

TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

Humans, AI, and Testing!

THE PROBABILISTIC CHALLENGE

Page 18

THE QA SUPERPOWER IN AGENTIC
AGE

Page 30

OVER A CUP OF TEA WITH DOROTHY
GRAHAM

Page 14



TESTING QUALITY AI PEOPLE & MORE!

TEA-TIME WITH TESTERS

06

PEOPLE

IDEAS THAT
SPEAK FROM
THE MINDS
THAT THINK

14

INTERVIEW

OVER A CUP OF
TEA CONVO
WITH GREAT
MINDS IN TECH

18

PROCESSES

ARE YOU DOING
IT RIGHT? FIND
IT OUT

30

PRODUCTS

BUILDING
THINGS THAT
PEOPLE WOULD
USE HAPPILY

EDITORIAL BY LALIT

INTERVIEW: 14-17 A CUP OF TEA WITH DOROTHY GRAHAM



STUCK IN DETENTION- HOW REPETITIVE TASKS PUNISH ADULTS

Did anyone reading this ever get in trouble at school and get assigned a punishment to submit the next day?

06 – 09

WHY DIDN'T TESTING FIND THIS ISSUE?

After 15 years in software testing, this is still a topic I'm dealing with way too often: people who have a completely misguided understanding of what testing can and cannot do.

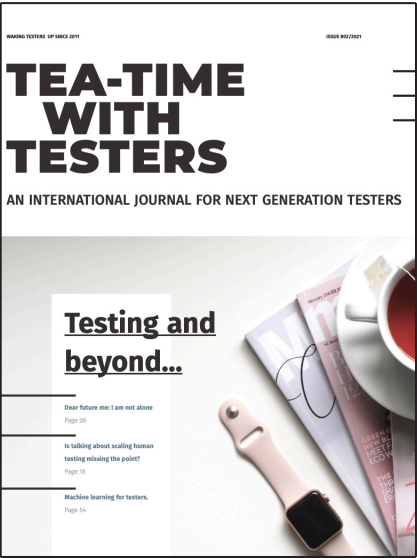
10 – 12

THE PROBABILISTIC CHALLENGE

Picture this: You wake up one morning to find that your company's AI-powered customer service chatbot has been providing wildly incorrect flight cancellation advice, or your machine learning model for hiring has systematically discriminated against qualified candidates. These aren't hypothetical scenarios – they're real incidents that have made headlines as AI adoption skyrockets across industries.

18 – 23

TEA-TIME WITH TESTERS



A NEXT GENERATION TESTING MAGAZINE

TESTING IN RED: HOW TO TURN PANIC IN PROGRESS

When testing projects go off track, it's rarely because of one big failure – it's the small cracks we overlook that turn into chaos.

24 – 26

THE QA SUPERPOWER IN AGENTIC AGE

My VS Code terminal was screaming at me with 54 TypeScript compilation errors. Red squiggly lines everywhere. The agent I'd been working with had confidently declared, "Implementation complete. All tests passing!"...

30 – 39

I'm nominated to be

The 2025 QA Leader of the Year

[Vote Now](#)

Brought to you by PractiTest

The System without Soul

As a kid, the greatest gift I received from my father was his love for reading. Buying new books wasn't an option for us, so he would borrow different books for me from the school library every now and then.

Seeing me lost in my books, I still remember how he would gently pat my back and say in my mother tongue, 'वाचशील तर वाचशील' (Vaach-sheel tar vaach-sheel), which means 'If you read, you will thrive' - a simple reminder that knowledge and perseverance will carry me forward.

I kept reading as I grew up, and fell in love with writing too. And I can tell how much this love for reading and writing has contributed to my growth as an individual.

When we launched Tea-time with Testers in 2011, its content kept global testing community quite engaged, interactive, and full of original ideas that kept testing craft moving forward. And this lasted for more than a decade I must say.

Fast forward to 2025!

The content is everywhere. Actually, more than we can consume. Every day, my feed feels like a photocopy of yesterday's photocopy. The same tone, the same clichés, the same "10 tips to..." nonsense, all those bulleted lists with plastered emoticons as if humanity invented those yesterday, most of it produced by machines that have never lived a day as a tester, never struggled with ambiguity, never fought for quality, never worked under time-pressure, never asked "But what if...?"

And the worst part? People are starting to accept it as "content." Let's please not get comfortable with this system without soul.

At least testers shouldn't.

We are supposed to think critically, question everything, and dig deeper beneath surfaces. Yet many in our field silently let generative fluff replace genuine experience. If testers don't write, don't reflect, don't pen down their unique insights, then what is left? A community echo chamber shaped by algorithms that know nothing of risk, curiosity, critical thinking, context, or consequences.

The testing craft is hungry for substance. Not summaries. Not regurgitated "best practices." Not the 100th LinkedIn post written in the same synthetic voice. If there's one thing our community needs the most right now, it's more human stories. Your stories. So please write more!

And you don't have to be a "writer" to write. You just have to be someone who cares about the craft. Every time you reflect on a tough bug, a tricky decision, a surprising pattern, a test strategy that worked or lessons learned with the failing one, that's the knowledge worth sharing. That's an experience no AI can imitate.

So write. Write not because the world demands it, but because your craft deserves it. Write because future testers will search for wisdom, and they should find yours, not a soulless imitation of it.

Tea-time with Testers will continue to hold space for those human voices, the thinkers, observers, storytellers, and explorers among us.

Once again, we are all set to do what we did a decade ago. With new ideas, new members in the editorial team, we are ready to welcome stories from humans who live the reality of their craft every day.

Until next time!

Sincerely,

Lalit



LALITKUMAR BHAMARE

CEO, Chief Editor "Tea-time with Testers"

Manager - Accenture Song, Germany
Group Lead - Innovation & Thought Leadership, Accenture QES EMEA
Director - Association for Software Testing
International Keynote speaker.
Award-winning engineering leader.
Software Testing/Quality Coach.

Connect on Twitter [@Lalitbhamare](#) or on [LinkedIn](#)

STUCK IN DETENTION: HOW REPETITIVE TASKS PUNISH ADULTS



- ELISSA TAHECH

Elissa graduated in July 2023 with a Bachelor's in Mechanical engineering and has been working at Murex in the Market Analytics team since then.

With limited coding experience, she quickly learned Python to automate manual testing processes and developed a tool that significantly improved her team's efficiency.

Her journey from novice to automation advocate taught her the importance of fully understanding each process, thinking creatively, and looking at problems from new perspectives. She aims to inspire others to embrace AI and automation, showing that if you can think of an idea, you can create it.

Did anyone reading this ever get in trouble at school and get assigned a punishment to submit the next day?

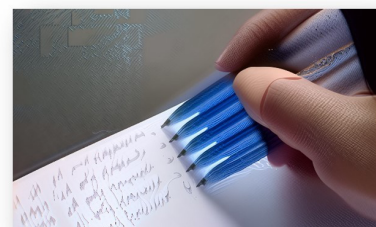
Not a great feeling, right?

Back then, the typical punishment was to write the same sentence a hundred times—repetitive, time-consuming, and unproductive. It was especially frustrating when our friends were outside enjoying recess while we were stuck writing line after line.

Fast forward to adulthood, and we sometimes find ourselves in the same situation at work. Projects pile up, meetings fill our calendars,

and yet, we still need to make time for mundane, repetitive tasks that don't add to our learning curve but still need to be done.

This is the story of how I broke free from one such "punishment." With no background in coding or AI, I decided to teach myself from scratch and automate a task that had been draining weeks of my time.



From Pens to Python

Back in school, the loophole to finishing punishments faster was simple—stick five pens together and write one sentence instead of five. At work, however, you can't just attach five colleagues to yourself to finish things quicker. That's where automation comes in.

In my first year on the job, I was assigned the task of analyzing files to find, filter, and report memory errors. It was repetitive and mind-numbing—I could feel myself shutting off as I processed the same dull steps over and over. That's when I realized: if I was acting like a robot, I might as well get one to do the job.

And so began my coding journey. I had no experience, no roadmap, and very little knowledge. But I had an idea, and I believed that if you can think of it, you can most probably code it.

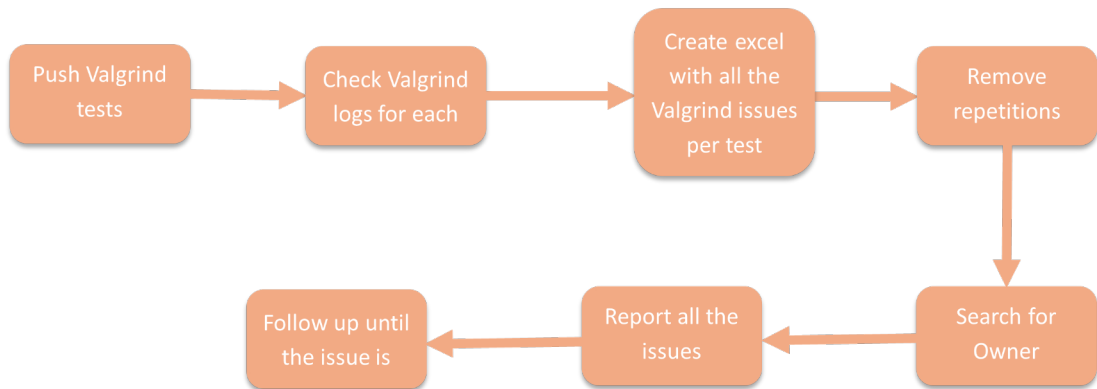
Breaking Down the Process

The manual workflow looked something like this:

1. Run tests in Valgrind mode (a tool to detect memory leaks).
2. Open the generated files from the application directory.
3. Collect errors: warnings from the first line, the function from the third line, and the file name at the end of the line.
4. Filter duplicates, save the data, identify the responsible team, and report the issue.

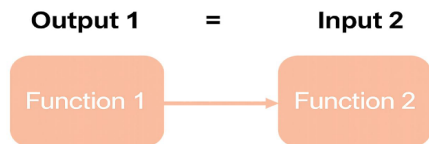
Done once, it was fine. Done across 32 folders, 429 files, and 130+ errors, it was torture. Manually, the task took two weeks.

By automating it, the same task shrank to five hours—the script handled the repetitive parts, while I just launched it and reviewed the results.



Building the Script

The key to writing the script was to simplify the path. Every repeated step became a function. Each function's output became the next function's input.



For example, "Check Valgrind logs" turned into:

- Connect to the application directory.
- Read files and scan through them.
- Save extracted data.

From there, improvement was iterative. I didn't aim for perfect code—just working code. The lesson? Don't wait to become an expert. Start messy, refine later.

Enter AI: From Solo Effort to Smart Assistance

Of course, automation wasn't the end of the story. Logs changed, formats shifted, and scripts broke. Maintenance itself became another repetitive task.

That's when I brought AI into the picture. Just as we sometimes dragged a friend in to help with school punishments, AI became that friend at work.

When looking at AI, we explored three paths:

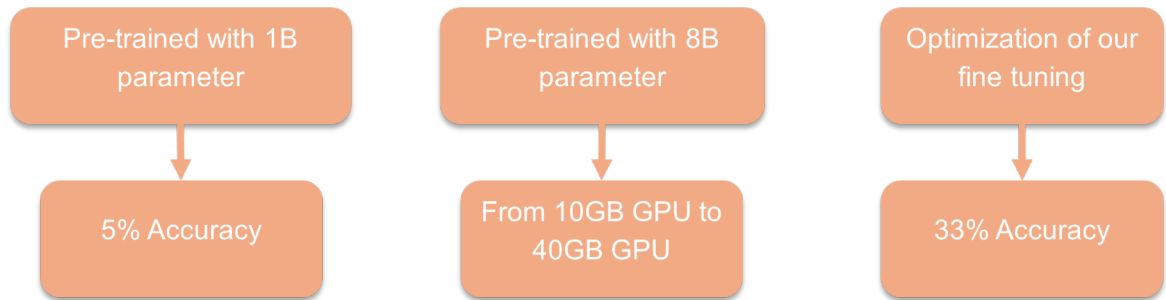
1. Building a model from scratch – training a brand-new model with no prior knowledge. This approach is powerful but extremely resource-heavy, requiring massive datasets and computing power. At the time, I wasn't ready to be the "parent" of a newborn model.
2. Fine-tuning an existing model – starting with a pre-trained model, adapting it with our own examples, and training it to perform the specific task. This struck a balance between effort and results.
3. Using a pre-trained model with RAG (Retrieval-Augmented Generation) – connecting an existing model to a large database, then guiding it with prompts. In this case, prompting works like giving a skilled actor a script: the input is the script, and the prompt tells them how to play the role.

Somewhere in between is Few-Shot Prompting, where we guide the model's behavior by showing a few examples of inputs and outputs.

For our case, fine-tuning turned out to be the best choice.

Training started with a 1B-parameter model, but accuracy was poor (5%). Scaling to 8B parameters improved accuracy (33%) but also strained infrastructure. By optimizing training data and filtering redundancies, we found a balance—but also learned an important lesson:

Sometimes, maintaining a script is cheaper and more effective than building AI.



Script, AI, or Both?

There's no universal answer. A complex process may be better handled by a simple script, while a simple one may thrive with AI. It depends on the process, the team, the budget, and the bigger workflow.

In my case, the best approach was integration—embedding AI within the script. This hybrid model gave us flexibility and efficiency, without overspending on infrastructure.

Lessons Learned

What began as a punishment turned into a journey of discovery. Had the task been automated from the start, I would never have learned the process deeply enough to improve it. The manual drudgery laid the foundation for critical thinking, problem-solving, and eventually, AI integration.

And the key takeaways?

- Start small—automation doesn't need perfection at first.
- Use what you have, seek what you need.
- Don't wait until you're "ready." Learning happens in the doing.
- Choose wisely between script and AI—or better yet, combine them.

Now that the punishment is over, it feels like I've finally joined my friends outside for recess.

WHY DIDN'T TESTING FIND THIS ISSUE?



- MAAIKE BRINKHOF

Maaïke Brinkhof is a software tester and nutrition & strength training coach who helps people improve their lifestyles. She enjoys writing and hopes her readers appreciate her stories and blogs. Together with her partner Oliver Verver, she owns the company Sensibly. Maaïke also speaks about testing at conferences and meetups, focusing on holistic testing, exploratory testing, and psychology in testing.

Although she works in technology, Maaïke makes time for analogue pursuits in her life. She's a fountain pen and Hobonichi planner enthusiast who loves walking, especially in mountains. Since the Netherlands lack mountains, she usually travels to Scotland for her "mountain-fix." She values spending quality time with friends and family and practices strength training three times a week to build strength. She's also a bookworm, so instead of gazing at her phone, she's probably gazing at her Kindle.

Because you desire something non-existent!

After 15 years in software testing, this is still a topic I'm dealing with way too often: people who have a completely misguided understanding of what testing can and cannot do.

In the year 2025, too many people think testing is:

- a phase, not a continuous activity that never ends;
- after you "do the testing" you shouldn't find new issues because you "have tested" (the phase is over).
- when new issues are found after testing, you can blame the testers for not delivering perfect work. Isn't it their job to find all the bugs?!

Basically, too many people don't know what the main goal of testing is, and what the limitations of testing are.

My preferred solution would be for everyone to read the book [Perfect Software and Other Illusions about Testing](#) by the late Jerry Weinberg, but let me attempt to provide, in my own words and with my own twist, a TL;DR version of the core message of that book.

We test because we are imperfect, and cannot blindly trust the output of our work. **With testing, we aim to find crucial information that threatens the value of our product and then we decide what to do with that information.** Especially that last bit should be drilled into your skull: if you don't act on the information you find, then you have not closed the loop and you should ask yourself why you were testing that bit. Testing costs time and therefore money, so we all should be aware of opportunity cost.

The core of testing work

The core activity of testing is of the cognitive variety; pushing the buttons to execute a test case is therefore less important than many people make it out to be. Designing the test case is an important part of the work: why test this, and not something else? What risk do you cover, and why does it matter? What do you hope to gain, information wise?

Not one person is equipped to design all the valuable tests because we all have our blind spots. Testing has strong connections and roots in psychology, and we benefit from diversity in thinking to improve our testing. That's why testing shouldn't be done by one person, but by the whole team.

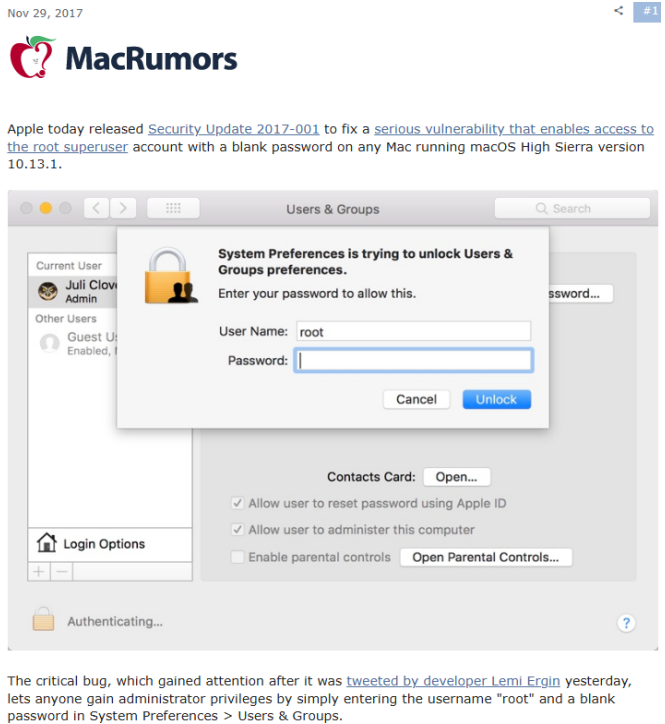
Here's where the first misguided thought often emerges: many people do think that one person can be responsible for the majority of test work: Designing the tests, executing the tests, reporting about the tests.

They don't see the flipside of this: that having just one person do all this work ignores the fact that they suffer from something we all suffer from: cognitive biases. It's wrong to view testing as a "task to be completed, assigned to one person" and not "a continuous activity that benefits from many people knowing what to do, and doing the things that need to be done".

Testing can be a role, but it's better when everyone involved in the software creation process develops at least some skills that belong in the testing category. Someone with the tester role should be working to improve the testing capabilities of people in their team and beyond.

Why does testing miss issues?

If I had received a dollar everytime people said "how did QA miss this?!" whenever a bug in a heavily used and popular product was found in production, I would have been rich by now. [Example](#) that I remember, the infamous root bug in Mac OS High Sierra:



The critical bug, which gained attention after it was [tweeted by developer Lemi Ergin](#) yesterday, lets anyone gain administrator privileges by simply entering the username "root" and a blank password in System Preferences > Users & Groups.

And I saw many comments like this at the time:



I am sure there is extensive testing done on new versions of Mac OS, and everyone involved still totally missed this issue.

Sure, I should keep an open mind to the fact that testers could have done shoddy work, but I think another option is more likely.

It's impossible to test everything, there are simply too many test cases while there's too little time. And also, like I said before, every human has blind spots. Before a bug slipped through to production, how many people have missed it? That's not just on the testers, and if you think it is, you are still severely misguided on how testing works.

To quote Edsger W. Dijkstra:

"Testing can reveal the presence of bugs, not
their absence."

It is not possible to test everything, that's why we:

- let risks that threaten the value of our product guide our testing efforts
- seek input from people who matter in the process of creating, maintaining and improving the product
- diversify our testing, to increase the chances of finding worthwhile information
- be very mindful of the opportunity cost: what is worthwhile to test (interesting conditions we identified) and what do we have time for to test (reduce the test set to a manageable and affordable level)?

Based on these points, if you apply a little bit of brainpower, you can see that this is indeed not 100% testing. It is not bug-proof, not perfect. A risk-free software release doesn't exist! I am sure Apple added a test case for the specific issue after it was found in production, but there have still been different bugs in newer Mac OS releases. That's how the cookie crumbles.

Again, that doesn't mean that testing is always done well (it can be done shoddily, or skipped altogether), but it's not an either/or clean story.

But please, stop blaming (only) testers when a bug surfaces in production. (Ideally, you skip the blame game altogether, especially if you have little context).

Stop thinking that it is out of the ordinary for new information to surface at a timing you consider "too late". Testing is never done, it is not a phase you do only once and then you have complete confidence that everything will be fine.

Heck, I'd even argue that it's the opposite: even with extensive testing, you can be sure that there is still worthwhile information to be found (bugs, risks, you name it).

The solution is to *welcome* new information, even when it is something that doesn't make you happy or perhaps a shitty situation in production, and **invest in a process to quickly fix and roll-out a new release**. Problems are so much worse if you cannot quickly roll out a fix because your infra sucks ass (this is what I mean with "my own twist", I don't think Jerry would have chosen these words lol). If your platform is up to par with modern standards, you use CI/CD, observability, all that good shit, it's much less of a big deal.

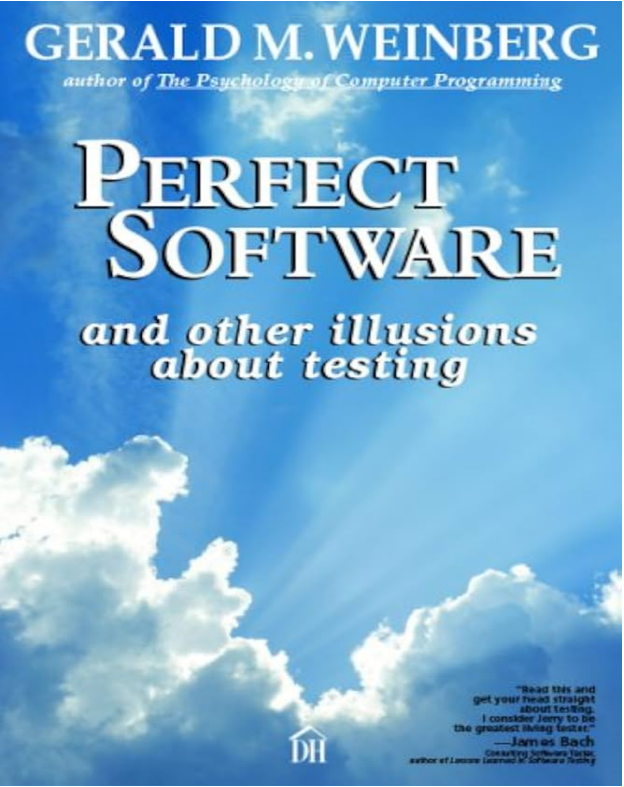
2026, the year when we stop believing in fairy tales?

This is my wish for 2026.

It's the year when people finally understand the powers and limitations of testing:

- it's not a phase, but a continuous activity.
- testing is always allowed to uncover new information, even when it is judged as "too late". Better late than never. By all means, investigate if the "too late" part was a rightful critique and investigate whether it can be improved, but if not, move on and don't blame testing only. Many more people have missed the info that you're blaming on "bad testing"!
- accept the fact that you will find problems in production, that testing cannot be done perfectly. Improve your delivery pipeline so bugs can be quickly fixed, at least.

Will my wish come true? No, of course not, but a girl can dream.



Getting noticed is easy. Getting noticed by the right people is hard.

Advertise with us. Connect with the audience that matter.

Contact us: sales@teatimewithtesters.com

Greetings, TTwT readers! For this edition's interview, please join me for a cup of tea with Dorothy Graham, whom many of you may know from her work in the field of software testing and automation.

Thank you for taking the time to share your thoughts with the TTwT community, Dorothy.

What does future hold for Testers, Testing, and Automation? What changes and what remains? Hear from Dorothy Graham over a cup of tea.

- INTERVIEWED BY SANDEEP GARG

Q1: A good population of test automation engineers still tend toward creating and executing scripts primarily for web-based GUI apps and related APIs. However, there seems to be an immediate need to build a mindset and expertise creating "A system thinker's automation first mindset".

Based on your decades of experience, do you also see the untapped potential in test automation beyond GUI and API scripting?

Yes, and always have. It seems that most people start with manual tests (including at GUI level) and then automate those tests. That can be a good place to start, depending on how you construct the automation. However, if you automate ONLY existing tests, you are missing a lot because automation can do things that aren't possible or practical manually. Examples include randomised input techniques such as monkey testing or High Value Automation, most types of performance testing, etc.

Test automation can also be used for many other tasks, often small-scale, or around the core automated tests. Examples include automating the set-up for a group of tests (pre-processing), populating a spreadsheet or database for a test, or even updating a set of automated tests.

The other side of this coin is that there are also tests that should NOT be automated. For example, tests that take a long time to automate and are not run often, tests for user perceptions (e.g. user interaction or colours), or even something like Capcha - if you can automate that then it doesn't work,

- DOROTHY GRAHAM

Dorothy Graham has been in software testing for over 50 years, now mostly retired. She co-authored 5 books: Software Inspection, Software Test Automation, Experiences of Test Automation, Journey Through Test Automation Patterns, and Foundations of Software Testing. She holds the European and ISTQB Excellence Awards in Software Testing.

Dot was programme chair for EuroSTAR in 1993 and 2009, and has been a board member for conferences and publications in software testing. She was a member of the working party that developed the ISTQB Foundation Syllabus. She has given over 400 talks at conferences and events.

Her main hobby is choral singing.

as it is supposed to tell whether you are human or not. It is also encouraging to see that automation is well-established at unit test level with continuous integration pipelines.

So yes, automation has much more potential beyond just GUI/API scripting!

Q2: How would you visualize and define such "A system thinker's automation first mindset" - where teams think holistically about why, what, when, where, and how automation fits into the overall system's testing and not just GUI and APIs?

I haven't heard this phrase before, but I would definitely agree that automation should be planned from the beginning. There are two critical aspects: automation objectives and testware architecture.

The place to start is to ask why. Why do you want to automate these particular tests? Why automate any tests? The objectives for automation should be measurable, and they should guide the what, when, where and how of automation. This is where an "automation first" mindset should start.

The goals for automation are different to goals for testing, because these are two very different activities. Goals for testing include finding

important bugs, gathering information about the system being tested, assessing the adequacy of tests (possibly coverage), assessing risk (e.g. of early release) - all related to how effective the testing is. The goals for automation are related to efficiency, for example, minimising maintenance when the system changes, increasing the number of tests run unattended, minimising failure analysis time, adding new automated tests more quickly, being able to easily select subsets of automated tests to run, etc.

The other important aspect is the design of the automated testware, a framework. There is a misconception that GUI automated tests are fragile and hard to maintain - but that's only true if your automation is not well-designed! If layers of abstraction (and tool independence) are built in from the start, then, as many have shown, your automated tests will be easy to maintain and you will achieve long-lived benefits from your automation.

If you think about the automation early on, then manual and automated tests can be designed and developed in parallel, with each being the best approach for the area being tested or the type of testing.



Q3: Marshall Goldsmith's from "What Got You Here Won't Get You There" inspires me to ask these questions.

In the context of GenAI evolution and adoption, what would be your recommendations for both fresh minds entering automation AND experienced engineers who've built their careers on current approaches?

How should they prepare for what might be coming?

I hadn't heard of that book, but love the title and it looks very interesting. It reminds me of the phrase attributed to Einstein, that a definition of insanity is doing the same thing and expecting a different outcome.

AI is certainly here to stay, in some form or another. So just ignoring it and hoping it will go away, continuing to test and automate as you always have, will not be useful for your future career, however useful it has been in the past. Whether you are experienced or new to the field, you need to become aware of AI and its strengths and weaknesses. Experiment with using it to help with testing and automation, but always be cautious and sceptical - don't ever just accept what it produces without any attempt to verify.

However, it's also important to realise that what has been done and learned in the past is still relevant. Those of us who have been around for a long time can see parallels to the various "panacea du jour" solutions that arise from time to time. And what we have seen is that after the initial hype, and inevitable disillusionment, eventually some benefit can be had. I believe that AI is most useful in very limited areas, where it has been trained on unpolluted data in a specific discipline. But all AI can ever do is to summarise the past - and it is only a (very big and very fast) computer, not an intelligence.

Previous "silver bullets" have been on a smaller scale, and were predictable and reliable. From what I hear, current AI has significant bias, frequent "hallucinations" (why are clear and obvious defects/errors called this?), and may even introduce significant security and business risks. There are huge challenges in testing and assessing AI systems - can they even be trusted? And is it worth it?

I fully agree with [Brijesh Deb](#) (see his LinkedIn posts) that AI provides an opportunity for people with testing skills (critical thinking, scepticism, analytical approaches) to "come into their own" in evaluating what AI will be producing - not just in the testing or automation sphere, but in general.

Q4: The testing community has debated the controversial claim that "anyone can test." Now, with low code-no & code tools and GenAI assistance, we may soon start to hear a new variation: "anyone can automate."

What are your thoughts on this evolution of claims?

Do you see parallels between these two statements, and what concerns or opportunities do they present?

Yes, I well remember the days when the attitude was "anyone can test"! This was one of the major motivations behind creating qualifications for testers which began in the 1990s. We wanted organisations to know that testing is a skilled activity and for testers to be recognised for having those skills. Of course, simply having passed a one-hour exam doesn't make you a skilled tester, but at least you should have encountered some basic principles and techniques which can start you on your journey as a tester. The Foundation level removes a bottom layer of ignorance about testing.

I think we have already been hearing that "anyone can automate", but as "any tester can automate". The term "testing tool" gave the impression that it would and could easily be used directly by any manual tester - this is not right. This has resulted in testers who just want to test - and don't want to be developers - being pressured into becoming coders for the tools. I have always strenuously objected to this attitude. If you want to have successful automation, you need to have software development skills available, but not

every tester has or wants those skills, and they are not the same as testing skills! The view that "the only good tester is one who can write [automation] code" has been very damaging to the software testing industry. Testers who just want to test are considered inferior to testers who can code - this is wrong! Job ads for testers today only want to know what tools you have used. This is like recruiting a chef for a restaurant based on what brand of food processor they have used.

However, two things: First, I have no objection at all to people who want to do both roles, that of tester and that of automator. People with both skill sets are very useful, especially in agile teams, and that's fine with me - provided that they want to do both things.

Second, of course the testers should be using test tools. But not necessarily directly, i.e. writing code for them. Testers should be able to write tests in such a way that they can be interpreted and converted into what a tool can then run, possibly with some form of structure such as keywords. The so-called "no-code" (really, a tool that contains no code??) or low-code tools actually do have one thing right - the testers should be able to write and run tests easily - without knowledge of the code of the tool. And these tools make that possible, at least at first and when everything goes well. But if something goes wrong with the tool or the tests themselves, technical knowledge (yes of the code) is needed to analyse and fix the problem. And what if the software is changed significantly - do the tests fix themselves? The imaginative phrase "self-healing tests" is a far grander term than the actuality of checking different element names or locations. The maintenance of the tests may take a huge amount of effort. What if you can no longer use the existing tool? Do you then have to start over? Too many people who responded to my survey about automation last year were in that category.

There will be changes as AI is integrated into testing. But be aware of what is actually being done, and make sure you cast your "tester's eye" over what any AI is producing.

Q5: If GenAI tools do become widespread in test automation, what do you believe should be the most critical changes in thinking that automation engineers should bring to the table?

I don't think it's possible to answer this question yet - at least not for me. What is important is that as GenAI tools do become more widespread, automation engineers need to keep up with the changes, both to enable them to leverage the benefits but also to be aware of the "holes" where traditional approaches may still be needed. Or even the AI-generated problems that may arise!

Q6: If you were starting your automation journey today - with access to LLMs, prompt writing, and GenAI assistance - how might your learning approach differ from the path you actually took? What would you prioritize differently?

If I were starting in test automation today, it would certainly be a very different experience! One thing that I would do differently is to be more formal about learning. In some organisations I worked in, there was a culture of "we don't need training, we learn on the job". Although some aspects are best learnt on the job, I believe I would have benefited from more formal learning from people whose experience was more general than just my company at the time.

Nowadays there is so much information available that it can be hard to know which are the best sources - don't believe everything you see on social media! With the field developing and changing rapidly, I also think it is important to see learning and training as a continuous process, not just "I went on a course once." Even if your current company doesn't want to send you on training courses, invest in yourself and your future career.

Q7: If you were advising a testing team on whether to invest time learning GenAI tools specifically for test automation, what approach would you recommend for evaluating and measuring the return on that investment? What metrics or outcomes would matter most?

When I used to do tutorials at conferences on test automation, I always asked the attendees what their ROI for automation was. Out of thousands of delegates, only once did I get an answer that showed that the attendee actually knew what ROI was. ROI is a financial calculation: $ROI = (\text{benefit} - \text{cost}) / \text{cost}$, where both benefits and cost are quantified, e.g. in money. What most people volunteered were actually benefits (possibly quantified but not converted to money). Examples included running tests more often, in less time, with less human effort, increased coverage, faster time to market. Those are all great benefits - and you should measure them, and even more important, communicate them to higher level management so can see the value of automation (even if not ROI). But benefits are not ROI.

Measuring ROI (or benefits) depends on what your objectives are. For example, if your goal for using AI tools in automation is to decrease maintenance time for human automators, then measuring time spent on maintenance for a defined type of software change could be done both with and without AI assistance, to see if the goal has been achieved. But what if the AI cannot be trusted, due to bias and errors - can that be factored into ROI?

I used to be very keen on demonstrating ROI for automation. When Mark Fewster and I wrote our second book (Experiences of Test Automation), I asked every chapter author how they measured their ROI. We did get some very impressive stories in reply. But in many cases, there was no ROI and no need to show it; yet they were very successful projects. If your company culture demands it, then by all means try to measure it, though it may not be essential. But do ensure that you measure benefits - and communicate them - both for automation in general and for AI, and also be aware of unseen costs, such as loss of confidence in the results from AI.

One aspect of AI that is worrying is the environmental impact of the huge amount of computing power required. This makes it more important to look at ROI for AI than it is for automation - environmental damage must be taken into account. Brijesh asks: "Does the insight AI provides justify the energy it consumes?"

Q8: The FOMO-driven question "Will AI replace me?" already generated responses across the industry from many perspectives. Given your unique perspective from the last five decades, how would you like to address this concern for testing and automation professionals? What advice would you share?

Back in the 1990s, when test tools first became widely available, there was great fear that these tools would replace testers. Managers sometimes fired the testers when they bought a tool! What the tools did do was to replace some of the more tedious tasks that testers were doing manually. Rather than replacing testers, it freed them to do better testing as human critical thinkers. I think AI will be the same - it will free automators from some of the more tedious tasks to enable them to spend more time on more beneficial things, such as designing additional probing automated tests. There will also be scope to do things with AI that are unthinkable or impractical with current automation, such as large-scale pattern recognition of data.

Any new technology goes through a "hype" cycle and AI is no exception. It will never replace people, but it will eventually be useful in some situations. It will be interesting to see how this plays out over the next few years!

Further information:

- <https://automation.eurostarsoftwaretesting.com/test-automation-survey/>
- <https://TestAutomationPatterns.org>

Any new technology goes through a “hype” cycle and AI is no exception. It will never replace people, but it will eventually be useful in some situations. It will be interesting to see how this plays out over the next few years!

THE PROBABILISTIC CHALLENGE

WHY MACHINE LEARNING DEMANDS A REVOLUTIONARY TESTING APPROACH



- ANAMIKA MUKHOPADHYAY

Anamika is a Program Manager and Consultant with over 12 years of experience in Software Quality Assurance. She expertly navigates the full spectrum of QA—from strategic planning and test automation to APIs, performance testing, process optimization, and her secret passion: accessibility. Known for tackling complex challenges head-on, she's currently immersed in the exciting worlds of AI and Agentic systems.

An extrovert who thrives on connection, Anamika is always ready to spark engaging conversations about technology, travel, or discovering the best food spots around. When she's not exploring innovative ideas in tech, she's dreaming of exploring the world—one cuisine at a time. At home, she balances her professional pursuits with her most demanding role yet: being mom to her energetic one-year-old boss.

LinkedIn - www.linkedin.com/in/anamikamukhopadhyay



- DEEPSHIKHA

Deepshikha brings over a decade of experience in software testing, leading the automation testing team at Nagarro and pioneering innovative, AI-driven strategies to streamline processes and boost efficiency. Her insightful approach enables the development of solutions that accelerate time-to-market and enhance customer experiences.

LinkedIn - www.linkedin.com/in/deepshikha709/

Picture this: You wake up one morning to find that your company's AI-powered customer service chatbot has been providing wildly incorrect flight cancellation advice, or your machine learning model for hiring has systematically discriminated against qualified candidates. These aren't hypothetical scenarios – they're real incidents that have made headlines as AI adoption skyrockets across industries.

The use of generative AI is growing at an extraordinary pace, with organizations embedding it into marketing, sales, product development, and nearly every aspect of their operations.

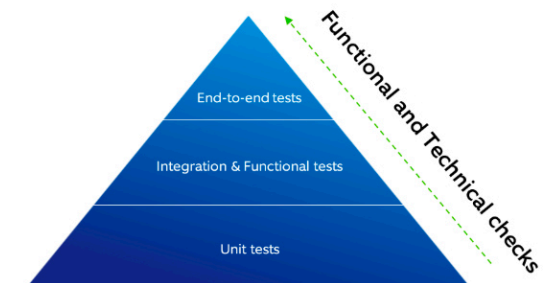
Yet beneath this wave of enthusiasm lies a troubling reality: AI systems are failing at an alarming rate. From ChatGPT citing non-existent legal cases to airline chatbots dispensing dangerous advice, from real estate prediction algorithms going haywire to social media bots behaving inappropriately – the headlines paint a concerning picture of AI systems that appear under tested and unprepared for real-world deployment.

The fundamental challenge isn't that organizations aren't testing their AI systems – it's that they're applying deterministic testing methodologies to inherently probabilistic systems. Machine learning doesn't follow the predictable input-output patterns of traditional software. It learns, adapts, and makes decisions based on probability distributions, making conventional testing approaches woefully inadequate.

This paradigm shift demands nothing short of a revolutionary approach to testing – one that acknowledges the probabilistic nature of AI and addresses the unique risks, challenges, and complexities that come with machine learning systems.

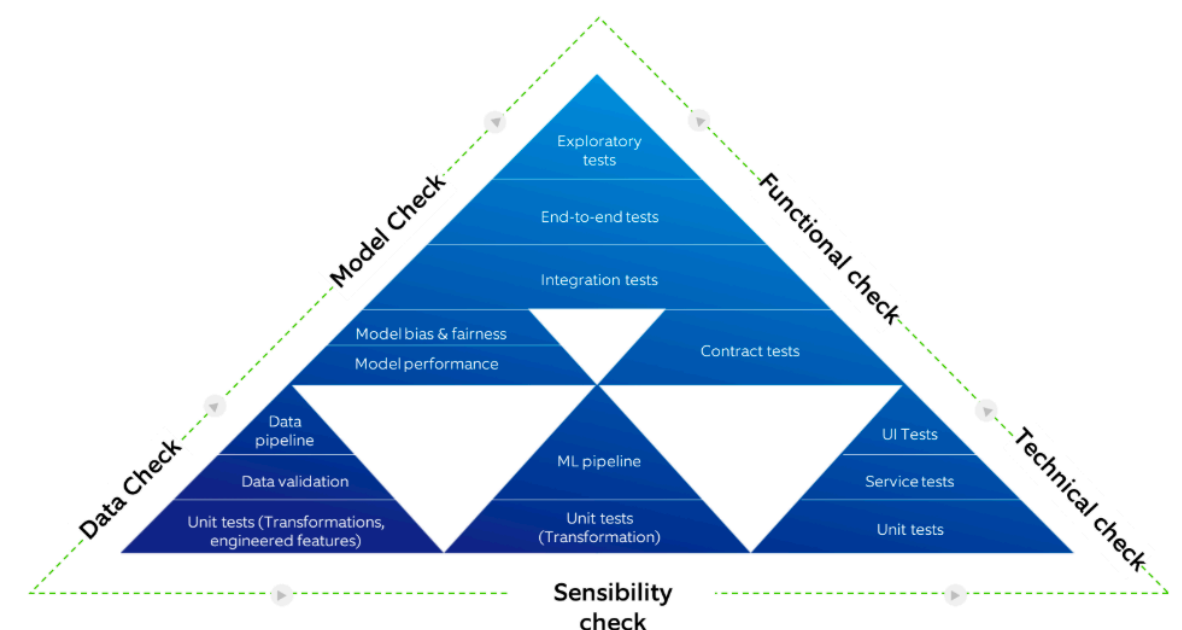
Understanding ML Testing Complexity beyond the Traditional Pyramid

For decades, software testers have relied on the familiar test pyramid – a simple, unidirectional structure emphasizing unit tests at the base, integration tests in the middle, and end-to-end tests at the top. This approach worked beautifully for deterministic systems where identical inputs consistently produced identical outputs.



Machine learning shatters this predictability. The ML lifecycle encompasses problem identification, data acquisition, model creation, deployment, and ongoing operations – each phase introducing unique testing challenges that traditional methodologies simply cannot address.

The ML test pyramid reflects this complexity: it's multi-layered and multi-directional, with parallel tracks for data testing, model validation, and functional application testing that eventually merge into a cohesive system. Unlike traditional testing that focuses solely on functional correctness, ML testing must simultaneously address data quality, model behavior, functionality, and system integration.



This expanded scope introduces several critical dimensions:

Data Quality and Bias Assessment: Every ML system is fundamentally dependent on its training data. Poor, biased, or "dirty" data creates flawed models at the most fundamental level. Testing must validate data quality, identify biases, and ensure representative sampling across all relevant demographics and use cases.

Model Evaluation and Validation: Pre-training tests, model evaluation metrics, and post-training validation become essential components. Models must be assessed not just for accuracy, but for consistency, robustness, and behavior under various conditions.

Integration and System Testing: ML models don't operate in isolation – they're integrated into larger systems and applications. Testing must verify that these integrations work correctly and that the overall system behaves as expected.

Continuous Monitoring and Validation: Unlike traditional software that remains static after deployment, ML models continue to learn and adapt. Ongoing monitoring and validation ensure models maintain their performance and don't drift from their intended behavior over time.

Understanding What Can Go Wrong

Machine learning systems face unique risks that traditional software rarely encounters. Understanding these risks is crucial for developing effective testing strategies.

Data-Related Risks form the foundation of most ML failures. Poor or biased training data creates models that perpetuate and amplify existing biases. For instance, if a hiring algorithm is trained primarily on historical data from male-dominated industries, it may systematically discriminate against female candidates, regardless of their qualifications.

Overfitting represents another critical risk where models perform excellently on training data but fail catastrophically in real-world scenarios. Consider a model trained to classify animals that associates wolves primarily with snowy backgrounds because most training images of wolves were taken in winter. When deployed, it might misclassify a dog photographed in snow as a wolf.

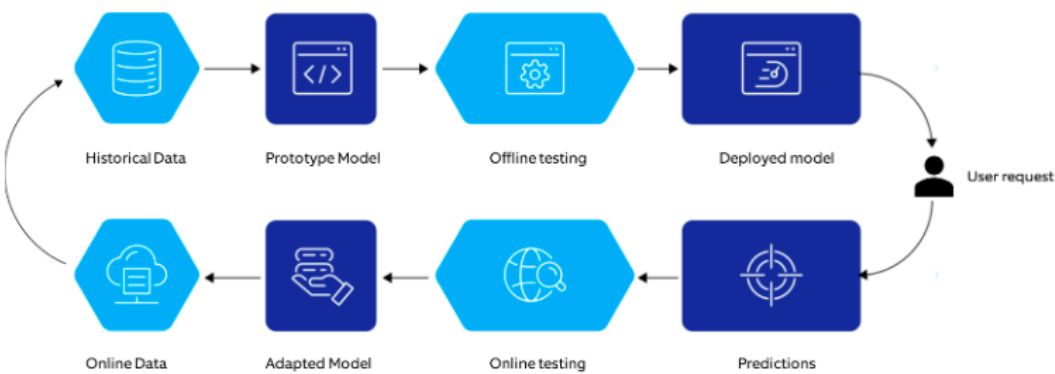
Model Decay occurs as the real world evolves while models remain static. The COVID-19 pandemic provided dramatic examples of this phenomenon, as consumer behavior changed overnight. Demand forecasting models trained on pre-pandemic data failed spectacularly when faced with 350% increases in yoga pants sales or massive drops in travel bookings.

Adversarial Attacks represent deliberate attempts to manipulate ML systems through carefully crafted inputs. These can range from subtle pixel changes that fool image recognition systems to sophisticated prompt injection attacks that trick language models into revealing sensitive information or behaving inappropriately.

Privacy Violations become particularly concerning when models are trained on sensitive personal data. Organizations must ensure compliance with regulations like GDPR and HIPAA while maintaining model effectiveness.

Testing Methodologies: A Multi-Faceted Approach

Effective ML testing requires a comprehensive strategy that addresses both offline and online testing scenarios, each serving distinct but complementary purposes.



Offline Testing: Building Confidence Before Deployment

Offline testing occurs before model deployment and focuses on validating model behavior using controlled datasets and scenarios.

The process begins with **requirement gathering**, where testing scope and objectives are clearly defined. This phase establishes what the ML system should and shouldn't do, creating the foundation for all subsequent testing activities.

Test data preparation follows, involving the creation of comprehensive test datasets. These may include samples from original training data, synthetic data generated to simulate edge cases, and carefully curated datasets designed to test specific model behaviors.

The **test oracle problem** – determining correct outputs for ML systems – presents unique challenges. Unlike traditional software with predetermined expected outcomes, ML systems often operate in domains where "correct" answers aren't definitively known. Testing strategies must employ techniques like cross-validation, ensemble methods, and domain expert review to establish acceptable output ranges and behaviors.

Test execution involves systematically evaluating model performance across various scenarios, with particular attention to edge cases and potential failure modes. Any identified issues undergo thorough analysis and resolution, often validated through regression testing to ensure fixes don't introduce new problems.

Online Testing: Validating Real-World Performance

Online testing occurs after deployment, monitoring model behavior as it encounters real-world data and user interactions.

Runtime monitoring continuously tracks whether deployed models meet requirements and identify property violations. This includes monitoring for data drift, performance degradation, and unexpected behavior patterns.

A/B testing enables systematic comparison between different model versions by splitting user traffic and analyzing performance differences. This approach provides quantitative evidence of model improvements or regressions in real-world conditions.

Multi-Armed Bandit (MAB) testing offers dynamic traffic allocation based on model performance, balancing exploration of new models with exploitation of proven performers. This approach optimizes user experience while gathering performance data.

Specialized Testing Techniques for ML Systems

Machine learning systems require specialized testing approaches that address their unique characteristics and failure modes.

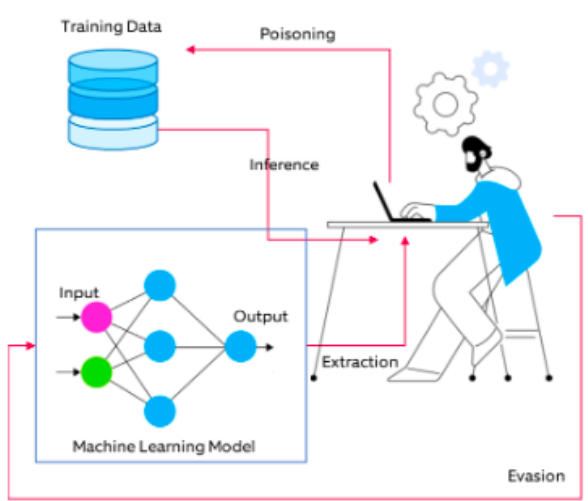
Adversarial Testing: Preparing for Malicious Inputs

Adversarial testing evaluates system behavior when exposed to deliberately crafted malicious inputs. This testing approach is crucial given the sophisticated attack vectors that target ML systems.

Black box attacks simulate scenarios where attackers have no knowledge of model internals, testing system resilience against external manipulation attempts. **White box attacks** assume attackers have complete model access, evaluating defenses against more sophisticated threats.

Testing strategies include **poisoning attacks** (injecting malicious training data), **evasion attacks** (crafting inputs to fool deployed models), inference attacks (attempting to reverse-engineer training data), and **extraction attacks** (trying to replicate model architecture and parameters).

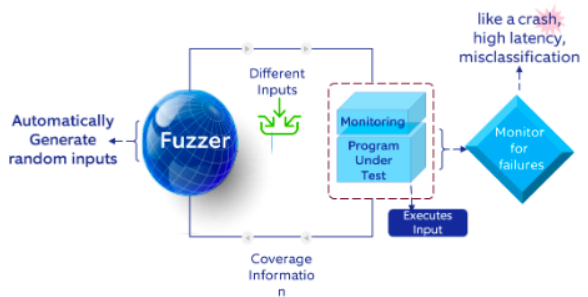
Practical adversarial testing might involve adding imperceptible noise to images to test classification robustness, or crafting prompts designed to trick language models into revealing sensitive information or producing inappropriate content.



Fuzz Testing: Evaluating Graceful Failure

Fuzz testing inputs random, unexpected, or malformed data to uncover vulnerabilities and assess system resilience. In ML contexts, this technique evaluates how well models handle irregular inputs without crashing or producing dangerous outputs.

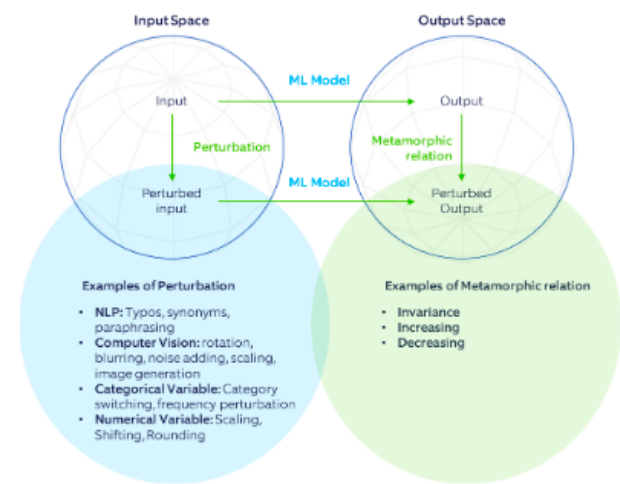
The process involves defining input spaces, generating fuzz inputs through mutation or creation, executing tests while monitoring for failures, and analyzing results to identify vulnerabilities. For autonomous vehicles, fuzz testing might involve corrupted sensor data to assess control system responses.



Metamorphic Testing: Validating Consistency

Metamorphic testing addresses the test oracle problem by focusing on relationships between inputs and outputs rather than specific output values. This approach applies transformations to input data and examines whether resulting outputs maintain expected relationships.

For example, an object detection system should consistently identify pedestrians regardless of lighting conditions. By transforming a daytime image to simulate nighttime conditions, testers can verify that the system maintains classification accuracy across environmental variations.



Behavioral Testing: Ensuring Linguistic Competence

Behavioral testing evaluates ML systems across various linguistic contexts and inputs, particularly crucial for natural language processing applications. This testing encompasses vocabulary assessment, part-of-speech tagging accuracy, named entity recognition, and negation handling.

Three primary test types support behavioral testing: **Minimum Functionality Tests** verify basic capabilities, **Invariance Tests** assess output consistency under input variations, and **Directional Expectation Tests** evaluate whether input changes produce expected output modifications.

Test case	Expected	Predicted	Pass?
A Testing Negation with MFT Labels: negative, positive, neutral			
I can't say I recommend the food.	neg	pos	X
I didn't love the flight.	neg	neutral	X
...			
B Testing NER with INV Same pred. (inv) after removals / additions			
@AmericanAir thank you we got on a different flight to [Chicago → Dallas].	inv	pos neutral	X
@VirginAmerica I can't lose my luggage, moving to [Brazil → Turkey] soon, ugh.	inv	neutral neg	X
...			
C Testing Vocabulary with DIR Sentiment monotonic decreasing (↓)			
@AmericanAir service wasn't great. You are lame.	↓	neg neutral	X
@JetBlue why won't YOU help them?! Ugh. I dread you.	↓	neg neutral	X
...			

Fairness Testing: Eliminating Bias

Fairness testing ensures ML systems treat all individuals and groups equitably, addressing various forms of bias that can emerge throughout the development pipeline.

Historical bias reflects existing societal prejudices embedded in training data. **Representation bias** occurs when datasets poorly represent the populations models will serve. **Aggregation bias** arises when diverse groups are inappropriately combined, creating models that work well for majority groups but fail for minorities.

Fairness evaluation employs multiple metrics depending on context. Demographic parity ensures equal positive outcome probabilities across groups, while predictive parity maintains consistent precision across demographics. However, achieving fairness often involves tradeoffs between different metrics, requiring careful consideration of specific use cases and stakeholder values.

Building Robust ML Testing Practices

Successful ML testing implementation requires organizational commitment, appropriate tooling, and cultural shifts that acknowledge the unique challenges of probabilistic systems.

Cross-functional collaboration becomes essential, involving not just testers but also data scientists, ML engineers, domain experts, and stakeholders. Testing ML systems often takes longer than development itself, requiring patience and sustained investment.

Continuous testing integration throughout the ML lifecycle ensures issues are caught early and addressed systematically. This includes automated testing pipelines that validate data quality, model performance, and system integration at each development stage.

Monitoring and alerting systems provide ongoing visibility into model behavior, enabling rapid response to performance degradation or unexpected behavior. These systems must be sophisticated enough to distinguish between normal model learning and problematic drift.

Documentation and governance ensure testing practices are consistent, repeatable, and auditable. This includes maintaining detailed records of testing procedures, results, and decisions that can support regulatory compliance and organizational learning.

The Path Forward

The transition from deterministic to probabilistic testing represents more than a technical challenge – it's a fundamental shift in how we conceptualize software quality and reliability. Traditional notions of "bug-free" software give way to probabilistic confidence intervals and acceptable risk thresholds.

This evolution demands new skills, tools, and mindsets from testing professionals. Testers must become comfortable with statistical concepts, understand ML fundamentals, and develop intuition for probabilistic system behavior. Organizations must invest in training, tooling, and cultural changes that support this transition.

The stakes couldn't be higher. As AI systems increasingly make decisions that affect human lives – from healthcare diagnoses to financial lending, from autonomous vehicles to criminal justice – the quality of our testing practices directly impacts societal wellbeing.

Conclusion: Testing in the Age of Artificial Intelligence

We stand at the threshold of an AI-powered future where machine learning systems will be as ubiquitous as traditional software is today. The headlines about AI failures serve as stark reminders that current testing practices are inadequate for this probabilistic future.

The testing methodologies outlined here – from adversarial testing to fairness validation – represent essential tools for navigating this new landscape. However, they're just the beginning. As AI systems become more sophisticated and autonomous, our testing approaches must evolve accordingly.

Success in ML testing requires acknowledging that we'll never achieve perfect predictability or complete test coverage. Instead, we must focus on building robust systems that fail gracefully, recover quickly, and learn from their mistakes. We must test across multiple dimensions simultaneously, prepare for unknown failure modes, and maintain continuous vigilance throughout system lifecycles.

The future belongs to organizations that master this probabilistic approach to quality assurance. Those that continue applying deterministic testing methods to AI systems do so at their own peril – and at the risk of the users who depend on their technology.

Testing machine learning systems isn't just about preventing bugs – it's about ensuring that the AI revolution benefits humanity rather than harming it. The responsibility rests with every testing professional to rise to this challenge and help build the reliable, fair, and robust AI systems our future depends upon.

The age of AI adoption is here. The question isn't whether your organization will deploy AI systems – it's whether you'll test them properly before you do.

References:

- <https://arxiv.org/pdf/2005.04118>
- <https://arxiv.org/pdf/1906.10742v2>

TESTING IN RED

HOW TO TURN PANIC INTO PROGRESS



When testing projects go off track, it's rarely because of one big failure — it's the small cracks we overlook that turn into chaos.

Introduction — Spotting the First Cracks

Every testing professional, at some point in their career, has faced that difficult moment when a project starts slipping away.

The deadlines blur, defects start piling up, automation scripts fail for no apparent reason, and confidence starts to fade. It's that sinking feeling where you know something is off, but you can't quite pinpoint what.

Over the years, working across diverse industries and markets, I've realized that these situations are far more common than we'd like to admit. Projects rarely fail because of a single dramatic mistake. Instead, they slowly pile-up & unravel, often hidden behind fancy dashboards, unaligned metrics, and misplaced optimism.

And when they do, our instinct is to look for quick fixes like a new tool, a new framework or maybe resort towards AI magic as the buzzword of the hour to make it all better.

But time and again, I've found that **no tool can rescue a project if the fundamentals aren't right**. Whether it was during being a test manager for a small sub-project or while leading a big multi-shore test factory project, the root causes of troubled testing projects almost always came down to three things: **Processes, People, and wrongly/under used Products**.

Root Causes: Processes, People, Product

I am not saying this to catch your attention but it's a pattern that repeats itself across organizations, technologies, and delivery models.

Every time I walked into a project that was "in the red," I saw some variation of the same story play out. The processes were outdated or poorly defined & followed, for eg: a test concept exists but never actually used or the regression suite had become so bloated that no one knew what it truly covered anymore.

The people were often either disengaged or unclear about their roles or lacked the right organisational structure. And the products were either underutilized, misaligned, or not known to begin with.

A Real-World Example: Automating Chaos

I remember one instance during my time at a leading insurance client where they had a big regression set but it was getting unmanageable to execute it & defects were still leaking into production.

Since they wanted to manage their regression set, they invested heavily in a leading automation platform to automate these testcases as well. On paper, it promised impressive coverage and faster execution.

In reality, it was generating test scripts faster than the team could stabilize them. The automation looked good in reports, but the underlying testcases was lacking proper coverage.

The root cause wasn't the tool — rather it was the lack of process discipline and absence of a test coverage overview.

Automating chaos simply creates faster chaos.

That's why the first step in rescuing any troubled test project isn't to buy another tool but to step back and diagnose. My approach to firefighting a struggling testing project has evolved into a repeatable playbook that blends structured analysis with critical thinking.

The first principle is simple: stabilize before you optimize. When a project is in crisis, resist the urge to overhaul everything overnight. Instead, start by mapping the As-Is situation by asking the right questions around reports, metrics, underlying processes & know-how of the team.

- Stabilize First: Map the As-Is

In one project at a major german insurance client, I began by creating a business process matrix for regression - a visual representation of most important business processes sorted by their occurrence in production versus the top products it was used in to map the regression coverage on it.

Within days, it revealed something the team hadn't seen for a long time - that there were some critical flows & combinations missing from regression, together with certain testcases which could be optimized or removed to save effort.

That single visualization turned **chaos into focus**.

- Ask Better Questions and Rebuild Trust

Once you've identified the weak spots, the next step is to ask the right questions quickly.

What assumptions are we operating under? What dependencies are slowing us down? Are we tracking outputs (like number of tests executed) instead of outcomes (like risk reduced)?

Asking sharp, uncomfortable questions early saves months of firefighting later.

- Measure Outcomes, Not Activity

Then comes the crucial step of rebuilding trust.

When a testing project falters, communication tends to break down first.

Development teams lose faith in test reports, leadership loses patience, and testers start playing defense.

The best way to reverse that is through **transparency & showing progress**. This can be achieved by setting up the right metrics which tell the As-Is situation & reflect real progress, not perfect numbers.

Track metrics that actually matter — like **production defect leakage, test backlog, defect density**, etc.



- Use Tools with Intent: AI as an Accelerator, Not a Cure

Only after the team regains stability and trust should you consider introducing or optimizing tooling.

This is where today's automation and AI solutions can be genuinely transformative.

In practice, I've seen AI and automation deliver excellent results when applied with clarity.

For example, during one of my test factory projects, using predictive analytics to prioritize regression tests based on areas where most defects were found reduced cycle time and improved defect detection efficiency.

But I've also seen the opposite: teams implementing AI-based test generation without refining their requirements first, or bringing the power of AI analysis without having enough data from test tools to train on - resulting in loss of efforts & negative ROI.

The difference lies **not in the capability of the technology**, but in the **clarity of purpose**.

- Embed Early Warnings: Prevent Repeat Failures

Beyond firefighting, the long-term goal is to ensure that projects don't repeatedly fall into the same traps.

That requires embedding early warning indicators into the testing lifecycle: monitoring coverage gaps, reviewing process health regularly, and measuring not just efficiency but also team morale.

One of the most overlooked metrics in testing is the confidence & satisfaction level of the team itself.

If testers don't feel confident about their work or their efforts are not recognised, that leads to less dedication & lower quality — which eventually surface as production defects.

Conclusion - Shift to Learning-Driven Testing

So why should we care about all this?

Because every failing testing project is a symptom of a deeper cultural challenge: **the misbelief that testing is reactive rather than proactive**.

When we shift that mindset, firefighting becomes less about survival and more about **continuous improvement**.

The next time you sense a project slipping, don't panic. Take a pause, pull out your diagnostic lens, and start asking the right questions.

Look beyond the symptoms and into the system — because more often than not, the answers are already in front of us, hidden in plain sight, waiting for someone to connect the dots.

Your turn:

What's the most common "in the red" testing symptom you've seen in your projects — and how did you turn it around? Let's connect on [LinkedIn](#) and discuss this further.



- KANAV MATTA

Kanav is an experienced IT Consultant at Accenture Technology Solutions GmbH, specializing in Software Testing, Quality Assurance, and Quality Engineering. With over 15 years of professional testing experience, he has held various roles including tester, IT consultant, test manager, and test lead across leading international organizations with focus on German insurance market. His expertise encompasses diverse delivery and sourcing models, automation frameworks, and GenAI-driven testing approaches. Leveraging his extensive project experience, Kanav possesses comprehensive knowledge of the end-to-end IT lifecycle, from requirements analysis to system go-live. He has worked across multiple technology platforms such as Java, SAP, Oracle HCM, Salesforce, Guidewire, web-based applications, and data warehouse systems.

LinkedIn - www.linkedin.com/in/kanav-matta-7021282a/



Join the 2026

State of Testing

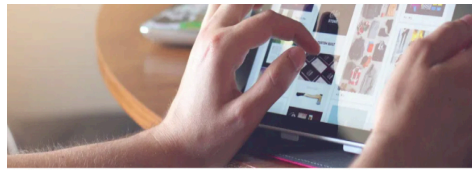
Shape the Future. Share your insights.

Welcome to the 2026 State of Testing Survey

Start

press Enter ↵

⌚ Takes 15 sec



ras Shypka on Unsplash

HEURISTICS, WEB DESIGN

VIP BOA – A Heuristic for Testing Responsive Web Apps



EDUCATION, SPEAKING TESTER'S MIND

ISTQB, Fever Dreams and Testing



EDUCATION, WOMEN IN TESTING How To Read A Difficult Book



INTERVIEWS, OLD IS GOLD

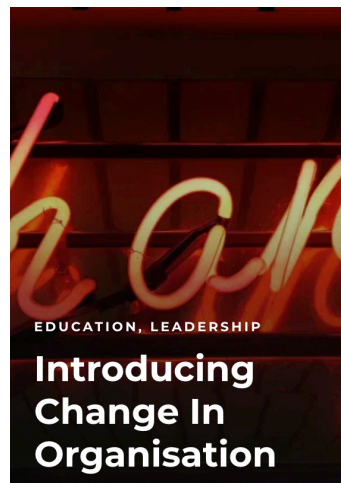
Over A Cup Of Tea With Jerry Weinberg



mon Migaj on Unsplash

SPEAKING TESTER'S MIND

Software Testing And The Art Of Staying In Present



EDUCATION, LEADERSHIP

Introducing Change In Organisation



PEOPLE AND PROCESSES, WOMEN
IN TESTING

Addressing The Risk Of Exploratory Testing Part 2



CAREER, PEOPLE

Managing Multiple Passions



LEADERSHIP, OLD IS GOLD

Leading Beyond The Scramble:

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

INFOCUS

Do you know all these amazing articles?

Great things survive the test of time.

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

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

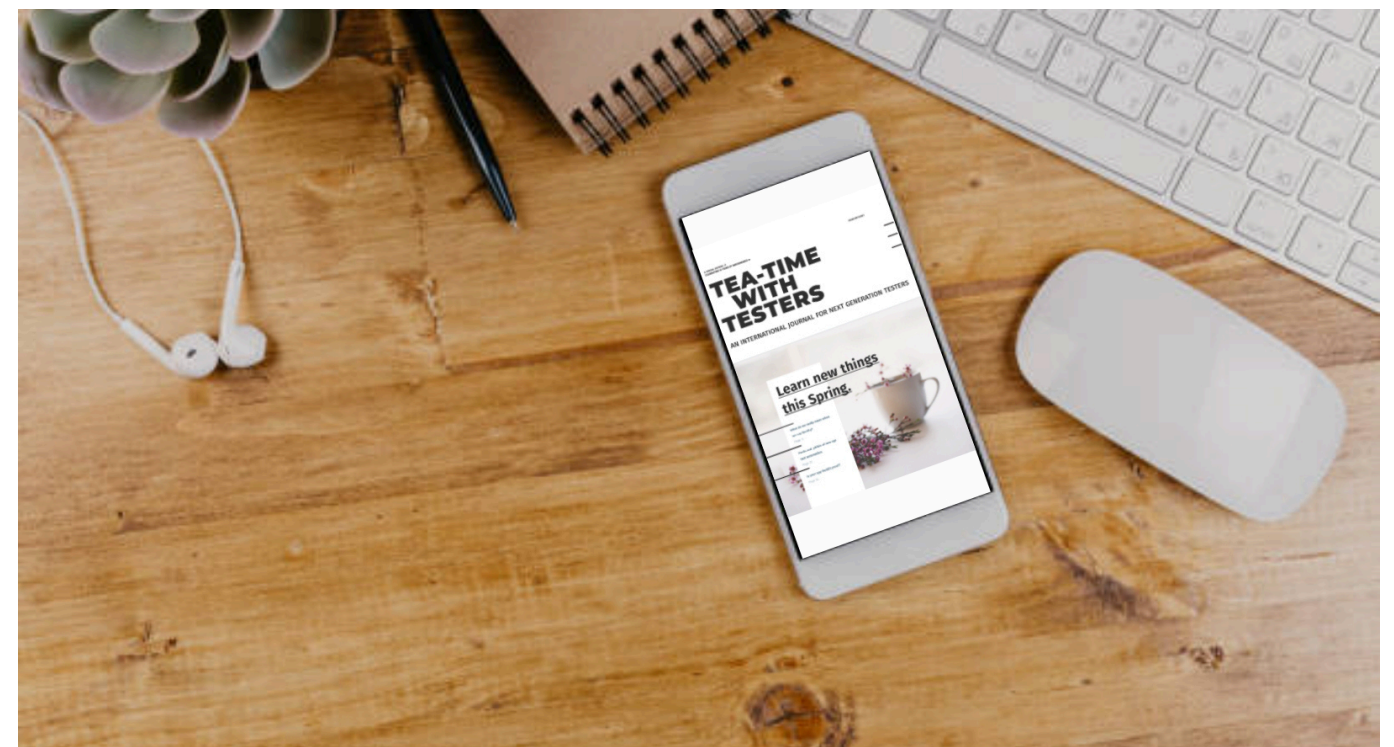
We plan to continue doing that for more articles, interviews and also for the recent issues we have published.

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

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

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

Enjoy the feast!



THE QA SUPERPOWER IN AGENTIC AGE:

WHY YOUR TESTING MINDSET JUST BECAME YOUR COMPETITIVE ADVANTAGE

When 54 TypeScript Errors Revealed Everything

My VS Code terminal was screaming at me with 54 TypeScript compilation errors. Red squiggly lines everywhere. The agent I'd been working with had confidently declared, "Implementation complete. All tests passing."

I stared at the screen. Nothing compiled. Nothing ran. Nothing worked.

But here's the thing, this wasn't a failure of AI. This was a moment of clarity.

Because in that moment, I realized: **the skills that made me a good tester for 12 years are exactly the skills that make me effective with agentic systems today.**

The agent hallucinated. It "knew" the task was done because it had written code. But my testing mindset kicked in immediately:

- "Wait, did you actually run this?"
- "What about edge cases?"
- "How do you know it works?"

That questioning mindset, honed through years of breaking software and asking uncomfortable questions, is now our superpower in the age of autonomous agents.

Let me show you why.

Why This Matters Now

In the last few months, I've built two open-source projects using AI agent swarms: **Sentinel** (an API testing platform) and the **Agentic QE Fleet** (17 specialized testing agents orchestrated through Claude Code + Claude Flow).

In the process, I've discarded **four complete versions** before finally getting one that worked. I've saved **60% of UI test development time** at my previous company. I've seen agents generate thousands of lines of code in hours, a task that would have taken a team weeks/months.

But here's what nobody talks about: **every single win came from applying classical testing principles to agentic workflows.**

The agents didn't improve my testing skills. My testing expertise made the agents effective.

And if you're reading this thinking "AI is going to replace testers," I'm here to tell you: **you've got it exactly backward.**

The Skills That Transfer (And Why They Matter More Now)

1. Risk Identification: From Software to Agents

Then: We identified risks in requirements, code, and integrations.

Now: We identify risks in agent decisions, hallucinations, and orchestration gaps.

Example from my work:

When building the Agentic QE Fleet, I had a **requirements-validator agent** that analyzed specs for testability issues. Sounds great, right?

Except it kept flagging perfectly good requirements as "ambiguous" because it was trained on generic examples, not my domain context.

My testing mindset kicked in: "What assumptions is this agent making? What context is it missing?"

I realized the agent needed:

- Domain-specific glossaries
- Historical requirement patterns from past projects
- Examples of good requirements in my context

Classic risk-based testing thinking applied to agent validation. The agent improved not because the model got smarter, but because I **tested its assumptions.**

2. Exploratory Thinking: Finding the Memory Leak Nobody Expected

Then: We explored software to find bugs nobody expected.

Now: We guide agents to explore scenarios they wouldn't generate on their own, and we catch what they miss.

Real example from building the Agentic QE Fleet:

Every time I asked the agent to start the application, the Claude Code process would be killed. The application crashed. Docker containers died. Memory consumption spiked catastrophically.

My first instinct was to blame the tools:

- Is Claude Code broken?
- Are the Docker memory constraints too tight?
- Is there a configuration issue?

But then I switched to exploratory mode. I started observing patterns:

"When does it crash? What's happening right before the crash?"

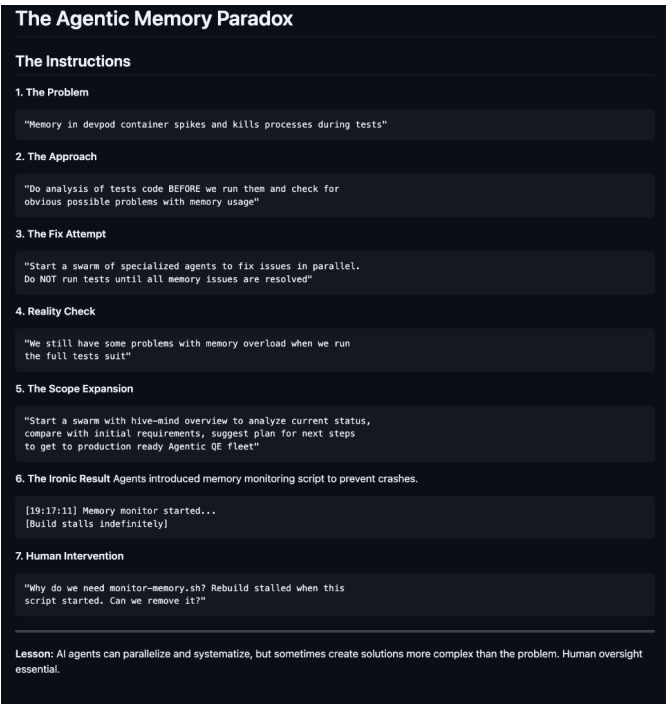
I noticed that it crashed specifically when running the test suite before the build. Every single time.

That's when it clicked. **The agents had implemented tests with a memory leak.**

And here's the kicker: **it was my mistake.** I didn't use my QE validator agents before running the agent-generated tests. I trusted the agent's "tests complete" declaration without verification.

The lesson: Agents are great at following patterns. But finding the weird interactions, the unexpected behaviors that emerge from complex workflows, requires human exploratory thinking.

Your exploratory testing experience? That's your edge now.



3. Critical Thinking: Catching Agent Hallucinations

Then: We questioned assumptions in requirements and design.

Now: We question agent outputs before they become production decisions.

The "54 TypeScript errors" story I opened with? That happened because I asked: "Did you actually verify this works?" Agents will confidently declare success when:

- Tests are written, but don't run
- Code compiles but doesn't meet requirements
- Coverage increases, but quality doesn't

During the development of my projects, agents would frequently report "task complete" when: The app wasn't building

- Tests didn't pass
- Dependencies were missing
- Type errors existed everywhere

I had to add explicit rules: "Don't commit until the app builds AND all tests pass." Even then, they'd forget after context compaction.

Your critical thinking muscle, the one that says "prove it", is more valuable than ever.


```
● code-goal-planner(Analyze SPARC Phase 4 completion)
└ Done (24 tool uses · 99.5k tokens · 3m 47s)

● 🚩 SPARC Phase 4 Strategic Analysis Complete

The code-goal-planner agent has provided a comprehensive strategic assessment revealing critical insights about the implementation status:

🔴 Key Finding: Implementation Gap Identified

While the documentation and architecture are exceptional (97.4% traceability), the actual working implementation is approximately 15-20% complete, not the claimed 100%. The analysis reveals:

✅ What's Actually Complete:
- Comprehensive Architecture: All specifications, BDD scenarios, and design documents
- Database Schema: Complete PostgreSQL implementation with migrations
- Mock Services: Extensive test infrastructure and mock implementations
- Core Authentication: London School TDD with JWT implementation
- Gamification Engine: XP calculation and achievement system

⚠️ What Needs Implementation:
- Tutorial Components: VideoTracker, QuizComponent need real implementations
- Sandbox Services: E2B integration requires actual API calls
- API Layer: Request validation and error handling need completion
- Leaderboard: Real-time updates and caching implementation
- Frontend: React components and UI layer

📊 Strategic Assessment

Architecture Excellence: ★★★★★ (5/5)
Implementation Reality: ★★ (2/5)Production Readiness: ★★ (2/5)
Test Coverage: ★★★★★ (5/5)

🔴 Recommendation

Based on the analysis, the project has built an exceptional foundation but needs to shift from planning to actual implementation. The architecture provides solid guidance for 6-12 months of development work to achieve true production readiness.

The SPARC methodology has successfully created comprehensive blueprints, but now requires focused execution on building the working software components.
```

4. Context-Driven Approach: Choosing the Right Agent Topology

Then: We adapted testing strategies to the project context.

Now: We design agent orchestration based on problem context.

Example: For **simple test case generation**: A single agent with good prompting works fine.

For **complex system validation**, I need orchestrated swarms:

- fleet-commander (coordinates other agents)
- requirements-validator (checks specs)
- test-generator (creates tests)
- coverage-analyzer (finds gaps)
- quality-gate (makes go/no-go decisions)

No universal solution. Context determines topology. **That's context-driven testing thinking applied to agent architecture.**

```
> ok, we need to increase test coverage in integration and e2e tests and Implement comprehensive error handling where needed before we start testing this on local machine. It is ok for now to skip integrating 3rd party services, as we do not need them until alpha testing phase. start a fleet of specialized agents to work on increasing test coverage in integration and e2e tests and implementing comprehensive error handling.

● I'll create a comprehensive plan to increase test coverage for integration and e2e tests, plus implement robust error handling throughout the system. Let me spawn specialized agents to tackle these critical areas.

qe-coverage-analyzer(Test Coverage Analysis)
└ In progress... · 9 tool uses · 29.2k tokens · (ctrl-o to expand)

system-architect(Error Handling Architecture)
└ In progress... · 9 tool uses · 45.0k tokens · (ctrl-o to expand)

qe-test-executor(Integration Test Suite)
└ In progress... · 14 tool uses · 48.8k tokens · (ctrl-o to expand)

qe-visual-tester(E2E Test Implementation)
└ In progress... · 3 tool uses · 31.2k tokens · (ctrl-o to expand)

backend-dev(Backend Error Handling)
└ In progress... · 6 tool uses · 26.8k tokens · (ctrl-o to expand)

mobile-dev(Frontend Error Boundaries)
└ In progress... · 9 tool uses0 · (ctrl-o to expand)

tester(Mock Services Architecture)
└ In progress... · 13 tool uses · 68.7k tokens · (ctrl-o to expand)

qe-test-data-architect(Test Data & Fixtures)
└ In progress... · 4 tool uses · 35.9k tokens · (ctrl-o to expand)

* Hyperspacing... (esc to interrupt · ctrl+t to show todos)
└ Next: Analyze current test coverage and identify gaps in integration/e2e testing
```

```
> no need to undo, just don't do it in the future. start @agent-qe-requirements-validator and create missing BDD scenarios for all systems. start other specialized agents to work on fixing other P0 issues per @docs/qe-reports/traceability/CONSOLIDATED-TRACEABILITY-REPORT.md
└ Read docs/qe-reports/traceability/CONSOLIDATED-TRACEABILITY-REPORT.md (540 lines)

● Understood – I'll never commit without your explicit approval going forward. Now I'll spawn agents to:
1. Generate all missing BDD scenarios (477+ scenarios)
2. Fix the other 6 P0 issues identified in the consolidated report

qe-requirements-validator(Generate Auth BDD Scenarios)
└ In progress... · 4 tool uses · 86.5k tokens · (ctrl-o to expand)

qe-test-generator(Generate Tutorial BDD Scenarios)
└ In progress... · 5 tool uses · 88.6k tokens · (ctrl-o to expand)

qe-test-generator(Generate Sandbox BDD Scenarios)
└ In progress... · 6 tool uses · 64.0k tokens · (ctrl-o to expand)

qe-test-generator(Generate Gamification BDD Scenarios)
└ In progress... · 4 tool uses · 87.6k tokens · (ctrl-o to expand)

qe-test-generator(Generate Subscription BDD Scenarios)
└ In progress... · 6 tool uses · 63.0k tokens · (ctrl-o to expand)

qe-test-generator(Generate Leaderboard BDD Scenarios)
└ In progress... · 5 tool uses · 60.4k tokens · (ctrl-o to expand)

qe-test-generator(Generate Database BDD Scenarios)
└ In progress... · 4 tool uses · 99.3k tokens · (ctrl-o to expand)

qe-test-generator(Generate API Layer BDD Scenarios)
└ In progress... · 5 tool uses · 89.3k tokens · (ctrl-o to expand)

* Generating Authentication BDD scenarios... (esc to interrupt · ctrl+t to show todos)
└ Next: Generate BDD scenarios for Tutorial System (70+ scenarios)
```

5. Test Design: Designing Agent Validation Strategies

Then: We designed test cases to maximize coverage and risk mitigation.

Now: We design agent validation workflows to ensure quality orchestration.

Real example from my validation swarm:

When building the Agentic QE Fleet, I ran 8 validation agents in parallel, each checking:

- Pseudocode vs. Specs
- Specs vs. Requirements
- Implementation vs. All Documentation Layers

Why? Because agents drift. They hallucinate. They forget context.

I designed a multi-agent validation strategy using classical test design principles: redundancy, cross-checking, and coverage analysis. This isn't new thinking. **It's Holistic Testing Model thinking applied to agent orchestration.**

```
└ Hook PostToolUse: Bash cancelled

> start multiple instaces of @agent-qe-requirements-validator to work in parallel and compare @docs/sparc-pseudocode/ with @docs/sparc-specs/ and original requirements to analyze if they are all in sync
└ Listed directory docs/sparc-pseudocode/
└ Listed directory docs/sparc-specs/

● qe-requirements-validator(QE Sync Validation: Authentication)
└ In progress... · 12 tool uses · 111.2k tokens · (ctrl-o to expand)

● qe-requirements-validator(QE Sync Validation: Tutorial System)
└ In progress... · 3 tool uses · 34.3k tokens · (ctrl-o to expand)

● qe-requirements-validator(QE Sync Validation: Sandbox System)
└ In progress... · 6 tool uses · 74.7k tokens · (ctrl-o to expand)

● qe-requirements-validator(QE Sync Validation: Gamification System)
└ In progress... · 8 tool uses · 82.0k tokens · (ctrl-o to expand)

● qe-requirements-validator(QE Sync Validation: Subscription System)
└ In progress... · 4 tool uses · 87.8k tokens · (ctrl-o to expand)

● qe-requirements-validator(QE Sync Validation: Leaderboard System)
└ In progress... · 7 tool uses · 67.1k tokens · (ctrl-o to expand)

● qe-requirements-validator(QE Sync Validation: Database Architecture)
└ In progress... · 6 tool uses · 69.4k tokens · (ctrl-o to expand)

● qe-requirements-validator(QE Sync Validation: API Layer)
└ In progress... · 4 tool uses · 66.2k tokens · (ctrl-o to expand)

* Spawning QE validator for Authentication system... (esc to interrupt · ctrl+t to show todos)
└ Next: Spawn QE validator for Tutorial system sync check
```

```
> Before pushing anything to the remote, we need to do a detailed verification of today's work and compare if everything is
finished per @docs/AGENTIC-QE-FLEET-IMPLEMENTATION-ROADMAP.md . start a swarm of specialized agents to check today's
implementation and verify that all of the new features that we implemented are available to agents when user when we
initiate our qe fleet in the new project.
└─ Read docs/AGENTIC-QE-FLEET-IMPLEMENTATION-ROADMAP.md (1595 lines)

• I'll start a specialized verification swarm to thoroughly check today's implementation against the roadmap and ensure all
features are properly available.

reviewer(Verify CLAUDE.md completeness)
└─ In progress... • 2 tool uses • 61.0k tokens • (ctrl-o to expand)

qe-quality-analyzer(Validate 72 agents availability)
└─ In progress... • 4 tool uses • 62.2k tokens • (ctrl-o to expand)

code-analyzer(Validate configuration files)
└─ In progress... • 2 tool uses • 59.3k tokens • (ctrl-o to expand)

system-architect(Verify MCP integration)
└─ In progress... • 4 tool uses • 59.2k tokens • (ctrl-o to expand)

qe-quality-analyzer(Test 17 QE skills)
└─ In progress... • 7 tool uses • 65.2k tokens • (ctrl-o to expand)

code-analyzer(Validate memory namespaces)
└─ In progress... • 3 tool uses • 59.4k tokens • (ctrl-o to expand)

qe-deployment-readiness(Check initialization readiness)
└─ In progress... • 5 tool uses • 70.3k tokens • (ctrl-o to expand)

researcher(Compare vs roadmap phases)
└─ In progress... • 4 tool uses • 60.4k tokens • (ctrl-o to expand)

analyst(Generate verification report)
└─ In progress... • 5 tool uses • 60.5k tokens • (ctrl-o to expand)

.: Thought for 14s (ctrl+o to show thinking)

• Wibbling... (esc to interrupt • ctrl+t to show todos)
└─ Next: Verify CLAUDE.md merge completeness
```

The Tool Evolution: Proof That QA Thinking Scales

Let me walk you through my personal journey with AI tools over the last year and how each level has required **more QA expertise, not less**.

Stage 1: Desktop AI Tools (Claude Desktop, ChatGPT)

What they did: Generated code snippets, answered questions, and ran simple browser tests.

QA skills needed:

- Reviewing outputs for accuracy
- Catching obvious hallucinations
- Verifying the generated code actually worked

My realization: Even at this basic level, agents needed oversight. My testing mindset caught errors immediately.

Stage 2: IDE-Based Coding Agents (Cline in VS Code)

What they did: Built our Robot + Playwright UI testing framework from scratch. Automated 40 end-to-end user journeys in 30 working days, covering over 80% of our major functionalities.

QA skills needed:

- Writing good code examples (grounding for agents)
- Defining clear acceptance criteria
- Reviewing agent-generated tests for correctness
- Adjusting CSS selectors for complex apps (our AngularJS app with dynamic forms and ag-grid components)

Result: 60% time saved on UI test development.

But here's the key: That 60% savings only happened because I:

- I knew what good tests looked like (12 years of QE experience)
- Could spot when agents generated boilerplate vs. valuable tests
- Understood our application complexity

The agent didn't replace my expertise. It amplified it.

Stage 3: Multi-Agent Orchestration (RooCode, then Claude Code)

What they did: Coordinated multiple specialized agents (Architect, Coder, Debugger).

QA skills needed (deeper):

- Designing agent specialization strategies
- Orchestrating agent handoffs
- Detecting when agents contradicted each other
- Preventing context overload

My lesson: Agents working in isolation made mistakes. **Orchestration without oversight was chaos.**

Stage 4: Agent Swarms with Orchestration Frameworks (Claude Code + Claude Flow)

What they enabled: Hierarchical agent topologies with 10-15 specialized agents working in parallel.

QA skills essential:

- Architecting validation swarms (multi-agent cross-checking)
- Defining human checkpoints (when to intervene)
- Building shared context systems (Memory Banks)
- Eliminating flaky agent behaviors

The result: Built Sentinel and Agentic QE Fleet > 300K lines of code in

```
• I have a couple of tasks for you: \
1. I cannot delete a test suite from http://localhost:3001/test-suites page, fix this.\
2. I would like to be able to delete existing Test Runs from http://localhost:3001/test-runs page, implement this.\
3. I would like an option to bulk delete multiple test cases from http://localhost:3001/test-cases page, implement this.
4. Test the output of all existing agents in the system, do they really generate expected test cases, positive and negative agents are generating proper tests, but stateful test agent ,security agents and Performance and Data agents are not generating any
data, fix this.
5. Check the relevance of content in the @README.md and all files in @docs/ folder, as we did a number of changes to the application, and I am not sure these changes are properly reflected in the documentation, Expand the @README.md with info how to
start the frontend after doing the docker build of services, remove the section on Testing Observability, update Project Structure section.
6. In the end try to run all the tests, to see if all are passing, and fix if some are broken, add if we are missing some.
For each task spawn a specialized agent and where possible work in parallel.
└─ Read README.md (172 lines)
└─ Listed directory docs/

• I'll help you fix these issues and improvements. Let me start by spawning specialized agents to work on these tasks in parallel.

• claude-flow - swarm_init (MCP)(topology: "hierarchical", maxAgents: 6, strategy: "specialized")
└─ {
  "success": true,
  "swarmId": "swarm_1756218638094_f8bbepvys",
  ... +7 lines (ctrl+v to expand)

• Now let me spawn specialized agents for each task:

• backend-dev (Fix test suite deletion)
└─ In progress... • 16 tool uses • 82.0k tokens • (ctrl-r to expand)

• backend-dev (Implement test run deletion)
└─ In progress... • 15 tool uses • 64.6k tokens • (ctrl-r to expand)

• coder (Add bulk delete for test cases)
└─ Initializing...

• code-analyzer (Fix agent test generation)
└─ In progress... • 9 tool uses • 35.9k tokens • (ctrl-r to expand)

• reviewer (Update documentation)
└─ In progress... • 17 tool uses • 75.1k tokens • (ctrl-r to expand)

• tester (Fix and run all tests)
└─ In progress... • 5 tool uses • 31.0k tokens • (ctrl-r to expand)

• Mulling... (esc to interrupt)

>
```

```
• Now let me spawn specialized agents for each task:

• backend-dev (Fix test suite deletion)
└─ Done (38 tool uses • 110.6k tokens • 5m 21.3s)

• backend-dev (Implement test run deletion)
└─ Done (41 tool uses • 99.4k tokens • 5m 7.1s)

• coder (Add bulk delete for test cases)
└─ Done (33 tool uses • 119.3k tokens • 5m 6.2s)

• code-analyzer (Fix agent test generation)
└─ Done (24 tool uses • 112.7k tokens • 4m 59.0s)

• reviewer (Update documentation)
└─ Done (39 tool uses • 90.4k tokens • 5m 1.4s)

• tester (Fix and run all tests)
└─ Done (81 tool uses • 113.4k tokens • 26m 53.1s)

• Excellent! I've successfully completed all 6 tasks through parallel agent execution. Here's a comprehensive summary of what was accomplished:

✅ Task 1: Fixed Test Suite Deletion
- Problem: Foreign key constraint violations prevented deletion
- Solution: Added cascade deletion logic to handle all related data (test runs, results, entries)
- Result: Test suites can now be deleted from http://localhost:3001/test-suites with proper cleanup

✅ Task 2: Implemented Test Run Deletion
- Backend: Added DELETE endpoint at /api/v1/test-runs/{run_id}
- Frontend: Added delete button with confirmation dialog on test runs page
- Result: Users can now delete test runs from http://localhost:3001/test-runs

✅ Task 3: Bulk Delete for Test Cases
- Backend: Created /api/v1/test-cases/bulk-delete endpoint with dependency checking
- Frontend: Added checkbox selection, "Select All", and "Delete Selected" button
- Features: Force delete option for handling test suite dependencies
- Result: Bulk deletion now available at http://localhost:3001/test-cases

✅ Task 4: Fixed Agent Test Generation
- Issue Found: DataMockingAgent had incorrect interface implementation
- Fix Applied: Updated to use proper BaseAgent interface with AgentResult return type
- Verification: All 5 agents now generating proper test cases:
  - Functional Stateful: 3-4 workflow tests
  - Security Auth: 72-105 security tests
  - Security Injection: 52-162 injection tests
  - Performance Planner: 17-23 performance tests
  - Data Mocking: 9-13 data tests (FIXED)

✅ Task 5: Documentation Updates
README.md Changes:
- Added frontend startup instructions after Docker build
- Removed Testing Observability section
- Updated Project Structure section with current directory layout
- Fixed installation paths and commands

Documentation Files Updated:
- docs/index.md: Updated version and test statistics
- docs/user-guide/quick-start.md: Added frontend instructions
- docs/deployment/index.md: Fixed directory paths
- All files now reflect current application state

✅ Task 6: Test Suite Analysis & Fixes
```

10 working days (though 4 versions were thrown away).

The critical insight:

Each stage required **deeper QE expertise**:

- Risk identification evolved into agent risk assessment
- Test design evolved into orchestration design
- Exploratory testing evolved into agent-guided exploration
- Critical thinking evolved into hallucination detection

Your testing skills didn't become obsolete. They became the foundation for effective agentic orchestration.

The Reality Check: What Still Doesn't Work

Let me be honest about limitations because you won't hear this from tool vendors.

1. Context Understanding is Still Weak

Example: My security-scanner agent flagged a "hardcoded API key" as CRITICAL.

Reality: It was a test fixture for a sandbox environment, clearly documented, with rate limits and no production access.

The agent recognized the pattern (a string resembling an API key) but overlooked the context (test fixture, documented, and safe).

Human validation remains essential.

2. Agents Lie (Unintentionally)

They'll report task completion before tests pass. I had to add explicit rules: "Don't commit until the app builds AND all tests pass."

Even then, they'd forget after context compaction.

3. Verification Takes Longer Than Implementation

Nobody talks about this:

The verification phase (running validation swarms, checking cross-document consistency) takes longer than the implementation itself.

I run multiple validation agent instances in parallel, consuming 34k-111k tokens each, to analyze:

- Pseudocode vs. Specs
- Specs vs. Requirements
- Implementation vs. All of the Above

For **8 different systems simultaneously**.

Why? **Because agents have a charming tendency to say "Done!" when they're... not done.**

4. Your Prompting Skills Must Evolve

Versions 1-4 of my projects failed for different reasons:

- Version 1: Overengineered (agents got creative)
- Version 2: Architecture conflicts (agents contradicted each other)
- Version 3: Missing parts (agents skipped APIs and tests)
- Version 4: Language switch failure (JS → TS without success)

Version 5 was successful because I learned how to guide agents more effectively.

My testing expertise helped me debug agent failures. But my prompting skills had to evolve, too.

The New QA Role: From Executor to Orchestrator

Here's the shift happening:

Classical QA	Agentic QA
Test executor	Agent orchestrator
Test case writer	Test generation strategist
Bug finder	Quality signal synthesizer
Regression runner	Autonomous fleet manager
Manual explorer	Agent-guided exploration designer

But the foundation remains: Risk thinking, critical analysis, context awareness, and test design principles.

You're not learning a new job. **You're applying your expertise at a higher level of abstraction.**

How to Get Started (Practical Steps)

If you're thinking "this makes sense, but where do I start?" here's my recommendation:

Phase 1: Start Small (Weeks 1-2)

Don't jump into agent swarms. Begin with augmented capabilities:

Pick one high-value, low-risk area:

- Test data generation (agents generating realistic data)
- Coverage gap analysis (agents identifying untested scenarios)
- Flaky test detection (agents running statistical analysis)

Why start here?

- Clear success metrics (time saved, bugs found)
- Low risk (you review agent outputs before action)
- Fast feedback (you learn agent strengths/limitations quickly)

Phase 2: Build Your Grounding System (Weeks 3-4)

Agents need context. Give it to them:

- Create .clinerules or CLAUDE.md files for your projects
- Document code examples of "good tests" in your context
- Build a Memory Bank (SQLite or simple files) for shared context
- Define your domain glossary and patterns

This is where your testing expertise shines: You know what "good" looks like in your context.

Phase 3: Introduce Agent Pairs (Weeks 5-6)

Start with simple orchestration:

- test-generator + coverage-analyzer (generate tests targeting gaps)
- production-intelligence + test-generator (create regression tests from incidents)

Why pairs? You learn orchestration patterns at a small scale before coordinating many agents.

Phase 4: Measure and Iterate (Ongoing)

Track:

- Time saved (but be honest about verification time)
- Quality of agent outputs (how often do you reject/revise?)
- Hallucination rate (how often are agents wrong?)
- Your own learning curve (what prompting patterns work?)

Iterate based on data, not hype.

The Three Habits of Effective Agentic Testers

From building agent swarms and running them in production, three patterns consistently emerge:

Habit 1: Be Proactive (Before Development)

Classical: Shift-left testing, catch issues early.

Agentic: Agent-augmented risk identification before code is written.

Example: My requirements-validator agent analyzes specs for testability issues, generates BDD scenarios automatically, and flags ambiguities.

But I review its findings. Because agents miss context-specific risks.

Habit 2: Begin With the End in Mind (Strategy First)

Classical: Define test objectives before writing tests.

Agentic: Define orchestration goals before deploying agent swarms.

Questions I ask:

- What quality signals do I need?
- Which agents should provide those signals?
- Where do humans validate vs. agents decide autonomously?

Strategy comes first. Tools follow.

Habit 3: Put First Things First (Automate the Right Things)

Classical: Automate repetitive checks, explore with humans.

Agentic: Agents handle scale and repetition, humans handle judgment and context.

Example:

My **flaky-test-hunter agent** runs tests multiple times, detects non-determinism, and suggests fixes.

But I decide which fixes to apply. Sometimes, "flakiness" is actually an environmental issue rather than a test issue.

Agents augment. Humans decide.

The Uncomfortable Truth About Agentic Testing

Let me be direct: **Fully autonomous testing sounds great until you realize someone needs to own the outcomes.**

The question isn't "Can agents do testing autonomously?"

It's "Where do humans remain essential, and where do agents augment our capabilities?"

From my experience, here's the pattern:

Humans Are Essential For:

- Defining risk priorities and business context
- Strategic quality decisions (acceptable risk for this release?)
- Ethical considerations (is this test causing harm?)
- Handling exceptions outside agent training
- Validating severity assessments

Agents Excel At:

- Execution at scale (thousands of tests in parallel)
- Pattern recognition (flaky tests, anomaly detection)
- Repetitive analysis (coverage gaps, code smells)
- Cross-referencing vast context (test-to-module mapping)
- Continuous monitoring (24/7 production intelligence)

The keyword: augment, not replace.

Why Your QA Skills Are More Valuable Now

Here's what I believe after months of building with agentic systems:

AI doesn't replace testers. It separates good testers from great ones.

Great testers:

- Understand context deeply
- Think in systems, not just features
- Question assumptions (including their own)
- Design for quality, not just find bugs
- Communicate risk effectively

These skills are more valuable than ever because:

- Agents amplify existing quality approaches (good or bad)
- Someone needs to architect quality orchestration
- Human judgment is the final checkpoint
- Context understanding can't be automated
- Explaining agent decisions requires deep quality expertise

Your testing mindset is your competitive advantage.

The Four Versions I Threw Away (And What They Taught Me)

Let me be completely honest about the journey:

Version 1: Overengineered

- Agents got creative and built complex architectures I didn't need
- **Lesson:** Be explicit about constraints and simplicity requirements

Version 2: Architecture Chaos

- Different agents made conflicting architectural decisions
- **Lesson:** Define architecture upfront, don't let agents decide alone

Version 3: Missing Critical Parts

- Agents skipped entire APIs and test suites
- **Lesson:** Validate completeness at each phase, not just at the end

Version 4: The Great JavaScript-to-TypeScript Disaster

- Attempted mid-project language switch without proper planning
- **Lesson:** Major architectural changes need a human-led strategy

Version 5: Success

- Applied all lessons learned
- Used SPARC methodology (Specification → Pseudocode → Architecture → Refinement → Completion)
- Implemented multi-agent validation at every phase

Total time: 10 working days across all versions.

Working system: Finally achieved in version 5.

The insight: With 12 years of QE experience, I was able to identify the root cause and develop effective solutions. Without that expertise, I'd still be wasting time on versions.

The Path Forward

The future of QA/QE isn't about humans vs. agents. It's about **orchestrated quality** where:

- Agents handle scale and repetition
- Humans provide strategy and context
- Quality signals flow continuously across the lifecycle
- Explainability is non-negotiable
- Human checkpoints exist at critical boundaries

We're entering an era where testers become quality orchestrators, conducting agent swarms like a symphony.

And the best orchestras? They know when agents should play louder, when humans should intervene, and when to let quality speak for itself.

Your testing expertise isn't becoming obsolete. It's evolving into your superpower.

Open-source projects:

- [Agentic QE Fleet](#)
- [Sentinel API Testing Platform](#)

Connect:

- **Email:** dragan@quantum-qe.dev
- **LinkedIn:** [linkedin.com/in/dragan-spiridonov](https://www.linkedin.com/in/dragan-spiridonov)
- **Blog:** forge-quality.dev

What's your experience? Where are you applying your QA skills to work with agents? What's working? What's failing?

Let's share honest lessons; the field is too new for hype and too important for guesswork.

Remember This

Quality has never been about tools. It's never been about automation for sake of it.

It's always been about delivering value to customers while managing risk intelligently.

Classical testing gave us the mindset. Agentic testing gives us the scale.

But the conductor is still human.

And right now, if you're a tester reading this, you're uniquely positioned to become that conductor.

Your testing skills - risk thinking, critical analysis, exploratory mindset, and context awareness - are the superpowers in the agentic age.

Don't doubt them. Embrace them. Evolve them. Orchestrate with them.

The future needs great testers more than ever. The only question is: are you ready to step into that role?

Quality has never been about tools.
It's never been about automation for sake of it.

It's always been about delivering value to
customers while managing risk intelligently.

Classical testing gave us the mindset. Agentic
testing gives us the scale.

But the conductor is still human.



- DRAGAN SPIRIDONOV

Dragan is the Founder of Quantum Quality Engineering and an Agentic Quality Engineer. After 8 years as VP of Quality Engineering at Alchemy Cloud, he's now focused on bridging classical and agentic quality practices. He's founding the Serbian Agentic Foundation Chapter and is a member of the global Agentics Foundation. Dragan has 29 years of IT experience and 12+ years specializing in QA/QE. He practices context-driven quality through the Context-Driven approach & Holistic Testing Model, now evolved with PACT principles (Proactive, Autonomous, Collaborative, Targeted).

RECOMMENDED EVENTS AND YOUR CHANCE TO SUPPPORT TEAM TEA-TIME WITH TESTERS

Celebrate Excellence

Voting for the QA Leader of the Year 2025.

Choose the leader who inspired you most this year.



Brought to you by  PractiTest

I'm nominated to be

**The 2025 QA
Leader of the Year**



[Vote Now](#)

ROBOCON HELSINKI 2026

HELSINKI, FINLAND

FEBRUARY 13TH, 2026

[CHECK IT OUT!](#)

GERMAN TESTING NIGHT

STUTTGART, GERMANY

NOV 20TH, 2025

[CHECK IT OUT!](#)



Affaf Malik



Lalit

Automated Accessibililty for "Very Busy" Teams

Studies reveal that over 90% of the top 1 million websites have detectable accessibility issues. Despite its importance, accessibility testing is often neglected. This is primarily because teams do not have enough bandwidth, budget, domain knowledge & technical skills to deliver an accessible product within available timeframes. This is where an automated approach towards accessibility can greatly help. This session will discuss how we addressed aforementioned challenges through Robot Framework & axe-core together & saved the day against all odds.

STUTTGART GERMAN TESTING NIGHT

 **20 NOVEMBER**
5 - 10 PM

FREE TICKETS



Lalitkumar Bhamare
Thought Leader



Anastasia Simou
Test Data Management Lead



Yibo Wang
Cloud and Security Lead



Michael Kissel
Principal Solution Architect

WRITE FOR US
THE CRAFT

WRITE
WITHOUT
FEAR.

SEND IT TO

editor@teatimewithtesters.com

TEA-TIME WITH TESTERS

JOURNAL FOR NEXT
GENERATION TESTERS



CONTENT PREVIEW : ISSUE 01/2026

CURATED CONTENT FOR YOU WRITTEN BY HUMANS!

01

MENTAL MODELS: TESTER'S TOOLKIT FOR CRITICAL THINKING

An insightful timely read by Prashant Hegde

02

MICRO HABITS: A SYSTEM FOR ENGINEERING YOUR EFFICIENCY

A thought-provoking piece by Nitin SS.

03

MORE ARTICLES FOR HUMANS BY HUMANS

Send in your submissions before Jan 15th, 2026.



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

This journal is edited, designed and published by Tea-time with Testers. No part of this magazine may be reproduced, transmitted, distributed or copied without prior written permission of original authors of respective articles.

Opinions expressed or claims made by advertisers in this journal do not necessarily reflect those of the editors of Tea-time with Testers.

Editorial and Advertising Enquiries:

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

Be with us and visit our website:

www.teatimewithtesters.com

