

No matter what profession or role you have you can always apply a coaching approach. This means how you act and relate to the values of coaching.

The root system represents eight characteristics to consider for constructive communication, which are used in a coaching approach.

I quickly realized why I have never been very fond of the title Test Coach. It doesn't fully reflect what I do or who I am. I am a **subject matter expert** in testing trying to help an organization, a team or an individual to improve their testing by guiding them and showing what to do, how they can do it and why.

But I am also a **Decision Maker**, a **Teacher**, an **Inspirer** and a **Mentor** (for those who choose me to mentor them). I shift a lot in between all of these. One role I have never used at work is the Coach. However I often apply a **coaching approach**. This is an approach where I ask questions, where I use my curiosity to understand where the team or individual is right now and where I display my courage to challenge and ask "uncomfortable" questions. Focusing on what works and what moves us forward is also part of what I apply in my daily job whether my title is Project Manager, Test Coach or Scrum Master.

The only time I have been the Coach and only a coach is when I am a professional Coach in an agreement with a client.

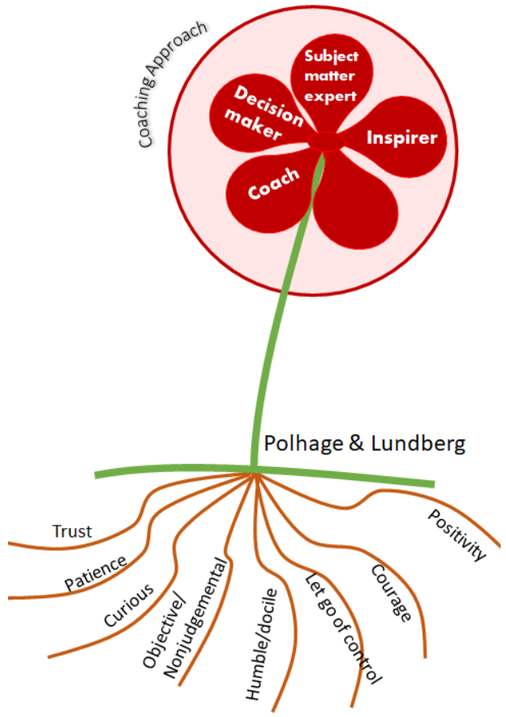
Coaching in software testing

I recently had a short assignment where I was asked to coach a tester. She needed someone to talk to regarding her own journey where she was leading a change in her organization. In the beginning I found myself struggling with who to be. Biased by my recent experiences as a Professional Coach I started off in that role but quickly understood that my client needed something different. The focus was more related to guidance around the change she was implementing at work rather than her own journey. Sometimes it was hard to separate her own growth from the approach to testing that she was implementing.

Something that is very important is the agreement that you come up with before starting the sessions. The purpose of that agreement is to build trust and set expectations. Though this situation was a bit new for both myself and my client we decided to keep an open dialogue along the way to make sure she got value from our sessions.

My learning experience here is that it is not as black and white. What title you carry is not as important as the approach you choose.

During these sessions I used a coaching approach – actively listening, asking questions, driving my client to find her own solutions based on where she and her team are right now. In the cases where she wanted me to share my experience and thoughts, I did that as well.



What are your thoughts regarding coaches in Software Development/Testing?

References

Polhage & Lundberg

International Coaching Federation

INTERVIEW

Where is testing profession heading to? What does it take to get the Devs do testing? How do we measure the success of Modern Testing? Let's ask Alan Page!

It's a pleasure meeting you once again (virtually this time), Alan. How have you been and what are you up to?

Like most people, I've been spending a lot of time at home over the past fifteen months. My dog, Terra, has been pretty happy with the situation, and has kept me company during a lot of long meetings on Zoom. I've been at Unity for over four years now, and currently lead teams that deliver Unity's service infrastructure, developer tools, reliability engineering, documentation, and quality coaching. Quite a variety of roles, but all stuff that leads to customer's having a quality experience using any of our app sdks or services.

Let me ask the most important question before we proceed. Do you like testing? Why if yes and why not if no?

Of course I like testing. I do it all the time. I'm a discerner, so I evaluate, judge, and question things constantly. While on the surface, I do a lot of things that aren't really "traditional" testing anymore (meaning I don't spend significant time exploratory testing or writing testing tools), I apply my hunger for knowledge in asking the right questions in order to help coach teams towards delivering better quality.

In the second principle of Modern Testing, we tried to capture this aspect of helping - or accelerating a team's ability to ship more quickly, and with high quality. Sometimes, but not always, the 'accelerant' here is the activity (and output) of testing.

Speaking of which (Modern Testing), could you tell our readers more about it? Where did the idea come from? What made you develop it further? Why is it called Modern Testing?

Years ago, my podcast partner Brent Jensen and I were beginning to see some changes in the way software was developed. As teams moved to Agile, we were seeing teams improve quality and delivery with fewer test specialists, and we were seeing some of those teams see success without dedicated testers. We saw teams use data and monitoring to accelerate their delivery of quality software.

We also saw teams making mistakes as they moved down this path.

We began our podcast as a way to help testers and teams navigate the changes we were seeing. Eventually, we began to refer to it as "Modern Testing" - even though it's not really about testing, and it's not really that modern. We just called it that to differentiate it from more "traditional" methods where testing was done late, and often by a separate - and sometimes isolated team.

Eventually we came up with a set of Principles to describe what we were talking about. They're based on what we were seeing as well as some of the books we were reading and that we found helpful (e.g. The Lean Startup by Eric Ries).

The very first time I spoke about the modern testing principles in public, I was afraid people would get scared, or I thought at the very least they would disagree. I was surprised that a bit of the opposite happened when several people approached me and told me I had just given a name to the way their team worked. Since then, dozens, if not hundreds of people have told me the same thing.

ALAN PAGE
–

Alan has been improving software quality since 1993 and is currently a Senior Director of Engineering at Unity Technologies. Previous to joining Unity in 2017, Alan spent 22 years at Microsoft working on projects spanning the company – including a two-year position as Microsoft's Director of Test Excellence.

Alan was the lead author of the book "How We Test Software at Microsoft", contributed chapters for "Beautiful Testing", and "Experiences of Test Automation: Case Studies of Software Test Automation". His latest ebook (which is a few years old, but still relevant) is a collection of essays on test automation called "The A Word: Under the Covers of Test Automation", and is available on leanpub.

You have been a musician too and have been working in the engineering field. What is easier? Composing music that people would love or building software that people would use happily?

When I studied composition, I studied the history of a lot of composers, including a lot of the more avant-garde 20th century composers. John Cage wrote a lot of great music that I like a lot, but he's probably most famous for his 4'33", which is 4 minutes and 33 seconds of silence, where the performers are instructed not to play their instruments during the entirety of the works' three movements. You could argue, "that's not music!", but Cage's intent - like a lot of the other avant-garde artists of the time was that whether it's a musical score, a painting, or whatever - the art is the audience's reaction to the medium. If the audience chooses to fidget in their seat, clear their throat, or yell obscenities and leave, they've had a reaction, and, in a sense, art has been made.

What does this have to do with the question? Well, I'd say that composing is more difficult. For most of the software I work on these days, the feedback loop from idea to customer feedback is pretty quick. Iterative and adaptive software development give us a big advantage in making software that people use happily. There are opportunities for a little of that same style feedback in composing music - I often would write something and then convince an ensemble to play it for me and offer some feedback - but usually around orchestration or idiomatic traits of the instruments - e.g. "that part in the middle felt awkward to play", or "I feel like the trumpet overpowers the flute in the coda". That's all great feedback that helped me improve compositions, but ultimately, those musicians aren't the ones who ultimately need to react to my art. It's much harder to get a room full of music enthusiasts to come to iteration after iteration of a piece of music and give me quantitative feedback on improving it.

They say culture eats strategy for breakfast. Do you think organizations' business models (or revenue models) eat testing culture for breakfast too?

I'm pretty sure the "they" in this case was Peter Drucker - who meant that the way your org works (the culture) will have far more influence on the outcome than strategy (or tools, for that matter).

I don't know if the same is true for business models and testing culture. In fact, I could argue that a culture of quality (which includes more than just testing) has a significant impact on business and revenue goals. It's the people that execute the strategy and processes - and the way people work that drives business revenue - not the other way around.

There are a lot of references to the cost of quality, but I believe that a good approach to quality and testing is a business advantage. When the entire team is focused on testing and quality, products ship more quickly, and require less re-work - making them cheaper to build in the long run. Furthermore, the tight feedback loops (which come from having the ability to update more often), give you a much better chance of solving your customers' problems in a way they enjoy.

You have been in the industry for a long time and have been in senior positions too. When it comes to making a trade-off, why is that it often happens at the cost of quality and then naturally at the cost of software testing?

As I mentioned in my previous answer, I don't think testing has to be a cost. I agree that it is often a cost though - especially when testing is done mostly at the end of the project or by a team isolated from the team developing the software.

I think teams choose to cut on testing because they way they've planned for it is expensive. I think if the whole team is involved in testing as a forethought then it's difficult (or impossible) to cut costs by sacrificing testing. It's just when testing is bolted on late, or approached as a separate phase that it gets expensive and more easy to cut.

With all the new tech and tools at our disposal, do you think the software testing industry is inclining a lot towards the "engineering" aspect of it and ignoring the "craftsmanship" part? How do you suggest we find the right balance between the two?

Time for my obligatory Robert Pirsig reference. In Zen and the Art of Motorcycle Maintenance, Pirsig says (paraphrased) that care and quality are two sides of the same coin. If you look at the companies who are successful with software right now, most of them care a lot about their customers, and how their software is made. Assuming that "craftsmanship" and "care" are the same thing, there's definitely room (and necessity) for a balance, but I think what's needed is much more of a blend than a balance.

Tech and tools definitely play their part. The tools teams use for code analysis and security analysis help teams get code correctness right much more often. On the other end of the spectrum, observability toolsets are making it easier for teams to understand how their software is used in production. Some of the automation tools are making it easier for the whole team to create stable and valuable tests quickly - giving the team time to spend time on deeper testing or analysis. The "tech" is making it easier for teams to put care into the software they deliver.

You have been a strong promoter of "whole team testing" especially the "developer testing" idea. Though I have some strong disagreements especially with the myth of mindset part, looking at your accomplishments it seems the idea worked very well for you. What are some critical elements or pre-conditions in your experience that must be present/fulfilled to succeed with developer testing?

I suppose I need to lead by saying that it's worked well for me with hundreds of developers and at two different companies.

With that out of the way, the most critical element is accountability. I frequently see someone speak out against developer testing by saying that developers don't want to test, or don't feel like testing is their job, or don't feel like they can do testing. On the teams I work with, I don't give them a choice. I treat every development lead as I would have treated a test lead years ago. I tell them that they are responsible and accountable for testing and quality of the software they deliver and give them enough information to get started, and an offer of help and advice any time they need it.

I - or a few test experts in my team offer coaching and consulting to help them improve their testing ideas. Frequently, we'll ask them to write test plans (which we happily critique), or brainstorm on test ideas for a new feature.

In the beginning, most developers learning testing are pretty similar to any new tester. They have a lot of unconscious incompetence in regards to testing, and don't think a lot about the testing they need to do. Sometimes they make mistakes, but they learn from those mistakes and get better. As a lot of us did when learning testing, they eventually get to a stage where they realize there's more to learn about testing than they can ever learn, but by that time, they've realized that continuous learning is the key to their success.

In short, tell them that they're accountable for testing and quality, and then give them enough help to be successful.

For organizations or industries where the idea of good-enough testing can be suicidal, do you think developer testing can still succeed there? What do such industries and organizations need to do differently to achieve that?

I think every developer - regardless of industry - should perform testing. They should create thorough unit tests at a minimum, but typically larger scale tests as well.

I could argue that the bar for "good-enough" testing changes with context - but maybe you're trying to get my opinion if dedicated testing professionals are still needed on software where errors may cause lives or massive amounts of loss. That answer depends on a lot more context, but I'll put it this way. Whether you are making an 8-bit mobile game for children, or a software engine control for a lunar lander, your goal is to make sure that you are identifying and mitigating risk to your customers' success. The risk with the game is low, so developer testing is more than sufficient. For the lunar lander, the business should determine whether developer testing is enough. I don't know those developers, and I don't work in that business, but my strong hunch is that they'd need a few more sets of eyes with critical thinking, systems thinking and some domain expertise to get to a level of risk the business is comfortable with.

Most of us work on software with a risk factor somewhere between those two examples. Like in those examples, our context (developer experience, risk factors, business goals, etc.) all dictate how we want to approach software testing and quality.

One of the principles of Modern Testing mentions that under certain contexts, teams may work without any dedicated tester. Are there any contexts where you think testers must be involved in teams?

This is answered partially, at least, in the previous question. But - this is a good place to point out that blindly removing testers from a team because "some company did it and it worked" is a really bad idea. If you don't have a culture that supports developer testing, or if the developers don't have the testing knowledge to properly assess and mitigate risk through testing, your team absolutely needs testers.

However - I also believe that with the proper level of training and quality culture that there are few industries or projects where dedicated testers would be required.

Typically in tech organizations, the quality is regarded as product quality and engineering quality. That apparently explains programming teams' inclination towards engineering quality. More often than not, this also results in the programming team not feeling responsible for product quality as such. How do teams solve this problem if there are no dedicated testers who are often the connecting link between both aspects of quality?

I feel like this is a bit of a strawman. In a vacuum, I agree - developers tend to focus on engineering-facing quality, because it's in their face. Compiler warnings, static analysis, code coverage, velocity and other measurements of developer quality are built in to so many of the tools and processes used by teams, it is certainly easy for them to focus there.

The previous question mentions the seventh Modern Testing Principle. It's often misunderstood, so let's take a closer look.

We expand testing abilities and knowhow across the team; understanding that this may reduce (or eliminate) the need for a dedicated testing specialist.

This principle is documenting what we've seen in many software organizations (note: the Modern Testing Principles are documenting what we're already seeing; not what we want to happen, or think should happen). What we've seen is that as development teams embrace more and more aspects of testing, that they may get to a point where they don't need dedicated testing specialists anymore. They don't get rid of testers for the sake of getting rid of testers - they just find that they can reach the levels of quality they are striving for without a dedicated tester. Most often, the testers who helped them get to this stage remain on the team in other roles. Often, they'll move into a new role, but occasionally move back into a testing or quality role if or when needed.

> The concept of Modern testing exists so that **we can help testers navigate changes happening across the industry.**

I guess a measure of its success would be for organizations to understand better how testing changes as they move to a world of quicker releases where quality is based on near real-time customer feedback.

Oh Alan, I would have liked to be wrong about my experiences which made me ask the previous question. More often that not, I met Devs/architects/Engineering leads who always shrugged things off blaming it all on the "product" team. Anyway, what contribution do you expect from testers in the team especially after the team switches to the whole team testing idea?

I will assume you're talking about an embedded tester on a team who have (or state they have), "whole-team testing". In this case, I like for the testers to drive the test strategy and planning, and pair with developers to write test strategies, test plans, or test charters for their components. They should have a heavy hand in defining what done" looks like. Given that testers are often the best systems thinkers on the team, they can also drive improvements through facilitating retrospectives and making sure that the team takes opportunities to learn from any setbacks they encounter.

Of course, they can (and should) do some testing too and use whatever they find to help refine and improve the team's test planning in the future.

I could be wrong but people who are not a fan of investing much in testing often argue that "customers don't care about testing". Do you think customers care about development either or the tech stack/tools we use or the best programming practices we follow or how much automation we have in place for that matter? As long as they get the quality product (or anything that solves their problem) at an affordable price in the minimum possible time, I guess they care about nothing mentioned above. Is it really because testing does not produce a production code, it becomes the natural victim of cost-saving strategies?

I don't think it matters whether you're a fan of investing in testing or not - it's a fact that customers don't care about testing. They don't care what kinds of testing you did, they don't care which tests passed, and they don't care about code coverage or static analysis.

They don't even want software. They want their problems solved easily and intuitively.

In modern agile teams these days, testers report directly to the engineering lead/managers who often know nothing much about testing (since they worked only as programmers and there was hardly any whole team testing happening back then). This often results in testers not getting the required support/guidance or even buy-in for their ideas at times. With your experience of leading engineering teams (with and without testers), what would be your advice to such managers?

I think it's the responsibility of those testers to help those managers - and their teams learn more about testing. I don't remember where I first heard this, but something I say often is that you can change your manager, or you can change your manager. Meaning, of course, that you can help your manager understand and learn more about the testing and quality side of the business - or, you can find a new manager who's less dense.

I think leadership and influence are essential skills for testers on Agile teams, and influencing the way your team develops and ships software is a core skill set for the modern tester.

How do we measure the success of Modern Testing?

It's never something we really thought about - but as I've stated before, the concept of Modern testing exists so that we can help testers navigate changes happening across the industry. I guess a measure of success would be for organizations to understand better how testing changes as they move to a world of quicker releases where quality is based on near real-time customer feedback.

What books would you recommend testers to read?

Like a lot of testers, I own a huge stack of Jerry Weinberg's books. All of those have some value for testers. I'm most partial to The Secrets of Consulting.

I think testers should understand how shipping more often and learning are correlated with software quality, and for that, I highly recommend Accelerate by Nicole Forsgren et. al. and The Lean Startup by Eric Ries.

For systems thinking and the theory of constraints (which are important aspects of shipping quality software), I recommend The Goal by Eli Goldratt, or The Phoenix Project by Gene Kim.

Your message to our readers would be?

There's a lot of change happening in the way software is delivered. As a tester, you could easily find a way to keep doing exactly what you're doing, or you could choose to adapt and change and grow along with the industry. With Modern Testing, we're just trying to help people in the latter path navigate the changes more easily.

You can be the butterfly, or you can be the wind.

AN EXPERIENCE REPORT ON R.S.T.

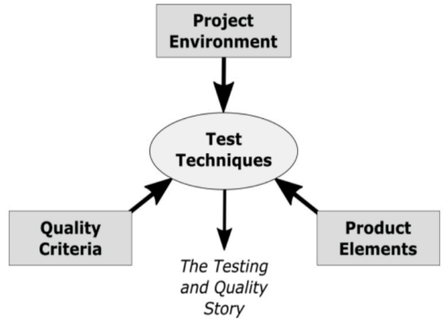
In early 2019 one of the Engineering Managers in my organization encouraged me to attend the Rapid Software Testing Applied online course. Initially, I was skeptical about how beneficial a remote course on testing methodology might be for me. I had several years of experience working in testing roles and I was quite comfortable with the way we were approaching testing in my current product team. I wasn't sure that there was much more for me to learn or things that would change my opinions on testing and the roles of testers. I was quite wrong in these assumptions.

By the end of 2019 myself and several colleagues were enthusiastically presenting a business case to management to advocate for further RST training within our organisation. In this article I'll highlight some of my experiences of the applications of the RST methodology, the challenges it's helped to overcome and how I've seen it benefit even very experienced testers.

Improving Strategies

"Your strategy contains the reasoning behind the testing that you do." (Kaner, Bach and Pettichord, 2002, p. 236)

There is a well established expectation that testers should be experts in the creation of test strategies for the products they work on. It's still listed as a (if not the) key responsibility for the majority of testing related job specifications. However it can be challenging for testers to define what the process of creating a strategy should actually look like in a practical sense. Is it just defining tests in advance of performing them? Is it deciding what tools and automation frameworks should be used?



[1] *The Heuristic Test Strategy Model*

One of the immediate benefits RST brought to our organisation was a new perspective on testing strategies. The Heuristic Test Strategy Model encouraged us to review and challenge our process of creating strategies. Prior to being introduced to this model most of our testers were already familiar with many of the common test techniques it mentions. They could easily articulate what these techniques were and how they would use them to test a product. However what they were going to focus their testing on and the reasons why they would do this were often less clear.

The Product Elements and Quality Criteria introduced in the HTSM became a powerful tool for framing our thinking around testing and helping us clearly define our reasoning. Specific tools such as Product Coverage Outlines and Risk Lists allowed us to capture these thought processes. Our strategies were no longer just conceptual and general, they were lightweight documents specific to the context of our products.

Risky Business

As testers we frequently talk about our work in the context of risks. The terms 'risk based testing' or 'risk focused testing' have become commonplace in the testing community. Whilst not being unique to RST, a core aspect of the methodology is risk analysis.

We found that the Heuristic Risk-Based Testing approach could be applied to our existing software products with very little overhead. Prior knowledge of statistical or quantitative analysis methods to identify and communicate risks weren't required.

Through experimenting with HRBT, we realised the importance of the social process involved in risk analysis. Discussing risks as a cohort of testers was exciting and a great way to share knowledge. However, at times it could become an echo chamber of testers opinions and ideas.

ANDREW JANUARY

Andrew January is a Senior Test Engineer at Simply Business.

He began his career in software testing in 2010. Since then he has worked at several London tech companies, coaching agile teams in the ways of testing and leading testing communities of practice. Outside of testing he has an interest in building Python applications for hobby projects.

You can find Andrew on Linkedin (https://www.linkedin.com/in/andrewjanuary/), Github (https://github.com/AndrewJanuary) and Twitter (https://twitter.com/AndrewJanuary)

"Risk communication and risk management efforts are destined to fail unless they are structured as a two-way process. Each side, expert and public, has something valid to contribute." (Slovic, 1987, p.285)

In order to make our risk analysis and risk based testing really effective we needed to involve more diverse perspectives outside of testing roles. Collaborative exercises such as the RST inspired Risk Storming have helped to enable this. It's now common in our organisation for testers to host risk analysis sessions for their product teams and stakeholders. Our discussions around product risks now involve more people who can actually help identify, rationalise and mitigate those risks. An additional benefit we've noticed is that these discussions provide opportunities for testers to subtly introduce or reiterate concepts such as Quality Criteria and Heuristics to our teams.

Jumping into the Unknown

Our organisation went through a rapid phase of growth during 2019. We hired lots of new engineers, rapidly formed new teams and developed new ways of working. As a result our testers occasionally needed to support multiple product teams, each with their own unique context and level of testing maturity.

The challenges of this situation prompted us to experiment with the Test Jumper role described by James Bach. This role proposes ways in which testers can add lasting value to teams, without needing to be embedded long term for the full lifecycle of a project or product. When combined with newly acquired knowledge of RST our testers were able to succeed in these high pressure temporary roles, despite working in the context of unfamiliar products, domains and technologies.

Continuous Learning and Community

In the two years since we first began to adopt the RST methodology the context of our organisation has continued to change. As a result we've found that we've frequently had to revisit the RST course materials, either to introduce it to new people, to revise our understanding or to improve how we are using it.

At the same time we've noticed more voices in the global community advocating for context driven and heuristic based approaches to testing. Conor Fitzgerald's talk at Test Bash Brighton 2019 highlights this and also hints at a growing interest amongst testers in the wider topics of cognitive science and psychology.

As the trends for adopting microservices architectures and DevOps methodologies continue, the expectations of testing roles are changing. There is a pressure on testers to provide value whilst not impacting the release cadence of product teams. Clokie describes the wider consequences this can have.

"The difficulty for testers is identifying what their role is, when testing is expected to be a part of everything." (Clokie, 2017, p.8)

The techniques offered by RST alone are not a direct solution to this challenge. However, the critical mindset it encourages and the shared language it provides can help testers to make lasting improvements in an organisation.

References

Kaner, C., Bach, J. and Pettichord, B., 2002. Lessons learned in software testing. New York: Wiley.

Bach, J. , 2020. The Heuristic Test Strategy Model v5.7.5.

https://www.satisfice.com/download/heuristic-test-strategy-model

Bolton, M. , 2020. Examples of Product Coverage Outlines. https://developsense.com/resources/pcos.pdf

Slovic, P., 1987. Perception of risk. Science, 236(4799), pp.280-285. https://doi.org/10.1126/science.3563507

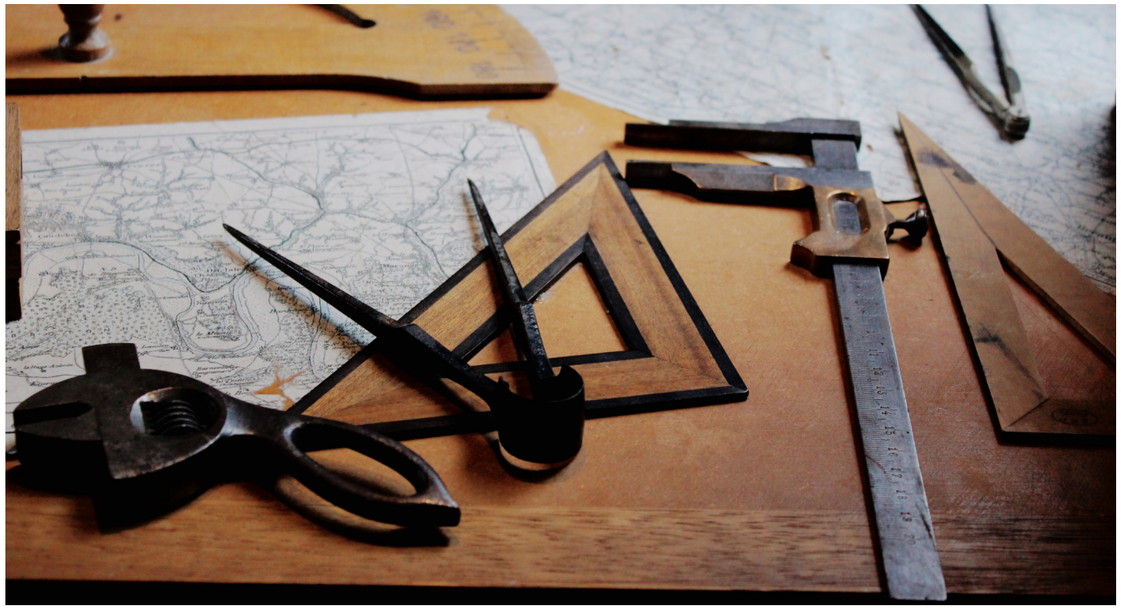
Bach, J. , 1999. Heuristic Risk-Based Testing. Software Testing & Quality Engineering, 23-28. http://www.satisfice.com/articles/hrbt.pdf

Gehlen, M. and Van Daele, B., 2020. Risk Storming.https://riskstormingonline.com

Bach, J. , 2014. Test Jumpers: One Vision of Agile Testing https://www.satisfice.com/blog/archives/1372

Fitzgerald, C., 2019. The Surprising Benefits of Exploring Other Disciplines and Industries.https://www.ministryoftesting.com/dojo/series/testbash-brighton-2019/lessons/the-surprising-benefits-of-exploring-other-disciplines-and-industries-conor-fitzgerald

Clokie, K., 2017. A Practical Guide to Testing in DevOps. [ebook] Lean Publishing. Available at: <https://leanpub.com/testingindevops> [Accessed 10 February 2021].



SEBTE: A SIMPLE EFFECTIVE EXPERIENCE-BASED TEST ESTIMATION - PART 2

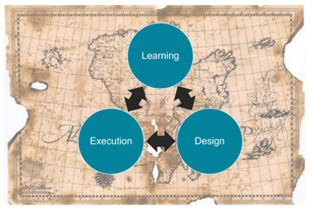
What is Charter Driven Session Based Exploratory Testing?

Exploratory Testing

There have been a few attempts to define exploratory testing. To begin with I suggest that all testing is exploratory and thus exploratory testing is just another word for testing.

I started using a definition from Cem Kaner in the mid-1990s to distinguish pre-scripted testing from exploratory testing.

Cem Kaner taught me that exploratory testing could be viewed as concurrent test design, test execution and test related learning which included planning and refocusing based on what we learn as we test.

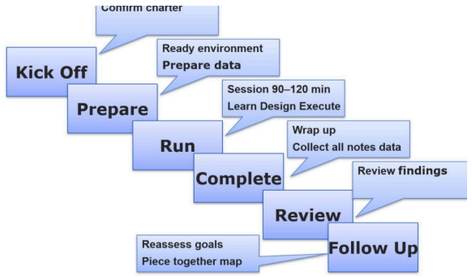


Cem Kaner eventually held a couple of small peer workshops about exploratory testing. Eventually Cem Kaner post the following description of exploratory testing on his blog entitled: *"On the craft and community of software testing."*

"Exploratory software testing is a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the value of her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project."

When I implement an exploratory testing framework, I try to use the following process steps:

Charter Driven Exploratory Test Framework



The first step is a kickoff meeting. The tester will be exploring. A collaborator will be commissioning the exploration. The collaborator can be a programmer as is more common when I implement CDSBET in an Agile team. The collaborator can be a test-lead as is more common when I implement CDSBET in a traditional or structured software development life cycle such as Waterfall or Rational Unified Process or V-Models.

Kick Off

The tester reviews the charter with the person commissioning the testing. We make sure that the scope and depth of the charter are discussed and agreed to. It is important at this time to also agree on the size of the upcoming session of testing in advance. I like to encourage the tester to discuss which variables, factors, conditions or data sources may be relevant.

Preparation

The tester gather whatever resource, data, tools or equipment are required for the upcoming session of testing. I never indicate a length for the preparation since it varies dramatically and it is really up to the team to manage and try to minimize this step. The tester will need healthy environments with appropriate data to full fill the charter.

Run

This is a time boxed session. I ask testers to implement this step with the cell phone, email, social networking and collaboration software turned off. I ask the tester to focus 100% of their attention of the session of testing. The tester will design and execute tests trying to learn about the charter. The tester will keep a record of decisions made, of tests attempted and of observations. The tester will use many diverse tools and technologies to complete this step. The step is always time boxed. A typical session is about 90 minutes long. Some sessions are short between 15 and 30 minutes. Some sessions are long between ½ and 2 days. Long sessions may be used for non-functional test charters. I do not expect all testing to be completed in one session. After the session the tester will review findings and decide whether it is necessary to invest in an additional session or perhaps to move onto a different charter instead.

Completion

The tester gathers their findings. The tester closed open files. The tester reports any bugs in bug tracking tools as required by the project team's workflow. The tester relinquishes the environment. Note that the tester must be able to reset the environment to a predicatabe state so very often at the completion step the testing will create virtual images of the test environment and data.

Review

The tester reviews their findings with the person who commissioned the testing. IN agile teams this is often the developer. In the review meeting all findings are reviewed and decisions are made on how to act on the finding In essence the review step is culling test findings and turing them into action. Test results are fed back to the person who commissioned the testing and also to any stakeholder would benefit from knowledge of the findings. This is a call for action. I recommend the review meeting take place on the same day as the session of testing. Some of my customers to the review immediately after the session of testing. Some of my customers review multiple sessions from the same day by the same tester at the same time.

The key decision made it – should we do another session on the same charter or should we move onto something else.

Follow Up

The team acts based on the finding of the tester.

The tester acts based on the feedback from the person who commissioned the work and any other stakeholder involved. This list will vary from charter to charter.

Charters

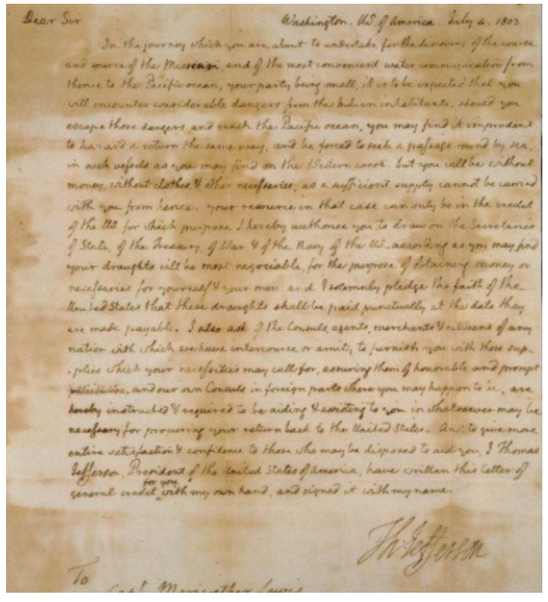
Note that charters derive from test ideas. One test idea can map to many charters. Multiple test ideas can map to one charter or there can also be a one to one mapping between test ideas and test charters.

A test charter is a mission statement for testing. It is a goal.

Chartering in 1803

The Lewis and Clark Charter to Explore

Mandated by President Thomas Jefferson


Dear Sir Washington the U.S. of America July a. 1803

In the journey which you are about to undertake for the discovery of the course and source of the Missouri and of the most convenient water communication from thence to the Pacific ocean, your party being small, it is to be expected that you will encounter considerable dangers from the Indian inhabitants. should you escape these dangers and reach the Pacific ocean, you may find it imprudent to hazard a return the same way, and be forced to seek a passage round by sea, in such vessels as you may find on the Western coast. but you will be without money, without clothes & other necessaries, as a sufficient supply cannot be carried with you from hence. your resource in that case can only be in the credit of the US. for which purpose I hereby authorise you to draw on the Secretaries of State, of the Treasury, of War & of the Navy of the US. according as you may find your drafts will be most negociable, for the purpose of obtaining money or necessaries for yourself & your men: and I solemnly pledge the faith of the United States that these drafts shall be paid punctually at the date they are made payable. I also ask of the Consuls agents, merchants & citizens of any nation with which we have intercourse or amity, to furnish you with those supplies which your necessities may call for, assuring them of honorable and prompt retribution. and our own Consuls in foreign parts where you may happen to be, are hereby instructed & required to be aiding & assisting to you in whatsoever may be necessary for procuring your return back to the United States. And to give more entire satisfaction & confidence to them who may be disposed to aid you, I Thomas Jefferson, President of the United States of America, have written this letter of general credit with my own hand, and signed it with my name.

Th Jefferson

To

"The object of your mission is to explore the Missouri River, and such principal streams of it, as,by its course and communication with the waters of the Pasific Ocean, wether the Columbia, Oregan, or any other river, may offer the most direct water-communication across the continent, for the purposes of commerce.

On an Agile team a charter statement can be the "title" of a testing task. In my practical experience charters can be expressed in 160 or fewer characters. This number dates to my earlier days using CRT terminals. The CRT terminals had 80 columns and 24 rows. I could always describe a charter in two lines of less thus the rule of thumb a charter can be expressed in 160 or fewer characters of text.

Session Notes

There are many ways that exploratory testers can express their findings. A charter can be represented by a task in a workflow management system, for example a Jira ticket. Each session associated with that charter would need to be a sibling object. In a session object the tester would include their findings, session notes, screen shots, screen videos with audio commentary, spreadsheets, data records, virtual images of system under test, pointers to bug descriptions and commentary from developers, product owners, teammates, and other interested project stakeholders. Collecting and recording findings should be a natural part of the testing workflow.

Session notes are like medical notes on a patient chart or a professional engineer logbook. This is a record of the testing done describing decisions made and trials attempted. The session notes do not need to include analysis or assessment, generally session notes focus on recording facts.

Text Files

In most session notes the tester puts a timestamp before each note and uses short bullet lists to describe test findings. Session notes are often recorded in simple text files but there are many variations.

Document Files

I have customers who use Microsoft Word and SnagIt (TechSmith) to keep session notes. The word document contains a chain of screen shots created by SnagIt. Microsoft Word macros ensure timestamps and document format is always consistent and reference-able.

Note taking tools like Rapid Reporter.

Rapid Reporter is a free tool to help testers record notes during a test session. The notes recorded automatically include a time stamp. Text, screen shots, rtf files can be embedded in the notes which are very professionally rendered as clean html tables.

Mind Maps

Many testers choose mind mapping tools such as XMind or FreeMind to capture visual representation of notes. Mind maps can include images, text, links to other objects and relationships between objects.

It is considered a good practice to keep track to time testing, start time, end time, timestamps during testing.

Many testers choose to include information in their session notes about their use of time: set up, on charter testing, off charter investigation, bug reporting.

When testing in a regulated environment consistent session notes can be used to demonstrate compliance to regulatory standards.

Some Other Test Estimation Techniques

There are many different test estimation techniques which may be relevant depending on your project context. As the EPA suggests - your mileage will vary. It is important to be aware of the many different approaches which can be applied.

ROB SABOURIN
–

Rob has more than thirty-nine years of management experience leading teams of software development professionals.

A highly-respected member of the software engineering community, Rob has managed, trained, mentored, and coached thousands of top professionals in the field. He frequently speaks at conferences and writes on software engineering, SQA, testing, management, and internationalization.

Rob authored I am a Bug! the popular software testing children's book. He works as an adjunct professor of software engineering at McGill University; and serves as the principal consultant (and president/janitor) of AmiBug.Com, Inc. Contact Rob atrsabourin@amibug.com

Test Estimation Technique	Description
Ad Hoc	I have used Ad Hoc estimation techniques when confronted with testing projects with loosely defined constraints. No clear goals. No clear quality perspective. Requirement turbulence. New or emerging technologies. With Ad Hoc techniques I always start with a guess. The key point to remember is to re-estimate frequently as you learn more about the project. With time the estimate will converge if the project converges.
Work Breakdown Structure	I use work breakdown on projects in which I can clearly organize testing into separately managed levels. For each level I define test objectives. For each test objective I define testing activities. For each testing activity I identify tasks that can be done by suitably skilled team members. Each task has pre-conditions and deliverables. Dependencies between tasks are understood. The granularity of a task is generally less than the reporting period of the project.
Programming Ratio	I have used tester programmer ratios to help me budget for equipment and office space. I have never used tester programmer ratios to estimate testing effort on a project. The idea is that I have a problem such as "How many square meters of office space will I need for testers in two years?" Since I have no idea which project is going to take place what I might do is inquire about how much space is being used by programmers and use a ratio to estimate the space required for testing. The estimate will be wrong, but it is probably better than nothing.
Wideband Delphi	Wideband Delphi estimation techniques involve a team of test, domain and technology experts who are provided the same source information and then asked to individually estimate the effort required to test a project. The individual estimates are then brought together in a facilitated team meeting. By studying the differences between the estimates, the team gains insights into the real work required to test the product. Individuals re-estimate and re-aggregate the results a few times until they converge on an acceptable estimate.
Planning Poker	Sizing requirements using story points is increasingly popular among my customers. A story point is a synthetic unit of size which blends notions of complexity and scope combining all the work a team is expected to do to fulfill the requirement. With experience, teams learn the relationship between size, in story points, and actual effort, in person hours. Productivity is often measured as story points per unit of time, which is metaphorically known as the team's velocity. As productivity increases, the effort required to complete the same story is expected to decrease. As the team's process improves their velocity increases. Mike Cohn published an interesting team approach to story point estimation, called planning poker. Planning poker cards have numbers on them between zero and one hundred distributed sort of like Fibonacci numbers. Each card represents the size of a story. Team members vote using the cards to define the size of each story during grooming, refinement, or estimation sessions. Team members advocate low and high estimates. They iterate to converge on an acceptable team estimate. The team approach assumes that the group has recent experience implementing similar requirements.

Test Estimation Technique	Description
Re-estimation Cone of Uncertainty	I continuously revise my estimates as I learn more about the product being developed and testing. I consider reality as it is uncovered. What are the real technical changes to the code? What is the real work being done by the user? What is the real environment the software is being used in? I revise my estimates and as work continues, I find the estimates converge.
Other Techniques	In engineering and project management there are many estimation techniques commonly used. As I learn new techniques, I seek opportunities to see how they may fit in the software testing domain.

Example Spreadsheet Available to Readers

Please contact the author, Robert Sabourin, via email at robsab@gmail.com to request a copy of an example spreadsheet illustrating how to use SEBTE.

References:

A guide to the project management body of knowledge (PMBOK guide). (2018). Exton: Project Management Institute.

McConnell, S. (2006). Software estimation: Demystifying the Black Art. Redmond, WA: Microsoft Press.

Humphrey, W. S. (1997). Introduction to the personal software process. Reading, MA: Addison-Wesley.

Sabourin, R. (2020). Charting the Course. Coming up with Great Test Ideas Just in Time. Montreal, QC: AmiBug.Com, Inc.

STORY OF A BOOK ON TESTING SKILLS

In this fast-changing world, you can ask yourself, what are the skills that I should have to be able to make a difference with the colleagues around me? Is it being an expert in test automation? Or able to read all kinds of different code languages? At least you know that the technology you learn today can be gone tomorrow. For example; how much time will it take to learn a new programming language? Probably you can learn the new language quite fast, just as many other people have done. So if you look at all kinds of hard skills; these are just abilities that can be acquired and enhanced through practice, repetition and education. Does it really surprise you that with these hard skills, you can't distinguish yourself so are you really prepared for the challenges of the 21st century? The challenges of this century have much more to do with the interhuman skills. As technology is already taken over our daily routines, how can you make a difference? Look at people around you in your organization. The kind of people that are successful, in the long run, are appreciated, valued and nice people. These people are using their soft skills! And with the soft skills you are able to make a difference. This is not just something that we made up. In general there is a lot of attention to define the right skills, to become "future proof". One of the models that is used to define the skill set for the future, uses the model of the 4 C's which stands for communication, critical thinking, creativity and collaboration.

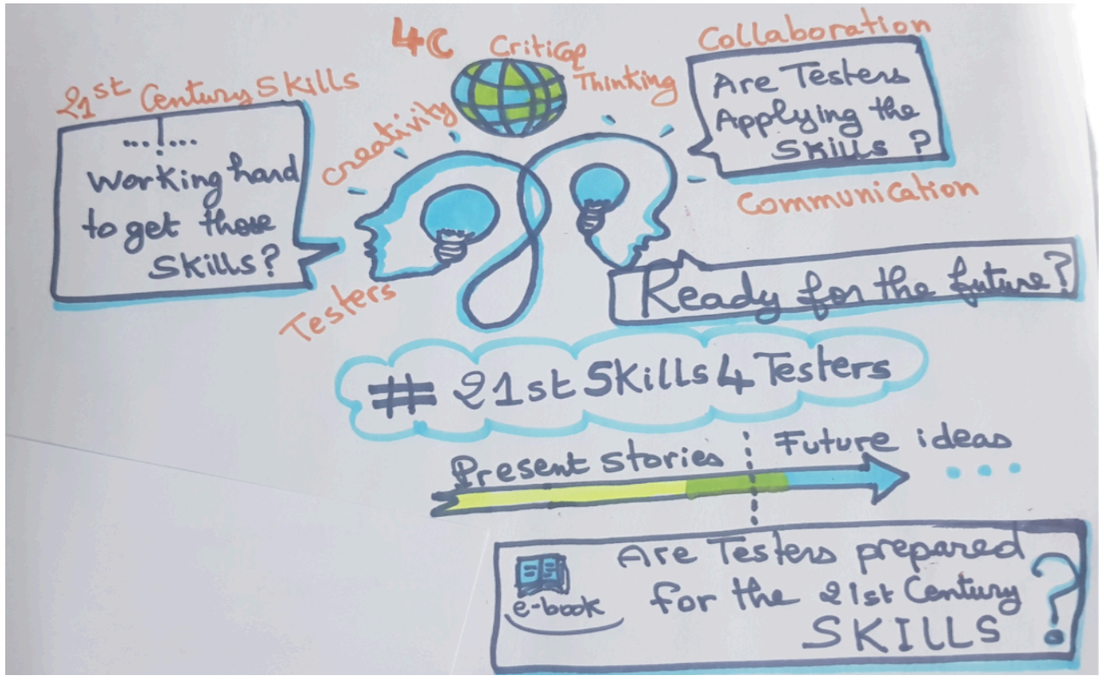
We will elaborate on these 4 C's in this article below and we were wondering if these skills are also applicable to make a difference as a software tester? We, Emna Ayadi and Ard Kramer started investigating this question. They came up with the idea to ask two questions to software testers all around the world: did they already apply these skills at this moment and how are they going to apply the skills in the future?.

How did we make the book (applying the skills)?

We created a website where testers could share their thoughts and ideas. They could submit with stories per one of the 4 C's and we notified our network by sharing our ideas on Twitter and LinkedIn. The reactions were quite overwhelming and we received 243 submissions from all around the world: from Canada to New Zealand and Argentina to Sweden. And as we said before the submissions were as well in English as in French..

For us the hard labour started: the editing. This meant that we had to look for a logical order of the stories and to put everything well readable in a book. Emna contributed with her sketch-notes to visualize the content (and after reading the stories we learned that testers had the common opinion that visualization is a very important way of communication with each other). In the paragraph below we will give you some elaboration on the 4 C's from a testing point of view and we shared some quotes of contributors.

	Critical Thinking		Communication		Collaboration		Creativity		4Cs	Total	
	Story	Idea	Story	Idea	Story	Idea	Story	Idea	Story	All	
ENG	18	19	23	19	22	19	17	17	3	157	243
FR	11	8	11	11	19	9	9	8	0	86	



What can you find in the book: Tips from authors per skill

Critical Thinking

It's the ability to look at problems in deeper and different ways by evaluating the possibility of failure also, finding gaps between expectations and reality. The goal here to find the unknown unknown by having an analytical mind and linking learning across your testing steps. James Bach says that

"Testing is an infinite process of comparing the invisible to the ambiguous in order to avoid the unthinkable happening to the anonymous."

"In the future, as applications/systems become more complex, automating them will be of the same wavelength. Thus, critical thinking will be essential for a QE to keep up with technology practices. Aside from the variety of software available to be developed, there are a number of automation tools being available to the public, some let you code, some make it easier. Regardless of which tool, like a QE, your critical thinking will be important not only in problem-solving but also in maximizing these tools to improve your productivity and efficiency" Gerald Habal (from the Philippines)

Communication

Communication is the set of interactions with others that transmit any information. Sharing your opinions regarding the software you are testing, be curious asking questions within your team and proposing your ideas or solutions.

***"My challenge in the future for my team, and testers worldwide, will be to speak up, raise the risks early, and never assume. Ask clarifying questions to ensure that you understand the purpose of the product. Talk with the stakeholders. Find out what they expect, and how they expect to use the product. Speed to market is driving teams to deliver faster, with shorter sprints. This will only continue to be aggressive. Being an effective communicator on the team will be critical for testers in the future."* Mike Lyles (from the USA)**

Collaboration

Collaboration is the act of working or thinking together to achieve a goal. Collaboration can be done with two or more people or organizations that have a common goal to achieve. Testing is not a single step in the process, it's a whole team task shared between all the development team and business side. In fact teams of people have a collective intelligence independent from the individual one and greater than the total of these parts.

***"Being experienced, I see "Collaboration" in the future as a very important skill. Something that makes me very happy is to be an inspiration for other professionals. Always try to be better, collaborating with others and giving peace of mind so that they collaborate with you too and everyone is a winner!"* Tatiana Ribeiro Nunes (from Brazil)**

Creativity

Creativity is the ability of an individual or group to imagine or construct and implement a new concept or object or to discover an original solution to a problem.

The ability to come up with new and useful ideas while exploring the software. Innovation is the successful implementation of creative ideas, this includes both incremental and radical change in systems and products to deliver better quality.

***"Being creative to me is mostly about idea generation and visualization. Finding new ways to approach and model our test approaches - there's more to testing than following agile and v-model approaches. There's more to testing than testing software development projects."* Jesper Ottosen (from Denmark)**

The definition that we derived in the conclusion

As we stated the book is a billagual book, it starts with stories in English and are divided in four chapter: per chapter a C will be covered and it contains two paragraphs: the first paragraph is about how the contributors already apply the skills and the second chapter how they are going to apply the skills in the future.

We also had some contributions that covered all the 4 C's: we have put them together in the last and fifth chapter. The same division in chapters is used for the second part of the book, which are the same stories but this time written in French.

For every story in both the French and English contributions, we looked at one or more keywords to provide the reader an overview. The keywords in the index helped us to visualize the topics that were used the most. With this overview we were able to create a kind of definition of the The tester of the 21st century:

"He or she is an open-minded tester, who values diversity because he/she wants to look at problems and challenges from different perspectives. The tester is aware of assumptions and tries to make those assumptions clear and does this by listening and by asking questions and for feedback if he/she understands what is said. The overview that is created can be at best visualized If you want to get the maximum out of working together, he/she knows that discussion is important, but that a discussion needs an environment where we can trust each other so we are able to learn and to improve."

But probably if you read the book, you can come up with more definitions or inspiration to create a profile for a future proof software tester. For that we challenge you to read the book and share your ideas or your perspective.

Definitely other people are interested in your thoughts so if you share them on social media using #21stskills4testers other people will get inspired by your thoughts just as we did this by composing this book.

We can imagine that you are quite curious what all these testers have written down in the book. There is an easy way to find out. You will find our book at Leanpub (https://leanpub.com/_21stskills4testers) where you can obtain the book for free (or pay as much as you want) and again, don't forget to share your thoughts too, about your experience while applying the 4 C's in your world! Let us know by using #21stskills4testers

EMNA AYADI
–

Emna Ayadi is a passionate software tester and detail oriented who loves to analyse root cause, test, collaborate with diverse people and investigate issues. She has five years of experience on different projects combined between testing and coaching roles. She appreciates delivering workshops about testing for her team and to the local community to make them aware about different trends in software testing and Ministry of Testing meetup organizer. Outside of work, traveling is her favorite pastime but when everyone is working .

ARD KRAMER

Ard is a software tester from the Netherlands. He works at OrangeCrest. He calls himself a Qualisopher which stands for "someone who loves truth and wisdom and at the same time is decisive to improve man and his environment". This means he is interested in the world around us, to see what he can learn and can apply in software testing. His dream is to participate, as a qualisopher, in all kinds of projects. Projects which add value to our community: he wants to inspire other people by cooperation, fun, and empathy, and hopefully some light in someone's life. .

COMMON TESTING MISTAKES - ARE WE REALLY EVOLVING?

Last year when I was working on creating the #TestFlix e-book with Sandeep Garg, we used to have a lot of candid discussions around Quality, Testing & Life in general. Sandeep told me about Jerry Weinberg during one such discussion. It was the first time I got to know about Jerry Weinberg. I started with reading his quotes, followed by some of his popular books like Secrets of Consulting & General System Design Thinking (still in progress :D).

A couple of months back I stumbled upon his book on Testing during a Happy Book Reading Club Session, i.e., Perfect Software & Other Illusions about Testing, which was released in 2008. At first, I thought that it might be outdated as we are in a technological era where things become irrelevant in months. Having an experience of Jerry Weinberg's books, I knew that there might be some good stories and incidents in this book. I decided to give this a read to enjoy the old stories and get a feel of the old days' testing issue(s).

I was enjoying the stories, incidents, and jerry's style of writing in this e-book until I came across a section of the book where Jerry talked about some Common Testing Mistakes. When I read that list, I felt all these are still the day-to-day problems for testers and/or testing.

I shared this post on LinkedIn that day and went to sleep.

The next day when I woke up, my phone was filled with LinkedIn notifications (Thanks to Michael Bolton for sharing this post and helping boost my post reach), and I understood that what resonated with me last night also resonates with a lot of testers out there. The post got 100 re-shares and was read by a great number of Testers, Developers, Managers, Consultants, etc. The comments on the post affirmed that these Common Testing Mistakes are still very common and highly relevant.

Over the last 2-3 decades, we have all seen the evolution of software testing tools, resources, processes, testing methodologies, and a lot of things. However, after witnessing 30+ years of software testing journey as an industry, if the mistakes are still the same then it posts a very serious question in front of us, i.e. Are we really evolving? and if not, then why not? I feel it's a lot because of getting caught up in the same old traps and ignorant behavior towards reality.

Here, I would like to elaborate on each of these Common Testing Mistakes, Traps, and Ignored Reality:

1. Thinking that locating errors can be scheduled

The only way we could know how long it would take to locate bugs would be to know where they are. If we know that, we wouldn't have to locate them at all.

Traps:

- Thinking of testing as a mechanical activity rather than a cognitive activity.
- Thinking that CI/CD/CT would eliminate the need for dedicated testing periods.
- Thinking that Green Automated Test Runs is equal to a quality product.
- Thinking that Automation will solve all our problems.

Reality:

- Testing is inherently exploratory.
- Scheduling locating errors is like Scheduling the arrest of criminals. NOT Possible for the majority of cases. Both need dedicated investigation, analysis, and evaluation.
- Testing is driven by Heuristics, which are fallible ways of solving a problem. They may fail too.

2. Not considering the time lost to task-switching

Task-switching can be beneficial, as we've seen, but like anything else, it has a cost. Each task switch loses a bit of time, so if you're switching among about five tasks, you may be accomplishing nothing. Most people react to that situation by simply dropping some of the tasks altogether, which can be dangerous.

Traps:

- Thinking of Multitasking as a productivity skill.
- Thinking of testing as a passive activity that can be clubbed with another activity.
- Accepting and acting on all the tasks as and when they around .
- Not planning your work.

Reality:

- If you multitask, you are bound to regret it, sooner or later.
- Responsible Testing requires a focused effort.
- Software engineering is a systematic application of engineering approaches to the development of software.
- If you fail to plan, then you plan to fail!
-

3. Treating testing as a low-priority task that can be interrupted for just about any reason.

Testing requires concentration if it's to be done reliably.

Traps:

- Thinking that anyone can do testing.
- Thinking of testing as checking.
- Measuring testing progress with the number of bugs.
- Thinking of testing as ONLY Verification.

Reality:

- Yes, just like cooking, anyone can do testing to some degree. But not everyone can do responsible testing.
- Quality is a multi-dimensional concept.
- The value of one bug could be sometimes greater than 50 other bugs. Bug count says NOTHING!
- Testing is NOT Verification, Testing includes Verification!

4. Demanding that testers pinpoint every failure.

Testers can help developers with this job if their time has been scheduled for it, but it is ultimately a developer's responsibility. At least that's what I've seen work best in the long run.

Traps:

- Thinking that testers should become developers and help pinpoint every failure.
- Thinking of problems as definite issues (Ex: True or False).
- Thinking that testers are mostly free and don't do anything else apart from test execution.

Reality:

- Testing & Development mindset are totally different. Testers are good at thinking about hypothetical issues (what if?) instead of Imperative thinking (do this!).
- Most problems are beyond the definite boundaries of Yes and No. They require human judgment and deeper investigation. Single observation can have multiple interpretations.
- Testing is NOT EQUAL to Test Execution. It involves a lot of other work like designing tests, preparing test data, studying and understanding requirements, writing bugs, tracking bugs, evaluating customer issues, etc.

5. Demanding that testers locate every fault.

This is totally a developer's job because developers have the needed skills. Testers generally don't have these skills, though, at times, they may have useful hints.

Trap:

- Thinking that all faults are straightforward.
- Many organizations fake such claims to customers while billing testers under the name of Full Stack, T Shaped, or Star Testers.

Reality:

- Most faults hide behind the multiple layers of abstractions of the software. They need an understanding of the different layers and knowledge regarding the flow and design of the software.
- If it sounds too good to be true, it's probably NOT true.

6. Repairing without retesting.

Repairs done in a hurry are highly likely to make things worse. If you're not in a hurry, you might be careful enough with the repairs not to need retesting, but if you're not in a hurry, why not retest?

Trap:

- Thinking that repairs have been done diligently and the testing phase can be bypassed.
- Developer Optimism
- Overconfidence on Unit Tests.
- Thinking that one more round of Testing might again destruct the product.

Reality:

- Most of the last moment fixes end up leaving side effects in the system.
- Over-optimism disillusions the reality!
- Unit tests are just small fact checks and nothing more.
- Testing never destroys anything except the illusions of overconfidence.

7. Ignoring cross-connections

Commonly, the actions of programmers drive the need for testing, so that testing and programming are bound together. For instance, if programmers deliver code to testers late or in sub-par condition, then you'll have to adjust test expectations.

Trap:

- Thinking of testing as an independent activity.
- Thinking of testing as fixed and defined.
- Ignoring the cost of bug analysis, investigation, advocacy, and retesting.

Reality:

- Testing is as much part of software development as development is. It's NOT independent.
- Testing is inherently exploratory and continuously unfolds in newer dimensions over time.
- Side activities around testing often take more time than test execution.

8. Paying insufficient attention to testability

Code that is designed and built to be testable can greatly reduce the time and effort associated with all aspects of testing.

Trap:

- Thinking that a magic framework would fix all testing challenges.
- Thinking that tools will amplify software testability.

Reality:

- The secret ingredient behind a good testing solution is a testable code.
- If testability is negative, tools would only amplify the negativity.

9. Insisting that all bugs be "reproducible"

Intermittent bugs need to be pursued with great vigor, not used as an excuse to delay testing or repairing. Use what information you have, and don't waste testers' time with unreasonable demands.

Trap:

- Thinking that each problem should be a result of some controlled parameters/configuration.
- Thinking that test system and reference system are equivalently configured.

Reality:

- During Testing, we only control some inputs and observe some outputs (results).
- Often the non-trivial seeming details like platform, runtime, supported libraries versions are not mentioned or ignored. Do you really know everything about your test system? Think again!

10. Confusing testing with "creating and executing test cases".

Much of the testers' work is not captured or encapsulated within the confines of an identifiable test case. Consider thinking instead in terms of test activities.

Trap:

- Thinking that Testing = Creating & executing test cases.
- Thinking of Testing to be revolving around test cases.

Reality:

- Testing is much more than test cases. Testing ≠ Test Cases.
- Good Testing involves many tasks like research about products, bug hunting, analyzing specifications, creating data sets, creating reusable tests, creating checklists, research failures, writing persuasive reports, etc.

11. Demanding process overhaul in your company

If you work in an organization that thinks it's the testers' job alone to pinpoint and locate bugs or fetch coffee and donuts for the developers, you may now be tempted to confront your boss with a crass, revolutionary attitude. Even if you're interested in increasing respect for testers in your organization, you're not going to change the environment with rude demands. You might say to your boss, *"I read that a tester is someone who tries different methods in order to make things work. I've come to suspect our testers could do better work if some things were different here. I'd love to have some time to discuss these things with you."*

Trap:

- Thinking that changes could be brought overnight.
- Thinking that new tools will solve all old problems.
- Trying to solve people's problems with technical tools.
- Looking for rational logic over reasonable logic.

Reality:

- Marvin's First Great Secret: Ninety percent of all illness cures itself - with absolutely no intervention from the doctor. Deal gently with systems that should be able to cure themselves.
- Most people can successfully absorb 10 percent into their psychological category of "no problem". Anything more would be embarrassing if you succeeded.
- No matter what or how they are saying, it's always a people's problem.
- If logic always worked, nobody would need consultants (or consultation). Consultants often come across contradictions.

After writing a lot about all the traps and reality, I would also like to share some references and must learn things based on my experience on how we can avoid these testing mistakes in our professional life:

- Learn and Master the Testing Foundations (Reference: BBST Foundations)
- Learn about Heuristics & Mental Models and why they matter (Reference: James Bach TTT Tribecast)
- Practice Time-Bound Charter Based Exploratory Testing (Reference: Explore It)
- Understand the importance of mindset for focused and good software testing (Reference: Blogpost on Mind, Matter, Testing and The Cargo Cult by Lalit Bhamare).
- Know that Testing is NOT Testcases (Reference: James Bach CAST 2014 Keynote)
- Learn and practice Bug Advocacy Skills (Reference: BBST Bug Advocacy)
- Learn about Consulting and some of its secrets (Reference: Secrets of Consulting)
- Learn Influential Writing (Reference: Blog post on Coaching testers how to email and influence stakeholders by Pradeep Soundararajan)

These topics and references are some of the most useful resources which have helped me to avoid many of the common testing mistakes and succeed as a professional tester. I hope that this would help you too in some way. If you feel there are more resources, which should be added to this list, please share your thoughts and feedback with me @parwalrahul.

All the best for your quality journey!

RAHUL PARWAL
–

Rahul is a Software Engineer by education and works with ifm engineering pvt. ltd., India. He is a Software Tester by trade, Programmer by practice, and a Mythology lover by heart. Rahul is a firm believer of Right Education and an ardent advocate of Open-Source mentality.
His latest ebook is available at leanpub.com/presentationheuristics
Twitter twitter.com/parwalrahul
LinkedIn: linkedin.com/in/rahul-parwal
Testing Blog: testingtitbits.blogspot.com

ras Shypka on Unsplash

HEURISTICS, WEB DESIGN

VIP BOA – A Heuristic for Testing Responsive Web Apps

EDUCATION, SPEAKING TESTER'S MIND

ISTQB, Fever Dreams and Testing

EDUCATION, WOMEN IN TESTING

How To Read A Difficult Book

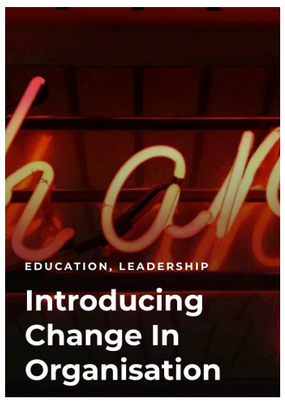
INTERVIEWS, OLD IS GOLD

Over A Cup Of Tea With Jerry Weinberg

mon Migaj on Unsplash

SPEAKING TESTER'S MIND

Software Testing And The Art Of Staying In Present



EDUCATION, LEADERSHIP

Introducing Change In Organisation

PEOPLE AND PROCESSES, WOMEN IN TESTING

Addressing The Risk Of Exploratory Testing Part 2

CAREER, PEOPLE

Managing Multiple Passions

LEADERSHIP, OLD IS GOLD

Leading Beyond The Scramble:

After nearly twenty years of working in software, I
many companies. One of them is what I call the sc

INFOCUS

Do you know all these amazing articles?

Great things survive the test of time.

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

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

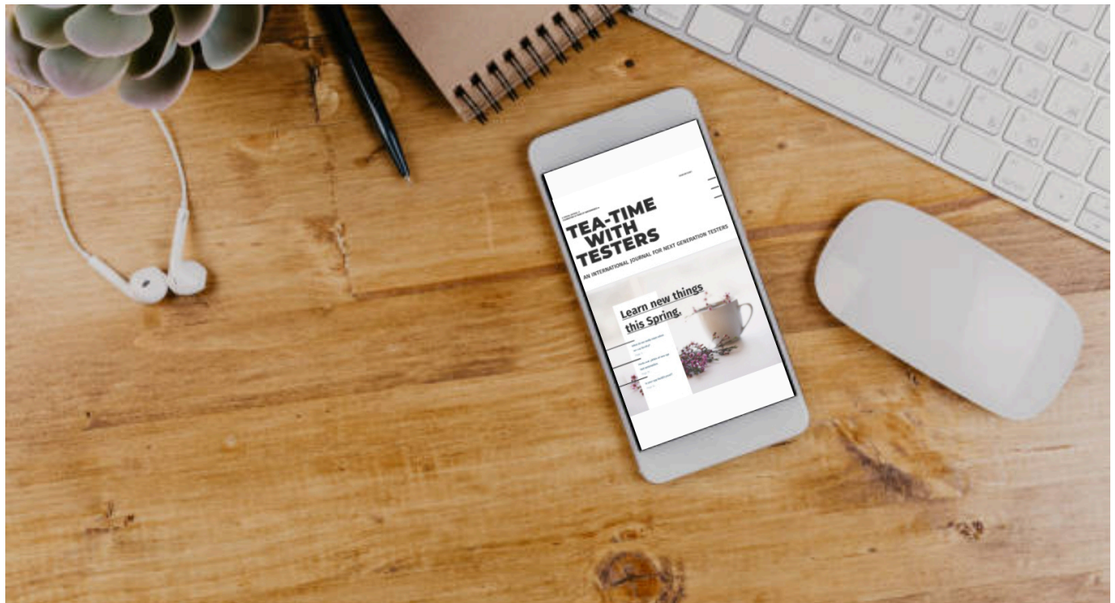
We plan to continue doing that for more articles, interviews and also for the recent issues we have published.

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

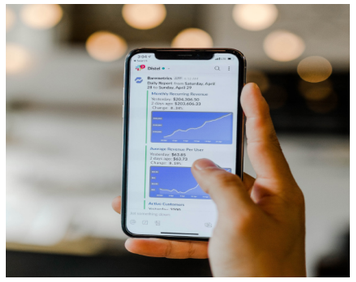
Let us know how are they helping you and even share with your friends and colleagues.

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

Enjoy the feast!



THE CHATBOT



QA has always been a specialized job, though not many people believe so, but it is for sure and has been proven all these years. The best of the developers cannot test and either they leave critical scenarios, or they leave testing the integration scenarios. To supplement the process and to ensure quality QA has been an utmost need of the hour these days for the products. All the QA in the world is specialized and to prove it we will see how challenging it is to make sure the chatbot works perfectly as it requires a lot of understanding on how a chatbot works its internals, various stack, algorithms, etc., and then generate scenarios to test the same.

However, if you do not know all that is mentioned above you can now see what's coming next, to develop an understanding of what things are required to make sure the chatbot is effectively tested. My expertise lies in automation and there are some questions which I will ask at the end of this article to make the reader think as to how we can achieve such things as they do not exist in the tools and are simple things that ensure maximum quality.

In today's chatbot QA most of the testing is a black box and is manual in the industry which the testers test the:

Tedious conversation flow of the users

Test small talk scenarios like Do you like people? or Do you know a joke?

Fallback checks if the chatbot can handle what it cannot handle

Integrations with APIs, Databases, Voice-based services, etc.

Moreover, the testers do not even know how the model works as the model engineers develop the underlying model and there is no further monitoring done by the QA. There is an analytics tool available to monitor but it needs technical expertise for the QA and QA needs to understand the internals.

The result 90% of the time the bot breaks and no one understands when it will break, most of the time the bot is stuck.

HOW TO EFFECTIVELY TEST

> "What's the deal with Chatbots?"

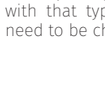
Soumya Mukherjee shares useful insights and tips to do it right. Read on to know more...

Before looking into the remediation let's see some more issues that face in doing the chatbot QA.

- One of the key issues is about continuous story creation as and when the bots are evolving. The bots are meant to converse like humans and there are limitations in QA to create or gather human behavioral data
- There are no tools to manage the story coverage which means that the testers do not even recognize that the stories are being missed and it keep on testing on the redundant data set
- The training data may not correspond to the new stories (end to end user chat sequence with the bot) which lead to the stale data being used each time. Most of the times the production data is never used to train the models leading to inaccurate results and faltering of the bot
- Most of the automation tools only offer record and playback and they keep on testing the same set of stories making it a pesticide paradox challenge which makes the bot always fail for the newer set of data and scenarios.
- In most cases, the stories are written in a text file and no automation tool can read through the existing story list and bring that intelligence into the tool.
- There is a common problem in QA which is the absence of a central dashboard to to check. From a chatbot context there is no generic dashboard that can trap the:
    - Intent matching
    - Check slots if they are filled as part of Entity testing
    - Entity validation
    - Check the confidence score of the model as when a test is executed
    - Confusion matrix which records the values of precision, recall, and F1-score
- There is no easy way to reset the bot and if that is stuck it is stuck during the conversation
- Multilingual bot QA is a challenge and companies have to build two separate bots most of the times
- Folks who dig deeper also find that higher the confidence score in all circumstances the bot is going to predict the same thing for multiple intents, and it will always predict with the one with the highest confidence score.

So, the question is how to make sure that the bot never breaks? Here are some ways to test your bot effectively:

- It is highly important to refresh the training data with the production data. If this is not done, then the bot will always be training on stale and out of context data for the evolving scenarios
- The QA needs to always create scenarios containing the happy path, contextual questions, digressions, domain-specific questions & stateless conversations
- QA needs to verify whether the entities are mapped for the scenarios. For example, if the scenario for school fees is considered the entities get wrongly mapped for bus fee or tuition fee, or any other fees
- Automated tests mostly API should consume all the stories and run them each time as part of the regression testing. The stories can be consumed from either directly from the stories file (Stories.md file in Rasa) or a repository where it is stored
- Story coverage visualization should always be part of the execution, which effectively shows how we are progressing with the testing and whether the tests can reach 100% story coverage. Although there are no story coverage tools available a graph database like neo4j can be implemented to store the stories and then the nodes can be traversed each time to run the stories against the BOT.



#	Story Description	#Stories	Coverage %
1	Happy Paths	28	89
2	Domain Specific	55	45
3	Digression Stories	80	25

- Most of the companies do not use Bot emulation platform for manual testing but tools like RasaX, BotFront can be used to visualize the execution even when the bot is in development
- One of the things which is very important, and which is to check the model accuracy with each conversation and then create a pattern to understand the rise and fall of the model along with:
    - Confidence score
    - Confusion Matrix including precision, recall, and f1-score
    - Cumulative accuracy profile
    - Cross-Validation results
- Exhaustive testing which checks bot resiliency is required to be part of the QA plan
- Integration checks with external database, services and most importantly all the webhooks are required to the part of the QA plan as well
- Fault tolerance testing by performing performance testing to verify bot response times, session management is required to be done. Most of the times during large volumes the bot response increased to a non-acceptance level and the session gets overridden which makes the entire bot infrastructure collapsed
- In case of live assistance, the handshaking and transfer of the flow needed to be checked as well
- One of the important aspects which the QA neglects is to perform various types of security testing. It has been seen that bots during a security assessment revealed a lot of information about user data that needs to be checked. Hence a level of security analysis on the APIs is needed to be performed along with that typing speed check, punctuations, and typo errors need to be checked.

Along with the above, there are a few other KPIs that are required to be tracked which are:

- Activity volume
- Bounce rate
- Retention rate
- Open sessions count
- Session times (conversation lengths)
- Switching of multiple stories in times of large conversations
- Goal completion rate
- User feedback (will be helpful for sentiment analysis as well)
- Fallback rate (confusion rate, reset rate, and the human takeover rate)

I am now sure that you would concur that chatbot testing requires much more effort to make it a specialized QA practice. With the set of standard practices, you can do much more effective testing on chatbots. In the later articles on this series, I will also discuss how you can make a story coverage tool for the chatbot and also, I would discuss some fancy automation tools which are there in the market and how it can be used to "EFFECTIVELY" drive your test automation for the "CHATBOT".

If you have any questions, feel free to reach out to me on Twitter (@QASoumya) or LinkedIn.com/in/mukherjeesoumya

SOUMYA MUKHERJEE
–
A passionate tester but a developer at heart. Having extensive experience of a decade and a half, doing smart automation with various tools and tech stack, developed products for QA, running large QA transformation programs, applied machine learning concepts in QA, reduced cycle time for organizations with effective use of resources, and passionate working in applied reliability engineering. Love to help others, solve complex problems, and passionate to share experience & success stories with folks. Authored books on selenium published by Tata McGraw-Hill's & Amazon.

A father of a lovely daughter.

MACHINE LEARNING FOR TESTERS - PART 1

Are you starting to see Machine Learning and AI more and more as a necessary or useful skill in job requirements and bewildered by it? Want to know more but don't know where to start?

Grab your starter kit by Paul Maxwell-Walters

There is probably no area of our lives these days not touched in some way by machine learning. Applications cover such wide areas as translation, speech recognition, forecasting, fraud detection, search engines, medical diagnosis, the financial markets, DNA sequencing and weather prediction for agriculture. The breadth of its potential applications is almost as large as the breadth of information technology itself. For this reason and others testers will be expected to have more (at least) conceptual knowledge of machine learning in the future.

This is an essay and tutorial-based version of a talk on Machine Learning I did for Sydney Testers Meetup Group in March 2021. I am a strong believer that, while machine learning and AI does require some undergraduate level computer science and mathematics capability to understand well, it is not beyond the capacity of most testers to learn enough to get to a point to at least conversing with data scientists and being "on the same page". In this regard I have tried to write an article that allows testers to understand the basic concepts.

What is Machine Learning?

"The use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data..."

Definition from Oxford University Press/ Lexico

The difference between Machine Learning (ML) and other algorithmic methods in computer science is the idea of an application using historical or example data to optimise its actions in some way without instruction. Often some sort of specified inputs and outputs are provided and a program asked to find relationships between them, however it is not obligatory and there are ML approaches where the computer develops its own relationships, sometimes with a stated overall goal and some limited feedback.

One point to be made is that Machine Learning and Artificial Intelligence, whilst usually considered synonymous in the media, are actually different things with some overlap. Machine Learning bases much of itself on statistics, an area not considered a part of AI, while AI includes areas not considered part of ML, such as expert systems and inductive logic.

The Machine Learning Timeline

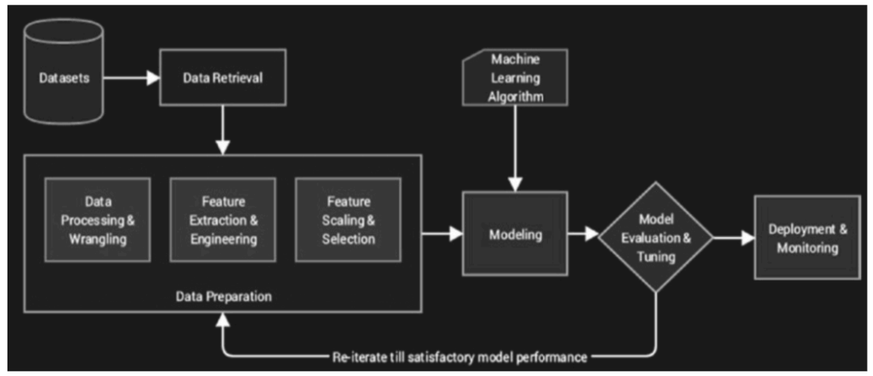
Since it has its roots in the history of statistics, ML can be said to date back to long before the dawn of modern computers, however most people treat ML as having started in the 1950s. Below is a timeline of important events in ML.

- 1805 - Adrien-Marie Legendre develops the Least Squares Method and thus Linear Regression
- 1951 - Marvin Minsky and Dean Edmonds create the first "neural network" machine. Evelyn Fix and Joseph Hodges create the k-Nearest Neighbour Algorithm
- 1957 - Frank Rosenblatt invents the Perceptron, the basis for all modern neural networks
- 1959 Arthur Samuel coins the term "Machine Learning" for the first time
- 1970-1982 - Backpropagation and the precursors to Convolutional Neural Networks and Hopfield Networks developed
- 1997 - IBM Deep Blue beats Gary Kasparov at Chess
- 2012 - Andrew Ng and Google Brain develop Neural Network to detect cats from unlabelled YouTube images
- 2016 - Google's AlphaGo beats a professional human player at Go for the first time

The Machine Learning Pipeline

Machine Learning in most places should be considered a process of data retrieval and manipulation. Then "features" (data variables) that can be modelled are extracted, filtered and put into a structure with metadata and parameters known as a "Model".

This is used to generate some sort of predictive output or classify some data, which is then compared to an expected output. In the case of differences the model parameters are amended and optimised (a process known as "training" or "learning") and the process repeated until the model output matches the expected output adequately.

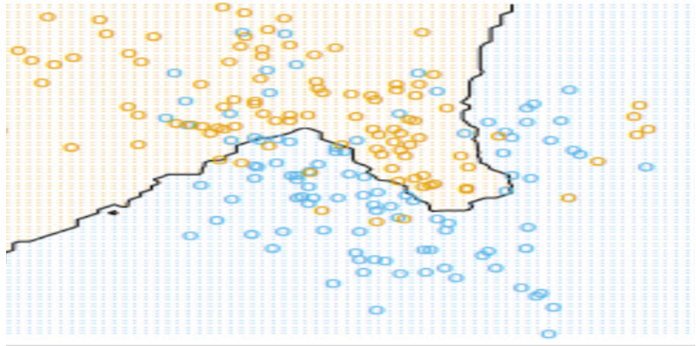


Being a specialised field, the terms of ML should be introduced now since they will be referred to during this essay. They are not difficult to understand at a high level.

- **Supervised Learning** - Computer given specified inputs and outputs and find relationships between inputs and outputs.
- **Unsupervised Learning** - No labelled outputs given, computer has to come up with its own relationships from the input data.
- **Reinforcement Learning** - Computer reacts with a dynamic environment in which it must perform a certain goal. Given "rewards" as feedback, which it has to maximise.
- **Model** - weights and parameters used by the computer to represent an assumption about the relationship between input and output data.
- **Training** - Comparing the model to the output and then optimising it to reduce errors.
- **Feature** - Some variable defined in the input data (i.e. size, colour, age etc.)

In addition to the above, ML can be usually broken down into three types.

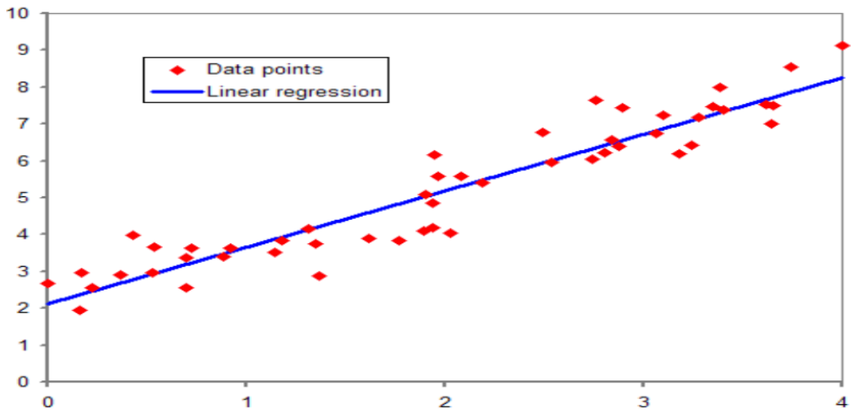
Classification - Estimating which category something belongs to based on a dataset of data already labelled into categories (i.e. classifying animals into dogs, cats, rabbits etc).



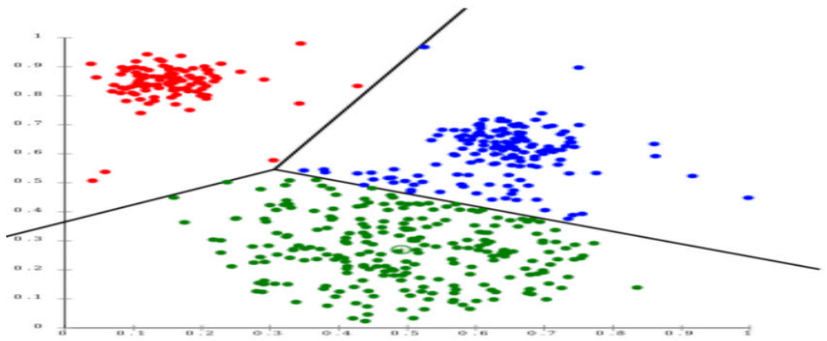
PAUL MAXWELL-WALTERS
–
A British software tester based in Sydney, Australia with about 10 years of experience testing in agriculture, financial services, digital media and energy consultancy. Paul is a co-chair and social media officer at the Sydney Testers Meetup Group, along with having spoken at several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

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

Regression - Estimating the relationship between an input variable and an output variable



Clustering - (Unsupervised) Grouping items into groups with similar properties, without labelled data to learn from.



On Attribution and the use of Mathematics

It is not possible in my view to write an overview of machine learning that is adequately explanatory without mathematics, so I have had to include some equations. Some knowledge of algebra and calculus will prove useful when reading this. However I have tried to keep them to a minimum.

For the images and equations in my examples I have often used Wikipedia. This is because they have either public or creative commons attribution. Where no mention was made it is taken from Shervine Amidi's Stanford Super ML Cheatsheet https://stanford.edu/~shervine/.

In all other cases I have included attribution as appropriate.

Machine Learning Techniques

Regression

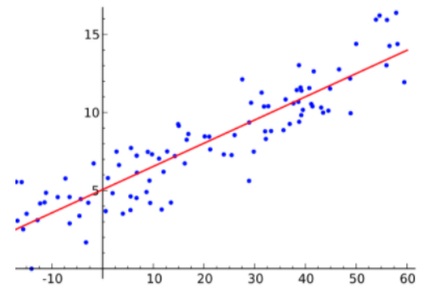
Regression is a technique where one has some sort of numerical input and output data and fits a trendline or curve to it.

Linear Regression - The Most Basic ML Approach

(images, equations and example taken from https://en.wikipedia.org/wiki/Simple_linear_regression)

The most basic technique is Linear (Ordinary Least Squares) Regression, taught in most school statistics classes.

Consider a series of x,y points on a 2 dimensional plane that we wish to model using a straight line.



Consider a one dimensional model (where alpha is the intercept across the y axis and beta is the gradient of the trendline) -

$$y = \alpha + \beta x,$$

..or if we expand to a number of data points i and include the random error term ε

$$y = \alpha + \beta x,$$

Find the optimum gradient β and y-axis intercept α such that ε is a minimum.

$$\text{Find } \min_{\alpha,\beta} Q(\alpha,\beta), \quad \text{for } Q(\alpha,\beta) = \sum_{i=1}^{n} \hat{\varepsilon}_i^{\,2} = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2 \,.$$

In order to find the minimum we need to solve for optimised gradient β and y-axis intercept α -

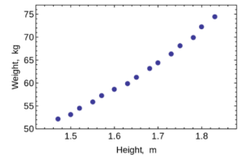
$$\begin{aligned}
\widehat{\alpha} &= \bar{y} - (\widehat{\beta}\,\bar{x}), \\
\widehat{\beta} &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\
&= \frac{s_{x,y}}{s_x^2} \\
&= r_{xy}\frac{s_y}{s_x}.
\end{aligned}$$

- $\bar{x}$ and $\bar{y}$ as the average of the x_i and y_i, respectively
- r_{xy} as the sample correlation coefficient between x and y
- s_x and s_y as the uncorrected sample standard deviations of x and y
- s_x^2 and $s_{x,y}$ as the sample variance and sample covariance, respectively

$$r_{xy} = \frac{\overline{xy} - \bar{x}\bar{y}}{\sqrt{\left(\overline{x^2} - \bar{x}^2\right)\left(\overline{y^2} - \bar{y}^2\right)}}.$$

A famous example of this is that of Height vs Mass for a sample of American women.

Height (m), x_i	1.47	1.50	1.52	1.55	1.57	1.60	1.63	1.65	1.68	1.70	1.73	1.75	1.78	1.80	1.83
Mass (kg), y_i	52.21	53.12	54.48	55.84	57.20	58.57	59.93	61.29	63.11	64.47	66.28	68.10	69.92	72.19	74.46



In my original talk I demonstrated this calculated on an Excel spreadsheet as follows,

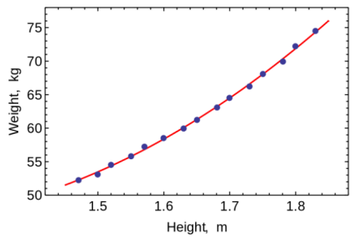
	A	B	C	D	E	F	G
1		x	y	x-mean(x)	y-mean(y)	(x-mean(x))^2	(x-mean(x)) x (y-mean(y))
2	1	1.47	52.21	-0.180666667	-9.868	0.032640444	1.782818667
3	2	1.5	53.12	-0.150666667	-8.958	0.022700444	1.349672
4	3	1.52	54.48	-0.130666667	-7.598	0.017073778	0.992805333
5	4	1.55	55.84	-0.100666667	-6.238	0.010133778	0.627958667
6	5	1.57	57.2	-0.080666667	-4.878	0.006507111	0.393492
7	6	1.6	58.57	-0.050666667	-3.508	0.002567111	0.177738667
8	7	1.63	59.93	-0.020666667	-2.148	0.000427111	0.044392
9	8	1.65	61.29	-0.000666667	-0.788	4.44444E-07	0.000525333
10	9	1.68	63.11	0.029333333	1.032	0.000860444	0.030272
11	10	1.7	64.47	0.049333333	2.392	0.002433778	0.118005333
12	11	1.73	66.28	0.079333333	4.202	0.006293778	0.333358667
13	12	1.75	68.1	0.099333333	6.022	0.009867111	0.598185333
14	13	1.78	69.92	0.129333333	7.842	0.016727111	1.014232
15	14	1.8	72.19	0.149333333	10.112	0.022300444	1.510058667
16	15	1.83	74.46	0.179333333	12.382	0.032160444	2.220505333
17	sum	24.76	931.17			0.182693333	11.19402
18	mean	1.650667	62.078				0
19							
20	**beta = sum((x-mean(x)) x (y-mean(y)) / sum(x-mean(x))^2**					**61.27218654**	
21	**alpha = mean(y) - beta*mean(x)**					**-39.06195592**	

This results in a regression equation that describes the Heights vs. Weight data as - **y = 61.272x - 39.062**

Non-Linear Regression

(Image and Equations from https://en.wikipedia.org/wiki/Ordinary_least_squares)

Maybe we decide that the linear regression trend is best described as a curve.



We can apply a curve trendline by using a quadratic function with a new term h2 for height.

$$w_i = \beta_1 + \beta_2 h_i + \beta_3 h_i^2 + \varepsilon_i.$$

!Which a statistical package can be used to reveal the following -

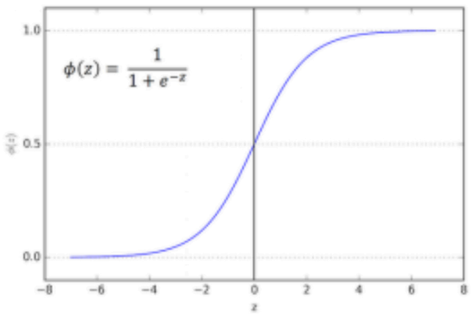
Method	Least squares
Dependent variable	WEIGHT
Observations	15

Parameter	Value	Std error	t-statistic	p-value
β_1	128.8128	16.3083	7.8986	0.0000
β_2	–143.1620	19.8332	–7.2183	0.0000
β_3	61.9603	6.0084	10.3122	0.0000
R^2	0.9989	S.E. of regression		0.2516

In this way we can extend a linear regression approach to finding non-linear (polynomial) data relationships.

Logistic Regression

Regression can be used for binary classification (i.e. between two classes) if a non-linear separation function (known as the logistic or sigmoid function) is used.



This is trained using a statistical technique called the Maximum Likelihood estimation. It is used for classifying loan customers to those likely to default, categorising voters by political party etc.

The results is a binary output where the probability "p(i)" of True/ False, Will Default/Will Not Default, Spam / Not Spam is a logistic function where z is the linear sum of data points multiplied by weights along with a bias beta 0).

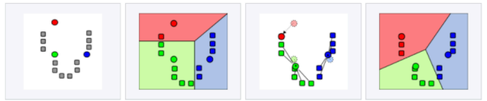
$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i})}}.$$

Clustering

Clustering is an unsupervised learning technique where a computer categorises a set of disparate data points into "clusters" related to some property the data has. It is commonly used in image recognition, computer science and astronomy.

K-Means Clustering

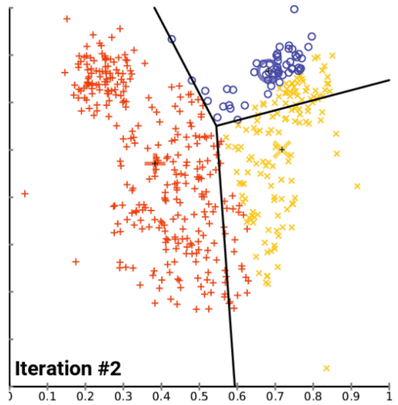
The most common of these, k-means clustering, calculates the mean between points and optimises this until the mean between points converges. It thus classifies the data into a number k clusters



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).
2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.
3. The centroid of each of the *k* clusters becomes the new mean.
4. Steps 2 and 3 are repeated until convergence has been reached.

By Weston Pace, https://en.wikipedia.org/wiki/K-means_clustering

A famous example of this is the (k=3) "Mouse" dataset.



By Chire, https://commons.wikimedia.org/wiki/File:K-means_convergence.gif

Classification

Classification is a supervised learning process of categorising some data into predetermined sub-populations based on some feature of that data. These could be binary (i.e. spam/not spam), real valued (i.e. length) or categorical (i.e. colour, blood type, type of voter).

Decision Trees

The most intuitive classification method is the decision tree.

The classic example of this taught at undergraduate level is that of restaurant customers. Because of this, I used data and images attributed to Alan Blair's AI course at UNSW. In this case, we classify them into whether they are likely to have to wait to be served or not.

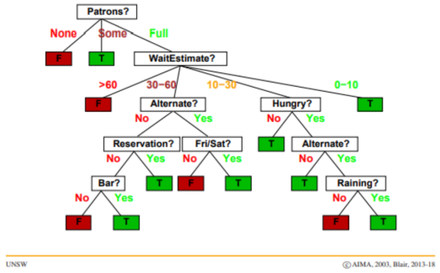
Restaurant Training Data

	Alt	Bar	F/S	Hun	Pat	Price	Rain	Res	Type	Est	Wait?
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

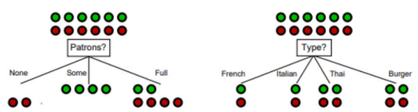
UNSW ©AIMA, 2003, Blair, 2013-18

The above table can be delineated into a tree-like structure known as a Decision Tree where each node is a possible option represented in the data above.

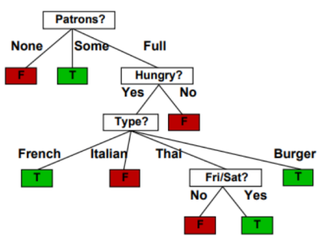
Decision Tree



In order that anyone traversing the tree can get to a solution quickly we need to choose and split attributes in such a way that the tree is as small as possible. In this vein, attributes that split data as fully as possible into sets of one type or another are seen to be more "informative" and "orderly". The measure of lack of order in a tree is called "entropy" and our aim is to reduce it at every step.



Splitting by "patrons", since it produces two branches with data of just one category, results in lower entropy than for Type.



PART 2 IN NEXT ISSUE

SHIFT-LEFT MET EARLY MODEL BASED TESTING

Nowadays more and more test activities are being automated, especially when it comes to executing the test cases and checking the results in the system under test. But what about the development of the test cases, the test case design? Are the test cases still developed in a structured way with a certain test coverage based on a well-considered risk? And is the impact on the test cases clear after a change in the test basis? And what about the Shift-left testing approach if we are mainly focusing on automating the test execution and checks? I think, with Model Based Testing (MBT) we already can answer many of these questions. In this blog I would like to give my vision and introduce the testing approach early Model Based Testing (eMBT).

Model Based Testing

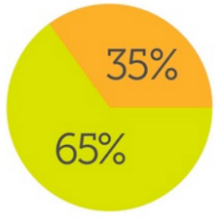
It is well known that Model Based Testing (MBT) has many advantages over other test approaches and that MBT offers a solution to automate the developing of test cases. Very important, because in the present there is no time anymore to manually set up the test cases and work them out using the traditional and formal test specification techniques. Despite the limited time, the goal remains unchanged; develop test cases with a certain test coverage in a structured way for the purpose of test execution.

If we, as testers, want to keep up with the ever-faster development cycles and still want to deliver the same test quality, we also need to automate the preparation of the test cases in the preparation phase. So, it is nice that MBT can help us to automatically generate the test cases from a model. And with any changes to the test basis, the maintenance is minimal. Only the model has to be updated and the test cases can be automatically generated again and again.

Shift-left test approach

Furthermore, MBT fits within the Shift-left test approach, which means that the test activities must start earlier in the Software Development Life Cycle (SDLC). By applying MBT you prepare a model in the test preparation phase and then derive the test cases from that model. With this you Shift all test activity (s) to the left. By start testing earlier, we ultimately ensure that we can provide feedback earlier and that we discover any bugs earlier in the process. Ultimately, any bug we find earlier is cheaper to fix, as also described in the well-known "Law of Boehm".

The question we must ask ourselves now is whether we will start our testing activities early enough, even if we use a Shift-left test approach and / or MBT. Especially when we realize that about 35% of all bugs in production can be traced back to the requirements. To avoid these bugs, we will therefore have to start testing the requirements (static test)! Ultimately, the requirements are also the basis for developing and testing the desired software. So, we need to make sure that this foundation is complete and clear and all stakeholders have the same understanding of it before we even start writing the first lines of code and testing it. Otherwise, we know for sure that the first bugs in the code will be introduced soon.



**Aditi Kulkarni, Global Assets Engineering Lead, Accenture – Software Intelligence Conference 2021. (According to their data based on 1000 projects)*

Early Model Based Testing (eMBT)

As described above, MBT fits perfectly within the Shift-left test approach. However, MBT is often started too late or used in a way without the approach of testing the requirements, but purely to generate automated (executable) test cases. We will therefore have to apply MBT in a specific way, at the earliest possible stage. I call this test approach early Model Based Testing, eMBT for short. By applying eMBT you will break down the requirements at the earliest possible stage by modeling them by means of a so-called eMBT-model. Such an eMBT-model has a high level of abstraction and by drawing it up you will quickly encounter ambiguities, contradictions, open ends, questions, etc. that you can then discuss with the stakeholders. When drawing up an eMBT-model, as a tester you will think differently about the requirements, and you will think more as the final customer. You are currently testing the requirements in an exploratory-like way and already give meaningful and early feedback, namely feedback on the basis. In addition, an eMBT-model is a user-friendly visual representation of the desired situation and is readable for all stakeholders. So not a technical and difficult to read model, as we often encounter within MBT. The user-friendly visual representation of an eMBT-model stimulates the communication and collaboration between all stakeholders with the aim of achieving a shared understanding of what needs to be built and therefore also tested.

Tooling

The eMBT approach does require a tool that supports this approach. For example, the tool must offer the possibility to draw a model with a high level of abstraction, the eMBT-model. This not only increases the readability of the model, but by drawing the eMBT-model you, as a tester, are also forced to think about both the happy and non-happy flow. Furthermore, the tool should offer the possibility to include the questions and comments you have, in the model itself.

Example

Below an example of a type of eMBT-model, based on the following requirements.

Requirements calculator app

If you open the calculator app, the calculator standard shows 0. You can make the following calculations:
1+1= with result 3
1+2= with result 3

1+4= with result 5
You can close the calculator app by clicking on the button 'close'

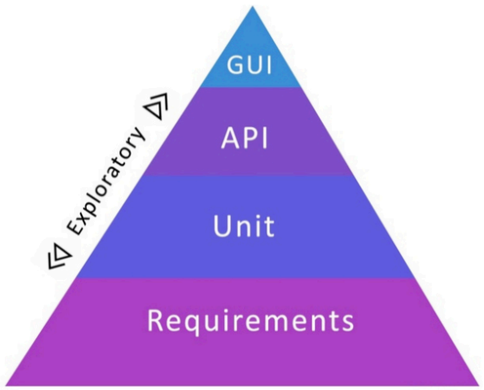
As you can see in the figure above, the eMBT-model is based on the principles of a flowchart, but with some specific conditions. This eMBT-model is not only easy to set up, but also clearly readable for everyone. Only three relevant nodes (action / status, decision and result) are used and furthermore, all questions / uncertainties etc. can be included directly in the eMBT-model via a separate node (balloon). As soon as all questions have been answered, all uncertainties have been removed and all stakeholders have the same understanding of what needs to be built, the eMBT-model is ready and the test cases can be generated automatically based on a pre-selected test coverage. These test cases then can be performed manually or being automated for the automated test.

To summarize

More and more automation is being done within the test process, especially when it comes to test execution and checks. But what about developing the test cases? Are we still doing this in a structured way and should we not automate this in order to keep up with the short development cycles? This is possible with Model Based Testing (MBT). MBT is now a well-known and proven test approach that offers many advantages over other test approaches. In addition, MBT fits within the Shift-left test approach because you can use MBT early in the process. However, we often see that technical models are used to derive the test cases and that the preparation of the model is started too late and therefore not used as a static test on the requirements. The technical model also does not stimulate the communication between the stakeholders, which is precisely so important at the start of the process.

early Model Based Testing (eMBT) can be the solution for this. eMBT is a software testing approach which starts at the very beginning of the SDLC with early feedback as an important goal. It supports a lot of important parts within the test process, such as collaboration, exploratory, study the requirements, determine risks, communication about the test basis and test object, determine test coverage, test case design and modeling. By using the right eMBT approach and tooling to start testing the requirements at the right time, unnecessary bugs are discovered at a very early stage. In addition, with this eMBT approach we quickly achieve a shared understanding of what needs to be built and tested, even before a line of code is written. If the requirements are clear and complete for all stakeholders, with one click you can automatically generate the test cases.

In the next figure an addition to the well-known test pyramid with the eMBT approach, in which testing starts with the (automated) testing of the requirements, the bottom layer.



If you would like to know more about the approach of early Model Based Testing (eMBT) or about eMBT tooling, please let me know.

SILVIO CACACE
–
Silvio Cacace is an experienced and passionate test professional with more than 26 years of practical experience. He has experience in manual testing and test automation within both traditional and Agile development processes and is the Founder of the Agile testing method APT® & DTM and the early based Model Based Testing tool DTM tool and TestCompass® https://www.linkedin.com/company/testcompass (www.compass-testservices.com)

ISSUE 02/2021
COMMUNITY

COMMUNITY

FailQonf. Story of Failebration by The Test Tribe.

On the 5th and 6th of June, The Test Tribe Community hosted FailQonf, a Conference dedicated to cover Failure Stories with Lessons around Software Testing and Quality.

FailQonf had a stellar Speaker Lineup deliver 18 Talks, an Expert Panel, a Fireside Chat, and an open space Failebration session.

FailQonf had close to 1200 Registrations and around 800 attendees from over 55 countries. It received a pouring love from all the attendees and the Average Event Rating of 4.84/5 and Average Session Ratings of 4.87/5 reflected the same.

If that's not enough, FailQonf Speakers were also interviewed before the Conference over many interesting questions. You can read them all, here- thetesttribe.com/tag/failqonf/

And, of course, you can re-live the FailQonf experience via this Roundup post - https://www.thetesttribe.com/failqonf-event-roundup/

TEA-TIME WITH TESTERS

JOURNAL FOR NEXT GENERATION TESTERS

CONTENT PREVIEW : ISSUE 03/2021

WILL YOU WAIT LIKE A CHILD ON CHRISTMAS?

01

MACHINE LEARNING FOR TESTERS - PART 2

Learn more about Machine Learning and Testing in this exclusive series. Paul has a lot more under his hat to share with us all. Hang in tight....

02

TEST CASE IS DEAD. LONG LIVE TEST CASE.

"For some time now and in light of my various readings, I have the feeling that the concept of test cases is evolving. Sometimes I feel like it's seen as a ball dragged underfoot, sometimes like the Holy Grail, sometimes it just doesn't exist. So what is the test case in today's software world?" .. Hear more from Benjamin Butel in this interesting article.

03

TEA AND TESTING WITH JERRY WEINBERG

We are bringing back the treasure of knowledge that Jerry Weinberg has left behind for us. Stay tuned for more...

INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

TEA-TIME WITH TESTERS

THE SOFTWARE TESTING AND QUALITY MAGAZINE

Created and Published by:

Tea-time with Testers.

Schloßstraße 41, Tremsbüttel

22967, Germany

Editorial and Advertising Enquiries:

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

Be with us and visit our website:

www.teatimewithtesters.com