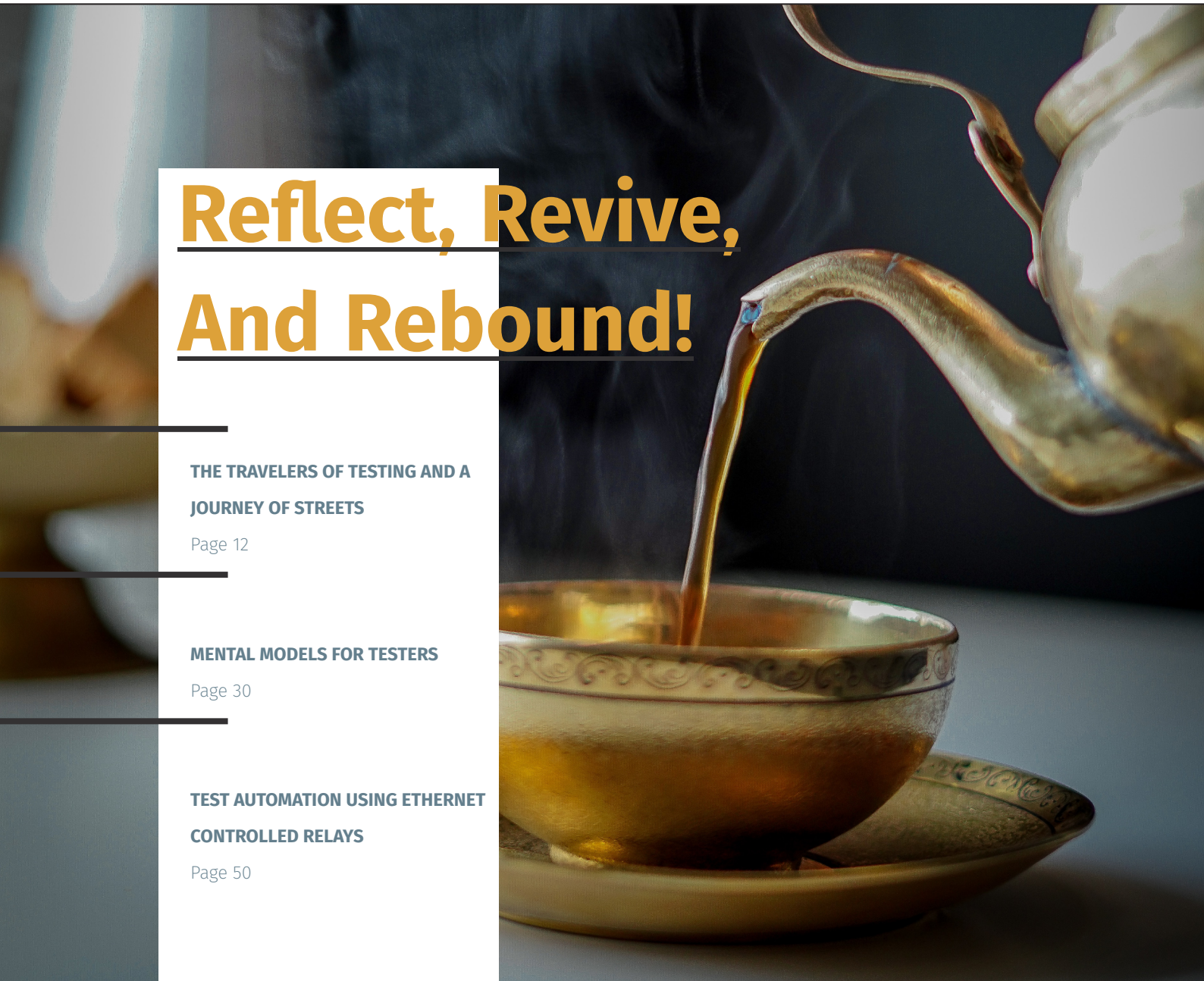
WAKING TESTERS UP SINCE 2011

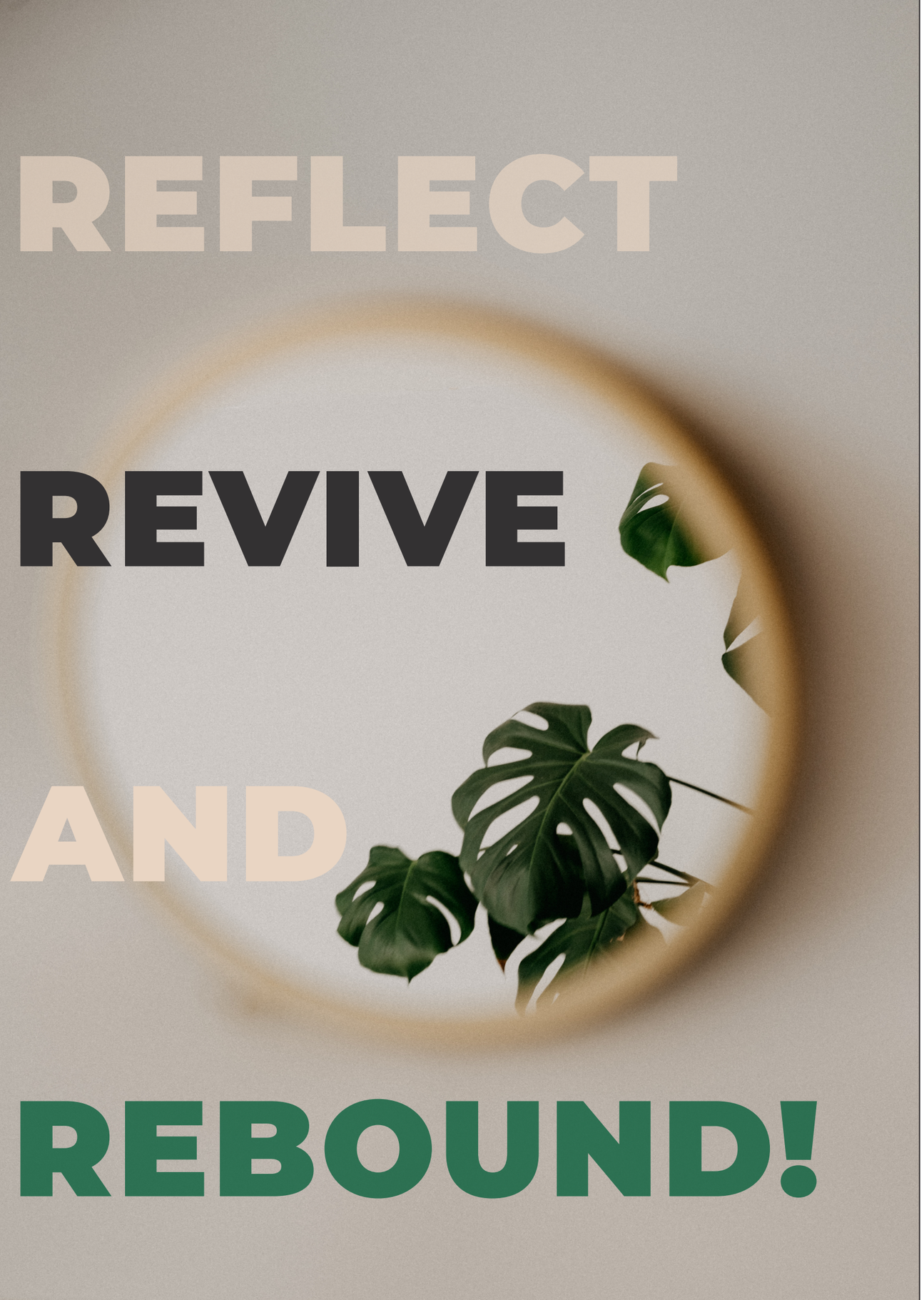
ISSUE #03/2023

TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS



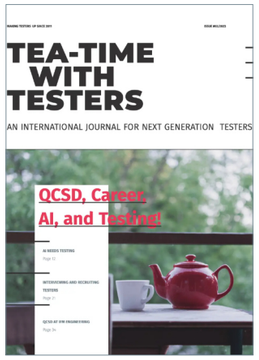
Reflect, Revive, And Rebound!



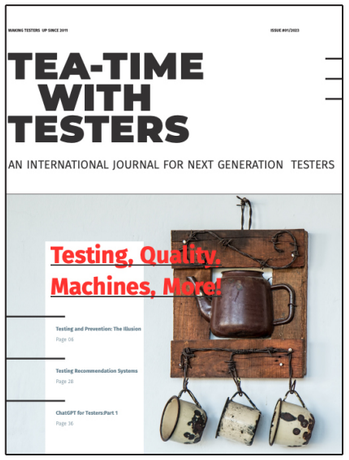
TEA-TIME WITH TESTERS

EDITORIAL BY LALIT

INTERVIEW: 22-24
A CUP OF TEA WITH GÁSPÁR NAGY



TEA-TIME WITH TESTERS



A NEXT GENERATION TESTING MAGAZINE

GenerAIting intrinsic motivation...

If you are reading this, it means a couple of things:

- Someone asked or encouraged you to read Tea-time with Testers.
- You searched for articles and news around software testing and found us.
- You are already a regular reader of the magazine.

Regardless of what your reasons are, I want to first congratulate you for doing the right thing. “What’s so rewarding about reading Tea-time with Testers, Lalit?”, you may ask! Well, while it is quite rewarding to read this magazine, I want to admire your “drive” and the “passion” for testing that brought you here.

In my recent interview, I was asked what differentiates an exceptional tester from good tester. And it was really not an easy question to answer. Considering the vast nature of Software testing as an intellectual field of work, there is not just one thing that could make someone exceptionally good. But the more I come to think of it, the more I realize that there is this “one” thing, that can help one become exceptionally good. What do you think it is?

“Intrinsic Motivation” is the answer. It refers to one’s inner drive and sincere passion that makes them pursue things for the sheer joy of fulfillment and satisfaction. It is the desire to engage in a task or activity because it is inherently rewarding, rather than for external rewards or pressures. I cannot imagine an individual to become exceptionally good at things without intrinsic motivation to reach there.

During my conversations, I often hear people talking about having a life beyond their 9-6 jobs. They have hobbies to pursue, family to spend time with, kids to raise, and anything else that they truly enjoy doing. And that’s an important aspect to consider too. However, I just do not believe that an intrinsic motivation necessarily means “extra work” or “over-time” or “grinding yourself towards a burnout”. I firmly believe that one can always make time for things they are truly passionate about. It’s all about managing your time and resources smartly, and being honest with yourself.

Sometimes people ask what will they get if they go extra mile to achieve something? I often urge them rather not to do it. Extrinsic motivation like money or praise is not sustainable as much as intrinsic motivation is. Intrinsic motivation prepares us better to handle challenges, setbacks, and obstacles because our drive comes from within. Daniel H. Pink in his bestselling book “Drive:The Surprising Truth About What Motivates Us” discusses this topic much in detail. I strongly recommend our readers to give it a read.

In fact, why to go that far? Simply reading the articles published in Tea-time with Testers might tell you about the “intrinsic motivation” of all our contributors. Or feel free to talk with any of our team members who have volunteered to support this community project.

Steve Jobs said, "Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work." To add to that I would say, your intrinsic motivation will help you do that great work.

Understanding the inner drive can help you reach higher levels of motivation, better performance, and greater satisfaction in work and life.

Give it a try before AI generates it on your behalf. Pun intended.

Sincerely,

Lalit

LALITKUMAR BHAMARE
CEO, Chief Editor “Tea-time with Testers”
–
Manager - Accenture Song, Germany
Director - Association for Software Testing
International Keynote speaker.
Award-winning testing leader.
Software Testing & Quality Coach.

Connect on Twitter @Lalitbhamare or on LinkedIn

AVO Automation™

TTWT SPONSOR OF THE YEAR 2023

Deliver Top quality Software with Test Automation

Avo offers 100% no-code test automation for:

Web | Mobile | API | Desktop and Citrix | Mainframe | SAP ORACLE salesforce and other packaged applications | and many more

AVO Assure

Avo Assure for functional testing:

Top features:

- In-sprint automation
- CI/CD Integration
- Smart scheduling and Parallel execution
- Upgrade Analyzer
- Intelligent reporting and dashboards

You achieve:

- >90% test automation coverage
- Improve team productivity by 2x
- Test 85% faster than manual testing
- Reduce production defects to <2%

AVO iTDM

Avo iTDM for next generation test data management:

Top features:

- Data discovery
- Data obfuscation
- Synthetic data generation
- Data provisioning
- Multiple security options

You achieve:

- Non-compliant data in non-production environments
- Data privacy regulations like GDPR, CCPA, and other compliances
- Compliance with continually evolving privacy regulations

Recommended by Industry Experts

SAP · ORACLE · salesforce · Capterra · IDC · G2 · Momentum Leader WINTER 2023 · Gartner · GetApp · GUIDEWIRE

seastarconf

OCTOBER 15-19 2023

Lalit Bhamare

Technical Arch Manager at Accenture Song, Germany

2-Day Certification Training

QUALITY-CONSCIOUS SOFTWARE DELIVERY

https://quality.seastarconf.com

USE PROMOCODE "ILOVETTIME" TO GET **10% OFF TICKET PRICES**

JAMBHAVAN & HANUMAN

3 LEADERSHIP LESSONS FOR TESTERS

- PRASHANT HEGDE

Prashant Hegde is a empathetic leader who finds fulfillment in helping others succeed. Prashant enjoys sharing his experiences by blogging and contributing to software testing communities worldwide. Prashant heads the testing unit at MoEngage, a leading insights-led customer engagement platform. Prashant is an avid blogger and a frequent speaker at software testing conferences.

LinkedIn: https://www.linkedin.com/in/prazhegde/
Blog: https://www.prashanthegde.biz/articles

INTRODUCTION

In today's technologically advanced world, leadership is crucial to a business's success. New generations of leaders are taking over the torch from established leaders, ready to make their mark. As your testing career advances, it eventually leads to more responsibilities, including management and leadership. A testing leader's role goes beyond being an excellent tester. As a testing leader, you are responsible for the success and well-being of your team. When testing leaders take on leadership roles, they usually do not receive any guidance or training on leading their teams. Whether you are a new testing manager, a seasoned one, or an informal leader, you will benefit from reading this article. What does true leadership mean?

Let me share a story with you.

In the epic Ramayana, there is an inspiring story about leadership that I often share with testing leaders. I would like to share this with the readers of Tea Time with testers. Let us set aside the spiritual part of this anecdote and focus on the leadership lessons it offers.

The story of Jambhavan and Hanuman

Rama learns that the demon king Ravana has abducted his wife Sita and is holding her captive in his island kingdom Lanka. To get Sita back, Rama allies with Vanaras (half-ape, half-human creatures). Rama seeks help from the Vanara army to confirm Sita's whereabouts..

The Vanaras found themselves constrained when they realized that they had to cross an ocean from the southern coast of India to reach Lanka to find Sita. Hanuman stands quietly in the corner as the vanara search party discusses flying across the ocean. He was one of the ministers in the Vanara army. Hanuman devoted his life to Rama and sought to help him in any way he could. It's important to note that Hanuman at the time was a different character from what we know today. His courage, strength, and bravery were still unknown to the world. As Hanuman stood on the shore, gazed out at the seemingly limitless ocean, and contemplated the impossible, a bear-like man emerged from the shadows. He was Jambhavan, the wise and experienced mentor of Vanaras.

Jambhavan motivates Hanuman to leap across the ocean. Jambhavan tells Hanuman that he's the only one in the entire Vanara army who can do this seemingly impossible thing. A sense of doubt and uncertainty began to creep into Hanuman's mind. "How could I possibly cross this colossal ocean?" Jambhavan calmly reminded Hanuman of his abilities. Jambhavan said, "Do not allow this vast ocean to intimidate you. You possess unimaginable strength within you". Jabhavan recalls the time when Hanuman was a child and wanted to swallow the sun, believing it to be a mango. Jambhavan recalled how Hanuman could fly and reach outer space as a child, triggering chaos among the gods. As Jambhavan builds Hanuman's confidence, he convinces him that crossing an ocean shouldn't be a problem for someone capable of reaching outer space. Hanuman was grateful to Jambhavan for seeing his true potential and giving him the confidence he needed. Hanuman took a deep breath and jumped across the ocean with unwavering faith. As Hanuman leaped over the ocean, Jambhavan watched with pride and awe.

As they say, the rest is history. Hanuman encountered several obstacles and demons on his journey over the ocean. There was nothing that could stop him. Besides finding Sita, Hanuman delivered Rama's message to her. While Ravana tried to capture him, he incinerated Lanka into ashes. In India today, you can find a temple of Hanuman virtually everywhere, and he is prayed for courage and strength on every street. The ex-president of the United States, Barack Obama, carries a statuette of Hanuman with him wherever he goes. It was Jambhavan who transformed a mere Vanara into the mighty god we know, love, and pray today.

LESSON 1: LEADERSHIP ISN'T ABOUT YOU, IT'S ABOUT OTHERS.

Leadership is often misunderstood as occupying a position and exercising authority. It is typical for traditional managers to direct and micromanage, to take credit for success and blame others for failure, and to want to be in control at all times. There is nothing inspiring or effective about this style of leadership. Several leaders believe that leadership is all about them. Their designation will make people respect and follow them.

Firstly, you must understand that leadership is not about you. It's about others and enabling them to succeed. A leader influences, inspires, guides, and develops others to accomplish a shared goal. A leader's role is to enable and empower his or her team members. Leaders help themselves and others to do the right thing.

Jambhavan selflessly thought about who could do the job and enabled Hanuman to succeed. Rather than making himself the center of attention, his focus was on enabling Hanuman and achieving the collective goal.

As a leader, your ability to unleash others is what matters. Develop the ability to see potential in others, ignite their inner fire, and guide them through the toughest situations.

Recognize and encourage your team to rise above their doubts and achieve the seemingly impossible

Choosing the right person for the job is an essential skill for a leader. Learn how to delegate effectively and get work done.

Leadership is not about taking credit. It's about giving others credit. It's about praising people who are doing exceptional work.

Jambhavan helped Rama with the collective goal of finding and rescuing Sita from Ravana.

Leadership does not come by designation, but by vision.

Learn how to harness individual motivations to achieve an organization-wide goal.

A leader's responsibility is to understand and prioritize the business's needs.

To make an impact as a testing leader, ensure your testing goals contribute to the larger organizational objectives.

As development cycles speed up and deadlines are tightening, testers are under significant pressure. It is possible for testing teams to become discouraged and less productive if they are not motivated.

LESSON 2: CONQUER YOUR SELF-DOUBT

Even the great Hanuman had to be reminded about his abilities. We are no exception.

New leaders and experienced leaders alike struggle with self-doubt. We doubt our own skills, and abilities, and downplay our accomplishments. This is also called imposter syndrome. Impostor syndrome is much more common than we think and it occurs at all levels. New leaders who fear not living up to expectations are particularly prone to this.

Self-doubt or Imposter Syndrome: How to deal with it?

Acknowledge your feelings of self-doubt. Distinguish facts from feelings. Introspect, think about the cause of your doubts, and figure out what's causing them. Work on those shortcomings.

The fear sometimes comes from the fact that you do not know certain things. Leaders of software testing sometimes become overwhelmed by so many things to handle. Leadership isn't about knowing everything and being an expert at everything. It is about influencing, inspiring, and motivating your team to work together for a collective goal. Instead of focusing on what you don't know, focus on what you do know.

Be vulnerable. Sharing your feelings with a trusted co-worker or a mentor(like Jambhavan) can help you overcome self-doubt.

Make sure you surround yourself with people who uplift and motivate you. Focus on building self-confidence, and developing a strong sense of self-worth.

Practice self-care. Remind yourself of your accomplishments and the hard work you've put in to achieve all the success you have gained.

I have found that offering positive thoughts to yourself while meditating helps you overcome imposter syndrome.

LESSON 3: GET TO KNOW THE TEAM YOU ARE LEADING

Jambhavan knew more about Hanuman than Hanuman knew about himself. As a result, Jambhavan was able to effectively coach Hanuman to reach his full potential. Hanuman's trust and devotion to his mentor are also evident in the story.

Understand who your team members really are. The importance of building a good working relationship with your team cannot be overstated. A strong team is built on a foundation of trust and connection. Invest time in learning about each team member as an individual and not just what they can do for you. By knowing your team well, you can create a working environment that empowers everyone to perform at their best. Furthermore, gain a deeper understanding of their strengths, weaknesses, perspectives, motivations, and goals. Knowing who your team members are will help you tailor your coaching or leadership style to be more effective and personalized.

Schedule regular one-on-ones with each member of your team.

Talk about what you can do to help the member reach his or her personal and professional goals during the one-on-ones.
Try to empathize with the other person's needs, emotions, and point of view,

Try to connect with your team members personally rather than just having formal conversations. Engage them in conversations about their interests outside of work. Discovering common interests that can help you build a stronger relationship.

Small talk, big benefits

Small talk at work matters. Make sure you take time for small meetings with your team members before and after meetings. Having small talk with your peers helps establish relationships and build rapport.

Value your team members at work and beyond.

Create an environment where team members know each other's interests. Make it possible for your team to interact over topics other than work.

Organize team outings, team building activities, informal get-togethers, and games for team building to encourage tester interaction.

Caring leaders bring better results

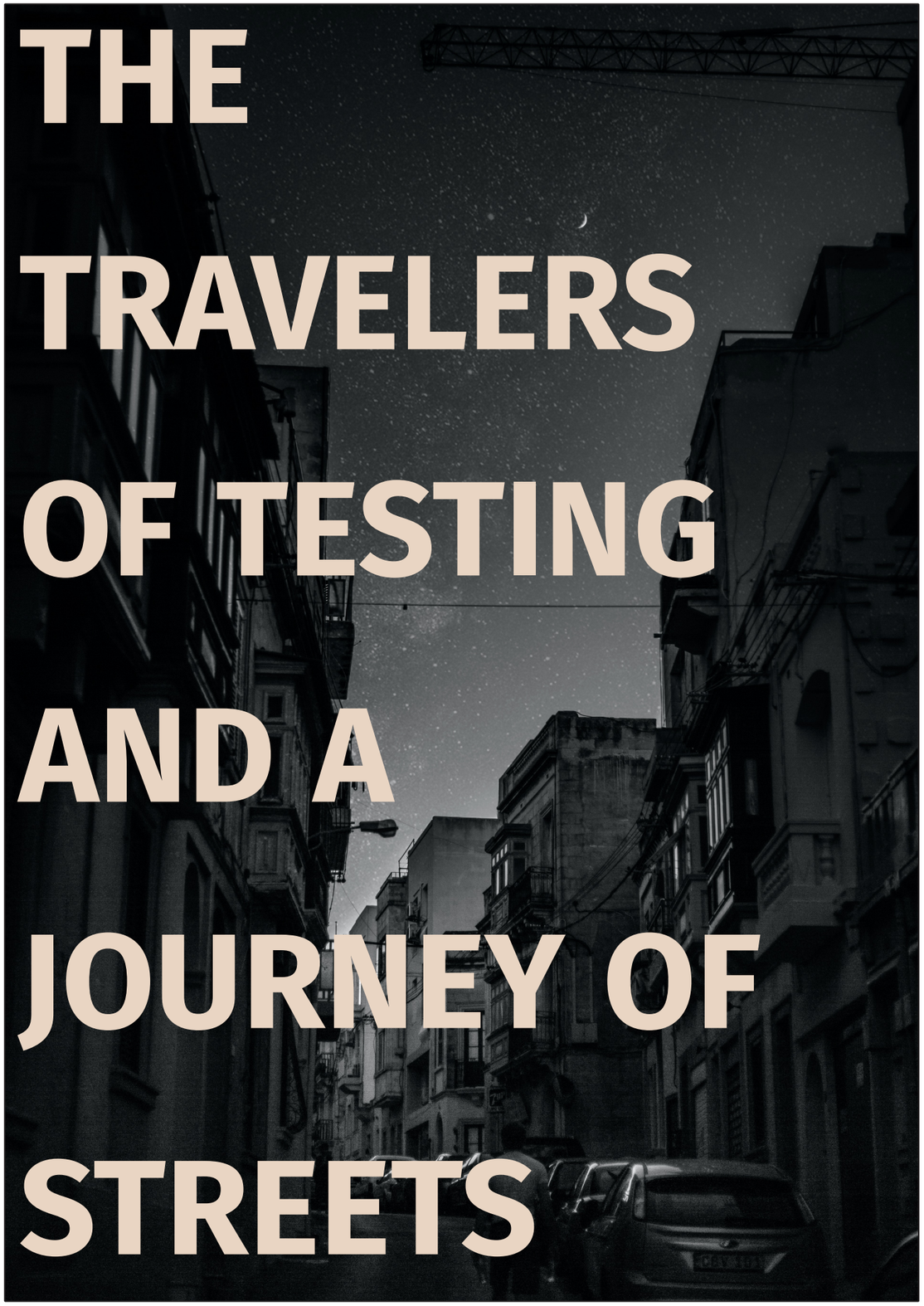
To be a good leader, genuinely care about your team and their long-term success. A growing number of top talent is leaving their jobs after a relatively short period. Caring leaders attract and retain top talent because they make them feel more involved, engaged, and valued.

CONCLUSION

Leadership is not about you but about others, your ability to inspire others matters. Don't let your self-doubt hold you back, and help those around you to do the same. Take the time to get to know your team members personally and establish a strong working relationship. Take a genuine interest in your team and their long-term success. Leaders also need mentors to advance in their careers. Find your Jambhavan - someone who cares about you and your professional development.

Leadership comes naturally to some people, while others learn the skills to become better leaders over time. The best way to discover leadership is through practice. Go ahead and put your learning into practice! As we take inspiration from this tale, let us strive to become today's Jambhavans. Our workplace and the entire testing community need more Jambhavans who can unleash Hanumans.

Be like Jambhavan!



THE TRAVELERS OF TESTING AND A JOURNEY OF STREETS

Twenty years is not a long enough time. It's too short to write the kind of things I am going to say here. However, twenty years of working in the field of testing is long enough to understand and realize how little I know. It's a long enough time to do experiments which are only partially successful. It's also a pretty long time to realize that there are people out there who have explored a dimension of it, which is completely alien to me.

A Poem of Travel

My mind is not wired for building long term memories. I forget a lot of stuff. The opening of the Punjabi poem that I am going to share is not even the exact poem that I had read. I was 17. That was a long time ago. I've forgotten it despite its impact on me and my style of thinking. What remains is a version of it in my mind. I don't remember its author's name. Some years back I tried finding that out. I couldn't.

ਕਾਢਾਂ ਦੀ ਗੱਠ
ਸਿਰ ਤੇ ਚੁੱਕ
ਤੁਰਿਆ ਸੀ ਜਦ ਇਤਿਹਾਸ ਰਾਹੀ
ਤਦ ਪਿੱਠ ਪਿੱਛੇ ਕਿਸੇ ਛਿੱਕ ਮਾਰੀ
ਤੇ ਤਾਰੀਖ਼ ਹਰ ਮੋੜ ਤੇ
ਰਾਹ ਭੁੱਲਦੀ ਰਹੀ

"The history traveler, with a baggage of inventions on his head, started to walk.

And then someone sneezed behind him, and history forgot its way."

(In many Indian cultures, sneezing at the beginning of anything is considered a bad omen in the sense depicted in the poem).

This poem is so important that I ended up reciting it during the only meeting (and a pretty short one) with James Bach years ago. The topic was belongingness. To explain why I don't belong to any particular school of thought, I recited the poem and translated.

The poem further goes on to tell that Time needs a highway to travel, but the travelers often venture into the streets and never come back.

I've always interpreted this poem in terms of what happens to knowledge in a profession. In short, this poem is the story of testing and testers for me.

I don't belong to any specific school of thought. I am a pluralist. I believe that testing, as a field of knowledge, seeks travelers to travel on its highway. Its ask is not the streets into which many and most experts ventured and alas, never came back. Followers followed.

The One-Way Streets

किसी को घर से निकलते ही मिल गई मंज़िल
कोई हमारी तरह उम्र भर सफ़र में रहा

Someone reached his destination as soon as he left home

Someone, just like me, remained in a journey for life

~ Ahmad Faraz

The journey of a highway can be boring and intimidating to some. The landscapes could be appealing. So, it's not uncommon to take a break, venture into the roads and streets to explore.

To manage complexity in the subject of testing, we coined terms and drew distinctions. The act of drawing such distinctions gave us an opportunity to go deeper. This is also the way humans innovate and develop unique styles. What did we do with these solutions later? Did we amalgamate these solutions or let the distinctions stay in their respective universes?

There is a fundamental question to be raised here. Do we have the intelligence for contradicting ideas to co-exist in our minds? At the least, do we intend to do so? If not, how do we plan to fill the inherent gaps in one style that could be filled from another style of doing things?

Around the time I got serious about testing, Exploratory Testing, Context Driven Testing and Risk Based Testing were some such distinctions and types. I am taking these names as examples, and also because the related work had a big impact on me. They highlighted important aspects of the art and craft of testing. They gave an opportunity for testers to develop methods, techniques, tools, heuristics and so on for focusing on those aspects. To this day, some of the related work I read during that time, helps me.

This was also the time when some of the well-known testing certifications were taking shape. The premise was to create a ground for organized learning paths and evaluation of a tester's testing skills. That was the promise they made to the overall ecosystem.

It was also the time when I was fighting my own demons. Rather than on a street of innovation, I was on a path of ego. I was focused on performance engineering and I looked down upon "manual testing" or even functional testing. I considered my own work superior to these and by its extension, me as a superior tester. I am not proud of that time.

RAHUL VERMA
-
Rahul is a Consulting Tester and Coach with an experience of 20+ years of experiments. He works with Trendig, a German company specialising in this space and also known to be the force behind the renowned Agile Testing Days conference. His consulting uniqueness lies in his experience-mix: Hands-on knowledge on business and technical perspectives of Quality with respect to its various dimensions, with a knack for programming and design. He has presented, conducted workshops and published articles on a wide range of subjects related to Quality in various conferences and forums, internationally.

As a coach he has trained hundreds of professionals on Quality from a pluralistic standpoint.

A few years later though, I came down from this high cloud. I realized how deeply flawed my thinking was.

It was the time when I expected the other streets to join the highway back. Exploration, context and risk should have flown back to the mainstream of knowledge rather than staying as a long-term distinction of types. They never truly did.

I started questioning this in my circle. I asked, shouldn't every tester develop exploration skills? If there is something called context-driven testing, what is non-context driven testing? Can we still consider that testing? If there is something called risk-based testing, what is non-risk-based testing? Can we still consider that testing? In a way I believed in the importance of these ideas more than their originators. That's the irony. However, humans are usually happy with categorized, even cut-down versions of concepts as they are more manageable. In short, I was shut down. We live in a world where more than the idea itself, what matters is who is promoting the idea. I was (and I am still) a poor, small promoter, I guess.

Another irony is that the followers of the leaders/adventurers who create and venture into these new streets end up picking only punch lines and one-liners from their leaders. They don't truly or deeply study their leader's work.

One such example is that when the main minds behind Exploratory Testing, conceptually merged it years later, and said it's what is Testing (although that means a street replacing a highway), many and most of the followers continued using Exploratory Testing as a term even to this day.

On the same lines, the certifications became more of a business and ecosystem problem rather than fulfilling the original premise of learning and self-evaluation. Certifications which tried to replace them mostly failed to scale but never admitted openly to how this absence of scale raised a question on the whole premise of those certifications in the context of overall ecosystem. The problem of a good certification at scale is an unsolved problem to this day. Whichever exist are street certifications. And more often than a not, even a poor version of what that street demands.

On the certifications front, I volunteered in various capacities to bring about some changes. I must confess, across all work that I've done in my career, I wasn't able to do much worthwhile in this area. The changes I could bring were too small. I created a certification too. It had its own flaws. I tried to improve it. That didn't sell because all participants continued to fail. Are certifications about knowledge or for practical purposes they have been systematically toned down to the level of knowledge that exists in the wild amongst testers? Is it a design/creativity constraint on part of the creators or is it a business compromise?

Automation in testing had the same fate. Black-box testing vs White box testing were such streets too. Static and dynamic testing were these streets as well. Pick a term and you'll find how silly, questionable walls have been created around those concepts. This caging has killed many opportunities of furthering knowledge in testing. The types of testing slowly became types of testers. The distinctions became self-imposed cages on skills. Black-box tester is not an uncommon word to hear these days, for example. Exploratory testers, White box testers, automation testers (what does it even mean?),... - the list goes on.

In the false dilemma of superiority in an x-vs-y game of words, common sense often takes a back seat. Critical concepts like controlled repetition are often frowned upon, treated as cheap skills and often made straw men to sell agendas. Well established concepts like that of regression testing are discussed less and less, letting misinformation propagate, as no one can take credit for inventing their names.

In short, everything else what one does not sell himself/herself is snake-oil, biased or plainly wrong. If the word context has any weight in testing, how can such extreme opinions of right or wrong exist in the same universe as the word context?

The distinctions and the vocabulary which were supposed to be furthering the craft ended up damaging it. This damage is at a point, where the highway is almost forsaken. What we are left with is a notional highway and most, if not all, travelers traveling on the streets.

The Ignored Streets

Most of the testers are focused on testing functionality by treating test objects as black boxes. These streets make the most noises. A damage of this majority bias was the lost potential of generalization of concepts from specialization streets. Those who tend to specialize in fields like performance engineering, security engineering etc stop using even the word Tester in their profile at some stage. In these fields, many non-testers have great innovations to their credit which we all as testers could have learned from. However, it has been as if those who were focused on functional testing took charge of what the core of testing should be about. The core test design techniques, for example, exist in the shape and form which suits basic functional testing needs and continue to be taught and discussed in that manner. We could have collaboratively helped these streets in joining the highway. But how we could have done that, when the foundational thought was to move away.

As an example, equivalence class partitioning, a classic foundational testing technique is heavily limited when it is seen only in the way it is seen in most testing books. There is functional testing bias. Usage of this technique can be seen in performance engineering world as well in a more open-ended manner. One has to apply sampling to use cases, test data, interactions etc. to come up with a workload distribution model. Similarly, the same approach is used in security engineering. That's the forward direction. In a backward direction of thought, there are many extensions to the technique used in these fields, e.g. the concept of payload construction, the special meaning of special characters, sequencing of characters etc which can be brought back to the mainstream stance on equivalence class partitioning. We can do the same to boundary value analysis, decision tables, state transition testing and so on. This bi-directional exchange of thought is lost when the streets decide never to join the highway.

It can be better understood with some further examples.

Most of the testers have heard and loosely used the term defect taxonomy.

Just do a cursory search. Try to find out which field of work and who has done decent work in the field of defect taxonomies. Barring a thesis guided by Cem Kaner, a few pages in Boris Beizer work, and some random articles/blogs in the functional testing world, there's nothing. Now check how the security engineering world has treated this area. Look at the maturity of taxonomies there. Check CWE website, for example. While we as testers were busy in proving which of our streets are better than others, there was relevant knowledge creation happening elsewhere. We could have learned the patterns. We could have built on those patterns to use them in various other areas of testing. We didn't.

Another example? Look at white-box testing. What have we done as a testing community in this area? Search for publicly accessible information. If one wants to do white box testing, do we really have rich white-box testing literature created by testers for testers? Compare it with the black-box counterpart to understand the majority bias.

What about testing tools? How many of these tools are created by testers for testers?

You might be tempted to take a few names as answers here. Pause and think. Is the totality of their work a justification to the questions I raise here? There are only a few and far pockets of hope. Not sufficient. The names that come to your mind need more encouragement, reach and space, otherwise the black-box, functional bias is here to stay and dominate most of the discussions in the field. If their knowledge remains in respective streets, I don't foresee any concrete large-scale change happening in the coming decades, in the way we treat overall testing knowledge.

For how long, are we going to be busy in debating and drawing distinctions amongst error vs mistake vs fault vs bug vs defect vs failure vs discrepancy vs issue vs incident? Manual vs automated testing? Can we throw these words/distinctions at a real-world problem and solve it?

In a world which does not take testing seriously and "anyone can do testing" is a popular opinion, should testers too behave in a way to treat testing in this manner? Yes, anybody can talk about testing. Doing testing and doing it in a focused way is what takes deeper thinking. When a tester makes testing as a street problem, s/he is (even if unknowingly) giving a nod to the statement that testing is easy.

James Bach in a recent comment on my post on LinkedIn said that he had expected that over a period of years, every tester will have his/her own mental model of testing and will defend it with reasoning. That's an important and beautiful thing to say. In my opinion the streets are not transient, because most testers are followers. The day when each tester has his/her own mental model, the highway will automatically unfold. As that is just hope, till that point I guess the leading voices themselves will need to show the path from the street to the highway.

We are testers. Despite carrying unique, and many a times contradicting, ideas, we share this bond. When I use the word tester here in this article, I am not addressing guests – those who are in testing temporarily and would walk away at the first opportunity. Those of you and me, who love testing and are here to stay, can't we just get along? To prove an idea is great, should we continue on the path of constantly proving that the other idea is less worthy?

> "For how long, are we going to be busy in debating and drawing distinctions amongst error vs mistake vs fault vs bug vs defect vs failure vs discrepancy vs issue vs incident? Manual vs automated testing?"

Let me take an example.

"Manual Testing" as a term is considered demeaning. So, some testers challenge this term. The thought process is more or less the same when somebody uses the word 'resource' for a human. If it is used to demean, I can understand the reasoning here. I can respect when someone interrupts and objects to the usage of this term to say - there's nothing called manual testing.

Having said that, I see this at the other extreme now as well. The moment somebody uses the word 'manual testing', many testers without even thinking or reading the content fully, pounce on it. "There is nothing called manual testing". "What do you mean by manual testing"? Such a question or remark is the only point of discussion they care about. All of this, without attempting to understand whether the person used the term to demean human role in testing. Goal displacement? Isn't ignoring a testing colleague's core point of discussion also demeaning? Aren't most of such remarks done by testers who want to sound smart and ridicule others? Without exaggeration, this extreme in the testing universe is seen when an influencer who himself shares copy-pasted contents on testing interview questions, says – "Humans have the complex neurological system which gives them an amazing sense of experience." This goes unchallenged as expected. After all, it will be politically incorrect to do so and will be an invitation to get trolled. The damage it does is that it dilutes similar commentary by those few testers out there, who have spent their career in exploring and talking about human role in testing.

Words are important. I get that. But only the words are not important. Context in which they are used is equally important. More than both of these put together, applied knowledge is important. Testing debates are frequently stuck these days in the war of word choices rather than the subject. If the goal is awareness, then let me point out something. There are only a few testers who are publicly vocal. The silent ones are increasingly becoming disinterested in testing subject knowledge. The reason is that the vocal testers give them the impression of more of an Instagram/YouTube influencer constantly either delivering the same message for the 100th time or blindly roasting others over word choices.

Are we selling popular opinions in the disguise of debates or ideas? Have we converted the idea of a "type of testing" to a "type of tester" and then to "type of testers" who cuddle together in costume parties? Do we actively ignore misinformation as long as it originates from testers in our own street and is in favor of what we promote? Are we serving testing or personal agendas?

A New Street

As we speak, a new street is already founded and is claiming its own share of travelers.

The Artificial Intelligence/Machine Learning street, already has and will have many experiments and innovations relevant to testing. Across the streets, there are contrasting emotions and reactions – that of excitement as well as that of fear. The experiments on the street are about testing AI/ML systems as well as using AI/ML systems in testing. There are a lot of opportunities.

Watch out, or this is going to be another one of those streets which does not come back to the highway. You'll think AI/ML will innovate new stuff in testing. It will do so. However, it will do so on its own street. If we don't change our mindset, this street will never join the highway.

Those who want to travel the street, remember that it has a testing context and in the absence of larger context of knowledge already created elsewhere, you will be either re-inventing some wheels unnecessarily or under-utilizing information and tools already available.

Evidence of both of the above problems co-existing is out there as I write this article. It's just the beginning.

A Call From the Highway

I am a minuscule part of testing community as well as what the testing highway is about. I don't wear an expert hat. I am a student. My relation with testing is that of servitude and love. In my limited circle, I show the variety of streets. I've raised the questions in this article numerous times. I've not taken sides. I don't have any favorites. I don't write to abide. I don't write to sound nice.

I decided during the first few years of my testing career to travel on the highway of testing. I constantly venture into the streets to learn new ideas but I come back to the highway with the new learnings.

Many things which need to be done cannot be done alone.

Do you empathize with the theme of what I wrote here? Are you in a street and would want your work to reach its full potential? Are you already a traveler of highway, trying to make sense of it all?

Let's join hands. Let's talk and put collaborative efforts in creating a body of knowledge which amalgamates knowledge from the streets back to the highway.

It will not happen automatically. Abandonment of the highway took explicit steps. Amalgamation will need explicit steps too and even more involved efforts.

In case you've read it all, thank you for your patience. That's all for now, with the following food for thought:

दरिया में यूँ तो होते हैं क़तरे ही क़तरे सब
क़तरा वही है जिस में कि दरिया दिखाई दे

Although there are drops and drops of water in a river

A drop is a drop in which one can see the river

~ Krishna Bihari Noor

A Fable On Testing

Two Cats, a Monkey and a Bread

Two cats got hold of a piece of bread. They were debating about it.

They wanted to settle on what it should be called before they eat it. As they didn't seem to settle on what is what, a monkey came in to resolve the matter.

"It's a manually created bread.".

"Oh, is it?", the monkey asked. "Let me taste and confirm", and took a bite.

"It's a bread created with tools".

"Oh?", another bite.

"No, no. Breads are supposed to be created by hand."

"Hmm", another bite.

"Come on, did the bread bake by itself? Some tool must have been involved."

"Yummy", another bite.

The piece of bread was gone. The cats were still debating.

The monkey thanked them for consulting him and said, "Next time you get hold of another piece, let me know. I'll be glad to resolve all your dilemma." Tasty dilemma I must add.

The cats, more hungry than ever, started their search for another piece of bread.

The monkey was waiting.

Tea and Testing with Jerry Weinberg

JERRY WEINBERG
October 27, 1933 – August 7, 2018
–
Gerald Marvin (Jerry) Weinberg was an American computer scientist, author and teacher of the psychology and anthropology of computer software development.
For more than 50 years, he worked on transforming software organizations. He is author or co-author of many articles and books, including The Psychology of Computer Programming.

His books cover all phases of the software life-cycle. They include Exploring Requirements, Rethinking Systems Analysis and Design, The Handbook of Walkthroughs, Design.
In 1993 he was the Winner of the J.-D. Warnier Prize for Excellence in Information Sciences, the 2000 Winner of The Stevens Award for Contributions to Software Engineering, and the 2010 SoftwareTest Professionals first annual Luminary Award.

For over eight years, Jerry authored a dedicated column in Tea-time with Testers under the name “Tea and Testing with Jerry Weinberg”. As a tribute to Jerry and to benefit next generation of testers with his work, we are re-starting his column.

To know more about Jerry and his work, please visit his official website http://geraldmweinberg.com/

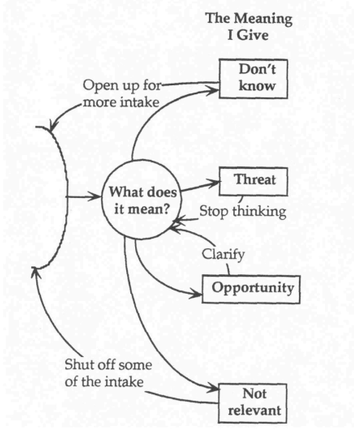
The Technology of Human Behavior - Part 2

Intake is an active, responding process, filtering some of what happens in the world and allowing other events to come into awareness.

To see this process at work, notice what happens next time you talk to someone and you become aware that the person isn't listening. Then say something that totally shifts the context, or change modalities by touching, or singing, or standing up and drawing a picture. Notice how the listener's response to your input changes.

Meaning

The Meaning step contains a response as well. All of the many meanings I can make from my Intake can be broken down into four major categories in answer to the question, "What does this Intake mean?" Each of these categories leads to a general type of response that is partly universal and partly particular to each person, as shown in figure below.



For me, the responses are, roughly

- "Don't know" leads to a desire for more data to clarify.
- "Not relevant" leads to shutting off some of the intake, turning my attention elsewhere.
- "Threat" leads me initially at least to stop thinking and go into an automatic mode in which I lose conscious control of my responses.
- "Opportunity" leads to more thinking, to clarify what external response to give next.

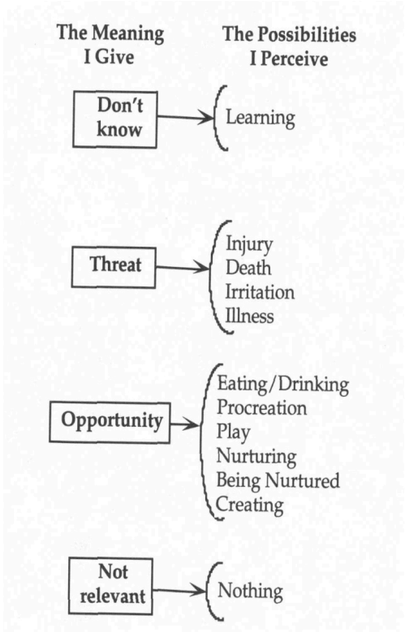
Making Meaning is also a responsive process. Some of the responses control the filtering of further data from the world.

Each person's responses are unique. Somebody else may respond to "don't know" by shutting off intake. My response labels me as a "curious" person who is attracted to things I don't understand.

My "stop thinking" response to "threat" is not a characteristic I value in myself. I prefer that "threat" would lead me to take more data and think more clearly, but my initial instinctive response is to shut down. I don't think I can change my basic pattern, but I can change my reaction by shortening the shutdown period. On the outside, it may now look as if I instantly move to take more data and think more clearly, but there's an internal struggle hidden beneath it.

Significance

The significance of each possible meaning can be considered in terms of the possible consequences to me, as shown in next figure. Of course, this is only a simplification of the thousands of possible consequences I may perceive, but my first response at this stage is to simplify into some broad but important categories of what might happen to me as a result of this interaction—learning, death, illness, play, creating, nothing, and so forth. It is the category I choose that determines the general pattern of my response.



Who is in charge of the response?

One of the reasons that "small brain" technology is so complex is that the human brain seems to operate not as one mind, but as a "multimind," or team of minds. If I am coping well, I have many different minds to put in charge for different situations. This, of course, is precisely what Ashby's Law of Requisite Variety says I must do if I am to be an effective controller of complex systems. This decision is sometimes conscious and sometimes unconscious.

Virginia Satir helped people access their different minds through an exercise called a "Parts Party." At the start of a Parts Party, the host thinks of real or fictional people to whom they have a strong emotional reaction—about half positive and half negative. For instance, during one Parts Party, I selected Albert Einstein, Sir Edmund Hilary, Rambo, Adolph Hitler, Mimi (from the opera La Bohème), Rasputin (the "Mad Monk" who aided the downfall of the last Russian Tsar), Billy Jean King, Woody Allen, Mother Theresa, Elizabeth Barrett Browning, Madame Curie, and Miss Manners.

The idea behind the Parts Party is that I will have a strong emotional reaction to a character who resonates with one of my own parts. A part I accept and value will produce a positive reaction; a part I reject and despise, a negative reaction. Thus, I would not have a strong reaction to Adolph Hitler if I didn't have a part that I identify with some aspect of Hitler. Through my own Parts Party, I learned that my Hitler part is the part that swats flies, and generally rids me of external irritations. Generally, my Hitler part is not the least concerned about how the fly feels, which might be okay for flies, but not for human beings who are irritating me.

To be continued in next issue...

INTERVIEW

How does BDD help software teams? How to deliver together as a team? Gáspár Nagy talked to us over a cup of tea.

- INTERVIEWED BY DAVE LEVITT

Hello, TTwT Readers! It is with great pleasure that I introduce Gáspár Nagy. He is a well-known author, coach, and consultant specializing in Behavior Driven Development (BDD) practices. He is also a major contributor to SpecFlow.

Greetings, Gáspár ! Welcome back from your vacation, and thanks for taking the opportunity to share your thoughts with the TTwT community.

Q1: I'd be interested to learn about your journey with BDD and SpecFlow. How and when did you get involved? What are you especially proud of? What do you wish you could do over if you could?

I started to get involved in 2009. At that time, I wanted to do better testing in an agile landscape, but Cucumber did not work well with .NET. I started development as an in-house effort at a company called TechTalk. It was then sold to Tricentis. I am now doing BDD training and consulting, I am also doing some tooling work.

I am especially proud of adapting Cucumber to the .NET ecosystem and developing a common Gherkin parser. If I were to do something different, I would develop a sustainable business model. For example: there's not much SpecFlow development at Tricentis these days.

Q2: The testing community has no shortage of opinions as to whether BDD is testing or not, but before I get to that, I'd like to set the stage and frame Specification by Example (SBE) and BDD as techniques teams can use to help drive ambiguity to clarity, specifically by crafting acceptance tests (a.k.a. scenarios) that leverages the language of the business. Do you care to offer any opinion or thoughts on this?

My background is in software development, but I've long been interested in testing. What I see happening is that the industry is trying to separate development and testing. To further that point, consider asking yourself what is the purpose of a particular test? A developer-facing test verifies what the developer wanted, while an acceptance-facing test verifies the expected behavior.

Even the best specification has ambiguities and context, so the best way to resolve this is for the team to collaborate and create a shared understanding. When this happens, development is no longer separate from testing. Constructive interaction results during the creation of concrete examples, which are in fact tests.

Q3: Continuing the above, it takes critical thinking skills to drive a rich and complete set of happy and unhappy scenarios. This is where testers can make a huge contribution. Have you encountered teams or organizations that still believe testers need to wait until the software is written to begin testing? If so, how have you dealt with this?

A culture won't change overnight. When testers collaborate during the creation of examples, the traditional testing, I.e., when the software has been developed, becomes easier. However, if you can't change the habit of testing at the end of a sprint (which is late feedback), try making the slices smaller, such as splitting up the testing by examples / scenarios. With this approach, the feedback loop becomes optimized. In this regard, testers can try to influence thinking by testing in smaller slices.

GÁSPÁR NAGY
–

Gáspár Nagy is the creator of SpecFlow, regular conference speaker, blogger (http://gasparnagy.com), editor of the BDD Addict monthly newsletter (http://bddaddict.com), and co-author of the books "Discovery: Explore behaviour using examples" and "Formulation: Document examples with Given/When/Then" (http://bddbooks.com). Gáspár is an independent coach, trainer and test automation expert focusing on helping teams implementing BDD and SpecFlow.

He has more than 20 years of experience in enterprise software development as he worked as an architect and agile developer coach.

> “Current AI technology requires a lot of learning, and it is effective in domains that are massively used. For example: driving a car or interacting with a login page. I don’t see AI being effective when domains are unique, which is what testers usually focus on. ”

Q4: I’ll cover automation shortly, but I have found over many years, and on many teams, that it is possible to create a shared understanding of what a Product Owner is looking for without automating your scenarios. What advice would you offer a team that is new to BDD?

I don’t have a cookie-cutter approach, but developing concrete examples and user journeys can be very useful, and there’s no better way to motivate a team than by developing successful products!

Q5: Software literature is filled with test automation failures. What advice might you offer to a team that wants to embark on automating their scenarios?

Users will be more tolerant of defects in new features, but less so if defects are found in regression testing. Yes, you can certainly do manual regression testing, but it does not scale. In contrast, when you automate your scenarios, you’re automating the requirements. Automation yields sustainability, and sustainability is certainly a dimension of quality. Put another way, sustainability drives value!

Q6 : SpecFlow has a very rich and interesting history. How did it all start, where is it now, and if you can share, what lies ahead?

There’s not much work going on now, but I do think things can be improved. For example: reporting. Another area I would like to see is support for other sources, i.e., markdowns.

Q7: It’s hard to go for a week without reading an opinion on testing and AI / ML. Have there been any discussions on adding some sort of AI / ML into the discovery process?

Current AI technology requires a lot of learning, and it is effective in domains that are massively used. For example: driving a car or interacting with a login page. I don’t see AI being effective when domains are unique, which is what testers usually focus on. On the other hand, AI might be able to help write the scenarios. Example: consistent scenarios.

Q8: I recently read a post that one of your BDD books just got translated to Japanese. Congratulations are in order! Finally, I want to wish you and Seb Rose all the best with your upcoming book. What will it be covering and where do things stand?

Our goal of the 3rd book is to collect automation patterns that are independent of a language, much like design patterns. Our work is slowing down though.

Q9: Is there anything I haven’t covered that you would like to share?

I would like to recommend to testers that they view testing as a living thing. Go to conferences, help influence developers, and keep an open mind.

Thank You, Gáspár, for sharing your thoughts and insights, and again all the best in upcoming endeavors!

~Processes

UNLEASHING THE POWER OF TESTING IN DESIGN THINKING

- LISA GUAN

Lisa Guan is a seasoned professional with 18 years of experience in R&D and finance industries. She excels in agile adoption, Design thinking, leadership consulting, OKR implementation, and performance management.

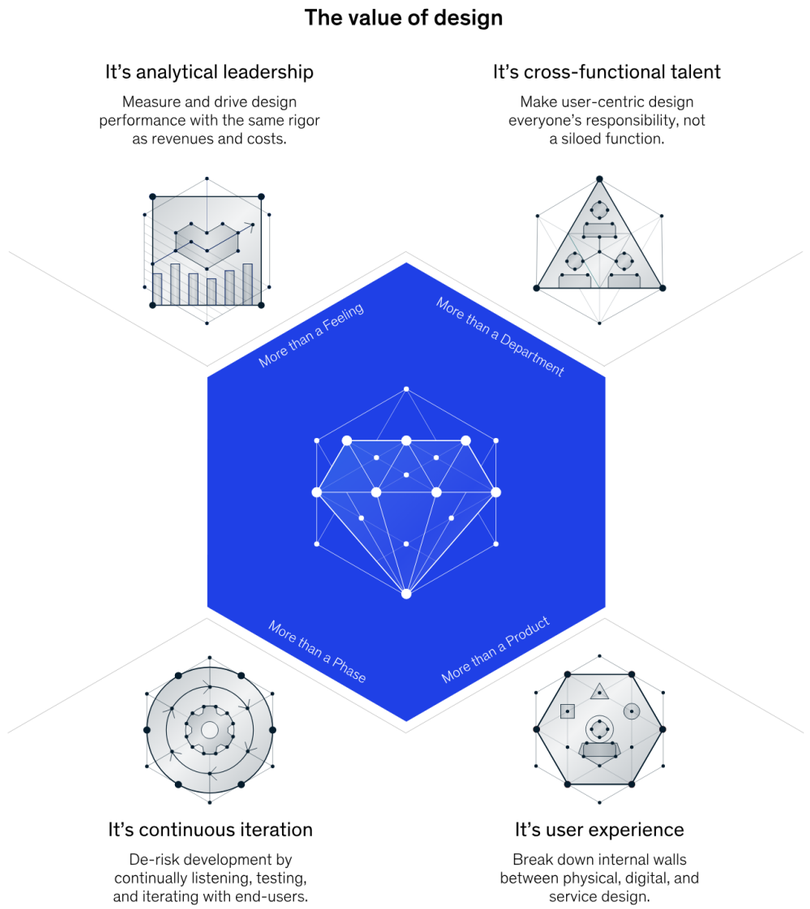
Lisa played a key role in leading IBM CIO's large-scale agile transformation from 2017 to 2019. She is a published author of the books "Performance Management for Agile Teams" and "20 Principles on the Practice of OKR", selling over 20,000 copies in China.

A Design Thinker's journey towards user-centric solutions

Have you ever heard of Design thinking? It's an incredible approach that puts humans at the center of problem-solving, focusing on understanding their needs and perspectives to create innovative solutions.

But here's the thing: just following the steps of design thinking won't automatically make you human-centered and problem-solving oriented. The real magic lies in the intelligence, collaborations, and mindset of the individuals and teams who embrace it.

The McKinsey's coverage of this collaboration aspect is quite convincing if you read their report - "The Business Value of Design". See the exhibit from McKinsey report below. Designing software is indeed a "cross-functional talent" and it's done better with continuous iterations.



I can't agree more. In my experience, bringing together individuals from diverse backgrounds and expertise has proven to be a game-changer in the design thinking process. And one key role I always would like to involve in my cross functional team is test engineer. Software tester role highly potent one that truly unleashes the power of design thinking.

Now, I know there might be some disagreement about this because design thinking is often used in the early stages of a project to identify problems and explore potential solutions. Skeptics may question the role of testers when there are no existing solutions available. However, I've come to realize that it's precisely during these uncertain times that testers can bring tremendous value to the table. Let me share with you why.

1. Facilitating better human-centered design:

To create truly superior human-centered designs, the involvement of testers is essential. Testers possess a deep understanding of user behaviors and expectations, making their integration into design thinking activities crucial for a comprehensive evaluation of the user experience.

As a UX designer, I have often partnered with the Product Owner and Software Architect during the product design phase. However, I occasionally felt the need for someone on the team who had UX/UE knowledge to brainstorm ideas together and challenge me on certain UX concepts. While the Product Owner and Software Architect could contribute, their limited UX/UE background prevented our discussions from delving deep into the subject.

Then, one day, we happened to include someone with a strong testing background in a design thinking workshop. This individual provided valuable insights on usability, intuitiveness, and potential pain points, enabling the team to refine and optimize the user experience. It was at that moment I realized the importance of testers in the design process. They offer unique perspectives, providing valuable insights on usability, intuitiveness, and potential pain points. By incorporating their viewpoint, the final product seamlessly aligns with user expectations, resulting in higher user satisfaction and adoption rates.

Ever since that experience, I never forget to invite a tester to participate in my design workshops. Their contributions have proven invaluable in creating user-centric solutions."

2. Making solutions more feasible:

A successful project needs to strike a balance between meeting user desires, business viability, and technological feasibility. However, one challenge that UX designers and product owners often face is becoming too focused on meeting user expectations or becoming enamored with their own business ideas, which can lead to unrealistic requirements being imposed on the development team. I have personally received numerous complaints from development teams about how my ideas are unrealistic in the past.

However, there is an effective solution to address this issue: involving testers in the early stages of the design process.

Testers possess a remarkable ability to proactively identify potential usability, functionality, or performance issues during the design phase – it's inherent to their job role. By raising concerns and suggesting improvements at an early stage, testers assist the team in addressing potential issues before substantial effort is invested, ultimately saving time and resources in the long run. Furthermore, their involvement provides valuable insights into the feasibility and practicality of design choices. Testers contribute to the creation of testable requirements, aiding in the definition of clear acceptance criteria and measurable outcomes."

3.Holistic validation of design concepts:

Another unique strength of testers is their skill set that enables them to identify potential issues and risks early on. When they are involved in the design thinking process, teams benefit greatly from their expertise in validating design concepts. Testers can evaluate the feasibility, technical viability, and testability of proposed solutions, ensuring alignment with both user needs and quality requirements.

In a nutshell, incorporating testers into the design thinking process enhances the overall effectiveness and success of the approach. Their insights, expertise, and ability to identify and address potential issues contribute to the creation of user-centric and high-quality solutions.

But it doesn't end there. Testers themselves can also benefit greatly from attending the design thinking process. You see, when testers lack knowledge of user needs, they may unintentionally test the software based solely on their own assumptions or technical requirements. This can create a disconnect between the actual expectations and preferences of the end-users and the testing efforts.

By participating in the empathize phase of design thinking, testers gain a deeper understanding of user needs, pain points, and preferences. This understanding enables them to approach testing from a user-centric perspective, leading to more meaningful and impactful testing efforts.

Of course, involving testers in the design thinking process does come with its challenges. Testers typically have limited collaboration with business roles, which can present obstacles along the way. However, I've found that by taking proactive steps, testers can overcome these challenges and enhance their understanding of business needs and communication skills. Some of these steps include seeking business context, improving communication skills, collaborating with analysts or owners, participating in requirements workshops, engaging in cross-functional collaboration, embracing an agile mindset, and pursuing continuous learning.

These actions align testers' efforts with business objectives, provide valuable insights, contribute to clear requirements, identify risks, stay updated, and foster collaboration.

In my own journey, I've witnessed the transformative power of involving testers in the design thinking process. It's a win-win scenario where testers play a vital role in creating user-centric solutions while also benefiting from a deeper understanding of user needs. Additionally, in his work around QX, Lalit has discussed several ideas around tester+designers collaboration. Give it a read.

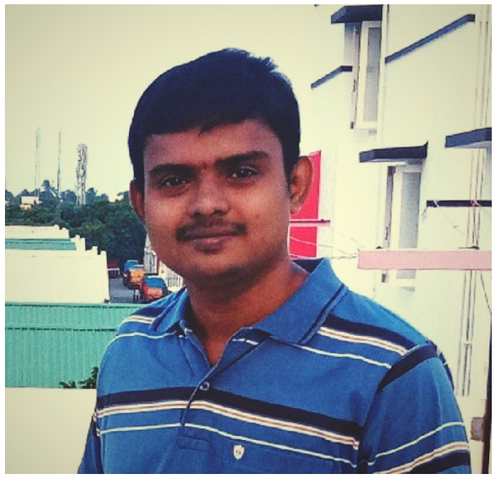
So, if you're embarking on a design thinking adventure, don't forget to invite the testers along for the journey. Together, we can weave a tapestry of innovation and create solutions that truly make a difference.

References:

"The Business Value of Design".- The McKinsey's Report

QX - QA+UX for Co-creating Quality Experience - Lalit Bhamare

THE ESSENCE OF MENTAL MODELS FOR SOFTWARE TESTERS



- BALAJI SANTHANAGOPALAN

Balaji Santhanagopalan is a passionate and experienced software tester in the field of Model-based systems engineering. He is working as Project Test Engineer at Siemens Digital Industries. He is also running a telegram group called Digital Testers 4.0 where resources related to Embedded systems software testing and IOT testing have been shared. He is passionate about teaching and writing about software testing.

He also loves to write Python scripts and work with Linux environments. He blogs at https://bjtestingtalks.wordpress.com/.

ISSUE 03/2023
PROCESSES

During TestAway Goa 2022 and Worqference 2023, I got an opportunity to attend a hands-on workshop on "Mental Models" by Ajay Balamurugadas. It made me curious to dig deeper into this topic. I have gone through multiple blogs, and videos, and read some books to explore more on this topic. In this article, I would like to share my learnings and findings about Mental Models

What are Mental models?

Mental models explain someone's thought process about how something works in the real world. To be more precise, Mental models are a representation of our thought process of something that is stored in our mind. As humans, we won't be able to process more information about the world in our minds. So, we use models to store complex information in our minds as understandable and organizable structures.

Why do software testers need to understand mental models?

Mental models help to practice critical thinking. It allows testers to come up with great test ideas and analyze risks. It makes us aware of our cognitive biases by guiding us to identify blind spots in our own thinking. Mental models provide the base for practicing exploratory testing. This is an excerpt from the book "A Practitioner's Guide to Software Test Design" that explains the importance of creating a mental model for designing better tests.

Key Question

What is the most important test I can perform right now?

A possible exploratory testing process is:

- Creating a conjecture (a mental model) of the proper functioning of the system
- Designing one or more tests that would disprove the conjecture
- Executing these tests and observing the outcomes
- Evaluating the outcomes against the conjecture
- Repeating this process until the conjecture is proved or disproved

The snippet below is from Software Testing Techniques book, that conveys that testing is a process of creating mental models.

3.8. The Role of Models

Testing is a process in which we create mental models of the environment, the program, human nature, and the tests themselves. Each model is used either until we accept the behavior as correct or until the model is no longer sufficient for the purpose. Unexpected test results always force a revision of some mental model, and in turn may lead to a revision of whatever is being modeled. The revised model may be more detailed, which is to say more complicated, or more abstract, which is to say simpler. The art of testing consists of creating, selecting, exploring, and revising models. Our ability to go through this process depends on the number of different models we have at hand and their ability to express a program's behavior.

> The exploratory process works even if you don't have a product to test. You can explore a set of documents or interview a programmer, using the same thought processes. You make progress by building richer, better mental models of the product. These models then allow you to design effective tests.

That's from Lessons Learned in Software Testing book. It highlights the essence of creating better mental models for exploring the software without any product documents.

All these points really helped me to understand the importance of mental models.

How to practice applying mental models?

Here are some mental models which testers can get started to practice with:

Inversion:

This mental model helps in inverting a problem and forces us to look at it differently which helps in unlocking new solutions. Practicing this mental model, allows us to avoid being stupid and help to think about the opposites of the action that we need to perform. Some simple examples can be like giving opposite characters other than characters supported for a particular line field, giving the opposite format of an Email ID to check whether that makes some problem or not etc...

Framing:

This mental model helps in presenting the same information in multiple ways. Bug Advocacy is one of the examples here. Because you need to present your bug findings and highlight their impact to Software developers and Product owners for convincing them to fix the important bugs. Another example is creating user personas as we need to frame personas for different kinds of users under different age categories for using a particular software. You can know more about framing mental model here: Framing mental model.

The Map is not the Territory:

This mental model helps to evaluate our assumptions and their reality, theoretical and practical approach. One example is whatever we learned from theory books, if we try to apply it in real life, It might not be accurate or practical or its behavior will be different from what we learned. So we need to know what exactly it is by applying it which is actually the territory. From a software testing perspective, requirements are a great example here because when we start implementing those requirements, they might get change over the course of time. Requirements are like maps here. It can guide us in creating test ideas but it can be inaccurate or not practical sometimes. You need to check the technical implementation of both the front end and back end of the software requirements which all are the actual territories.

Second-order Thinking:

This mental model helps to practice deep thinking. It can help us to identify risks when we are going to perform some actions or take some decisions. It can help to evaluate the long-term consequences of the decision that we make. What If? and 5 whys? can be good examples and a great way to start practicing this mental model. It's like charting out the next connecting thought by asking questions like 'What if we perform this action? What if this action cause this effect?'

Opportunity cost:

This mental model helps to evaluate the benefits of the options that we are going to select and we are going to miss. There are two options A and B. If you select option A, then what is the cost of missing option B? If the cost of missing option B is lesser, then there is no problem. Or else there might be a problem.

Fixing a bug earlier than fixing it during production is a better example here. Because fixing the important software bug during the production period proves to have the most cost than fixing it during the software development period.

First principles thinking :

This mental model helps to break down complicated problems into basic elements and then reassemble them from the base. It's a very useful mental model for software testers which helps in Test design and Automation framework development. The Product Coverage Outline can be a great example here.

I have shared only a few mental models here but there are 100+ mental models which we can learn to improve our thinking.

Relationship between mental models and heuristics:

I had a long discussion with Michael Bolton regarding the difference between mental models and heuristics. During that conversation, I was able to figure out the relationship between mental models and heuristics. I learnt that,

"Mental models are representations of something complex in simplified form and Heuristics can be a set of tools to develop and evaluate our mental models"

Through all these learnings, I understood the importance of mental models. I also found that practicing mental models help software testers to get better at their test craftsmanship and enhance their thinking skills. I hope this article benefits software testers to get started with mental models and help them to become great thinkers.

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

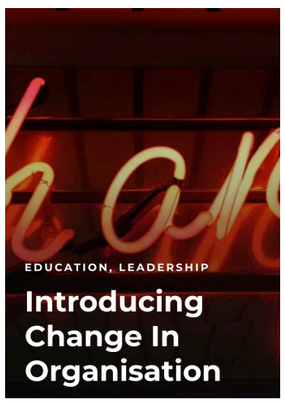
INTERVIEWS, OLD IS GOLD

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.

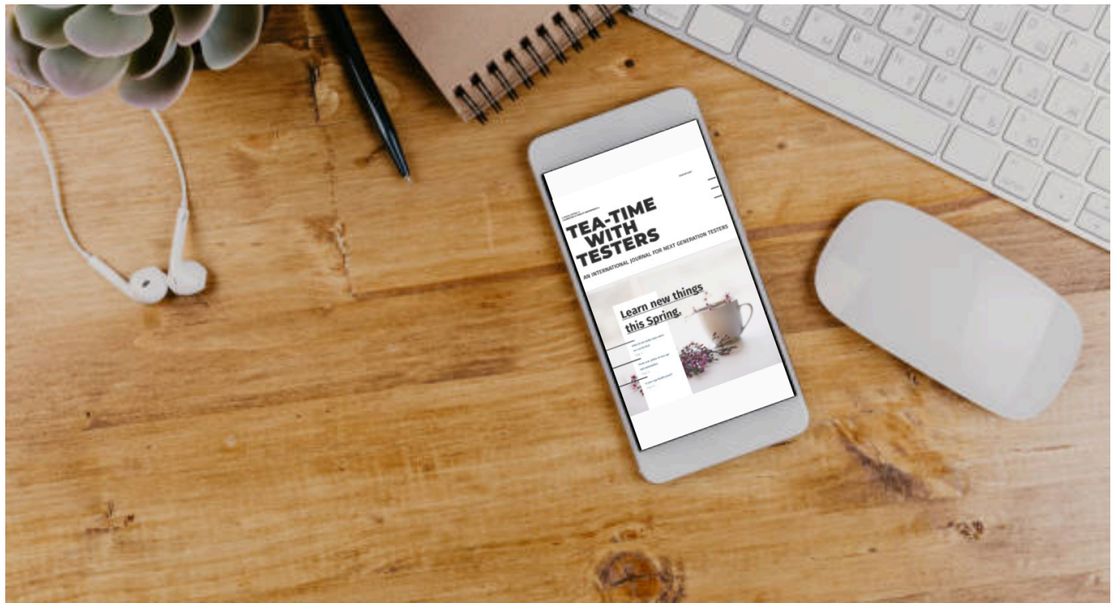
We plan to continue doing that for more articles, interviews and also for the recent issues we have published.

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!



GETTING STARTED WITH KARATE

Few years back, I was trying to find out solution for below problem statement -

1. Low code API tool for team with less/no coding knowledge.

2. Tool should have good community support.

While working on tool evaluation I came across Karate Framework which was exactly satisfying above two requirements.

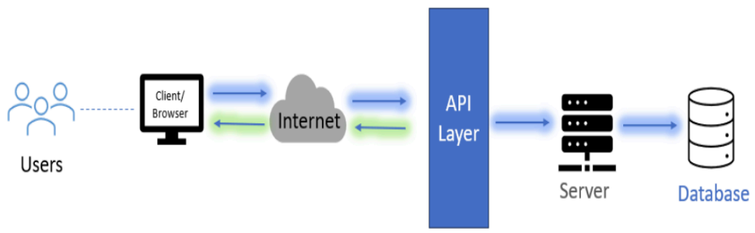
Well while we are going to check out APIs with Karate Framework, let's quickly touch base it again.

What are APIs?

API specifically stands for Application Programming Interface.

According to Wiki, an Application Programming Interface (API) is a way for two or more computer programs to communicate with each other.

Below picture illustrates generic communication between Client/Server for a Web application.



Types of API -

1. Libraries and Framework – Software libraries which we consume are API's. Even frameworks. e.g., Language bindings.
2. Operating System – These API's act as interface between an application and OS. e.g.- Win32 (older applications may run on newer versions of Windows)
3. Remote APIs – Allows manipulating remote resources through protocols. e.g., Java Remote Method Protocol
4. Web APIs – They are service accessed from client devices (Mobile Phones, Laptop, etc.) to a web server using the Hypertext Transfer Protocol (HTTP). They are further classified in public, partner, private and composite.

We are specifically interested in Web APIs as this is where Karate Framework is going to rescue us.

What is HTTP?

Hyper Text Transfer Protocol (HTTP) is a method to communicate between web clients and server. It is the primary protocol for communication and data transfer across the Internet (WWW).

HTTP has a sibling called HTTPS, which provides more secured mode of communication.

For Testing Web APIs, HTTP protocol provides us with CRUD operations.

What is Karate Framework?

My customized definition for Karate Framework would be "A low code framework which can help you to test UI, API and performance tests."

I have personally experienced below capability whilst using Karate Framework, the list is exhaustive but sharing few important ones -

1. Uses gherkin keywords while writing tests.
2. All-in-One framework.
3. Easy to learn.
4. Easy to integrate and adapt.
5. Good documentation.
6. Good community presence and support.
7. Frequent enhancements and bug fixes by creators.
8. Easy cloud integration.
9. Karate Reports for easy reporting.
10. Multi-env support.
11. Parallel execution is supported.
12. Allows logging.
13. Allows JavaScript and Java method calls within the test script.
14. Capable of handling various HTTP calls.
15. Open source and paid versions are available.

Getting started –

Note – Karate supports both MAVEN and Gradle, but for sake of simplicity I am using MAVEN as build tool and https://dummy.restapiexample.com for examples illustrated below for performing CRUD operations. Working example can be found at https://github.com/JyotiShah/KarateAPITests link. As a pre-requisite, ensure you have Java 11 or higher installed and additionally, you can use either of Visual Studio Code or IntelliJ as and IDE for writing your scripts.

1. Dependencies –

You can get the latest version of Karate at - https://mvnrepository.com/artifact/com.intuit.karate/karate-core

Or simply google “Maven Karate Dependency” which will take you to above link. Current latest version is 1.4.0, which I would be using in my GitHub repo as well.

```xml
<groupId>com.js.karate</groupId>
<artifactId>karate-CRUD</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>11</java.version>
    <maven.compiler.version>3.11.0</maven.compiler.version>
    <maven.surefire.version>3.0.0</maven.surefire.version>
    <karate.version>1.4.0</karate.version>
</properties>

<dependencies>
    <dependency>
        <groupId>com.intuit.karate</groupId>
        <artifactId>karate-junit5</artifactId>
        <version>${karate.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <testResources>
        <testResource>
            <directory>src/test/java</directory>
            <excludes>
                <exclude>**/*.java</exclude>
            </excludes>
        </testResource>
    </testResources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>${maven.compiler.version}</version>
            <configuration>
                <encoding>UTF-8</encoding>
                <source>${java.version}</source>
                <target>${java.version}</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>${maven.surefire.version}</version>
            <configuration>
                <argLine>-Dfile.encoding=UTF-8</argLine>
            </configuration>
        </plugin>
    </plugins>
</build>

project>
```


2. Config

The heart of Karate Framework is **karate-config.js**.

This file has a function which returns JSON Object and all variables configured in the function in key and values. This file is mandatory as Karate Framework reads this file upon initialization.

It contains information about -

1. Environment with default environment
2. BaseURL, appID, secret, credentials etc.
3. Global karate configuration like connectTimeout, readTimeout etc.
4. You can have customizations per environment using karate-config-env.js where env stands for env like “test”, “UAT”, “E2E” etc.

A sample config file looks like this – (source – https://github.com/karatelab)

```
function fn() {
  var env = karate.env; // get java system property 'karate.env'
  karate.log('karate.env system property was:', env);
  if (!env) {
    env = 'dev'; // a custom 'intelligent' default
  }
  var config = { // base config JSON
    appId: 'my.app.id',
    appSecret: 'my.secret',
    someUrlBase: 'https://some-host.com/v1/auth/',
    anotherUrlBase: 'https://another-host.com/v1/'
  };
  if (env == 'stage') {
    // over-ride only those that need to be
    config.someUrlBase = 'https://stage-host/v1/auth';
  } else if (env == 'e2e') {
    config.someUrlBase = 'https://e2e-host/v1/auth';
  }
  // don't waste time waiting for a connection or if servers don't respond within 5 seconds
  karate.configure('connectTimeout', 5000);
  karate.configure('readTimeout', 5000);
  return config;
}
```

3. Karate Runner

This is another important class where you should be able specify the features you want to run as below –

```
class TestRunner {

    @Karate.Test
    Karate testRunner() {
        return Karate.run("demo").relativeTo(getClass());
    }

}
```

4. TESTING CRUD OPERATIONS

```
Feature: sample karate test script

    Background:
    * url = 'https://gorest.co.in'
    * token = '757966449a9a507fdb808687516e52a7873cb545b3faf9d62c16c8e606e7216a'

        Scenario: Validate that an user is created using post call for dummy.restapiexample.com
          Given path '/public/v2/users'
          When headers Authorization = 'Bearer ' + token
          And request {"email":"abc@gmail.com","name":"FirstName LastName","gender":"male","status":"active","message":"usercreated"}
          And method post
          Then status 201
        * def id = response.id
        * print 'created id is: ', id


        Scenario: Validate get call for API
          Given path '/public/v2/users'
          When headers {Authorization: '757966449a9a507fdb808687516e52a7873cb545b3faf9d62c16c8e606e7216a'}
          And method get
          Then status 200


        Scenario: Validate that an user can be modified using patch method
          Given path '/public/v2/users/2227'
          When headers Authorization = 'Bearer ' + token
          And request {"email":"abc@gmail.com","name":"NewName","status":"active"}
          And method patch
          Then status 200

        Scenario: Validate that an existing user can be deleted using delete method
          Given path '/public/v2/users/2227'
          When headers Authorization = 'Bearer ' + token
          And method delete
          Then status 204
```

Explanation –

1. Our base URL is https://gorest.co.in
2. Endpoints are specified using path keyword e.g.

   Given path '/public/v2/users'

3. Authorization is handled using bearer token.
4. Requests are sent in JSON format for POST and PATCH methods.
5. Existing resources are modified/deleted in patch and delete requests

I hope with above examples, you should be able to create your first test script using Karate Framework.

- JYOTI SHAH

Jyoti is a passionate test practitioner. With over 13 years of experience She has had various roles from a Tester to a Test Consultant/Test Lead.

Jyoti has been extensively involved in Delivery and Test Management in a multicultural environment supporting stakeholders across different geographies.

ISSUE 03/2023
PRODUCTS

BDD REQUIREMENTS DISCOVERY WITH TESTCOMPASS

Often there appears to be confusion about the concept of Behavior Driven Development (BDD) and far too often BDD is seen as a testing approach. Is this maybe because BDD grew from a response to Test Driven Development (TDD), as explained by BDD pioneer Daniel Terhorst-North?

Although there may be misunderstandings regarding the concept of BDD, this blog does not intend to delve deeply into the complete concept of BDD or provide a comprehensive guide on how to perform BDD correctly. Numerous other blogs have already extensively covered these topics. However, for the purpose of this blog, it is important to at least acknowledge that BDD emphasizes the importance of collaboration between developers, testers, and business stakeholders. It centers around creating a common language that is easily understood by everyone involved, to get a deeply shared understanding of the requirements.

TestCompass

In this article we explore how the collaborative modeling tool TestCompass, in addition to the early Model Based Testing (eMBT) approach (see our other blog https://www.compass-testservices.com/embt-with-testcompass-in-practice), supports BDD in a very easy to use way. And specifically for the BDD phases Discovery and Formulation. Therefore we will take a closer look how to perform the BDD requirements Discovery practices 'Example mapping' and 'Feature mapping' (known as 3-amigos sessions) in TestCompass and how to turn the results (concrete examples) of these Discovery practices automatically into business readable language (as Gherkin feature files), in the Formulation phase.

As described in the intro of this blog, Behavior-Driven Development (BDD) is a powerful approach to software development that emphasizes collaboration and communication between all stakeholders (business and technical). However, executing the 3 different phases of BDD (Discovery, Formulation and Automation) can be challenging. Fortunately, there are tools and practices available to help teams execute these BDD phases more effectively.

TestCompass is such a tool, which can help you streamline, simplify and automate the BDD phases Discovery and Formulation.

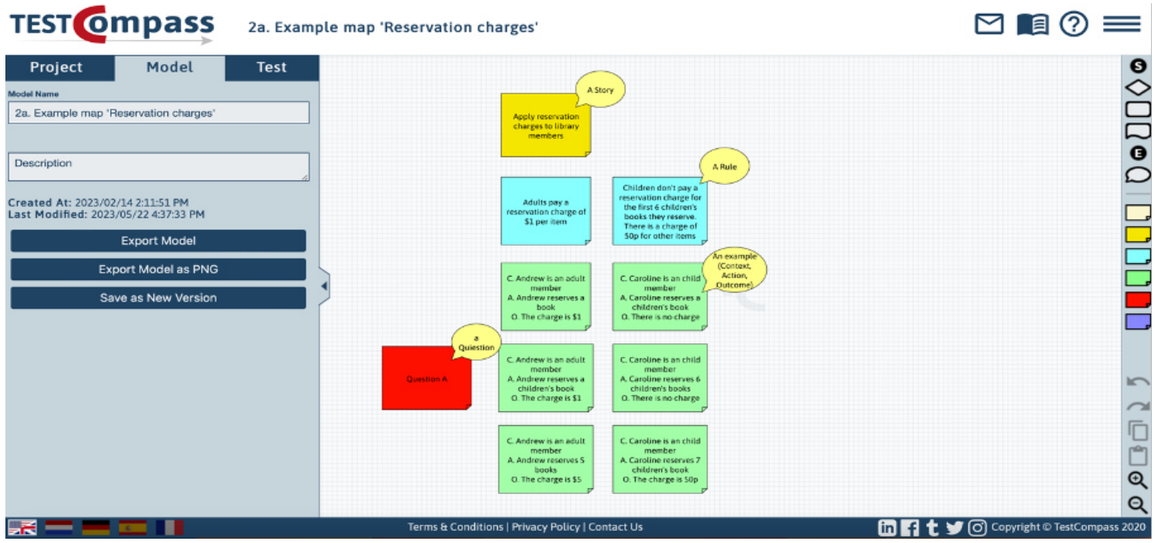
Discovery

In the BDD phase Discovery, we need to answer the question “What could it do?” and collaboration between business stakeholders, developers and testers is essential here. The goal of this phase is to ensure that everyone is on the same page regarding the requirements. To facilitate this collaboration, often workshops or meetings are hold, like ‘Example mapping’ and ‘Feature mapping’ (also known as requirements discovery workshops and 3-amigos sessions). In this meeting, a group of individuals (including at least a business stakeholder, developer and tester) convene to discuss a user story and document specific examples on index cards or sticky notes that serve as illustrations for that user story. These examples, typically associated with a particular business rule, generally comprise of the context, action, and outcome, effectively demonstrating the behavior described by the story.

TestCompass can support these requirements discovery practices, ‘Example mapping’ and ‘Feature mapping’, by the possibility to model out the Example map and Feature map by simply drag and drop different used colored sticky notes onto the canvas. For ‘Example mapping’ normally yellow sticky notes for the story, blue for the rules, green for the examples and red sticky notes for the questions that arise, are being used. In `Feature mapping’ normally yellow sticky notes for the story, blue for rules, green for examples, yellow for the steps and purple for the consequences are used. Also here red sticky notes for the questions that arise.

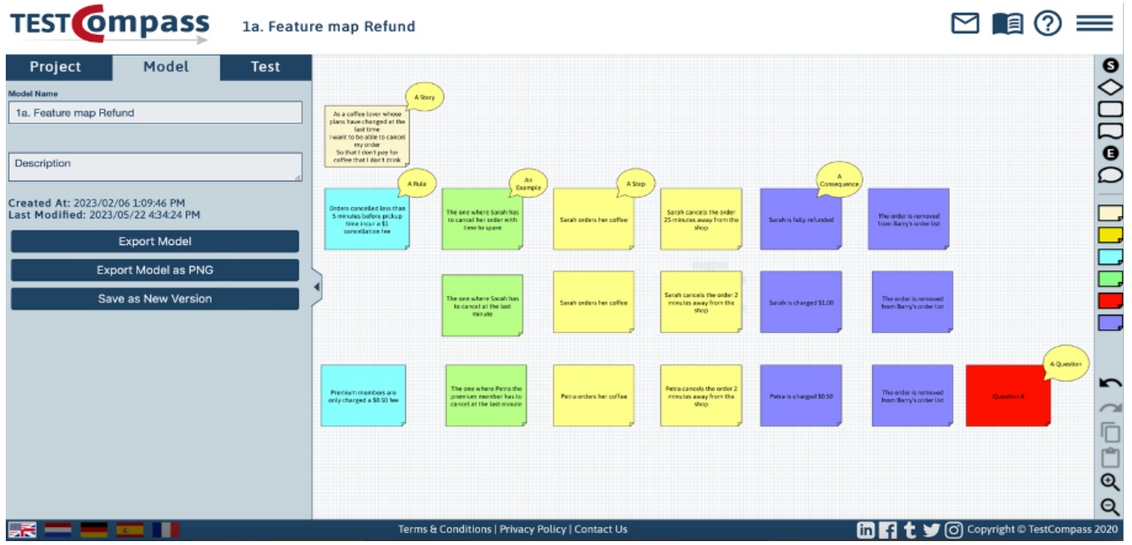
See below an example of an Example map (figure 1) and a Feature map (figure 2) modelled in TestCompass.

Example map ‘Reservation charges’



Modeling the Example map or Feature map directly in TestCompass is very easy and works really intuitive. It has many advantages over running a manual requirements discovery session. Besides the fact that all information from the sessions is automatically documented and saved in TestCompass, there are many other advantages. For e.g. in TestCompass it is easy to make changes or add extra comments to the Example map or Feature map. But also better visibility is an advantage, especially when the session is done online. And better visibility makes it easier to share and discuss ideas and examples. Another advantage is reusability. TestCompass allows the sessions to be reused for similar project or features. This can save time and effort in future projects and help to ensure consistency across different teams and projects. And do not forget a lower chance of making typos in the next phase Formulation, where the examples will be described in a formalized language. In TestCompass we can re-use a lot of the text from the Example map or Feature map (see next phase Formulation).

Feature Map Refund



Formulation

The BDD phase Formulation will start once the ‘Example mapping’ or ‘Feature mapping’ session in the BDD phase Discovery, is completely done and the goal of this phase - a shared understanding - has been achieved. In this phase we need to answer the question “What should it do?”. Now all the examples (and counter examples) created in the Discovery phase will be turned into a more proper and formalized language. And often this is done by the so called ‘Gherkin-gang’ by describing all the examples in Gherkin syntax (Given-When-Then format), so that they later can be used as executable tests.

During the formulation phase, TestCompass can help to convert the Example map or Feature map into a graphical model with a high level of abstraction and thus readable for both business stakeholders as technical stakeholders. This promotes the ability to have the graphical model reviewed within the team and ensure that all results from the requirements discovery practice have been properly interpreted and worked out in the graphical model and there is a deep shared understanding of what needs to be built (requirements). After all, a graphical representation is still much easier to read and to understand than a text, even if this text is plain English. It is also possible to add extra comments or new upcoming questions in the model itself by using a special balloon node. This makes the result of this phase even more readable and well documented.

See figure 2b for an example of how the first business rule of the Example map ‘Reservation charges’

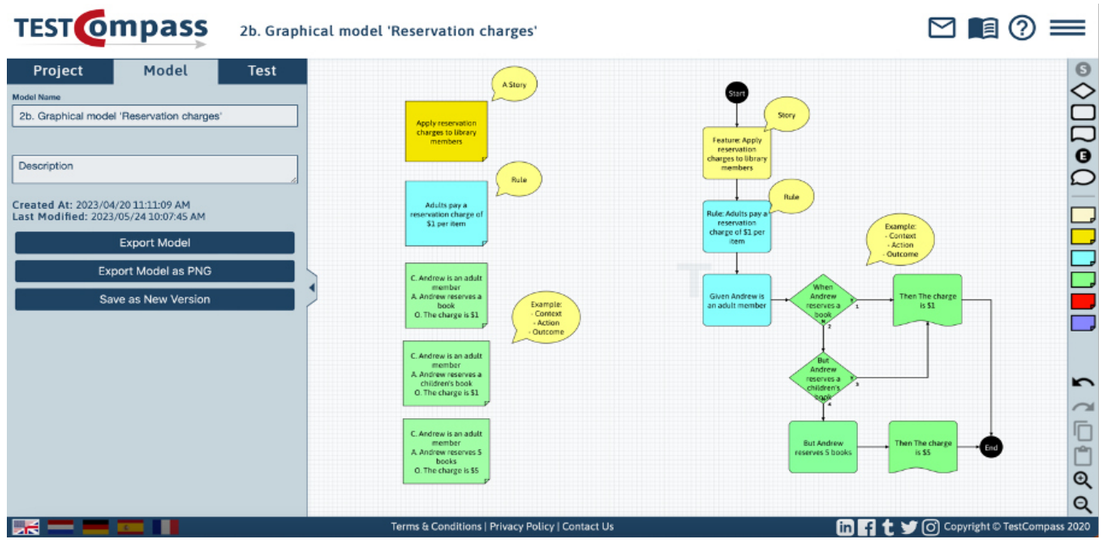
(figure 1) has been converted to a graphical model in TestCompass (figure 3). For clarity, the relevant part of the Example map is also shown.

And what about the Gherkin feature files, which are a common delivery of this phase? Well, after the Example map or Feature map in converted to a graphical model in TestCompass and has been reviewed, the Gherkin feature files can be automatically generated from the model. This is of course a great advantage. You no longer have to write out all the Gherkin feature files by hand and thus less time consuming and less chance of making writing errors. Furthermore, within TestCompass it is possible to select a requirements coverage form (from weak to strong) before the Gherkin feature files are generated. With this, the generated Gherkin feature files are related to the pre-selected coverage and therefore all the different scenarios in the Gherkin feature file are coverage based.

It is also possible to include the background in the graphical model. In addition, you can include any examples and tables in the details of the model. These are then automatically included in the automatically generated Gherkin feature file (outline scenarios).

See in figure 4 on next page, an example of the automatically generated Gherkin feature file with 3 scenarios from TestCompass based on the selected requirements coverage form ‘Path Coverage’. Also included is a general section containing the date, name of the project and model and the coverage used for generating the Gherkin feature file. Of course, it is always possible that later, one or more changes may need to be made to the requirements, and as a result, the Example map or Feature map may also require updating. This can potentially impact the scenarios in the previously created Gherkin feature files. However, in TestCompass, implementing such changes is a breeze, as you can effortlessly incorporate them and instantly generate the Gherkin feature files automatically once again. This means you don't have to manually update existing Gherkin feature files or create new ones from scratch.

A significant additional advantage provided by TestCompass is its ability to perform an Impact analysis. This means that when a change occurs, TestCompass automatically generates a comprehensive overview of the scenarios from the related Gherkin feature files, highlighting their new status, such as updated, unchanged, removed, and added. With this feature in TestCompass, it becomes entirely transparent which scenarios of the previously generated Gherkin feature file were affected by the change and in what manner. This allows for a clear understanding of the precise impact brought about by the change in question.




```
Feature: Apply reservation charges to library members

  TestCompass
  Project - 4. Example mapping (BDD)
  Test Model - 2b. Graphical model 'Reservation charges'
  Date - 2023-05-24

  Test Coverage - 100% (Path Coverage)

  Rule: Adults pay a reservation charge of $1 per item

  Scenario: 2b. Graphical model 'Reservation charges' - TC1
    Given Andrew is an adult member
    When Andrew reserves a book (YES)
    Then The charge is $1

  Scenario: 2b. Graphical model 'Reservation charges' - TC2
    Given Andrew is an adult member
    When Andrew reserves a book (NO)
    But Andrew reserves a children's book (YES)
    Then The charge is $1

  Scenario: 2b. Graphical model 'Reservation charges' - TC3
    Given Andrew is an adult member
    When Andrew reserves a book (NO)
    But Andrew reserves a children's book (NO)
    But Andrew reserves 5 books
    Then The charge is $5
```


To summarize, some of the key advantages of using TestCompass for the Discovery and Formulation phases of BDD:

- Streamlined collaboration: TestCompass facilitates communication and collaboration between business stakeholders, developers, and testers. It provides a platform where everyone can work together to create a common language that is easy to understand and interpret.
- Automated documentation: TestCompass automatically documents all information from the requirements discovery sessions. This means that there is no need to manually record or transcribe the results, saving time and effort.
- Better visibility: TestCompass provides better visibility of the requirements discovery and formulation processes. This makes it easier to share and discuss ideas and examples, especially when the session is done online.
- Reusability: TestCompass allows sessions to be reused for similar projects or features. This saves time and effort in future projects and helps to ensure consistency across different teams and projects.
- Lower chance of errors: By using TestCompass, there is a lower chance of making typos or other errors in the next phase of BDD formulation, where the examples will be described in a formalized language.
- Graphical representation: TestCompass can convert the example map or feature map into a graphical model with a high level of abstraction. This makes it easier for both business stakeholders and technical stakeholders to understand and review the requirements.
- Automatically generate coverage-based Gherkin feature files directly from the graphical model.
- Automated Impact analysis after a change in the requirements. Provides an overview which previously generated Gherkin feature files were affected by the change and in what manner.

Overall, TestCompass is a powerful collaborative modeling tool that in addition to the early Model Based Testing (eMBT) approach, can support the Discovery and Formulation phases of BDD in a very easy to use and intuitive way. TestCompass can be an excellent choice for organizations that are looking to adopt a BDD approach and improve collaboration and communication between all stakeholders involved in the software development process.

References:

Example mapping - https://cucumber.io/blog/bdd/example-mapping-introduction/

Feature mapping - https://johnfergusonsmart.com/feature-mapping-a-lightweight-requirements-discovery-practice-for-agile-teams/

- SILVIO CACACE

Silvio Cacace is an experienced and passionate context driven test professional with long experience in software testing field (since 1994).

Silvio is also founder of TestCompass®, the easy to use and early Model Based Testing tool (eMBT) and Behavior Driven Development (BDD)-supported Collaborative modeling tool, in the cloud.

PHASE SPACE -

THE CONTROLLED CHAOS

ARSLAN ALI - PRESALES AND SOLUTIONS CONSULTANT

Arslan Ali is a veteran software tester and solutions consultant from Pakistan. Having more than 20 years of diverse experience in information technology services and IT education, Arslan's main focus now is community building and consulting.

He is the founder of the BeingTester initiative and also co-founder of the Pakistan Software Testers Society and SQAs from Pakistan, where testers from around the country share experiences, assist other testers and help build a stronger QA community.

FAIZA YOUSUF - PRODUCT MANAGER AND COMMUNITY BUILDER

Faiza Yousuf is a software engineer with 13 years of experience building products and teams. She is a serial founder focusing on women's financial inclusion initiatives and improving gender parity in tech. Faiza is a multi-award-winning community leader. She founded WomenInTechPK and cofounded CodeGirls and CaterpillHERs.

Faiza is an Expert-Vetted freelancer on Upwork and was featured in many of their campaigns. She is a well-known international speaker, loves to read, and enjoys writing about tools and practices.

The greatest challenge for testing software is controlling the time alongside the completeness of test cycles, in contrast to the complexity of the software application.

In the case of development, this is resolved by adopting agile methodologies or continuous delivery methods, but if we look closely, it is easily perceivable that these practices are exerting an enormous amount of pressure on quality assurance and testing practices, not only in terms of delivery timelines but also in addressing the definitions of "done."

Most of the decision-making is on retesting a certain feature while regression is happening inside a tester's head. They (testers) take these decisions as on their experience with that specific application or any other applications they have tested before that.

So, there is no Project Manager who would know about the density of testing, and this is not something that is properly discussed. So, it can create loopholes.

Impact analysis and risk analysis of certain features in testing are also not conducted formally, so again, implicit knowledge is only in the tester's head and if the tester is not experienced, then you are in dangerous territories.

This also never becomes part of the conversation between developers, testers, and PMs. No matter if they write test cases or test scenarios or if the product has a requirement document or not.

We need to address this problem from the testers' and users' perspectives. To achieve this, there is a need to view the application as a complex interconnected ecosystem. Once the application is contained in a finite space of time and context, it will be easier for testers to determine which objects to be tested for which release, even though for the development team those components are not appended within the scope.

We believe that there is a presence of "gravitational" force in effect within software objects, and if we can determine this delicate relationship, then explore phase space and complexities of the system embedded within that phase space in order to determine the full coverage of testing in a limited timeline might become possible.

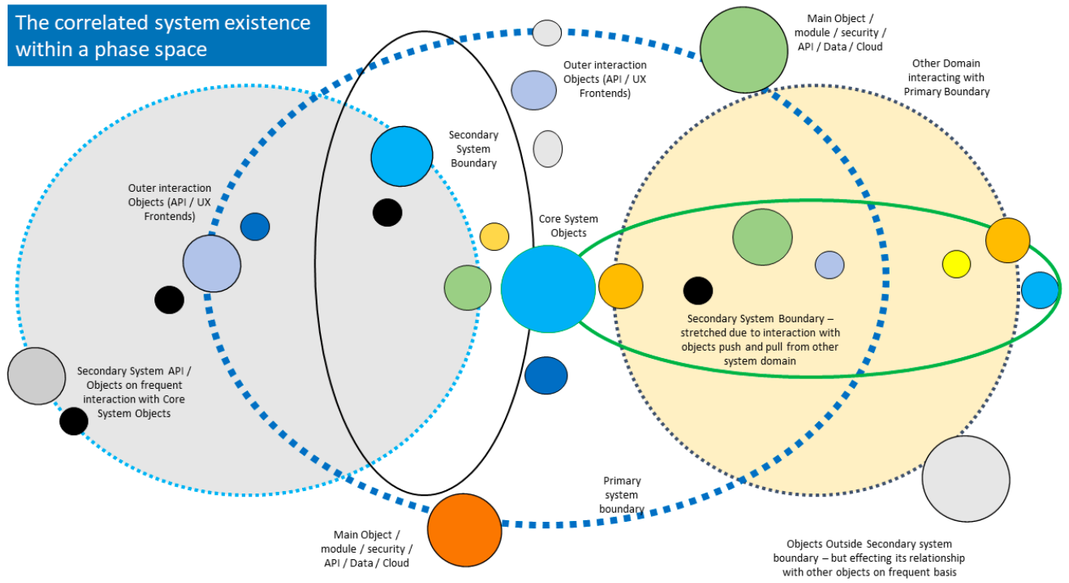
This research will assist software testers and business analysts in determining the layered structure of the application, help them discover areas of vulnerabilities and bugs, and also propose a suggestive approach for the development team.

We have extended the software definition and heuristics from the Context-Driven School of thought to a much more in-depth approach to understanding the existence of software application components and their relationships.

Introduction:

Software Systems are dynamic by nature which eventually makes them complex. This aggressive behavior is relative to users and market evolution. The dynamics of a software application depend on many factors, such as its architecture and coding structure, the data it is going to process, the function it's going to perform, its platform dependencies, its users. Its users' behavior and its life cycle.

The concept of containing software systems in phase space is to visualize systems with all their state and complexities. This complexity can be calculated down to singularity to the extent of the entire software application, subsequently providing testers with an approach to drill up and down the details as per their testing requirements, requirement specifications, context, and scope of testing.



The Challenges for Testers and Test Teams:

Thence, if we put things in perspective, testers can focus/de-focus on features we need to test more often than the others;

- As there are frequent changes reported by the clients and/or internal teams there is a high risk associated with the feature, and the vulnerability assessment is extremely necessary to be tested and verified
- The Impact zone of the system object(s) expands beyond system boundaries, and a failure can affect customers, data, and internal functions. At the time of scheduled builds or main application releases the test team has the burden to find these impact zones, but this is easier than said than done. No matter how many test cases are written the business and human context keep contracting and expanding the impact boundaries. Usually, people find solutions to this in automation, but in reality, a good map of the application with the identification of go-no-go areas can help testers better.
- The high business value of a certain application component, a marketable feature, or a claim that works as a potent to sell the system to its client, increase sustainability and retain existing market clients
- For humans, the interaction with these applications differs due to several contextual dependencies. Therefore, the perception of value for each user becomes dependent on several known and unknown factors.
- It carries independent relational factors from entity to entity, developers, testers, business analysts/product owners, customers, and users.
- We can measure this with certain inflation factors affecting requirements, release reworks, and the number of changes being incorporated after the market release of the application.

The Test Coverage Concerns: (Infinite Space / Finite Time)

Thence, if we put things in perspective, testers can focus/de-focus on features we need to test more often than the others; As there are frequent changes reported by the clients and/or internal teams

There is a high risk associated with the feature, and the vulnerability assessment is extremely necessary to be tested and verified

The Impact zone of the system object(s) expands beyond system boundaries, and a failure can affect customers, data, and internal functions. At the time of scheduled builds or main application releases the test team has the burden to find these impact zones, but this is easier than said than done.

No matter how many test cases are written the business and human context keep contracting and expanding the impact boundaries. Usually, people find solutions to this in automation, but in reality, a good map of the application with the identification of go-no-go areas can help testers better.

The high business value of a certain application component, a marketable feature, or a claim that works as a potent to sell the system to its client, increase sustainability and retain existing market clients

For humans, the interaction with these applications differs due to several contextual dependencies.

Therefore, the perception of value for each user becomes dependent on several known and unknown factors.

It carries independent relational factors from entity to entity, developers, testers, business analysts/product owners, customers, and users.

We can measure this with certain inflation factors affecting requirements, release reworks, and the number of changes being incorporated after the market release of the application.

Reworks, Cost of Quality, and Software Release Inflation:

The following data is extracted from JIRA trends of application maintenance release trains. The data reflects a sequence of scheduled releases for the clients during a period of two years. What the project managers needed help to grasp was the effect of reworks on each release due to limitations in JIRA data.

For this, a complete combined picture of data was required. The table, therefore, represents an exclusive chronological representation of the data of main releases and further on the subsequent patches of these releases.

In the table below and the graphical representation, one can see that the rework creates each release to inflate around 20% on average. Small releases with very less amount of testing and regression effort and quick timelines have even a higher percentage, as the testing time applied on these releases due to their volume is reduced, in contrast to the impact created by each release.

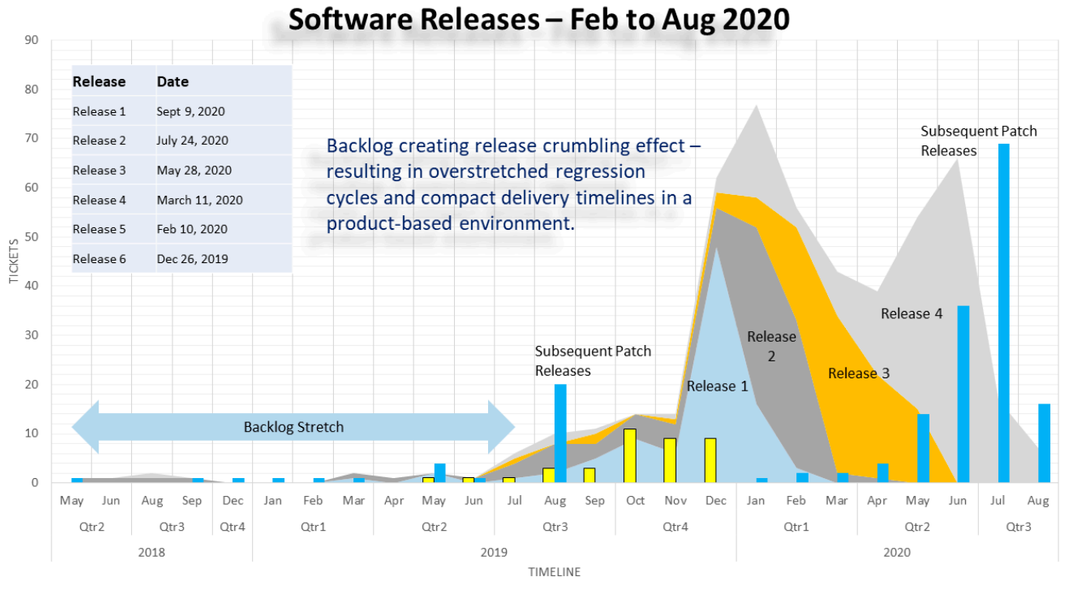
Also to be noted is the "Code Fix" percentage per release, meaning to fix the reported bug, the development has to do code changes in the application, and with this, the testing time and test coverage concerns for the test team become big challenges.

This graph represents the chronological release-wise data. As a release manager, the biggest challenge is to manage the content and time of each release.

There are five releases here, and the contents overlap each other, meaning the requirements established even at the same time are not delivered to the client together. They are either prioritized or delayed. The priority is manageable, whereas the "delay" creates backlogs for the test team, and this creates an overhead.

The above pattern reflects that the drag of these backlogs results in a crumbling effect toward the release date and exerts a force on both release boundaries.

In case of minor code changes resulting in modification of a multi-connected function will eventually result in testers executing all the test cases to provide test coverage, and this also means the expense of more time and effort, even if these test cases are automated.



Release Inflation Matrix

Release	Total Tickets in Actual Release	Code Fix in Actual Release	Code Fix %age	Release Inflation by Tickets	Release Inflation %age	Inflated Code Fix	Inflated Code Fix %age
Release 1	38	24	63.16%	6	15.79%	4	16.67%
Release 2	95	26	27.37%	23	24.21%	10	38.46%
Release 3	107	62	57.94%	39	36.45%	17	27.42%
Release 4	100	77	77.00%	22	22.00%	6	7.79%
Release 5	189	119	62.96%	84	44.44%	24	20.17%
Release 6	359	205	57.10%	60	16.71%	20	9.76%
Patch 1	13	13	100.00%	30	230.77%	3	23.08%
Patch 2	66	44	66.67%	39	59.09%	12	27.27%
Patch 3	26	12	46.15%	12	46.15%	3	25.00%
Release 7	131	47	35.88%	6	4.58%	0	0.00%
Total	1124	629	56%	321	28.56%	99	15.74%

On the other hand, the tickets other than Code-fixes, such as "understanding issues", "system configuration", "environment setups", "not reproducible", and "system updates", actually become overhead for testing efforts because as a preventive measure, it is still considered as a good practice where test team close all the tickets whether these are not created nor reported by the Test team members. Consider something marked as a bug and is in an open state but with a flag as "Configuration", which will be an overhead to the QA team unless it is rechecked and closed.

This creates pressure, not only on the QA team on their regression and automation timelines but also creates inefficient implementation schedules for business teams.

The determined numbers of changes here are contextual as different software applications and their release strategies may differ as per their Industry/domain, team capacity, and business objectives.

Dissecting the challenges:

Incorporating changes or new enhancements in an in-use application also depends on how much expansion or wobble can be introduced by the solution providers where the application can pass its sanity tests, and the code remains controllable.

Now, to understand the below-mentioned challenges it is important to understand that there can be several examples to depict here, but we have chosen two distinct scenarios; first, a traditional ERP development approach where there are sequenced releases with fixed schedules and the risk of inflated release is pretty much controlled. And second, where the development is chaotic and the release frequencies depend on the customer request for changes, this is also scheduled - today we called this continuous delivery with continuous integrations.

There are a couple of aspects that can fluctuate application complexity and how much development can expand with changes

Common code-base for several different clients operating on different business domains:

This approach is fast in delivery but higher in risk, as one change on a single client may affect many clients, the same happens in terms of bug fixes. Therefore, test teams are faced with the challenge of impact analysis, and multi-domain-based testing (in a limited timeline), meaning separate lines of tests to be executed for different business domains in which the application is running, the example of such domains is "Finance", "Textile", "Education", "Services", "Retails", "FMCG", "Pharmaceuticals", "Human Resource", etc.

Thence expansion and wobbling are very carefully managed by the development manager, and the system can stretch its boundaries to a certain limit only. Not everything can be changed as per the client's requests and wish lists, therefore some of the system aspects are either pushed back to the timeline while others are deferred.

The most efficient way to cater to multi-domains on a single code base is a "configuration dashboard" approach. Where the delivery team can toggle the features of the entire application and customize changes after every gap analysis they are doing.

The changes are also done on form levels where certain features and UI aspects are carefully marked for the clients, so if an FMCG client is using a Purchase order, they may not use/want to use certain buttons/options/ fields on the form, as they may belong to "Textile" domain.

Multiple branches cater to several clients, although the business domain remains the same

The second approach for release managers is to create multiple code branches and keep updating the main branch with the changes to code and delivery lines, this works for smaller applications, but the system can experience wobbling of the boundaries as soon as the data, the issue tracker and release timelines get entangled with each other. This phenomenon occurs when a feature for a certain client is to be delivered and is not scheduled for other clients due to its lower priority.

In another case, a completely independent release train is managed with all independent deliveries for one specific client. However, on a "major release" event, the changes are merged with the master branches. This entanglement is tricky and creates reworks on maintenance and regression levels for both development and testing teams, respectively.

The Layered Cake Approach:

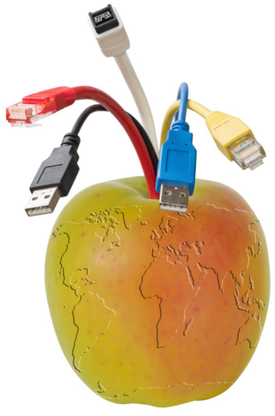
On one hand, this mix is highly creative and interesting, but on the other hand, it is a daunting challenge, as finding balance for the application for each entity, namely, customers, developers, business teams, and testers, is not something that can be dealt with at the coding level, we need to see things from a different vantage point.

It is similar to viewing applications in the form of a layered cake, where the outer layers represent a more presentable form and the inner layer represents a stable structure and a solid base. Viewing each of these layers is contextual in nature. For frontend developers, testers, and customers, the outer layers matter the most, but for core developers, and automation engineers, the inner layers become the focal point of change and chaos.

This also depends on the problems and the discovery of the bugs, therefore in so many words, the balance is to keep the application sanity in order, and to an extent, work with the controlled chaos.

TEST AUTOMATION USING ETHERNET CONTROLLED RELAYS

- **RAHUL PARWAL AND ABHISHEK BAWKAR**



- RAHUL PARWAL

Rahul Parwal is a Software Tester from India. He is a recipient of the prestigious Jerry Weinberg Testing Excellence Award.

Rahul is an avid reader, blogger, and conference speaker who likes to share his thoughts on various social media platforms. Recently, he has also been inducted as a LambdaTest Spartan, & a Browserstack Champion for his work in the field of software testing. Presently, he works as a Senior Software Engineer with ifm engineering in India.

Software testing of any Hardware in Loop (HIL) system is a very critical and challenging task. Primarily, during the testing of the software, other critical aspects related to hardware such as the state of device, power supply control, communication channels / modes and much more need to be managed continuously. All these other aspects create a challenge if we want to scale automated software tests on n hardware devices as they include a manual test step per device per session. Moreover, manual intervention is required for software tests related to device.

This article focuses on an in-house developed Ethernet controlled relay-based mechanism for achieving a fully automated test execution process. In this solution, relay boards are used for controlling the device process along with any test-specific step. Although, standard ethernet controlled relay solutions are available in the market, we decided to create a custom board solution as it offers the following advantages:

1. Customization: A custom-made relay board can be designed to meet the specific needs of a particular application, whereas a purchased relay board may not have all the desired features or may have features that are not needed.

2. Cost: Depending on the complexity of the relay board, building one may be more cost-effective than purchasing a pre-made one.

3. Quality: A custom-made relay board can be made with higher quality components and more precise construction, which may result in improved reliability and longer lifespan compared to a purchased relay board.

4. Learning opportunity: Building a custom relay board can be a valuable learning experience, as it allows you to understand how the board works and how to troubleshoot any issues that may arise.

5. Unique solution: Building a custom relay board allows you to create a unique solution that may not be available on the market.

Introduction

A controller may be a microchip or separate hardware device for the control of a peripheral device. For our system, we are using the ifm ecomat mobile series controllers. The ifm ecomat mobile series controllers are designed for use in harsh and rugged conditions. They are suitable for direct installation in vehicles and mobile machines. For safety-critical tasks, safety controllers are also available. The signals produced by sensors are quickly and reliably processed by the controllers and provided to the actuators. Besides multifunctional inputs and outputs for different applications, each controller has several CAN interfaces and an RS232 (Serial) interface. Some controllers also support Ethernet interface. The programming tools are in line with IEC 61131-3 for control systems used in mobile machines provide the programmer with a variety of programming languages for fast project implementation and management through an SDK API solution, which is known as **ifm SDK API**. A powerful visualization module for graphic visualization of the machine and installation functions, also known as **Maintenance Tool**, completes the package. ifm's control systems can be completely operated via the programs developed using ifm SDK API. The connection to the device is established via a standardized serial interface (RS232) or a CAN interface and in some cases using an Ethernet interface also. The software allows setting of all communication parameters of the connected controller, programming of the controller, and diagnosis/visualization of the available data in the controller.

MNTT Auto Tester is a test automation hardware setup built to meet testing requirements of Maintenance Tool and ifm SDK API project. Using the Auto Tester, it is possible to execute positive as well as negative tests on both Maintenance Tool software and ifm SDK API software. The commands supported on the controller are available under three categories, i.e., read, write, and execute software commands.

Hardware Block Diagram and Description

Auto Tester consists of following hardware components:

a. A Host PC with configuration:

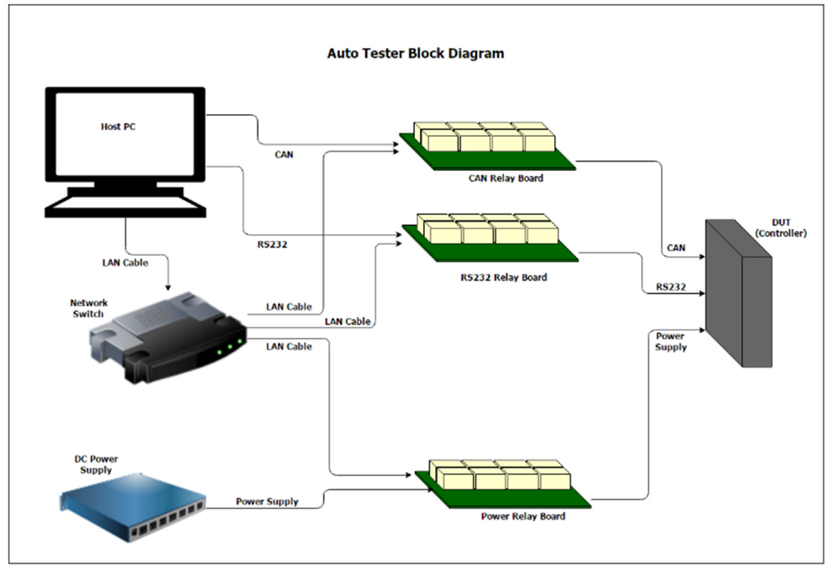
- Processor: Intel Core i5-4590 CPU @ 3.30 GHz
- Internal RAM: 8 GB
- Operating System: Windows 10 Pro 64-bit

b. NCD ProXR Enhanced Ethernet Controlled Relay Controller.

c. Industrial Rack by APW President.

The Host PC and the relay controller boards are routed via LAN. The Power Supply, CAN, RS232 (Serial), and Ethernet connections from the Device-Under-Test (DUT) are routed through the relay.

Using appropriate commands from the host PC, the relays can be turned ON or OFF, which in turn connects or disconnects the DUT's power supply and communication supply buses.



1. Controller Classification

The Controllers can be classified on various parameters such as:

1. Family
    i. Basic System
    ii. Classic / R360 System
    iii. Ecomat / R360 III System
2. Safety Type: Safe, Non- Safe
3. Architecture: 16 Bits, 32 Bits
4. Communication Support
    i. CAN Support only
    ii. CAN + RS232 Support only
    iii. CAN + RS232 + Ethernet Support

2. Communication Interfaces Classification

1. CAN (Controller Area Network) Communication

A Controller Area Network (CAN bus) is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other. Popular CAN interfaces for which the controllers are tested are as mentioned below:

i. CANfox
ii. CAN-PEAK
iii. IXXAT
iv. Kvaser

Each CAN cable has two main wires for data communication:

i. CAN High (CAN_H)
ii. CAN Low (CAN_L)

For successful communication between receiver and transmitter using CAN protocol, the CAN High (CAN_H) of transmitter device must be connected to the CAN High (CAN_H) of the receiver device and CAN Low (CAN_L) of transmitter device must be connected to the CAN Low (CAN_L) of the receiver device. There should be a 120 ohm termination resistance between the CAN_H and CAN_L lines.

2. RS232 Communication

RS232 is a serial information transfer protocol standard that defines the protocol (method of transmission of data) as well as the physical hardware required. Fundamentally it is a method of transferring data across a single wire. Data is transmitted serially in one direction over a pair of wires.

Each RS232 cable has two main wires for data communication:

i. Transmission (Tx)
ii. Reception (Rx)

Data going out is labeled Tx (indicating transmission) while data coming in is labelled Rx (indicating reception). For successful communication between receiver and transmitter using RS232 protocol, the Tx of transmitter device must be connected to the Rx of the receiver device and vice versa.

3. Ethernet Communication

Ethernet is a standard communication protocol embedded in software and hardware devices. Ethernet is widely used for home and industry. The Internet Protocol is commonly carried over Ethernet, and is considered one of the key technologies that make up the Internet.

Each Ethernet cable has four main wires for data communication:

i. Transmit Data Plus (TxD+)
ii. Reception Data Plus (RxD+)
iii. Transmit Data Minus (TxD-)
iv. Reception Data Minus (RxD-)

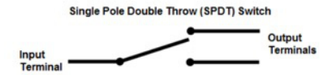
For successful communication between receiver and transmitter using Ethernet protocol, all the main wires for data communication of transmitter device must be connected to the same wires of the receiver device, i.e. TxD+ of transmitter device with TxD+ of receiver device, RxD+ of transmitter device with RxD+ of receiver device, TxD- of transmitter device with TxD- of receiver device and RxD- of transmitter device with RxD- of receiver device.

3. Relay

A Relay is an electrically operated switch. It consists of a set of input terminals for a single or multiple control signals, and a set of operating contact terminals. A SPDT (Single Pole Double Throw) Switch and DPDT (Double Pole Double Throw) Switch are the basis of Relay / Relay Board.

1. SPDT (Single Pole Double Throw) Switch

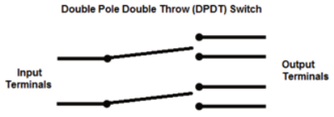
A Single Pole Double Throw (SPDT) switch is a switch that only has a single input and can connect to and switch between 2 outputs. It has one input terminal and two output terminals.



The input terminal is known as COM (Common). The output terminal connected to the internal terminal is known as NC (Normally Closed), whereas the output terminal not connected to the internal terminal is known as NO (Normally Open).

2. DPDT (Double Pole Double Throw) Switch

A Double Pole Double Throw (DPDT) switch is a switch that has 2 inputs and 4 outputs; each input has two corresponding outputs that it can connect to.

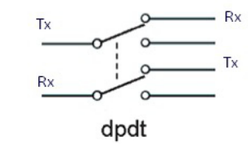


Each of the terminals of a double pole double switch can either be in one of the two positions. This makes the double pole double throw switch a very versatile switch. With two inputs, it can connect to four different outputs. It can reroute a circuit into two different modes of operation. A Double Pole Double Throw (DPDT) Switch is a combination of two single pole double throw (SPDT) switches. This DPDT switch arrangement is called a Relay.

Similar to SPDT, in the above diagram, each input terminal is known as COM (Common). Each output terminal connected to the internal terminal is known as NC (Normally Closed), whereas each output terminal not connected to the internal terminal is known as NO (Normally Open).

3. Relay (DPDT) connection for RS232 and CAN Communication

A DPDT arrangement can be used as a relay for connecting the RS232 and CAN Communication wires:



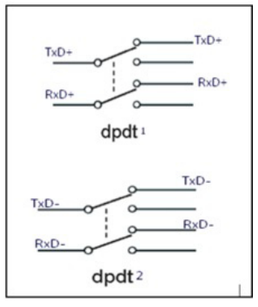
In the above figure(s):

i. CAN_H --> CAN High
ii. CAN_L --> CAN Low
iii. Rx --> Receive
iv.Tx --> Transmit

Note: Tx is connected to Rx and vice versa for Serial Transmission and Reception.

4. Relay (DPDT) connection for Ethernet Communication

Two DPDT(s) arrangement can be used for connecting single Ethernet Communication wire as shown in the figure below:



In the above figure:

i. TxD+ → Transmit Data Plus
ii. RxD+ → Reception Data Plus
iii. TxD- →Transmit Data Minus
iv.RxD- → Reception Data Minus

5. Relay Board And Auto Tester

Relay Boards are electronic boards with an array of relays and switches. They have input and output terminals and are designed to control the voltage supply. Relay boards provide independently programmable, real-time control for each of several onboard relay channels. A single relay board is a collection of eight relays.

Single Relay Board = 8 Relays = {R1, R2, R3, R4, R5, R6, R7, R8}

Auto Tester is a collection of two or more relay boards i.e. no of relay boards * 8 relays.

Two types of Auto Testers have been developed to control devices having CAN and RS232 (optional) support and CAN, RS232 and Ethernet Support.

The Auto Tester that supports only the classic industrial communication protocols (i.e. CAN and RS232) is called as the **Classic Auto Tester.** The Auto Tester that supports all the industrial communication protocols supported by the ecomat controllers (i.e. CAN, RS232 and Ethernet) is called as the **Ecomat Auto Tester.**

The Relay Boards are classified into the following types as per their functionality:

i. Communication Relay Boards
ii. Power Supply Relay Boards

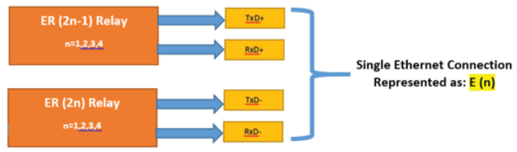
a. Communication Relay Boards

Communication Relay Boards is a relay board for connecting communication interfaces.

For Classic Auto Tester, Communication Relay Board = CAN {R1, R2, R3, R4} + RS232 {R5, R6, R7, R8}

For Ecomat, Auto Tester, Communication Relay Board = CAN {R1, R2, R3, R4} + RS232 {R5, R6, R7, R8} + Ethernet {(ER1, ER2), (ER3, ER4), (ER5, ER6), (ER7, ER8)}

Each Tuple of Ethernet Relay in the above statements forms a single Ethernet Connection:



Each CAN Relay is a DPDT with CAN_H and CAN_L:

Each RS232 Relay is a DPDT with Rx and Tx:

b. **Power Supply Relay Boards**

Power Supply Relay Boards is a relay board for connecting power supply cables.

Power Relay Board = Power Relays {P1, P2, P3, P4} + Free / Empty Relays {P5, P6, P7, P8}

Each Power Relay is a DPDT with GND and Vcc:

6. Classic Auto Tester

Classic Auto Tester consists of two relay boards (1 Communication Relay Board + 1 Power Relay Board)

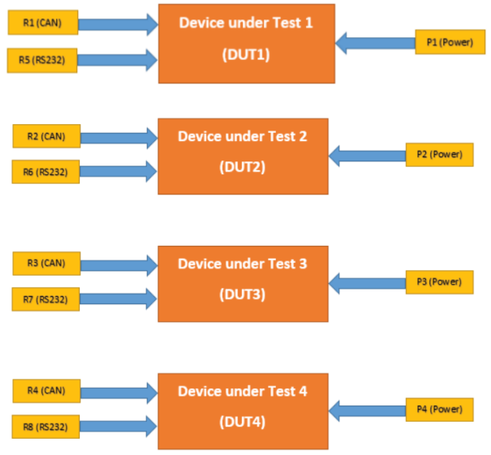
Communication Relay Board = CAN {R1, R2, R3, R4} + RS232 {R5, R6, R7, R8}

Power Relay Board = Power Relays {P1, P2, P3, P4} + Free / Empty Relays {P5, P6, P7, P8}

The classic Auto tester (Basic / R360 System) can thus be visualized as in the chart diagram given below:

Assume Device Under Test as D1, D2, D3, and so on.

Each device would have 1 RS232 Relay, 1 CAN Relay (For Communication Interface) and 1 Power Supply Relay connected to it.



The Classic Auto Tester only supports maximum four devices.

We currently have four Classic (Basic+R360) Auto Testers = 4 * 4 = 16 devices support.

Each Auto tester is assigned a particular IP address.

7. Ecomat Auto Tester

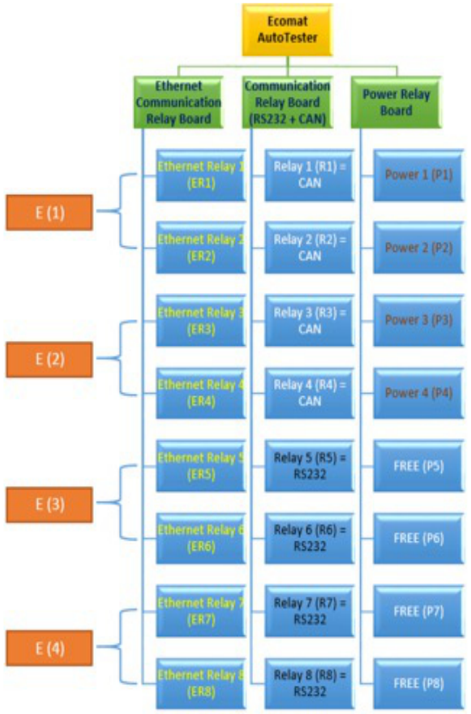
Ecomat Auto Tester consists of 3 Relay Boards (2 Communication Relay Boards + 1 Power Relay Board)

Communications Relay Board = CAN {R1, R2, R3, R4} + RS232 {R5, R6, R7, R8} + Ethernet {(ER1, ER2), (ER3, ER4), (ER5, ER6), (ER7, ER8)}

Each Tuple of Ethernet Relay in the above statements forms a single Ethernet connection.

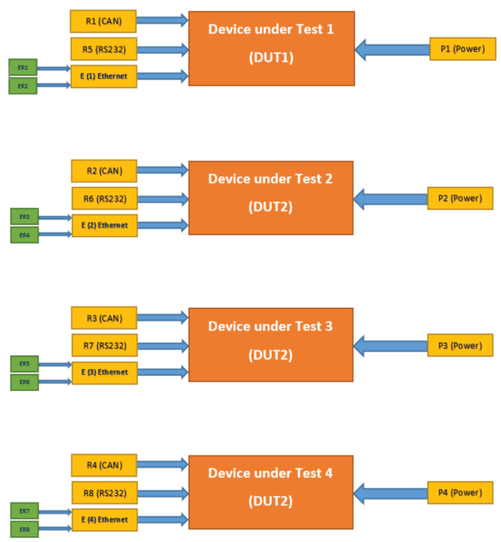
Power Relay Board = Power Relays {P1, P2, P3, P4} + Free / Empty Relays {P5, P6, P7, P8}

The Ecomat Auto Tester can thus be visualized as per the chart diagram you see next.



Assume Device Under Test as D1, D2, D3 & D4 and so on.

Each device would have 1 Ethernet Connection (2 Ethernet Relays), 1 RS232 Relay, 1 CAN Relay [For Communication Interface] and 1 Power Supply Relay connected to it.



Therefore, it is evident from the above representation that one Ecomat Auto Tester supports maximum four devices only.

The below image shows the fully implemented autotester rack with all the relay boards (communication, power) and autotesters (classic, ecomat):

8. Software System Overview

Autotester PC uses different test solutions for testing the ifm SDK API and Maintenance Tool. In both the testing solutions, a byte array for the appropriate command (such as Start or Stop Power, Start or Stop Communication, etc.) are sent to the socket of the relay board. The socket communication over ethernet controls the various relay boards and autotesters.

The Autotester PC has the following software(s) installed for the test execution on the ifm SDK API:

a. Python 3+

b. Pytest 5+

The Autotester PC has the following software(s) installed for the test execution on the Maintenance Tool:

a. WinAppDriver

b. .NET Framework 4.7+

c. NUnit

A JSON configuration file is maintained for defining the environment parameters for the test execution. It contains parameters like:

- Test PC MAC address
- Communication type to be used (0 = CAN, 1 = RS232, and 2 = Ethernet)
- Autotester(s) available for testing

- IP Address of the Autotester(s) to test (To establish Ethernet connection with the Autotester)
- Port of the Autotester (For socket connection)
- Execution flag per Autotester
- Controllers available per Autotester for testing
- Execution flag per controller
- Controller type (16 bit or 32 bit)
- Controller safety supported info (To run safe controller related tests also)
- Controller node id
- CAN baud rate
- RS232 baud rate

```
{
  "TestPC": {
    "MAC Address": "F4-8E-38-7C-73-14"
  },
  "CommTypUnderTest": 0,
  "MaxPorts": 4,
  "AutotesterConfig": {
    "Autotester_1": {
      "IP Address": "172.29.169.51",
      "Port": 2101,
      "MAC Address": "00-80-A3-AC-70-FF",
      "CommunicationRelayBank": 1,
      "PowerSupplyRelayBank": 2,
      "TestExecution": true,
      "CommConfig": {
        "types": [
          {
            "Comm_Type": "CAN",
            "channel_win": "canfox(1) ch01",
            "channel_lin": "ican0"
          },
          {
            "Comm_Type": "RS232",
            "channel_win": "com4"
          }
        ]
      },
      "Controllers": {
        "CR0032": {
          "TestExecution": true,
          "Port": "1",
          "type": "32Bit",
          "Safety": "NonSafe",
          "NodeID": "127",
          "CAN_BaudRate": "125",
          "RS232_BaudRate": "115200"
        },
        "CR0232": {
        "CR0020": {
        "CR0200": {
        }
      },
    "Autotester_2": {
    "Autotester_3": {
    "Autotester_4": {
  }
}
```

Based on the values in the configuration file, the test engine initiates an auto tester setup function which performs the setup operation for the specific test device.

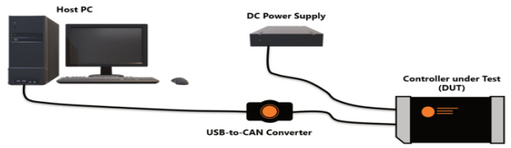
For example, In the above sample file, the test execution for CR0032 controller is set to true. Also, the CommTypeUnderTest is set to 0, which implies that the test must be executed on a CAN communication interface. Thus, the auto tester setup function would initiate a socket command to turn ON the power and communication (CAN) supply lines for CR0032. This would set up the environment for test execution to work on CR0032.

The test scripts call the power or communication ON & OFF blocks based on the test flow. These functions read the parameters from the autotesterconfig file and accordingly send the commands to the appropriate relay lines of the specific device under test.

For manual troubleshooting, we also have a GUI application that comes along with the NCD relay boards. However, the GUI application needs to be turned off when controlling the NCD relay boards via socket commands as the NCD relay boards can only have one exclusive connection at a time.

9. Results & Analysis

The initial basic setup for the testing without any relay board or autotester setup involved a lot of manual testing or human interventions during the entire test execution cycle. There were repeated manual interruptions on communication channel or power supply for executing the test cases. The initial basic setup for testing is shown in the below figure:



Initial Setup:

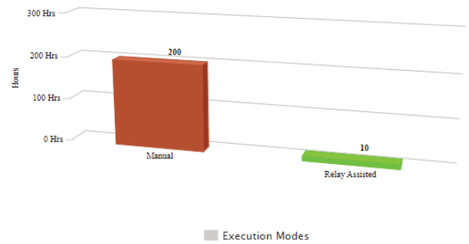
- Total test cases: 150 (approx.)
- Time taken to execute a single test case on a single controller: 15-16 mins (approx.)
- Total execution time per controller: 150 x 16 mins = 2400 mins = 40 hours = 5 person-days (or 1 week)
- Total number of R360 Classic series controllers (device under test): 5
- Total execution time for complete test cycle: 5 x 1 week = 5 weeks

After establishing communication channel or power supply through Ethernet controlled switches i.e. relays and performing communication channel or power supply failure via Ethernet controlled relays, we are able to significantly reduce the overall execution time and improve the overall efficiency of our testing.

Our Python and C# scripts now, programmatically control controller's power supply and communication lines removing the manual interventions and need to do attended testing. Now, we can run most of our test cycle tests in an automated and unattended mode.

Relay Assisted Test Execution Setup:

- Total test cases per controller: 150 (approx.)
- Time taken to execute a single test case on a single controller: 50-55 seconds (approx.)
- Total execution time per controller: 150 x 50 seconds = 7500 seconds = 125 mins = 2 Hours (approx.)
- Total number of R360 Classic series controllers (device under test): 5
- Total execution time for complete test cycle: 5 x 2 Hours = 10 Hours



Overall test cycle time has been reduced from 5 weeks (200 Hours) to 10 Hours. % Time Saving = x 100 = 95%

10. Conclusion

This paper presents test automation using ethernet controlled relays for hardware in loop testing. The relay-based mechanism is a time and cost-effective solution for the repetitive manual interventions required in any hardware in loop testing setup.

We have found several benefits of using a relay-based automation mechanism over a manual approach:

- **Increased efficiency:** Automated systems can operate faster and more accurately than humans, leading to increased efficiency and productivity.
- **Reduced errors:** Automated systems are less prone to errors than humans, leading to improved quality and accuracy.
- **Reduced costs:** Automated systems can operate around the clock without the need for breaks or time off, leading to high availability and provision for scheduling in off work hours.
- **Increased flexibility:** Automated systems can be easily programmed and reprogrammed to perform a variety of tasks, increasing the flexibility of the system.
- **Enables remote working:** This setup can also enable engineers to work remotely once the setup is ready. During the Covid Pandemic phase, where teams were forced to work from home and the physical access to the controller device setups in office was not available, relay-based auto tester solution proved to be highly robust, reliable, and efficient. This solution is a great enabler for virtual mode of working.

Also, one of the most common challenges for any IoT Testing Team is the tasks to manually setup the tests that involve devices as they often need power or communication breaking and connecting mechanism for various types of tests. With a relay-based mechanism, it is possible to easily create remote setups that can be controlled using socket commands. Also, as the relays are controlled via socket programming (commands), it is possible to control them using any high-level programming language that facilitates socket programming.

When compared to the existing ready to use hardware in loop testing solutions that are readily available in the market, our evaluation shows that the relay board is highly customizable and cost-effective solution. The simple and easy customizability gives a testability edge for any application that needs to be tested using such a hardware in loop setup.

We have been using the relay-based auto tester setup for test automation since 2017 and all our evaluations demonstrate that it is an effective, practical, and promising solution for any hardware in loop testing requirement. Currently, we are in the process of adopting a similar setup for our IoT testing requirements.

Acknowledgment

We would like to thank, Aniket Patil (Team Lead, ifm engineering, Pune) and Kalpak Nikumbh (Associate Director, ifm engineering Pune) for their guidance, review, and continuous support from the draft to completion.

This article is co-authored by Rahul Parwal and Abhishesk Bawkar (abhishekbawkar@gmail.com)

References

[1] B. S. Achary, S. Mishra and A. Kumar, "Real time hardware in loop testing of single phase grid connected PV system," 2014 Eighteenth National Power Systems Conference (NPSC), 2014, pp. 1-6, doi: 10.1109/NPSC.2014.7103878.

[2] TPT | Testing Hardware-in-the-Loop (HiL) via ASAM XIL API (piketec.com)

[3] S. Palla, A. K. Srivastava and N. N. Schulz, "Hardware in the Loop Test for Relay Model Validation," 2007 IEEE Electric Ship Technologies Symposium, 2007, pp. 449-454, doi: 10.1109/ESTS.2007.372125.

[4] J. Wu and N. N. Schulz, "Experimental Design for Remote Hardware-In-the-Loop testing," Proceedings of ASNE Reconfiguration and Survivability Symposium, Jacksonville, Florida, Feb. 2005.

[5] A. Clerici, R. Chiumeo and C. Gandolfi, "Real Time Control Hardware in The Loop test of a novel MVDC solid-state breaker," 2020 22nd European Conference on Power Electronics and Applications (EPE'20 ECCE Europe), 2020, pp. 1-9, doi: 10.23919/EPE20ECCEEurope43536.2020.9215863.

[6] CP9030 - maintenance for PC with operating system WINDOWS or LINUX - ifm

[7] CP9031 - maintenance for ifm displays with LINUX - ifm

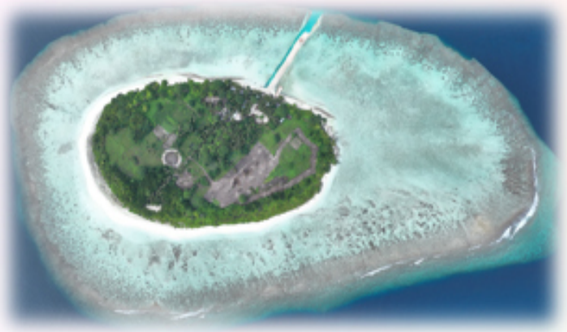
[8] Controllers - ifm

[9] Shop - store.NCD.io

[10] Double Pole Double Throw (DPDT) Switch (learningaboutelectronics.com)

COMMUNITY

ISSUE 03/2023
COMMUNITY



Register now!

Unlock your **dreams** with a chance to win a **fully sponsored trip** to **Maldives** and earn **rewards as you learn**!

TEA-TIME WITH TESTERS

JOURNAL FOR NEXT GENERATION TESTERS

CONTENT PREVIEW : ISSUE 04/2023

MORE AWESOMENESS IS ON YOUR WAY THIS SEASON!

01

DATA MIGRATION TESTING

Begin with end in mind strategy by Sandeep Garg..

02

TEA AND TESTING WITH JERRY WEINBERG

We are bringing back the treasure of knowledge that Jerry Weinberg has left behind for us. More awesomeness on its way....

03

YOUR ARTICLE HERE

Got ideas worth sharing with rest of the world? We are here to help

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