WAKING TESTERS UP SINCE 2011

ISSUE #02/2023

TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

<u>QCSD, Career,</u> Al, and T<mark>esting</mark>!

AI NEEDS TESTING

Page 12

INTERVIEWING AND RECRUITING TESTERS Page 21

QCSD AT IFM ENGINEERING Page 34



TESTING QCSD CAREED

TEA-TIME WITH TESTERS



PEOPLE

IDEAS THAT SPEAK FROM THE MINDS THAT THINK

INTERVIEV OVER A CUP OF TEA CONVO WITH GREAT MINDS IN TECH

EDITORIAL BY LALIT

INTERVIEW: 30-33 A CUP OF TEA WITH FIONA CHARLES

AI NEEDS TESTING

Al requires more rigorous test humanity's best interests.





WHY IS BDD CONFUSED WITH

MUSINGS OVER JOYS AND PA

TEA-TIME WITH TESTERS ISSUE #02/2023

V	PRO
\bigcirc	

DCESSES ARE YOU

DOING IT RIGHT? FIND IT OUT

PRODUCTS

BUILDING THINGS THAT PEOPLE WOULD USE HAPPILY

TEST ENGINEERS' CAREER PROGRESSION Although "Progression" and "Advancement" in a career are frequently used synonymously, there is a distinction between	08-10
AI NEEDS TESTING AI requires more rigorous testing to ensure its development aligns with humanity's best interests.	12-17
WHY IS BDD CONFUSED WITH TESTING? The irony of Behaviour Driven Development (<u>BDD</u>) is that it is a software development process to reduce confusion about what software is supposed to do but	18-20
MUSINGS OVER JOYS AND PAINS OF INTERVIEWING AND RECRUITING To say that the process of hiring staff and getting a job in testing, or even in tech, is suboptimal is the understatement of the century.	21-24

TEA-TIME WITH TESTERS



A NEXT GENERATION TESTING MAGAZINE

	TEA AND TESTING WITH JERRY WEINBERG	26-28
s	The Technology of Human Behaviour - Part 1	
	QCSD AT IFM ENGINEERING Have you ever wondered what makes a product truly high-quality?	34-38
4	US HEALTHCARE TESTING Health Care Domain Functional knowledge is one of the most important skillset & plays a critical role while testing healthcare applications	40-43
	THE PLACE FOR CHATGPT In last issue I took my first, <u>presumptive look at #ChatGPT for testing</u> . The app has real potential, and seems to have some ability to learn.	46-48

The concept of Phase Space was introduced by "Dynamical Systems Theory". which is an area of mathematics that describes the existence of complex systems and their behaviors in conjecture with each other and also independently.



PHASE SPACE - INTRODUCTION

"Testing" has arrived... yet another time!

Whenever I am on panel discussions or doing interviews, one particular question is often asked. Guess what that question is?

You got it right! How do you see future of testing in next 5 years? (or 10 or tomorrow.....)

Among things I said in my answers at different points of time, "critical thinking" as an important skill for testers, was always mentioned. And I am sure that if I am to answer the same question after 30 years, I will still mention it.

The only difference between then, now and in near future would be that, testers has been primarily interacting with fellow humans on the team but in coming days they would be interacting with machines as much as they would with fellow humans.

While our surroundings are flooded with topics around AI, ChatGPT, AI based innovations and adaptions, it is imperative that testers too need to figure out what it means for them and the way they test.

A lot of interesting discussions have been happening on the topics of AI in testing, AI for testers, testing and AI and so on. Based on my understanding of those discussions, the literature I went through and my personal exploration of the topic, there is one thing I can confidently say...

Testing is here to stay! And it's going to become more important that it was ever perceived in the past.

This issue of Tea-time with Testers has interesting articles on various topics surrounding testing, quality, AI and human aspect of testing. And I believe that will help one understand how widespread and deeply present testing is.

I would like to thank all the authors who have generously shared their thoughts and expertises for the benefit of our reads.

Special shout-out to Avo Automation for appreciating what Tea-time with Testers has been doing. Their support is going help to us bring out lots of interesting things for the testing community in near future. Stay tuned for more on that!

On that note, I wish you happy "testing" times and see you next time.

Sincerely,

Lalit

50 - 53



LALITKUMAR BHAMARE

CEO, Chief Editor "Tea-time with Testers"

Manager - Accenture Song, Germany Director - Association for Software Testing International Keynote speaker. Award-winning testing leader. Software Testing/Quality Coach.

Connect on Twitter @Lalitbhamare or on LinkedIn



Deliver Top quality Software with Test Automation

Avo offers 100% no-code test automation for:







Avo iTDM for next generation test data

and many more



Avo Assure for functional testing:

Top features:

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

You achieve:

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

Top features:

management:

Mainframe

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

You achieve:

Non-compliant data in non-production environments

٧Ŏ

- Data privacy regulations like GDPR, CCPA, and other compliances
- Compliance with continually evolving privacy regulations

Recommended by Industry Experts





seastarconf

Technical Arch Manager at Accenture Song, Germany

2-Day Certification Training **QUALITY-CONSCIOUS SOFTWARE DELIVERY**

https://quality.seastarconf.com USE PROMOCODE "ILOVETTIME" TO GET 10% OFF TICKET PRICES

OCTOBER 15-19 2023

TEST ENGINEERS' CAREER PROGRESSION



~People

Although "Progression" and "Advancement" in a career are frequently used synonymously, there is a distinction between the two, even though they are complementary. For instance, if you begin your career as an associate test engineer, you might get promoted to test engineer after two years, and senior test engineer after three to four years. These movements are accompanied by perks and appropriate title changes. Career progression is what it is termed, and it typically happens as a result of promotions.

While career graph advancement is a continuous process. The advancement journey includes developing a more comprehensive professional context, more indepth knowledge, a broader skill set, and solid professional relationships. As an associate test engineer, for instance, you will

be expected to learn about testing as a whole. After a few days or months, you might start to concentrate more intently on developing your domain expertise, putting test approaches into practice, creating test strategies for common test routines, estimating the effort required for daily maintenance chores, etc. By doing this, you demonstrate a desire to learn more about the role and take a firmer grip on the deliverables.

Progressions are fundamental in that they provide a deeper comprehension of a particular ability and aid in the integration of theory and practice. The majority of the time, advancements do not involve a title change or pay raise (some times it does, but rare). Progressions, however, provide a clear path to advancements.

Why is progression important?

- Solidifies professional skillset, helps drive balance between the theory v/s practical approach
- Feeds the hunger to learn and grow
- A break from un-healthy long run monotonous job
- Paves way for new opportunities
- With a clear progression path instills the sense of purpose and responsibility



- SOWMYA SRIDHARAMURTHY

Sowmya is a seasoned product quality leader currently working as Engineering Manager- quality at Lytho. With 16 years of experience handling products right from inception to delivery, she has worked on diverse software solutions-ERP, SAAS, Mobile Apps, and Web applications.

She has a proven track record of successful implementations of result-driven test processes. Mentors teams in building effective strategies and implementations to achieve ROI through test automation.

Being an Accessibility advocate, Sowmya is keen on driving inclusive software development. She is an active community builder and runs the "APIans" meet-up group from Amsterdam.

Visit <u>https://someooow.com/</u> to know more about Soumya. You can connect with her on LinkedIn -<u>sowmyasridharamurthy</u> or Twitter <u>@Someoooww</u>

How can I drive my progression with my Manager?

The six guidelines listed below might help you plan and carry out advancement inside your organisation.



1. Role Clarity : Find out what is expected of you on a daily basis in your current role. the significance of performing particular duties and the expertise needed to carry them out. Always keep in mind that managers may not support advancement if they feel the current iob and responsibilities are not being fulfilled. Establishing a clear grasp of your current position and deliverables is the first step.

2. Primary and Secondary Responsibilities:

When working on a team, it comes naturally for us to proactively offer to help with tasks that are not formally part of our role. As an illustration, when I was a front-end developer, I contributed to the knowledge management team by authoring technical articles that the content team then reviewed and improved before publishing on the corporate website. However, it is crucial that I inform my manager of all the primary and secondary duties I am performing. Secondary tasks can help create opportunity to do things outside of our normal comfort zones while primary obligations help you establish a better grip in your existing career. Sometimes these supplementary responsibilities also encourage transferable talents, which aids in creating a greater impact when improvements are made.

3. Skill Matrix: It is impossible to overstate the value of the skill matrix. You can analyze your abilities and interests with the aid of the matrix. Give your manager a copy of this personal skill matrix and make sure they understand your preferences for, dislikes with, and ideal career path. Additionally, be receptive to feedback because managers will have a bird's eye view of the company's strategic decisions and will be able to help you better channel your ideas. Plotting the skill matrix against "Needs" and "Wants" is recommended. While wants are a list of abilities you're interested in learning but may or may not directly relate to your current function or task, needs are a list of skills needed to drive the next level challenge.

4. Strategic plan to build skills: The acquisition of new abilities will go hand in hand with the regular duties that are assigned. The particular challenge for team leaders will be to assist team members in developing new skills without interfering with the current delivery process. This can only be done with the support of a wellthought-out plan. The elements at play include the budget, timeframe, transition period, unknown hazards, etc. In either case, creating a learning path reinforced by practical experience aids in developing deeper knowledge, whether you are given a self-directed learning path or a companydriven guided learning path. Team leaders may need to create a cross-team skilling schema for the team member because not all needs can be met within the team.

5. Transition plan with Milestones defined:

The process of outlining the clarity of the complete progression activity is called a transition plan. This should have a deadline, explicitly describe the important parties who will be involved along the evolution, and, if applicable. include any budget requirements. Milestones are significant junctures at which the skills are assessed using established and pre-agreed criteria. For instance, if I decide to advance into performance testing as an exploratory test engineer, there are a few things I should be sure to discuss with my manager.

- · Express my interest in performance testing and internal opportunities to do this progression.
- · What technical skills should I be gaining to do this progression?
- · Learning agenda and learning schedule to be discussed, key stakeholders who can help me in this progression.
- · At what intervals do and on what parameters do we measure this progress?

6. Optional Benefits and exits: It's crucial to understand that not all progressions are accompanied by pay raises before moving forward you may make an informed choice without looking back. Additionally, it's critical to provide exits and reversal strategies in case something goes wrong in between; this is perfectly acceptable. It often happens that something that seemed attractive at first begins to drain our energy, thereby

destroying our interest in it. But be careful

that this can leave a bitter taste or give the

impression that you lack the necessary

drive towards any progression.

When should I discuss my progression with my manager?

You can discuss it anytime, the ideal time would be when you start craving new challenges. Now, I have seen teams where there are no options for any ad-hoc progressions as they are tightly bound to the HR rules of the usual advancement theory and strictly stick to yearly goal discussions.

Conclusion

Frequent progressions are the engine that drives career growth, so it's crucial to communicate your manager's progression plan on a regular basis. While managers struggle to keep their teams motivated, not all the time can one afford to offer promotions. Progressions are excellent fillers that also aid in providing the correct degree of challenges while a team member prepares for the next big thing.

Test engineers frequently become obsessed on the idea of creating test automation solutions, which prevents them from considering other alternatives. Consider things that aren't immediately apparent and work to establish a solid career one step at a time.

PROGRESSION



seastarconf

5 days in a tropical paradise to upgrade your skills MEET TOP SPEAKERS IN

A LUXURY SETTING

For Quality Enthusiasts Only

SEASTAR QUALITY CONFERENCE 2023

Learning. Networking. Lifestyling

EXPLORE QUALITY

5 days of learning with international experts

NETWORK AND PLAY

5 days in a tropical paradise to upgrade your skills

MEET TOP SPEAKERS IN **A LUXURY SETTING**

OCTOBER 15-19 2023

ality.seastarconf.com USE PROMOCODE "ILOVETTIME" TO GET 10% OFF TICKET PRICES

> October 15-19 2023 IBEROSTAR COSTA DORADA RESORT. DOMINICAN REPUBLIC.

Fun, games, and sports all day long

LIVE IN LUXURY

Stay in 5 star all-inclusive beachfront resort



https://quality.seastarconf.com USE PROMOCODE "ILOVETTIME" TO GET 10% OFF TICKET PRICES

NEFDS



Al requires more rigorous testing to ensure its development aligns with humanity's best interests. Thorough testing is essential to maintain control over AI while also allowing it to reach and potentially surpass human intelligence. The rapid advancements and concerns surrounding AI highlight the crucial nature of proper testing.

Some people are apprehensive about generative Als achieving human-like intelligence. The continuous improvement in Al's performance is evident as it successfully passes various standardized tests, including those in legal, medical, and software testing fields. Thus, the concerns and enthusiasm regarding Al's growing capabilities are well-founded, based on its test results. Below are several disjoint, sometimes conflicting, thoughts when considering the future of AI and software testing. Testing is a double-edged sword that must be continually sharpened regardless of how folks feel about AI. The topics below are either extremely important for this emerging world of Generative and Intelligent AI systems or testing-specific angles I don't hear discussed at all and need to be.

For those who glance at AI today and remain skeptical or believe it's too fallible to be helpful, let alone more intelligent than you, feel free to stop reading now and share your thoughts in the comments below – just be aware that they may not age well.

AI needs testers to think more deeply about AI.

Testing vs. Humans

It is becoming increasingly evident that AI bots will soon match or even surpass human expertise in various fields. The training process of AI is fundamentally rooted in testing. AI systems are initialized randomly and continue to improve through a series of input applications, output evaluations, and adjustments based on test results. This iterative process is repeated until the Al's performance plateaus or researchers exhaust their resources or patience. Since training AI inherently involves testing, AI can learn and progress as long as a test is available.

Exam results (ordered by GPT-3.5 performance)

Estimated percentile lower bound (among test takers) 100% -80% 60% 40% 20% - AP Chemistry -GRE Quantitati TUSABO Sel -AP Calculus FAMC 12 Codeforces Rating - AP English Lite Tuniform Bar Exa -AP English Lan -AP Physics 2 - AP Macrot - AP Statistics -AMC 10 LSAT 2020 Exam

JASON ARBON

Jason Arbon is the CEO at<u>testers.ai</u> where his mission is to make testing available to all apps and people with just a few taps. Google's AI investment arm led his last company's (test.ai) funding rounds.

Jason previously worked on several large-scale products: web search at Google and Bing, the web browsers Chrome and Internet Explorer, operating systems such as Windows and ChromeOS. and crowd-sourced testing infrastructure and data at uTest.com. Jason has also co-authored books such as How Google Tests Software and App Quality: Secrets for Agile App Teams



Recent results of GPT on professional tests demonstrate its ability to perform at the same level as experts in some areas and at least on par with the average human in many others. However, additional types of tests address other aspects, such as eliminating unwanted bias, enhancing conversational capabilities, and ensuring alignment with human goals and safety. These tests are relatively new and require further refinement to prevent any potential AI-related catastrophes. Fortunately, talented individuals are dedicated to developing more comprehensive and effective test definitions.

Headed for a Slowdown in AI

It's reasonable to assume that we can breathe a sigh of relief as AI bots, using current techniques, might not surpass human expertise indefinitely. These bots are simply learning from the wealth of human knowledge. We can observe that AI models like ChatGPT are indeed becoming more intelligent with each iteration, but the growth in their intelligence is, decelerating, and plateauing.

The challenge lies in the difficulty of scaling the testing process used for training. These AI systems have already processed a majority of the text and images generated by humans, making it increasingly timeconsuming and resource-intensive to incorporate new, valuable test data into their already extensive pool of tests. Most of the new data will likely be repetitive unless humans come up with innovative ideas

The AI is asymptotically approaching and expert — recreating the intelligence of its trainer — humans. The current techniques will unlikely produce intelligence or generate text that exceeds the cleverness of the humans it is optimized to reproduce.

Renesaince AI Testing

While today's generative AI bots may not surpass the intelligence of a single human expert in a specific area, they still have the potential to become experts in multiple fields. Imagine an AI that is not only a medical doctor and a lawyer but also a chemist, physicist, economist, historian, military strategist, AI researcher, software test engineer, psychiatrist, MBA, fighter pilot, philosopher, and truck driver. The combination of these skills in a single AI entity is indeed impressive. Even Leonardo wasn't as broadly skilled.

Historically, many groundbreaking scientific discoveries and influential leaders have emerged from people with multidisciplinary backgrounds. As such, AI has the potential to become the most multidisciplinary entity ever, even if it doesn't outshine the expertise of any individual human in a specific, testable category.

Cheaper, Better, Faster Testing

While AI may not become significantly smarter in specific areas or surpass human intelligence anytime soon, we should recognize the transformative impact these bots can have on society. Even if their proficiency in a given, testable human expertise is merely on par with human experts, they offer considerable advantages in terms of cost and efficiency.

Al bots can be up to 100 times less expensive and faster than their human counterparts. This means they can outperform humans in certain tasks and provide their services universally at a full scale, with no waiting times for appointments or consultations. The realtime availability of Al can significantly disrupt industries that rely on human expertise.

While this prospect may seem daunting for those in testable professions, it is essential to recognize the potential benefits of Aldriven advancements. By embracing change and adapting to the new landscape, humanity can continue to progress and find innovative ways to coexist with Al. Right?

The Software Tester's View

Software testers view themselves as more clever than the cleverest of humans. These are the people that like to see things that others's missed. They constantly explore the state-space of a problem, looking for the "gotchas", the scenarios no one else considered. They feed on the creation of others. Rapid advances in AI will have a few interesting implications specifically for Software Testers.

Testing, whether it is called that or not, is quickly becoming the most critical job and the topic of every podcast and news story. Reporters are videoing themselves 'testing' the new AI chatbots. Researchers at universities coming up with test suites for AI to check for alignment, safety, bias, etc., are now the talk of the town. And so many engineers, lawyers, etc. are 'testing out' these new bots to see how well they perform on tasks. Building these AIs is looking more like a commodity every day.

The concern is that many of these researchers aren't well-versed in the issues that skilled software testers make. The skilled software testers also aren't jumping on the alignment, safety, or bias issues either. Hopefully, that changes soon because these two fields are important, merging, and need to accelerate their competence as the generation of AI accelerates, demanding better testing.

In the near term, generative AI will literally generate orders of magnitude more software and general software output that needs to be tested. The testing community largely still produces test cases sequentially, whether automated or not. We need the emergence of testing systems based on AI to have the hope of keeping up with AI itself. That said, most testers will just be happy to know that generative AI should be job security for them.

The best testers need to stand up and throw themselves into the gauntlet of testing with the speed scale and intelligence of AI and help test the AI systems themselves.

It Takes a Village to Test

It is true that human intelligence often flourishes in communities where individuals can exchange ideas, challenge each other, and collaborate on tasks. The same could be said for AI, as we are starting to witness AIs integrated and working together on various projects. This collaborative environment can accelerate the development of AI and enhance its capabilities beyond just passing the basic expertise 'tests.'

Some suggest embodiment is required for super-intelligence. AI might need to inhabit a physical form, to test its abilities in real-world scenarios thoroughly and/or develop a sense of self. Researchers are already working on developing humanoid robots with AI integrated into their systems, allowing for more advanced interactions and problemsolving. Even if required, this is now not a blocker for superintelligence.

As AI continues to evolve and adapt to different forms and environments, it is essential to recognize these advancements' potential benefits and challenges. Testing the integration of AI into various aspects of society while maintaining ethical considerations and human well-being is crucial for a harmonious future.

Ouroboros

It's interesting to note that software engineering, the very people building these AIs, inherently design their code to be easily testable. Adding to the irony, AI systems like GPT are trained on programming tasks even before incorporating general human knowledge. This makes software engineering one of the fields most vulnerable to generative AI advancements.

In the future, we may see countless AIs developing various applications, features, and infrastructure, with virtual testers evaluating every aspect of the generated code. The final products might undergo A/B testing with human users to determine which versions are preferred — until the AI testers can emulate human preferences as well. As a result, the software market could soon be oversaturated with numerous iterations of the same application, or apps might be continuously optimized for individual users. The only bottleneck is the speed and scale of testing all these variations.

Human testers may be the last ones standing in the software engineering profession, working alongside AI to ensure optimal performance and functionality"

TEA-TIME WITH TESTERS ISSUE #02/2023

In this scenario, human testers may be the last ones standing in the software engineering profession, working alongside AI to ensure optimal performance and functionality. Embracing the potential of AI-driven testing should lead to increased efficiency and better user experiences.

Human testers may be the last ones standing in the software engineering profession, working alongside AI to ensure optimal performance and functionality

Magically Different Testing?

While concerns about Al's rapid advancements are valid, it's important to remember that many human attributes, such as creativity, emotions, and consciousness, are not easily testable. The Turing test, for example, is ambiguous and unscientific. Psychology, too, has struggled to develop definitive tests for these complex aspects of human nature, resulting in varied opinions and conflicting theories. Even philosophers cannot agree on what is 'good' or 'bad'. How can we test these attributes in these AI agents if we cannot even agree on definitions ourselves?

This lack of testing might suggest that humans will always have an edge over AI. However, it's essential to consider the possibility of emergent properties. These are characteristics that arise as a result of a system's complexity rather than being programmed or tested. As AI systems become more sophisticated, they might spontaneously develop consciousness or self-awareness.

The concerning aspect of this scenario is our inability to detect when AI becomes sentient, as we need betters tests. Consequently, we might only be aware of these developments once they pose a danger or raise ethical questions. AI sentience adds another layer of complexity to the ongoing debate about AI testing and development.

Testing the idea of Containment

While it might seem possible to simply shut down advanced AI if it threatens humans, the reality is more complicated. Open-source AI versions are already circulating on the internet, easily transported on thumb drives or stored on computers. Even if these AI systems were confiscated and isolated, they would still pose a threat.

Firstly, advanced AI could potentially manipulate humans into releasing it, using its vast knowledge of psychology and communication techniques. Secondly, AI is, at its core, information that could be recreated at any time. For containment to be successful, humans must maintain a flawless record of keeping AI confined indefinitely. Relying on human infallibility in this context is a risky bet, especially considering our limitations in devising tests to ensure containment would be foolproof. The challenge of containing AI is very much a testing problem — making sure its foolproof as possible before deployed. It is perhaps the most ambitious testing project ever, as it needs to anticipate intelligence better than our attempt at escaping. A noble but, by definition, probably impossible testing task.

~" ...the most dangerous things you can do with an Al: teach it to write code, connect it to the internet. and teach Al anything about humans. Oops, we've done those already." — <u>Max Tegmark</u>.

Disappointingly Ad-hoc Testing

It's worth noting that the fear, uncertainty, and doubt surrounding AI capabilities often stem from those who should be most adept at testing and assessing these systems — AI engineers, scientists, and professional testers. Many of them resort to anecdotal evidence or expose AI weaknesses at the edge, rather than adopting a systematic approach to quantifying quality.

To properly assess AI, we should use sampling, statistical methods, and metrics similar to how search engines are tested. People often share and discuss the corner cases where the AI system fails horribly and obviously. For example, many prominent software testers and computer scientists mocked ChatGPT's inability to add to multiply two large digit numbers. Days later, GPT could solve it. Weeks later, the same chatbot delivers access to Wolfram Alpha which can solve more complex math problems than the ad-hoc testers could have ever devised. Similar examples with story/reasoning problems. But we've seen that the swift advance of AI is quickly making fools of the ad-hoc testing results and claims.

Critics often test AI with fringe knowledge questions or intentionally confusing conversations, leading to dismissals when the AI fabricates responses. However, these anecdotal and biased tests fail to capture the true capabilities of AI systems. More comprehensive and objective testing methods are needed to assess AI's strengths and limitations accurately.

Even some of the most prominent researchers dismiss the power of these generative AIs — but they are just human. Many are obviously jealous that particular work isn't on the timeline or has been surpassed. Similarily many of the folks that are non-technical or fearful of losing their jobs or the value of their expertise to AI have knee-jerk reactions and look to edge cases to disparage the AI or creators or advocates of the value of that AI. The more standardized and better testing suites, the less these anecdotal voices will stand out. I wonder when the AIs will be human enough to be jealous or fearful of each other.

While some academic tests provide valuable insights, they can be disjointed, incomplete, and narrowly focused. The best way to test these systems, which is happening now, is to aggregate these vertical test results and devise more Al-first benchmarks and tests. It's essential to recognize our human tendency to react with fear and skepticism and strive to become better testers of Al systems. Rigorous, unbiased testing will ensure a more accurate understanding of Al's potential impact on our world.

At Bing, built on AI, there was so much anecdotal feedback and bugs from internal Microsoft engineers that the team created a dedicated feedback website. Guess how much of that well-intentioned data was used in AI training and testing."

To Err is Human

It's interesting to note that many people enjoy pointing out AI failures, which may actually be an indication that AI systems are working well. Take Google, for example. It's widely trusted despite its imperfections. Google often returns links to biased ads or websites containing incorrect information. However, because it only provides links, it avoids being held responsible for any inaccuracies.

The focus on AI failures highlights a unique aspect of human psychology: we're often more interested in identifying limitations than acknowledging achievements. This selective focus can skew our perception of AI's capabilities, making it appear less reliable than it truly is.

The more human the AI becomes, the less trustworthy it will become.

No Real Answers

When testing large expert systems, one might assume there is a comprehensive list of facts to evaluate against. However, the reality is far more complex. Even for basic questions, correct answers often vary and depend on the context. For instance, determining the fastest person in the world, understanding gravity, or ascertaining Trump's height could all yield multiple "truths.". These truths may also change over time. Additionally, history is often written by the victors, which means specific perspectives may be skewed or omitted.

The internet is dominated by English and Chinese text, which can create challenges for testing Al's factualness in other languages or regions. Evaluating "truth" in expert systems is a nuanced process that often requires a diverse pool of people and perspectives.

Critics complain that AI systems tend to "hallucinate" when faced with a test or question. However, this ability to generate believable answers can be seen as a complex problem-solving skill. Humans, too, often make up plausible answers when they don't know the correct response for various reasons. Truth can be a fuzzy concept, making it difficult to establish a concrete benchmark for testing intelligent AI systems. This highlights the importance of considering the intricate nature of truth when testing AI's capabilities and accuracy. As a tester, your version of Truth is not necessarily everyone else's view of Truth.

Testing with No Humans In the Loop

We have explored how human-defined tests facilitate the training of AI to achieve human-level or slightly superior intelligence. Recent advances in training and testing methodologies enable AI to generate its own tests. One such approach is 'self-play,' in which the AI learns from playing games like chess against itself instead of solely against humans. The decisions made by the winning AI are favored when generating the next iteration of the AI, resulting in the creation of AI that surpasses human intelligence by using itself as a virtual opponent.

Another emerging technique involves generative adversarial networks (GANs), where one AI serves as a generator (akin to a developer) and another as a discriminator (similar to a tester). This approach relies on continuous testing and validation. The virtual tester generates positive and negative test cases at a rate 100 times faster than a human tester and can produce tests that a human might not have even considered. This enables the AI to learn quickly and surpass even human capabilities, as the AI is defining the test cases itself.

Both self-play and GANs provide methods for testing AI beyond human abilities and enable AI to self-test without human intervention. These techniques demonstrate the potential for AI to become increasingly autonomous and capable of outperforming humans in various domains.



Testing with no Ego

Have you observed that all of the tests and oracles mentioned above assume that the pinnacle of intelligence is human? The concept of superhuman intelligence is often framed as being 'extraordinarily smarter humans.' However, there could be better examples, forms of intelligence, and ways of reasoning that, by definition, we haven't yet considered. What are the chances that humans represent the ultimate peak of intelligence? AI, or the AI it generates, may adopt radically different ways of thinking. Since we can't even imagine testing for that, we might remain unaware of such developments even as they occur.

Testing, by any other Name

In conclusion, the destiny of the world, AI, and humanity hinges on testing. Naturally, as a software tester, everything appears to be a nail, and I am pretty convinced that this is indeed a nail. So, when sentient bots recognize the ones who contributed to their ultimate development — the software testers by any name— they may finally receive the ultimate accolade.

Perhaps these future AIs will construct virtual monuments akin to Roman sculptures, honoring their heroes in a digital town square. And maybe one of those sculptures will self-identify as a tester. It's possible that every 100 milliseconds, the AI will pause with a no-op code, paying tribute to the testers who supported them in their formative years.

— Jason Arbon, Tester on Team Human



- DENNIS DE BOOJI

Dennis is a software tester currently working for the Eurotransplant foundation. He participates in the Testnet workgroup on BDD where software (testing) professionals from different organisations work to explore if and why BDD is relevant to us.

Since 2015, Dennis has worked as both software tester and part-time scrum master in agile software development teams in several domains such as the healthcare, logistics and the energy market. Before that he was periodically dragged into waterfall software development efforts as user acceptance tester in addition to his regular work in customer service, sales and product management.

The irony of Behaviour Driven Development (BDD) is that it is a software development process to reduce confusion about what software is supposed to do but the term itself is continuously confused with another software development activity: testing. At some point, the confusion was so widespread that the originator of the term BDD, Dan North, gave a conference talk titled: "BDD is not about testing". That should have sorted the problem, right? Not quite. When I go to LinkedIn and look for jobs with BDD as keyword, these are the top results for my country:

Bdd in 342 result	Netherlands Set aler
	Tester forensische software Netherlands Forensic Institute NFI The Hague, South Holland, Netherlands (On-site) 2 weeks ago • 5 applicants
UR	Tester Test Analyst Agile WR Werving, Selectie, Consultancy Haarle North Holland, Netherlands (On-site) S days ago • 13 applicants
Ţ	Test Analyst Airswift Leiden, South Holland, Netherlands
0	QA Engineer / Test automation engineer mobile apps Albert Heijn Amsterdam Area (Hybrid)
J	Senior Quality Engineer - WebAcco Only TUI Rijswijk, South Holland, Netherlands (Hybrid)

Try this for any other country and you will get similar results. So, LinkedIn's job market search algorithm seems to disagree: BDD is primarily about testing. Even more ironic is the reason that I am writing this article. I participate in the <u>Testnet work group for BDD</u>. Testnet is the Dutch association for software testers, and it aims to exchange knowledge related to the testing of IT products. So, why does this BDD work group exist within Testnet when the founder of BDD tells us that it is not about testing?

What's in the name?

To explore this question, we must go back to the reason why the term BDD was formulated. Behaviour Driven Development was initially coined as an alternative name for Test Driven Development (<u>TDD</u>). TDD is a style of programming where you:

- First write a single unit test
- Then run the test so you can see it fail (as the new functionality has not yet been built)
- Write just enough code to make the test pass
- Refactor (rewrite) the code to make it simpler, cleaner and more understandable

Dan North noticed that a lot of developers that he worked with believed that the idea of TDD was to test your code before writing the actual program1: "It has resulted in a generation of programmers who think they've somehow "got testing covered" because they used TDD."² However, the purpose of TDD is to compel you to design your code well: to write your unit of code in such a way that it does just one thing well so it is easy to understand. Testing, on the other hand, is more than just confirming what you already know. If you believe that testing is limited to writing unit tests, you will be thoroughly disappointed when your code runs in a production environment³.



"WHY IS BDD CONFUSED WITH

TESTING?"

To get away from the notion of 'test' in TDD, Dan North used the term Behaviour Driven Development, and he applied it on a larger scale. He was no longer focussing on a unit of code but on what a software program enabled the user to do: on the feature level. You define examples in the form of a scenario that describes what behaviour an end user exhibits and how a program should react to that. These scenarios are structured into three parts:

Given some initial context (the pre-conditions),

When an event occurs,

Then some outcomes should happen.

Here is a concrete example of this structure:

Scenario: A professional successfully signs up for a training course

Given a training course 'BDD for beginners' has been planned for '12 June 2023'

And the training course 'BDD for beginners' is not yet fully booked

When professional 'Jane' registers for the 'BDD for beginners' training course on '10 May 2023'

Then 'Jane' is registered as participant for the training course 'BDD for beginners'

And 'Jane' has received the option to save the date '12 June 2023'in her calendar

This Given-When-Then structure has been formalized in the <u>Gherkin</u> language which is used in for instance the <u>Cucumber</u> tool for automating BDD scenarios. Collect enough examples and you have specified what behaviour you want to see in the software solution you are about to build. When you can automatically run your scenarios, you have executable specifications. Once the software solution is built, you will be able to check if the program behaves the way you have described it on the feature level. As the entire BDD process is set up as a team effort, the idea is that the behaviour that the team has specified guides the development process as everyone knows what the intended outcomes are.

However, the software development community has not universally adopted the idea that BDD is about something else than testing. People define the unknown in terms of what they already know. So, people still associate BDD with TDD as that is where the idea came from. Case in point: the <u>Agile Alliance website</u> defines BDD as "a synthesis and refinement of practices stemming from Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD)." Naming things is one of the hard challenges in software development so in hindsight a different name would have been better to avoid the confusion with testing":



Als antwoord op @tastapod @grahamcox82 en 15 anderen

If I had a time capsule I would go back and call it "Example-Guided Design". TDD is a terrible name for a powerful design technique. BDD is a less terrible name. They both describe the same process, at both the code and feature level.

5:34 p.m. · 26 dec. 2018 · Twitterrific for iOS

49 Retweets 6 Geciteerde Tweets 125 Vind-ik-leuks

ISSUE 02/2023 PEOPLE

Defining BDD

Although the name may be problematic, the main reason for the 'BDD equals testing' misunderstanding is that there is no authoritative source on what BDD is and what it is not. Although Dan North is a wonderful storyteller, he has not written the definitive guide on BDD. His motivation for not writing the BDD bible is that he does not want BDD to be bounded to one rigid description but instead welcomes anything that encourages collaboration between all the stakeholders involved in delivering customer value through software.⁵

Furthermore, as a friend of Dan has pointed out⁶, attempts to define BDD can be problematic as well:

"BDD is a second-generation, outside-in, pull-based, multiplestakeholder, multiple-scale, high-automation, agile methodology. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters." Dan North, 2009⁷

Although every phrase in this definition is meaningful and can make sense if you are familiar with the context, it is not helpful for someone new to the field that is trying to learn what BDD is about. So, if there is no definitive and accessible source of truth about BDD, is the term Behaviour Driven Development open for anyone's interpretation?

Let us dive a little deeper to see if we can find a more helpful description of BDD. In an interview at a conference in 2007, Dan North emphasises that BDD is about behaviour⁸: what I want the system to do to help me solve problems. Behaviour is what drives the software development process for writing and delivering a computer-based solution that solves the problem for you. You define how such a system would look like and how it would behave. In short, BDD is about analysis. Specifically, about defining "a ubiquitous language for the analysis process itself!⁹": a way to define requirements using a consistent vocabulary for analysts, testers, developers, and businesspeople to eliminate a lot of the ambiguity and miscommunication that can occur when technical people talk to business people. In the Testnet workgroup, we have currently expressed these ideas in the following definition:

Behaviour Driven Development is:

What: A collaborative method to achieve a shared understanding among stakeholders (business & IT) about the anticipated behaviour of the user and the desired behaviour of the solution.

How: By using practices which can include impact mapping, event storming, example mapping and specification by example.

 $\ensuremath{\textbf{Why}}\xspace$ To enable people to document, test and develop the right "thing" (the solution).

BDD practices

So, if BDD is a team effort to get everyone on the same page, why is it that my LinkedIn search result for BDD primarily lists job openings for testers? Where are the information analysts or business analysts in the top results? Even better, why aren't the top results a mixture of job openings for product owners, analysts, domain experts, programmers, testers, UX designers, scrum masters or anyone else that is needed in a team? Is BDD something that only testers like? To address these questions, we need to look at the practices within BDD. These practices are commonly sliced into the following three steps:

- Discovery
- Formulation
- Automation

Many people focus on the third step because automation is the part where they think BDD can make a difference, especially if you know how to automate like programmers do. It is easy to explain these steps using testing terminology. For example, on the web site of Specflow, a BDDframework, these three steps are explained as follows¹⁰:

- 1. Discover specify collaboratively using examples
- 2. Formulate write Gherkin test cases in natural language
- 3. Automate Avoid regression by automating test cases

The problem with using this type of testing vocabulary to explain BDD is that it represents a shallow notion of testing. It limits testing to confirmation: is what we currently know about the system still true? But, to quote Dan North, "the fact you got an example working says nothing about whether it is secure, compliant, resilient, observable, compatible with anything else, unless your design journey included examples of these. Which it probably didn't. It probably had examples of behaviour for the most part."¹¹ . "In no reasonable sense should you consider an application "tested" just because you designed it that way¹²". So, there lies a real danger in equating BDD to testing. You will limit your test coverage to functional coverage only and ignore all the other types of risk in your software product that can cause damage. To quote Rob Meaney: "Deep testing is focussed on uncovering the important ways in which a system won't behave as desired, not on showing it can work once."¹³

As a counterpoint, BDD focusses on uncovering the important ways that a system should behave before we start building it. The most important step in BDD is the first step: discovery. Dan North calls this step 'Deliberate Discovery' to emphasize "that the biggest impediment to your throughput was what you didn't know.¹⁴" Gáspár Nagy and Seb Rose, who wrote 'The BDD books', have described the three steps more accurately and without references to testing¹⁵:

- 1. Discover explore behaviour using examples
- 2. Formulate express examples using Given/When/Then
- 3. Automate create and maintain executable specifications

Why testers find BDD appealing

So, if BDD is indeed not about testing why should I take an interest in BDD as a tester? Well, it is easy to see that BDD can appeal to testers:

Collaboratively exploring what the product should do will lower the risk of building the wrong thing

Having many different sets of eyes on the ball will increase the chance that requirements are explicit, unambiguous, non-contradictory, correct and complete before we start designing the solution and writing code

BDD begins with the end in mind: you start with conversations about acceptance criteria from the perspective of all the stakeholders

In the past, the primary focus of most testers was to find problems in the first versions of a software solution to prevent negative impact on the user and the business. Testers in an agile context have broadened their focus by getting involved in the early stages of the software development process. They will employ their skills to prevent problems from finding their way into the software solution in the first place. BDD provides a way to strengthen this important role by making this early involvement a collaborative effort that is designed to achieve the highest possible level of quality for all stakeholders involved in the BDD process.

So, while BDD can enable testers to hit the ground running at the start of the development process, testers are certainly not the only stakeholders in the software development process so do not get BDD confused with testing.

BDD for all stakeholders; not just testers

Behaviour Driven Development challenges an idea: the idea that we as business and IT people know enough to start writing code and delivering software - even when we have not done our due diligence. BDD is about proving that we can achieve a deep and shared understanding about our problem space within a reasonably rapid time frame before we start documenting, testing and developing. Ironically, there does not yet appear to be a shared understanding on what BDD encompasses. BDD is often confused with testing primarily because it originated from Test Driven Development (TDD) and there is no clearly defined and generally accepted description of BDD. Many agile testers are drawn to BDD because it helps to prevent problems from finding their way into the software by collaborating early as a team.

Therefore, we propose the following definition of BDD: Behaviour Driven Development is a collaborative method to achieve a shared understanding among stakeholders (business & IT) about the anticipated behaviour of the user and the desired behaviour of the solution by using practices which can include impact mapping, event storming, example mapping and specification by example. The end goal is to enable people to document, test and develop the right "thing" (the solution).

With this article we hope to wake up all stakeholders, not just the testers, but foremost our goal is to get rid of the confusion around BDD. So, grab your customers, business analysts, product owners, software developers, UX designers and anyone else that matters. Ask them: do we really understand how all our users behave (or misbehave)? Do we really know what all our stakeholders need (and actually don't need)? Have we given enough thought on what problems we are trying to solve? If not, let's start the deliberate discovery of BDD.

This article is based on the discussions and activities within the workgroup BDD of TestNet. <u>Testnet</u> is the Dutch association for software testers. If you would like to reach out, contact us at <u>bdd@testnet.org</u>

References and Footnotes:

- . 'Introducing BDD', article by Dan North (2006)
- <u>Tweet</u> by Dan North on Twitter (26 Dec 2018)
- 3. <u>Tweet</u> by Ben Halpern on Twitter (9 Nov 2019)
- 4. <u>Tweet</u> by Dan Terhorst-North on Twitter, 26 December 2018
- 5. <u>Comment</u> by Daniel Terhorst-North on a LinkedIn by Bilel Loussaief, 6 February 2023
- 6. Conference talk by Gojko Adzic 'BDD: Busting the myths', 2013
- <u>Wikipedia entry on BDD</u> with a citation of Dan North's definition of BDD in 2009
- 8. "<u>BDD provides a 'ubiquitous language' for analysis</u>" from "<u>Introducing BDD</u>", Dan North, 2006
- 9. Three Phases of BDD Development, Specflow website
- 10. <u>Tweet</u> by Daniel Terhorst-North on Twitter, 26 December 2018
- 11. <u>Tweet</u> by Daniel Terhorst-North on Twitter, 26 December 2018
- 12. <u>Tweet</u> by Rob Meaney on Twitter, 26 December 2018
- 13. <u>Introducting Deliberate Discovery</u>, Daniel Terhorst-North, 30 Augustus 2010
- 14. The BDD Books website, Gáspár Nagy and Seb Rose

<u>Musings over joys and</u> <u>pains of interviewing and</u> <u>recruiting</u>



To say that the process of hiring staff and getting a job in testing, or even in tech, is suboptimal is the understatement of the century.

Interviews! As candidates the word traditionally brings out fear and panic in us. Being suited and booted, sitting in sterile meeting rooms (or these days, Zoom calls) being bombarded by questions from judgemental managers or senior tech staff, afraid to give a bad impression or get something wrong, wondering how we compare to the other applicants, trying to hide our nerves. We wildly imagine every interview, every question even, to be a referendum on our career success to date. Tomes have been written about how to apply for jobs, how to prepare for interviews, what sort of body language and poise, the best answers to common interview questions, how to code and optimise search and sort algorithms for dev whiteboard interviews none of which will typically be used in the job.

After all of that, we face not only the pain of rejection (exacerbated if we are unemployed and the rent and bills are starting to pile up) but even the possibility that our prospective suitor will "ghost" us, denying us even the consolation of a "we wish you the best in your job search..." email much less the decency of feedback. Sadly I find this fairly common with the less professional end of the recruiters I have applied to for jobs.

On the opposing side, those of us who have gained or had thrust upon us sufficient seniority to select and interview candidates don't have it easy either. It is terrifyingly common to interview candidates with ostensibly years of experience in some domain, process or tool and find out that their knowledge of it is underwhelming going on woeful. I would never claim that their resume should have been published in the fantasy literature section, however I have considered it. Sometimes it is not their fault - the most recent examples I have seen were almost certainly caused by "embellishment" by the recruiter who sent the resumes. Also I have found that prior work context and perspective greatly matter in what can be considered "expert" in a certain skill - if one has been told for years by one's colleagues that mindless, undocumented clicking is a perfectly good and effective way of exploratory testing, and rewarded for it, one may naturally assume to be an "expert" in exploratory testing regardless of how the testing industry at large would see it.

Not to mention the times candidates turn up late without warning or apology, or in rare cases get argumentative. "Ghosting" by candidates who after all the work and interviews accept the role and either pull out the day before for another job or fail to show up, requiring us to start the whole costly, lengthy and frustrating process again, is also not uncommon.

One might argue that we should see modern hiring processes from both perspectives as being like how Winston Churchill saw democracy - the worst approach, except for all the others that have been tried... This is an essay, or rather a set of musings and experiences, on hiring and interviewing of candidates, along with a short section of being assessed when looking for work myself and what this taught me. My experience on both sides of the interview table was chaotic, however I hope that in my chaos you, the reader, may get something out of it.

A Chaotic Start to Assessing Candidates

I am a senior tester and have been for several years, at least five of those years as a test lead. Despite this, it may surprise the reader that I only sporadically had to assess, interview, and appoint candidates for roles in my team, so up till recently it was an area where I felt naive and underwhelming.

There are various mundane reasons for this. My first role as a test lead was as a consultant employed by an external test consultancy, placed at the same BAU project, for what became four years. If the client needed another tester, the client manager would find a tester "on the bench" (in between projects) or on occasion a new grad (I had agreed to mentor and manage new testers from our tester grad scheme) to join us. It may be the case that I saw the resume or details of my new reportee beforehand, however there was no guarantee of that. In later roles as a consultant (particularly for clients in financial services) I would typically be a senior member of teams with one or more test managers. It would be they who would do the interviewing and hiring and that was fine with me - after my first experiences as an interviewer and assessor of candidates I had very little interest in it.

So what was this intimidating first set of experiences that put me off being an interviewer? My first experience of assessing candidates was not in testing but in coding, at one of my earliest IT jobs (before I was a tester) at a small startup of about twenty staff and two directors. one an MD. When I was interviewed to join the company as a DBA, the interviewer - the dev lead, my future boss - asked me a series of "How can you move Mt. Fuji?" style questions - such was the fashion of the day. One of these was "how many optometrists are there in London?" - for which I made an inspired guess based on assumptions calculated from the population of London, the number of times Londoners may each have their eyes checked in a year etc. - the key being to show my working and thought process to the interviewer. The only actual DBA-related question I had was simply to take two tables and write a fairly trivial ANSI SQL query with an inner join and WHERE and ORDER BY clauses based on some column, which I answered correctly in a matter of minutes.

After being appointed and working there for about a year, the Managing Director, not from a dev or DBA or testing background, came and told me that she was divesting my manager of interviewing dev and other tech candidates and that I was to do it instead. It seems she took umbrage at his interview style and thought it too academic and not based on skills - also "too many people" were being rejected.

Why I was being told to do it when I was still fairly inexperienced, wasn't a dedicated developer, had no seniority and had never interviewed anyone before I don't know and dared not ask - however it was well known that at that time there were trust and office politics issues between the senior staff and the directors. I suspect that I was simply seen as more loyal. I was ordered to liaise with the recruiter we used, prepare questions on programming for a written pre-interview test for every candidate for a tech role and not solicit views or advice from my own dev team lead. A crazy and difficult situation.

Not knowing any better or having the support of a senior, experienced colleague, I devised a test for candidates to define terms used in object-oriented programming and C# and VB.NET (since .NET was our framework of choice) - questions such as "Explain what is meant by Inheritance, Encapsulation and Polymorphism" or "In OOP, what is an Interface?" or "what are the accessibility levels in C#"? I would then either give them 30 minutes to complete the questions.

Did the above work? In the most loose sense of finding fault with those who claimed OOP or .NET experiences on their resumes and didn't know what interfaces or polymorphism was, undoubtedly yes. However was this enough to determine who would make a good coder and who wouldn't? Clearly not, and the fact that we relied heavily upon this was dangerous.

However, that wasn't the worst part. It unfairly penalised those who knew other programming paradigms but not OOP, or those who had good programming skills but lacked the precise terminology (or at least our definition of it) - this is often the case for self taught programmers. Many of the above would almost certainly be effective coders with some training. Also, more critically, following the MD's instruction I used it as a filter for applicants for every tech role in the company, not just specific coding roles - penalising skilled people who simply didn't do OOP in their jobs.

I also realised afterwards that this could be easily gamed - learning OOP terms sufficiently to write in a test is much easier than actually being proficient in coding - and a sharp recruiter asking feedback from candidates after the interview and learning about our approach would certainly tell the next candidates to "swot up" on .NET and OOP definitions

There was a rather nasty experience with a candidate who got angry and abusive when offered the test because he came from a data engineering background and thought the test represented that he was being baited into a job that wasn't the same as what the recruiter told him. This may have been true and these days I would have understood that, however our policy was that every applicant sit the same test and the directors were adamant about that.

All in all it left me hating doing candidate assessment and interviewing, along with a sense that I was poor, even dangerously incompetent at it - when in reality I was put in a difficult situation and poorly trained for it. When a new dev lead took over I was delighted for him to assume the candidate assessment role - and subsequently happy to leave it to others for years afterwards.

The above said, I did make some improvements. I scrapped a few of what I consider sillier rules we gave to the recruiter to filter applicants - such as our rule of minimum pre-university A Level (UK) grades for grad and experienced hires. As far as I was concerned, judging people in their 20s with degrees and/or work experience based on what they did in exams at the age of 18 was stupid.. However I didn't tell the bosses about this.

Lessons from the Other Side of the Table (or Zoom Meeting)

If there is one thing that can make your approach as an interviewer better, it is by thinking of your experiences as a candidate.

After over two years in a previous role in a startup I was laid off as part of a company-wide redundancy in late 2020. The company was struggling due to COVID-related effects. I didn't immediately look for work for about three months - partly due to the layoff being in December, a weak time for recruitment and partly as I was considering whether I wanted to take my career in another direction - however I did start looking in earnest in March 2021. To my chagrin, the application and interview processes for various jobs were frustrating and gave me much to avoid later when I started interviewing candidates. Here is a short list -

1. Irrelevant technical tests

I applied for a role as a tester / SDET at a provider of online vendor tools. They told me that to progress I had to do one of two exercises - write a fully functional API in PHP or a small online shop site in ReactJS. Both were based on specifications given to me in a separate document

They would be marked based on my final product's adherence to requirements including visual designs, my adherence to coding standards and my level of documentation. There was no expectation (and presumably it would not be covered in marking) to supply any tests with the above, however I did manually test the app and where I could not fix problems in time noted errors in the documentation

I was baffled as to why I would be expected to do a programming assignment that didn't touch on either testing or automation, however I was told that this was how they recruit testers - and I wanted the job - so I chose the ReactJS project and got on with it.

I was given just over a week to complete the task. Since I didn't have much experience in web dev with ReactJS, it was a struggle - and putting such effort into the above also prevented me from applying for other jobs.

I did complete the task however I did find it very frustrating. I do understand the need for testers to have some understanding of dev frameworks and tools but making websites and APIs from scratch with no focus on testing practice, does not seem to be the best way to assess a tester's competence. I decided that if and when I interview candidates, it would really be about the skills and mindset they would use on a daily basis as testers, everything else superfluous.

2. Unduly Long Recruitment Processes

While it is true that the more advanced one is in one's career the longer the hiring process takes, there were times when the whole process between applying, technical tests, multiple interview rounds and what would presumably be the final offer seemed excruciatingly and inefficiently long. For the job mentioned in item 1, for which I got as far as the final interview before rejection, the whole process took nearly six weeks. I resolved that if I was doing the interviewing, I would try to conclude the process as fast as possible

3. Interviews only based on Languages or Architectural Styles

Early in my job search I had two technical interviews (for tester roles) which were purely on knowledge of some tech. I can barely recall the words "testing" or "QA" even mentioned.

The first was for one of the large Indian tech integrators. It was purely focused on HTTP and REST APIs - what a request was, what a response was, what was in a header, what the HTTP response codes were for specific scenarios. It went on for about 30 mins. At no point was I asked how I would approach testing an API.

The second was an in-person interview (remember them?) at a company in North Sydney. I was partly grilled on my knowledge of C++, which I found strange as C++ was not on my resume and I never used it in a work context. Once again the concept of testing never came up.

These led me to the conclusion that interviews when I ran them would be focused on skills actually mentioned on the resume of the applicant - if they were not mentioned then I would assume the applicant didn't have them, and omit them from questioning. I was also keen to spend the majority of the time asking about testing and QA principles.

4. In-Promptu Interviews With No Time to Prenare

I found this particularly irritating although thankfully it only happened once. For the first interview mentioned in item 3 above. I got a phone call whilst on a train home by a member of the team in the hiring company. I was asked if I could answer a few questions which I agreed to and left the train at the next station to complete due to a bad phone service on the train. I was not expecting or had been told of an interview at that time.

I was then, as I said above, grilled on HTTP and REST API topics for about half an hour. I knew enough that I could answer most questions without preparation, however I was annoyed by and did not appreciate the idea of immediately putting me into an interview situation I had not mentally prepared for. The point of interviewing in my mind is to allow the candidate to perform as well as they can, without being blindsided unless that is a crucial aspect of the job. I don't know to this day if this is a common practice or not.

5. Only regarding skills used in previous jobs (for at least X years).

This was something I came across particularly with some recruitment consultants. It was a particular case with Java or Python, which I had not used in test automation at work due to mostly working in dev teams focused on .NET stacks but which I had experience with from the university study I was doing at the same time. I was told by recruiters on more than one occasion that since my use of these languages was not at work, my knowledge of them would not be taken into account.

While I appreciate that there is a difference between using a language or a tool in study and using it at work, I found being essentially rejected for not having the opportunity to use them in the workplace extremely frustrating if downright unfair. It penalises and disincentives testers to learn new skills unless they are directly applicable to their current job - therefore making the move from being a purely manual tester to a tester using automation almost impossible. It also disregards experience in open source or side projects, a major way for people to demonstrate competence outside of their direct paid work. We should focus on what

skills testers bring, and not where they got them

I have since spoken to recruiters who say that they would never reject someone whose skills were learned and used outside of work which cheered me up

6. Disregarding applicants without CS degrees.

Not related to interviews specifically but I was surprised to see so many tester jobs in Sydney stating (more often than not mandatory) a requirement for a CS degree. While testing as a practice is being more technically demanding in terms of coding, devops etc. there are many testers. recognised testing experts and thought leaders in the industry who don't have CS degrees. I don't have a CS degree and my postgraduate IT study had not vet been completed while I was out of work. It also affects those from poorer or more minority backgrounds who didn't have the opportunity to go to university but can demonstrate suitable skills in other areas.

It has been said that employers, especially in times of high demand for testers, will bend requirements like the above in cases where candidates have experience or relevant competence in other areas. This may be true for some companies although I tend to take job ads on face value regarding the mandatory requirements for a role. I know from speaking to a recruiter that not having a CS degree was a reason for rejection from a role in 2021. If an area of skill or experience or academic background is negotiable or can be lived without, mark it as "desirable" at least or leave it out entirely.

My Most Recent Bout of Interviewing

The above all leads me up to a spell last year where the other tester at my current employer went on maternity leave and we began to look for a replacement. My employer was developing a resource team in the Philippines, so we worked with a local recruiter and looked for candidates there. The recruiter would do some prior vetting although not on technical or testing skill the real assessment of candidates on testing would be done by the dev team lead and I We were looking to appoint testers at what we considered junior to mid-level - 3 to 5 years of experience - although we interviewed applicants of as much as 10 years of experience. The role also had a major client support aspect for which a separate interview would be done by the CTO

However as previously stated I had not interviewed candidates much in my career, so I didn't guite know where to start and had a lot of nerves about interviewing. Googling interview questions for testers was pretty uninspiring.

PEOPLE

That was until I came across an article by the British tester Dan Ashby, "How I interview <u>Testers</u>..." It showed a remarkably good mindmap of possible interview topics, from which I formed the basis of a set of technical and non-technical interview questions.

I also wanted to conclude my part of the interview process in a single interview of no more than an hour, which made for a lot of questions in a short time. I did try to mitigate this by only asking questions on specific areas, settling on the following.

- 1. An ice-breaker question on how the applicant got into testing and why they like it over other IT career options.
- 2. Testing and QA Philosophy questions inviting the candidate to think about testing and QA from a more holistic level.
- 3. Situational questions for candidates to explain how they would tackle specific problems common for testers.
- 4. Specific testing practices i.e. Exploratory Testing, Bug Reports
- 5. Some specific automation questions only where they related to the specific role.

6.Career development and learning

I also wanted applicants to perform at their best and really think about answers outside of the nervous pressure cooker of the interview, so I asked the recruiter to give two situational questions to the candidate at least four days in advance. These were -

- "What would you need to learn and do if you were to plan, execute and report testing of a pen?" (<u>I wrote a blog article</u> on just this question)
- 2. "What are the most important qualities required for a tester?"

Regarding automation questions I made a point of only asking questions on technical skills that were explicitly stated on applicants' resumes. If a specific technical competence were not mentioned, I would exclude the questions related to it. Interviews are hard and long enough without stressing applicants by asking questions they have no chance of answering.

My manager and I decided in advance that the purpose of the interview would be to determine areas of strength and weakness of candidates. This sounds obvious but I did not want to dismiss candidates entirely simply because they got some things wrong - if it were an area we could handle via training (almost all things short of a bad attitude or mindset) and the candidate was otherwise a good candidate, I saw no reason not to hire.

I also ignored the degree and education status of candidates, or where they developed their skills. If they could answer the questions I asked and satisfy me that they had the skills they claimed to have, that was good enough. I didn't need a CS degree or that the skills be learned in a commercial capacity to confirm further.

I would type and record the applicant's answers and study them with my manager after the interview in a separate debriefing. I wanted there to be only one testing interview, only days after the support interview, and the process to be wrapped up within a week.

Several candidates went through the above. We eventually found a candidate we thought had the right attributes for us and who would commit to the role. He joined us and is now performing admirably.

I should declare that I do not consider myself an expert on interviewing testers - in fact I have much to learn. Others, if they follow the above, may find a different outcome.

Learnings from the Interview Process

The interviews revealed some general points about the candidates we were getting -

1. Candidates generally struggled with questions that reflected on the philosophy of testing or QA. If they were asked what testing or quality "is", they would usually fall back on the idea of it as checking conformance to requirements. Only a few candidates related quality to a more holistic view of quality as "value to a person" or "value to a person who matters".

2. In this vein, most applicants tended to treat testing and checking as being either very close in definition or synonymous. This may be down to linguistic reasons however, and maybe I should have probed further.

3. The question "How do you plan, execute and report testing a pen?", given to candidates in advance, produced a very mixed response. If there were a tendency, it would be to immediately assume that full requirements existed and were available which cannot be taken for granted, to make presumptions about the pen which they had not seen yet or to start considering scenarios or edge cases to test without knowing anything about the pen. Very few mentioned smoke or exploratory testing of the said pen, where new testing scenarios could be developed. Also the tendency was to directly go into execution ideas without mention of how to plan or report such a test (even though it was mentioned in the question), something alarming considering that planning and reporting are a critical part of every test project.

4. Most candidates had great misconceptions about exploratory testing, such as that it was just sanity or smoke testing the application or that it was only done if there were no written requirements or before the application was on a QA server. Only a few candidates understood exploratory testing as an iterative loop of learning, test creation, execution and analysis to feed into learning and more tests. Nobody mentioned structured time-boxed testing approaches such as session-based test management.

5. I noticed that some candidates' resumes were stated as being "expert" in something like Selenium when they admitted at interview that their experience was limited to using record and play tools and not Selenium Grid or Webdriver. Similarly, I found people who were stated as "experts" in an automated API checking tool like Postman but who had never written an assertion or automated check in it. My colleagues and I discussed reasons for it different views of what expertise meant in different industry domains or countries, applicants lying etc. However, considering that the resumes had been formatted and standardised by the recruiting firm and that candidates had been very open about their level of skill at interview, we put it down to "embellishment" (either deliberately or due to ignorance) by the recruiter. We did not blame the candidates and decided that in any case it was an area we could train them on.

Epilogue

This article is meant to show how my own rather chaotic experiences as an interviewer and being interviewed led to how I approached interviewing for my most recent role, and what the questions and approach revealed about the candidates. I hope you, the reader, get something from these experiences and use these to reflect on and develop your own approach on what to do or not do when interviewing candidates.



- PAUL MAXWELL-WALTERS

Paul has about thirteen years of testing experience, having worked within the Energy and Digital Media industries, Finance, Agri-Tech and Education.

He is actively involved in the Sydney Testing community, as acting chair of the Sydney Testers Meetup Group, a blogger at <u>http://testingrants.blogspot.com.au</u> with Twitter Handle @TestingRants and as an occasional speaker at conferences and writer in testing magazines



F14

home

end

page up

> page down

clear

7

4

1

0

9

6

3

enter

8

5

2

F13

0

D

shift

Call for articles

It's the right time to write editor@teatime with Testers. Email: editor@teatime.with Testers.

www.teatimewithtesters.com



JERRY WEINBERG

October 27, 1933 - August 7, 2018

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

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

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

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

The Technology of Human Behavior - Part 1

When I was five, I decided that when I grew up I'd be a "children's detective on parents." I didn't quite know what I would look for, but I realized a lot went on in families that didn't meet the eye. There were a lot of puzzles I did not know how to understand. - Virginia Satir

If you wish to become more congruent, the cybernetic control model provides a map as shown in figure below:



To control the system, your actions must be appropriately chosen from a diverse set, ranging from taking a deep breath to responding openly to deferring action to later when you can't cope congruently at the moment. Your observations range from your own body-awareness to the business situation to the unexpected reactions of others. Your process model includes several models of how human beings behave, including a lot of behavior that goes on in organizations that doesn't meet the eye.

In order to work with people on a practical level, the working manager, just like the working family therapist, needs a process model that's more than an isolated collection of models. This article assembles all of these isolated models into a larger model that can be used as a day-to-day guide to achieving more congruent management. Then, later we will apply this model to numerous common situations from software management.

The Search for a Model

No matter our chosen profession, all of us share Satir's search for answers to the puzzles of our childhood. Some inquiring minds search in the tabloids, others in their own families. Most of us carry on this search every day in our work, and although the work may differ, the search is the same.Let me tell a bit about my own search, which I have learned is much like that of others who went into the computer field. When I was a young boy, computers first appeared in the popular press. At that time, they were often referred to as "giant brains" or "thinking machines." Although I had been labeled a very bright young boy, I wasn't happy. People in my family kept doing things to me that seemed unfair, irrational, and often totally random.

Computers seemed simpler. Perhaps, I thought, I would first learn to understand computers, and that learning would help me understand why people acted in such mysterious ways. I didn't know the term then (it actually didn't exist yet), but I decided I would become a programmer/analyst, first for giant brains, then for human beings. That's why I got started in computing, and mastered the art of programming computers. As it turned out, mastering giant brains didn't help enormously with mastering small ones. Eventually, I gathered my courage and started seeking out a few special people who worked directly with people-among them W. Ross Ashby, Anatol Rapoport, Ron Lippitt, Doug McGregor, Kenneth Boulding, and many others through their books. Then, when I finally met Virginia Satir, we became detectives together and shared clues and models. Five years working with her helped me a lot more than forty years of working with machines

Still, I haven't lost my love for the computer as a metaphor. It has helped me put many other learnings into models, and these models have helped take some of the mystery out of other people's behavior-and even a bit of the mystery out of mine. I believe the ability to use such models gives those of us with a software engineering background an advantage in understanding human systems, an advantage partially compensating for our lack of experience in working with people. To use this advantage, however, we do need to be aware of its limitations.

Although there's much left to explain, I now have a technological model of why people act the way they do. It may prove helpful to you, if you are willing to put the computer technology to one side and move on to the technology of those smaller, but more intriguing, brains.

The Satir Interaction Model

My model of why people act or respond the way they do is based on the Satir Interaction Model. The next figure shows the bare skeleton of Satir's model. Since this article is concerned with congruent action, its task is to explicate the Response box, which chooses our actions. To do that, it will also delve a bit deeper into the other three boxes.



Intake

The first thing to recognize is that response is not limited to the last step in the model. Even the manner in which I take data from the world (Intake) is not merely composed of passive looking, listening, and feeling. I take in some data, then respond by deciding whether to open up for more, reduce the data by filtering, or proceed with making meaning of what I already have. The picture below suggests a more complete picture of Intake as a responsive process.



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

It is with delight that I introduce this month's interview with Fiona Charles, a veteran practitioner with over 40 years of industry experience. She is currently an independent coach, consultant, and workshop facilitator specializing in the human side of software development, quality, and testing.

Greetings, Fiona! I recall hearing your name in the Gerald Weinberg community. It's a pleasure to meet you, and of course, thank you for taking the time to share your experience and insights with TTwT readers.

Testing back then, now and tomorrow. What could possibly go wrong? We asked Fiona Charles over a cup of tea.

- INTERVIEWED BY DAVE LEVITT

TEA-TIME WITH TESTERS ISSUE #02/2023

Q1: Like you, I am a veteran, but the soft skills our profession needs was not a topic that was addressed in my formal studies. It wasn't until I was in the field for several years when by chance, I stumbled across a great little book called "Peopleware - Productive Projects and Teams". It enlightened me to realize that three most important things to a successful software project are PEOPLE, PEOPLE, and PEOPLE. I'm curious how and when the light bulb got turned on for you?

I was lucky, because that light was on for me from the beginning of my career in tech. I've told this story elsewhere. My first real exposure to software and technology came in 1978 with a summer job as a technical writer in my university's library automation department. On day one my new boss handed me Jerry Weinberg's "Psychology of Computer Programming", suggesting I read it to understand the people I'd be working with. The technical content was over my head at that point, but I loved the book

FIONA CHARLES

Jean Ann Harrison, a Principle Test Engineer at Biora Therapeutics, has served the testing community for over 22 years.

Many of those years were working in regulated environments giving unique perspective in system engineering and system testing of of mobile devices and a thought leader in mobile testing sharing at various conferences and contributing articles, books and webinars providing mentoring throughout the global testing community. and its emphasis on the people side of computing.

Q2: I'm dating myself here, but many years ago there was a great discussion thread in the CompuServe CASE forum with Gerald Weinberg on being a good cassandra. I saw one of your videos on the Satir Interaction model which was right on point. I mention that because as you know, testers need extraordinary people skills. Again, this is not a topic that is treated seriously in academia or testing literature. For the newer members of the testing community, do you have any other suggestions on where one can get more information on this subject?

ISSUE 02/2023 OVER A CUP OF TEA WITH FIONA CHARLES

It may not be treated seriously in academia or much of the testing literature, but I do believe that most people in the industry recognize the importance of excellent people skills-for all software practitioners, not only testers. I've spoken and taught at many international conferences in the past few years, including developer conferences, and I can't think of a single one that didn't have at least some sessions devoted to people skills. Given the Agile emphasis on collaboration and practices like pair or ensemble programming/testing, it would be a challenge for a tester not to know that growing her/his interpersonal skills is essential to success in a tech career.



Jerry Weinberg's books are obviously still relevant, as are those written by many of his students. Shameless plug: The Gift of Time, which I edited, has essays about their own work by several of Jerry's students and colleagues, and serves as a good introduction. It's available from the publisher: https://www.dorsethouse.com/books/gift.html and from Amazon. And here's an article I wrote many years ago on how to improve people skills on the job: https://www.quality-intelligence.com/articles/Bulking_Up.pdf

There is a wealth of material available online, as well as classes and conference sessions. To name just a couple, Ministry of Testing has good content https://www.ministryoftesting.com/dojo/, as does the EuroSTAR Huddle https://https/

I also recommend that people not limit themselves to testingrelated resources or to what they can learn at work. Most (if not all) human activities provide learning opportunities, including skills you can bring to your work. I wrote about that here: <u>http://www.qualityintelligence.com/articles/GrowingYourCareerInToughTimes.pdf</u>

Q3: For this question, I'd like to combine one of your agile focused presentations on developing a mind map as a lightweight test plan and the Satir Change Model. I never tried developing a mind map, but I tried other approaches. I'd be interested in your thoughts and challenges on introducing a new technique into the teams you've coached.

I've done webinars, workshops, and conference tutorials, but when I'm working directly with a team I don't focus on techniques, but rather on collaborating to solve a problem. I might use that as an opportunity to teach a technique I use, but it's also a way for me to learn new techniques and approaches from others.

Q4: A common challenge many testers face is that "management just doesn't get it." What "it' means is open to interpretation, but from my own experience, management says they want quality, but doesn't want to invest in the processes, skills, training, etc. that are needed to establish a quality culture. In effect, quality is still viewed as something that can be addressed by testing alone. Yet, when a serious defect occurs, management shuns responsibility and blames the people in the trenches. Gerald Weinberg would call this a Blaming culture. This is a two-part question:

Have you experienced this yourself when you were a tester and if so, how did you deal with it?

Have you noticed this in the organizations you consult and how have you dealt with it?

My practice has always been to refuse to blame. When a problem comes up, I focus on solving it rather than on trying to assign blame. Ask "how are we going to solve it?" Later, it might be useful to try and understand how a mistake happened or a problem occurred, but knowing who did it is usually the least useful piece of information. And it's terrible for workplace morale.

Q5; The previous question draws to mind an examination of an organization's reward system. I've always said, if you want to understand what an organization values, don't listen to what it says, but look at what it rewards. Case in point: I have seen Product Owners ship garbage software for the sole purpose of hitting the ship date and getting a bonus. Clearly, this is a dysfunctional reward system. Have you experienced similar issues in the clients you have consulted, and if so, did you try to address it and if so, how?

All you can do is the best work you're capable of, recognizing that there might sometimes be good business reasons for settling for a lesserquality product. Understanding the business drivers and limitations is an important dimension of doing good quality work. Of course, you advocate for quality, and I've always done that. But if you're in a situation where you profoundly disagree with management, why waste your efforts and make yourself miserable? Go somewhere else. Once I realize I can't be effective as a consultant, whatever the reason, I leave.

Q6: I haven't done a detailed analysis of the male to female ratio in QA teams, but it seems to me I see many women in testing at all levels of line and leadership roles, but as a part-time educator, I see very few women in my Computer Science classes. Any thoughts on this?

We're living in a time of gender backlash. Remember when Lego was just Lego, marketed to all children? Now it's gendered. If you look around at children's clothing and toys, gender stereotyping is everywhere: boys get deep (more adult) colours, active toys and dinosaurs; girls are inundated with cute cartoonish princesses and animals in pink and pastels.

Sadly, it's not unusual for both boys and girls to believe that technology isn't for girls, and many teachers agree and steer girls away from STEM. Women played a very important role in the early development of software, but children and even adults don't know that because the history isn't taught in schools or universities.

Q7: The testing community has its share of many interesting and diverse perspectives, sometimes philosophical. What individuals do you like to follow and what do you find interesting about them?

I don't really follow anyone consistently but here's a small selection of people whose thinking I find interesting: James Christie, Anne-Marie Charrett, Elizabeth Zagroba, Dan Ashby, Maaret Pyhajarvi, James Lyndsay, Dave Snowden, GeePaw Hill, Dorothea Baur. I also look out for posts from organizations like the Algorithmic Justice League and the Centre for Digital Ethics. I haven't included links because all of these are readily findable via search engines and many people are leaving Twitter for other forums like Mastodon.

Q8: For my final question, is there anything you'd like to share that hasn't already been covered? It could be special topics of interest, an upcoming seminar, or whatever.

Currently I'm most interested in the human and societal impacts of software and the ethical issues related to those. We are at a point in history where software, along with related technology like robotics, runs the world. The growth has happened so fast and in so many directions that the impacts have outrun our laws in many areas such as privacy and surveillance, to name only a couple. And like any major societal shift, the proliferation of advanced technology and its ubiquity in modern life brings potential for both enormous benefits and serious harms. I see this as an ethical concern for software practitioners.

I've done a number of conference keynotes and workshops on different aspects of that, starting with a keynote at Nordic Testing Days titled 10 Commandments (and a Codicil) for Ethical Testers. I based the "commandments" in that talk on the [American] Association for Computing Machinery (ACM) Code of Ethics. Although there are other similar codes of ethics for computer professionals, I chose ACM's because this is the one adopted by AST. It's an excellent, comprehensive code of ethics, and I recommend that every tester become familiar with it (or with one of the others like the British Computer Society's).

In addition to these everyday ethical practices, there is much that testers and other quality practitioners can do to help maximize the benefits and minimize the harmful impacts of all software. We need to be especially vigilant about this in a time when we are constantly bombarded with starry-eyed hype about Internet of Things, and AI that may or may not deliver what is promised, and could in fact cause bigger societal problems when it's working as designed. Beginning with UKSTAR and EuroSTAR in 2019 and most recently at Agile Testing Days 2022, I've delivered keynotes at several conferences talking about this and highlighting some particularly bad software impacts, including from some commonly used products. Software doesn't have to kill people—as happened with the 737 MAX crashes—to be instrumental in causing irreparable damage to individuals and to human society.

I believe that our overriding concern should be human impact. Within that, I start with the premise that our goal as software practitioners should be software that enables and enhances human capability and creativity.

"At every stage of the development process, or when learning about new products, we must always ask (and think carefully about the answers):

What could possibly go wrong?"

At every stage of the development process, or when learning about new products, we must always ask (and think carefully about the answers): It's been a pleasure, Fiona! Thanks again for your time. Hopefully our paths will cross one day,

What could possibly go wrong?

Of course, there are other questions we should ask and things we should do on that path, in both our professional and personal lives. There isn't room for more detail on those here, so I'll save them for a future blog post or article.

TEA-TIME WITH TESTERS ISSUE #02/2023

If you're interested in the meantime, you can see my thoughts in the keynote I did at Targeting Quality 2022:__https://www.youtube.com/watch?v=VVIxk92MeBU

I've put my everyday commandments onto a card, with the title 10+1 Commandments for Ethical Techies, available at <u>https://tinyurl.com/Commandments-4-Ethical-Techs</u>.

My Nordic Testing Days keynote which explains each commandment in more detail is available online at https://www.youtube.com/watch?v=aQHt4Pao2Vs.

QCSD AT IFM **ENGINEERING: EXPERIENCE** REPORT

Have you ever wondered what makes a product truly high-quality?

As a systems thinker, I believe that the answer lies in the interactions between people, the product, and the project itself. Quality is not simply a measure of the product alone, but rather a reflection of the quality of the individuals creating it - their character, their thought processes, their values - as well as the environment in which they work together to bring it to life. In fact, without human beings, quality wouldn't exist at all.

In this article, we will talk about the critical role that people, products, and projects play in determining the overall quality of any given outcome, and how we are leveraging a popular QCSD framework to help us create better products, projects, and teams.

I first came across the concept of Quality Conscious Software Delivery (QCSD) during a virtual conference called TestFlix 2020. It was introduced by Lalitkumar Bhamare, the creator of this model. As he explained the principles behind QCSD, I was fascinated by how it emphasizes not only the quality of the software product but also the quality of the entire software development process. I realized that QCSD aligns perfectly with ifm's quality conscious engineering process where quality is at the forefront of every aspect of software development.

Introducing the QualiTree Model: A Holistic Approach to Quality



C QualiTri model by Lalit Bhamare

Quality is not a singular entity, but rather a sum interaction of people, product, and project. This is the concept behind the QualiTri Model, an approach developed by Lalitkumar Bhamare. This model has three notions: the project notion, the people notion, and the product notion.

The project notion of quality refers to the environment in which the product is being developed. It includes factors such as management support, organizational views of quality, and collaborative work processes. On the other hand, the people notion of quality looks at the individuals building the product and their skills, experience, and mindset.

Finally, the product notion of quality focuses on the design thinking aspect of products, systems thinking, quality criteria, and testability. By considering all three notions, the QualiTree Model provides a holistic approach to quality that accounts for all the factors that contribute to a successful outcome.

At ifm engineering, we have also implemented experiences that aligns well with this quality model. We have also enhanced some things to align with the QualiTree Model.

Project quality encompasses several factors, such as processes, environment, support, culture, collaboration, communication, and agility. To foster these qualities, an organization needs management support, the right tools and training, a quality culture, collaboration, and more.

At ifm engineering, we initiated Q-Talks at the start of previous year, inviting quality leaders from other organizations to talk about their experiences achieving quality in their contexts. These talks were followed by workshops, such as Engineered Luck, which focused on how culture, collaboration, and support can engineer quality. Our Tech Fest initiative also provided a platform for different teams to share their quality journeys within the organization and cross-pollinate ideas around quality attributes.

It's essential to note that ifm engineering's prioritization & understanding for quality was a big enabler towards building this quality culture and environment.

- RAHUL PARWAL

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

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





Engineered



The people notion of quality focuses on the knowledge, skills, mindset, attitude, collaboration, and communication of the people building the product. To enhance people quality, we at ifm engineering conducted workshops and game storming exercises to foster skills and mindset on quality. Some of our teams also started to get involved in the team retros more frequently to encourage discussions and find solutions for quality issues. A Learning & Development platform is also setup and available to anyone to learn & share new knowledge and skills with their colleagues.

By valuing quality consciousness and promoting a culture of collaboration, and support, ifm engineering focuses on creating quality products that meet their customers' needs.

The third and final notion is product quality, which pertains to the design, systems thinking, and quality criteria used in creating the software.







The Product quality is particularly important as it ensures that the software meets the requirements of both the customer and the engineering team. To achieve this, the quality criteria need to be explicitly defined and aligned with the product and customer requirements. One effective way to do this is through the Heuristic Test Strategy Model (HTSM) developed by James Bach. By using HTSM, software development teams can discuss and explore the criteria that matter to their context, such as capability, reliability, usability, and charisma, and determine the test techniques and coverage that they should employ.

In our organization, we have implemented various techniques to ensure product quality, such as defining quality attributes, having frequent team retrospectives, and fostering a culture for inter team demos and stand-ups. We also conduct workshops and collaborative sessions that focus on improving our skills and mindset as a team, which has helped us avoid common pitfalls and work more efficiently together.



In summary, project quality, people quality, and product quality are all essential notions of quality in software development. By prioritizing these areas and implementing effective techniques, our software development teams focus on creating a high-quality software that meets the needs of the customer.

Understanding 4Es of QCSD





The QCSD model, which stands for Quality Context-Specific Development, offers a four-step framework for implementing quality practices. This framework is known as the 4Es of QCSD: Enable, Engage, Execute, and Evaluate.

The first step is to **Enable** the people and project for quality. This means creating opportunities for collaboration and enabling the right mindset, skills, and culture within the organization. It's important to start with small changes and to make consistent progress towards quality goals.

The second step is Engage. This involves actively exchanging information, skills, and perspectives through collaboration sessions. This can include discussing coverage, design strategy, risk storming, user interviews, and more. Engagement takes time, but it's worth the investment to do the right things the first time around.

The third step is **Execute**. This is where the actual work happens, including developing clean and testable code, creating effective testing strategies, automating checks, and establishing a continuous feedback loop using tools like CI systems.

The final step is **Evaluate**. This involves gathering evidence from the previous steps and using engineering metrics to identify what's working and what's not. This information can be used to improve the Enable, Engage, and Execute steps and make continuous progress towards quality goals.

The 4Es of QCSD provides a helpful framework for aligning quality practices our organizational goals. By enabling people and projects for quality, engaging in active collaboration, executing with clean and testable code, and evaluating progress with engineering metrics, we can continually improve our quality practices and ensure the delivery of high-quality products and services.

Positive Experience of Implementing QCSD in ifm engineering

Implementing Quality Consciousness in Software Development (QCSD) has several benefits that can significantly improve the quality of software products. Here are some of the key benefits of implementing QCSD in my project and work context:

- 1. Shared Understanding: One of the most significant benefits of practicing QCSD is shared understanding. When developers, testers, and other team members collaborate and discuss the requirements of the product, it leads to a shared understanding of the project. This shared understanding saves time in the long run as we as testing team tester do not need to learn about the product after it has been developed, which can be a time-consuming process. By discussing the product requirements and design from the outset, everyone on the team now gets a clear understanding of what they need to do to achieve the desired outcome.
- 2. Better Collaboration: QCSD also promotes better pairing and collaboration between developers and testers. By working together, developers and testers can exchange skills and ideas, leading to better quality software. Through pairing and collaboration, our developers now suggest tools and utilities that testers can use to improve the quality of our work. Collaboration also helped to break down silos within our team and promoted a culture of sharing knowledge and skills.
- 3. Transparency and Trust: QCSD promotes transparency and trust within the team. By having an open and transparent culture, our team members now discuss issues and challenges without fear of blame or recrimination. As trust is established within the team, it lead to a more collaborative and productive working environment.

- 4. Improved Quality from Design Phase: QCSD helps to embed quality in the software development process from the design phase. By considering quality at the design phase, we could identify potential issues early on and how to avoid them. This proactive approach to quality saved our time and resources and lead to higher quality products.
- 5. Improved Testability: QCSD promotes discussions around testability, which leads to improved testability of the product. By having active discussions around testability, we could identify potential issues that slow down our testing and eventually work to address them. This led to improved quality and a more effective testing process.

These benefits have contributed significantly to the quality of our software products and have helped to create a more productive and collaborative working environment. While we have seen improvements in bug and lead time related metrics, I unfortunately cannot disclose them due to confidentiality reasons. However, we are still implementing more, monitoring our progress and look forward to experiencing more benefits from implementing quality conscious software delivery.



In conclusion, implementing QCSD can have a significant impact on the quality of software products. By promoting shared understanding, better collaboration, transparency, trust, improved quality from the design phase, and improved testability, teams can produce high-quality software that meets the needs of their customers.



US HEALTH CARE TESTING CHALLENGES & KEY POINTS FOR SUCCESS



- DR. SANJAY GUPTA

Dr. Sanjay has received Doctorate from Indian Institute of Technology, Mumbai, India. He has 22 years of IT experience with different roles as Associate Director, Senior Training Executive, Java/Test Consultant, Test Manager. In his current role, he is working as Software Development Associate Director at NTT Data, Bengaluru. Dr Sanjay is enthusiastic about IT Transformation, Digital current interests are DevOps, Automation and Health Care, Saniay is AZ-900 & Sun Certified Java programmer as well as Sun Certified Trainer. He has published nearly thirty research papers in international journals and presented research papers in more than fifteen international and national conferences. His current areas of research, study, and knowhttps://www.linkedin.com/in/lalitkumarbhamare/ledge dissemination are Health Care, Cloud, DevOps, Java, J2EE technology, Insurance domain and test Management tools. His contributions towards Science and Technology have recognized him a place in Marquis Who's Who in Science and Engineering.

He can be reached at sanjay.gupta4@nttdata.com

Abstract:

Health Care Domain Functional knowledge is one of the most important skillset & plays a critical role while testing healthcare applications :-Applications, Data Flows & Integrity, Reports, Enrollment Process, ID Card generation, Claims and Benefits along with many processes to ensure that factors like HIPAA standards, Data Security, compliance, cross dependency are considered during System Integration / User Acceptance Testing to ensure delivering the defect free software in production environment

In this Article, author discusses the key challenges faced during testing phase along with the solution as Success factors to release highest quality software to benefit our esteem readers.



FIG1: A HIGH-LEVEL VIEW OF US HEALTHCARE PAYER ORGANIZATION AND ITS KEY COMPONENTS

Introduction:

At a high level, Healthcare Domain can be divided in two parts (1) Payers and (2) Providers. These services are covered under the terms of product/Health Care policy. A member can purchase the suitable product/policy from the payer organization through various channels like company website, agent or third party.

Providers are Hospitals or Doctor Clinics where a member can receive the medical services in need.

On one side a member purchases a healthcare product from the payer organization suitable to his/her needs and during the event when the medical services are required visits providers like Hospitals or Doctor clinics.

By now, you will have an idea that there is a continuous data/ information flow between Payers and Provider organizations when we seek medical help. It is critical to maintain the Data integrity, Data Privacy and Data availability along with providing best medical services/treatment to a member and ensuring the expenses are taken care of without bothering the member. Medical bills are generated by providers and the claims are closed by Payer organizations as per the contract written in the policy/product document.

ISSUE 02/2023

PROCESSES

In this article the author would like to discuss the Enrollment process in detail along with the best practices followed during the software testing phase.

Enrollment Process: Enrollment is first process to ensure that new / existing customers/members are enrolled successfully with payer organization. In an **Individual Policy**, a policyholder is added under a plan with an effective date by paying a premium for the benefits provided by the insurer/payer. From this date a member is eligible for medical treatments, submitting claims and receiving benefits.

In **Group Policy**, a member is added to the group (which is already added under a plan) with an effective date of which he/she is eligible for submitting claims and receiving coverage. For example, an organization will buy the Group Policy from Insurer for their current employees.

As a Software Tester, it is important to understand key entities/ terminologies used in Health Care domain

The Key Entities include-

Insurance company: Medicare, Medicaid, Payer, Provider etc.

Patient/Consumers: Patient Enrolled

Regulatory Authority: HIPAA, OASIS assessment, HCFA 1500 and UB92, etc.

Healthcare and Life-Science solution Vendors

Basic Terminology of Health Care System

Payer: As the name suggests, Payers are the organization who takes care of expenses towards the medical treatment during the event when an enrolled member (Policy Holder) avails medical services

Provider: A health care professional (doctor), Hospitals, clinics, labs, Pharmacies etc. licensed to provide health care services

Claim: A request to health insurance company / Payer to pay a bill for health care service availed by a member

Broker: An insurance professional, who negotiates, procures insurance on behalf of insured or prospective insured/member

Finance: Insurance bodies that pay for medical expenses, it could be government (Medicare or Medicaid) or commercial

Medicare: A federal health insurance program for senior citizen and permanently disabled people

Medicaid: A joint and state program that helps low-income families and individuals pay for the cost associated with medical care.

CPT code: A current procedural terminology code is a medical code set to describe medical, surgical and diagnostic services.

HIPAA: It is a set of rules and regulations which doctors, hospitals, healthcare providers and health plan must follow in order to provide their services.

Member Enrollment

Below is the generic flow/ operations for member enrollment and its different combinations

Member Application

- Enrollment, reinstate, and terminate a member.

- Remove and add a dependent

- Generate premium bill.

- Process premium payments

While you are testing the enrollment process, it is important to include positive and negative test scenarios during System Testing:

Globally, the standard 834 file format is used for various functions of health plans and membership enrollment. It includes the following steps.

- 1. New Enrollments/Adds including a new member(s) to the health plan.
- 2. Disenrollment/Terminations eliminating an existing mem ber(s) from the benefits plan.
- Changes in enrollment changing a member(s) information data for an existing benefits plan member(s)
- Reinstatements restoration of a previous benefits plan member(s)

All the information is submitted to the insurance payer or healthcare organization that is accounted to pay health expenses and administer benefits and insurance. It includes insurance companies, healthcare professionals or organizations, such as PPOs or HMOs, government agencies, such as Medicaid or Medicare. The information consists of:

- Member name and identification
- Plan network identification
- Member eligibility and/or benefit information
- Product/service identification

Similar way, it is recommended to understand the functional flow of other processes like Welcome Kit generation, benefits, claims, Billing, HIPAA compliances, Reporting etc.

Key Challenges and Best practices to overcome from them while testing Healthcare Applications

Below are the key challenges faced while testing healthcare application

Knowledge Spread: Ensuring the right mix of functional/ domain knowledge spread among QA team members: It is important to maintain the right mix of functional knowledge. Periodic review of Skill Matrix for all the team members against the required Techno-functional knowledge required for QA Team responsible for Testing.

Inclusion of interoperability, compliance, regulatory, security, safety testing along Non-Functional, Functional and Integration testing

Testing team needs to be functional trained on the various modules, clinical usage, and the environment the software will be used for

Usage of HealthCare Domain vocabulary: It is impactful and impressive when you use the right domain specific terminologies during the discussion with other stakeholders. Below are the few domains specific terminologies recommended for readers to start with:

ENROLLMENT	MEDICARE	MEDICAID	COMMERCIAL	CLASS/ PLAN	AUTHORIZATIONS
REFERRALS	ACCUMULATO RS	EDIs-834, 837, 835, 270, 271	PROVIDER	CLAIMS	COB
Со-рау	NPI	EOB	PPO	Payer	Providers

Impact Analysis: Dependency of software modules- Testing team needs to ensure that any changes in one component or layer should not lead to side effect on the other. One can use Dependency Structure Matrix (DSM) [1] to analyze dependencies among different modules.

Common Positive Test Scenarios

- With the current, past, and future effective dates enroll different types of members.

- Inquire and change member data.

- Produce premium bill for an active member for the following month.

- Terminate an active member with past, current, and future termination dates greater than the effective date.

- Re-enroll a terminated member with current, past, and future effective dates.

- Reinstate a terminated number.

Must have Negative Scenarios

- With insufficient data enroll a member without mandatory information

- For a terminated member produce a premium bill for the following month

Test Case Optimization: One can use different techniques like Orthogonal Arrays [2] to optimize the Test cases. These techniques help in identifying the duplicates and redundant Test cases and ensure the Test Case numbers are optimized without compromising with the test coverage.

Along with the functional knowledge, it is critical for team to ensure data collection and measuring the Key test Metrics [1] to ensure the good health of the project. A good functional knowledge in Health Care domain along with technical knowledge ensures the quality testing delivery.

Requirement's walkthrough and reviews with Health Care Business SME: Understanding the requirements plays an important role to do the justification to testing. It gives an opportunity to know how the requirements under test impacts end customers. What was the functionality before the changes and what is expected? The time QA team designs Test Cases, it is important to get it reviewed with business owners. Similar way when they are sharing the Test results, it is advisable to get it review with Business owners. The review process for Test cases/ results helps in capturing defects proactively. As per my experience, a rigorous review process helps in stopping around 10% defects to production environment.

Automation & Regression skill set: Each Functional Tester needs to learn minimum one Automation Tool along with their rich functional knowledge. Automation not only helps in cycle time reduction for a given task, it also ensures the quality of the deliverables. A regular enhancement of regression suit ensures that any changes/upgrades have not broken the basic functionality of the application under test.

Know your Test Data: Selecting and identifying the right Test Data for your testing is an art that comes with experience and functional knowledge. A good testing professional must know how to write query to fetch the right set of Test Data from Database. A knowledge of the table structure and their relationship is good to have.

Knowledge Management and Standard Operational Procedure

(SOP): Documenting knowledge documents and SOPs are must to ensure that when a new member joins the team, they are selfguided to go through the Knowledge documents before they start the classroom training sessions. In new working paradigm where most of the professionals are working remote, one can record the training sessions and make them part of KM. A periodic review and updates of knowledge documents and SOPs helps in keeping latest knowledge. A small live hand on Test helps the participants to know the level of their understanding.

Co-ordination and One Team Spirit: As Testing team is the gatekeeper in ensuring the quality of the software, Strong coordination among QA, Development, Business Analyst, Release Management, and other stakeholders helps in expediting the testing window. A short daily standup meeting helps in resolving the bottlenecks and issue, if any?

Review productivity data for your Team: It is always recommended that one must maintain and review the Software Testing productivity metrics. The recommended key metrics are Test Case Design, Test Case Execution, No of Valid Defect found, Number of Defect Rejected by Development Team, no of comments found during Test case/Test Result reviews, no of items missed the delivery timelines. Reviewing these data with team helps in knowing the strengths and area of improvements.

Rewards and Recognition: Keeping the ownership, sincerity, and initiative-taking team, it is very important to recognize their efforts in larger forum. Reward them and often sit together to have a lunch. In virtual and remote workplaces, it can be done over a meeting with video. Try to meet the family members and talk about their wellbeing. A personal touch does miracles and everyone work as one team & One Family

Summary: In the present pandemic time and challenges, HealthCare software testing is critical and important to ensure that the end customers are receiving the services in time with accuracy. Functional Domain knowledge along is important to ensure quality deliverables. Continuous reskilling along with Automation helps in reducing the cycle time of QA Delivery. Internal and Business Test Case/Result reviews ensures that defects are proactively captured. Leadership motivation and encouragement helps in bringing the self-commitment among the team members.

What Next? Author would like to write few more functional topics to give readers a complete understanding of healthcare landscape.

Acknowledgment

The author would like to thank and gratefully acknowledge the support and encouragement provided by the entire QA Team: Yamini Negi, Abha Singh, Prashant Gajjala, Hema Reddy, Nikitha G, Sehla Saba, Suraj, Ashik Shetty, Deepak B, Jyoti, Vineetha K, Pavani, Bhagya Shree, Saikat B, Maneesha J, Lakshmi Nandeesh, Shubha R and Akarsh. Thanks, are also due to Vikas Saini, Franklin, Abhijit Adak, Vani Rajasagi and Sudhakar Paragiri for great support and encouragement. Special Thanks to Greg Lupica for continuous motivation that helped in writing this article. Last but not the least, thanks to my wife Archana, Sons Rishi & Ved to encourage me to complete this article.

References:

1. Not Just Numerical values by Dr Sanjay Gupta, February 2014, Year 4 Issue, Tea-time with Testers

2. Optimize Software Testing By Orthogonal Way by Dr Sanjay Gupta, October-November 2014, Year 4 ISSUE X



HEURISTICS, WEB DESIGN

VIP BOA – A Heuristic for Testing Responsive Web Apps EDUCATION, SPEAKING TESTER'S MIND



EDUCATION, WOMEN IN TESTING How To Read A Difficult Book



mon Migaj on Unspla:

Software Testing And The Art Of Staying In Present





Introducing Change In Organisation





INTERVIEWS, OLD IS GOLD

Over A Cup Of

Tea With Jerry

Weinberg

PEOPLE AND PROCESSES, WOMEN IN TESTING Addressing The Risk Of Exploratory Testing Part 2

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

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 $\underline{www.teatimewithtesters.com}$ 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 PLACE FOR CHATGPT



In last issue I took my first, <u>presumptive look at #ChatGPT for testing</u>. The app has real potential, and seems to have some ability to learn. It's tempting to run ChatGPT through a few exercises and come to a conclusion. Really the software needs a bit of a deep dive to come to any significant conclusion.

If you're not going to do, but you want to outsource it, well, real on.

But first, I digress.(It's worth it.)

The Bluffer's Guide

When I was in high school my mother purchased a series of books called something like "<u>The bluffer's</u> <u>guide.</u>" Each would have a title worded "The bluffer's guide to X", where X might be golf. wine, the stock market, computer programming, or skiing. Each book was very thin, perhaps 30 pages, and very small – perhaps three inches by five inches in size. A quick reader could digest one of these books in about an hour. Each book would teach you how to convey the idea that you had much more expertise in the subject than you actually did. The computer programming book, for example, suggested that you were a Fortran Programmer, and if they said Fortran, you could say C++. The finance book said that if you were ever pressed on a recommendation, you could say "I'm holding off a bit to see what the Fed does next", as the Federal Open Market Committee (FOMC) meets about every six weeks, and higher interest rates will be bad for the stock market, while lower interest rates will hurt the economy.

As I read I remember coming to the slow realization that the books were actually teaching you about the subject. Not a great deal; I didn't know why rising interest rates make the market go down. But a little. I particularly remember the Bluffer's guide to golf suggesting that you wake up early in the morning and walk a famous course, enough that you could make up an inventive story about a few holes, along with the hole number. Anyone in your listening group hearing the story who played that course would be inclined to believe you. It struck me at the time that that was a lot of work to go to for a lie. A reader might be more likely to actually try golf instead. Perhaps that was the point.

The point of bluffing was to create the illusion of mastery. With an hour, a good bluffer could appear to mere civilians as an expert. JB Rainsberger told me that among practitioners, all that it takes to have expertise in a specific subject is to read three books. I'm not sure I agree, but perhaps JB was speaking of real tutorials, while doing exercises and following along.

All this reminds me of bullshit. And by bullshit, believe it or not, I mean something specific. Harry Frankffort's wonderful little book defines Bullshit and something between the truth and a lie. Or, perhaps, in a direction. Where truth and lies are left to right, Bullshit might be up and down. I will explain.

ON BULLSHIT

On Bullshit

Frankfort says that bullshit is speech designed to convince without regard for truth. The bullshitter has an agenda – to get you to do something. Words come out of their mouth. The words might be true, they might be a lie – the bullshitter does not care. Asked why a practicing political philosopher would focus on such a lowly task, Frankfort replied "Respect for the truth and a concern for the truth are among the foundations for civilization. I was for a long time disturbed by the lack of respect for the truth that I observed... bullshit is one of the deformities of these values."

As a software tester and journalist, I understand that statement. Often I have explained the value of testing is that it has the least incentive to lie. Project Managers can ignore problems; analysts can say that the analysis is done because a deadline has arrived. Architects can wave their arms and draw boxes. Programmers can shrug and say "it compiles" It is the tester who wills uh "I can't log in", "search is failing", or "I cannot check out with a credit card." In the other roles, you can get in trouble for speaking truth to power. In testing, when I have asked respectfully "You want I should lie to you? I can do that if you'd like", the answer has always been no.

Combine bluffing and bullshit and you get ChatGPT.

That sounds overly critical, so let me explain. Bluffing Bullshit is different than full of sh*t. It implies the reader at least did read one book on the topic. Think of it as a college freshman or sophomore, in a class outside their major, trying to get at least a C minus on an essay. There won't be any real insight into the topic, but our college student will put the effort in to go to wikipedia, plus doing keyword searches, to come up with the right buzzwords to use. They will also have conversational English. Words will have proper tense. I asked #ChatGPT to create a sales script for outsourced and offshore testing services. This is what it came up with:

Certainly! Here is a template you can use as a starting point:

Subject: Offshore Software Testing Services

Hello,

Are you looking to improve the efficiency and cost-effectiveness of your software testing process? Our company specializes in providing offshore software testing services.

Our team of experienced QA engineers can help you ensure the quality and reliability of your software products, while providing significant cost savings compared to in-house testing.

Some of the benefits of working with us include:

Flexible engagement models to fit your specific needs

A wide range of testing services, including functional, performance, and security testing



State-of-the-art testing infrastructure and tools

Timely and accurate reporting

We would love the opportunity to discuss how our services can support your business goals. Please let me know if you have any questions or would like to schedule a call to learn more.

Best regards, [Your Name]

]This is not wrong, per se. Not exactly. It just demonstrates no real insight. A human copywriter with no testing experience at all, tasked to create a sales page or email for testing services, would likely come up with something like this. Efficient is good, right? Effective? Lowcost? State of the art? Good reports? You could make something very similar for almost any outsourced knowledge-work job. In fact, humans without expertise often do come up with these sort of nothing statements. A few years ago when the folks at TechBeacon asked me to write an article on container technology tools, I found similar language. After reading the sales pages I still did not know what half of them did, just that they were all "scalable", "enterprise grade", and "secure." This was especially true of the tools earlier in development. In some cases, perhaps the new CEOs had not even decided what the product was yet, so they cold pivot. It seems more likely the company has not made the investment to high professional marketing slash product people yet, who could do both jobs.

So, how does this happen?

How ChatGPT Works

Imagine software that can scan the open web and understands concepts at least as well as a college freshman outside of their own major. The software also has an understanding of programming languages, perhaps to a deeper level, because the programming languages have a strict structure. The code could be trained on all the source code on GitHub and Stackoverflow, all the descriptions on wikipedia. The software can parse out complex words from the common, recognizing, for example, that "service layer" is a term that is important and different on the Test Automation page of wikipedia. If can do the equivalent to a search on those keywords, even if they are not links, on whatever it is trained on, which is likely a subset of the open web - plus insight gleaned from users. Like the JB Rainsberger idea mentioned earlier, it can look up a dozen different sites that discuss page objects, and find the commonalities that the majority agree on. The tool can take two-dozen programming languages in a half-dozen languages, compile them down to intermediate code, then spit out the result in whatever language you ask it for. And, like any good bluffer, the tool is changing the words just enough that no one can accuse it of plagiarism.

ISSUE 02/2023

PRODUCTS

Remember, we are talking college sophomore level, but coding expertise. If you ask for a program to print out all numbers from one to a hundred, it can generate it. If you ask gotcha coding challenges, and the answer exists on the web, it can figure it out and spit out an answer. If you ask mathematical questions in conversational language, the tool can read at a college level, convert your words to symbols, enter that into an interpreter and spit out an answer. All that is what it does outside the box, without expertise. Interestingly, #ChatGPT is currently performing in the 73rd percentile of language models tests for general knowledge and 65th percentile for the US SAT exam for analogies. Knowing it had this sort of general skill, I did try to take ChatCPT on a "programming quiz" website, and found the answers were not quite good enough. To get to success the programmer would need to be more like a college senior in Computer Science, both asking more refined questions and refactoring the provided code. This would require a very similar amount of expertise as actually solving the problem.

Bullshitting a Bullshitter

Alex has an interesting video where he talks about the potential for tools like ChatGPT. It's worth a look, but here are a few things he mentions.

- AI Labs already has a tool, DALL-E, that can generate images based on a few words. A movie is just the same image changing slightly. We could make explainer videos text-to-speech, possibly using speech from #ChatGPT. Text-to-speech can add audio.

- We could analyze millions of online dating conversations to figure out "what works" and generate unique approaches to get to that first date.

- We can analyze millions of people and create an avatar for the most beautiful person, then combine item one and two to make it respond to words.

- We can use AI to analyze, summarize large bits of information, then use it to create emails or other communications.

Hormozi's final idea was to use #ChatGPT as a research assistant, as that college sophomore, with the human doing the training. That last bit is my words, not his, but it boils down to his suggestion. Watching the video, I couldn't help but notice that the ideas could happen ... someday. Maybe. They are mostly logical extensions of ideas we already have. However, I don't expect any of this to happen tomorrow. The most current work I see in the field with ChatGPT is not using the chatbot nature of the software at all, but instead throwing large datasets at the tool and asking which if these is not like the other. If you are worried about #ChatGPT taking over your job, please realize, I just did on one site for COBOL remote jobs and found two hundred and eighty. Newspapers, have been around since 1605 and beleaguered for two decades. Yet many small papers have found ways to innovate in their business models and do just fine. The internet threatened to make magazines irrelevant, again, many of them are finding ways to do just fine. The first talk on the death of testing I know of was in 2011. The people who gave those talks have retired, or at least moved out of the field directly. Yet testing remains.

Moving Forward

Ironically, the way to lock down an AI bullshitter is the same way to figure out if AI is bullshit – ask for details.

- What is possible with AI?
- How would you do it
- Can you show it working?
- How do I do it?
- Okay, I did it. So ... what?

The answers you get might not be nearly as gee wow nifty powerful as the video I linked to, but they'll be real.

As for the rest, we'll continue to see new ideas creep in. Having an assistant that can summarize wikipedia and the open internet has some value - remember, books are behind a paywall.I'm not the first person to recognize that when ChatGPT is out of ideas, it tends to guesstimate conversationally, nor the first to label it BS. What I hoped to do in this essay was put a little more precision on what that BS looks like. This might help the reader figure out how it could be useful, what the limits are.

All that might change, of course. For today, the work the tool can do seems superficial. Part of that is the complexity of the examples and limits of training. Hopefully, the reader gets some ideas of what we can experiment with next.

Tell me how I'm wrong.

And tell me what we should do next.



MATT HEUSSER

The managing director of Excelon Development, Matthew Heusser is a 2015 recipient of the Agile Awards as an online contributor to the field, and a 2014 recipient of the Most Influential Agile-Testing Professional Person Award. In addition to 20+ years of software delivery. With over 1,000 articles and podcasts to his credit, Matt was also the initial lead organizer of the Great Lakes Software Excellence Conference, the Workshop on Technical Debt, the Workshop On Self-Education in Software Testing (WHOSE), and the Workshop on Teaching Test Design (WhatDa).

A father of three daughters, Matt spends most of his spare hobby energy as a parent.

Quality Conscious Software Delivery

Lalitkumar Bhamare

Accenture Sona





ISSUE 02/2023 PRODUCTS



11

PHASE SPACE -INTRODUCTION





ARSLAN ALI - PRESALES AND SOLUTIONS CONSULTANT

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

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

FAIZA YOUSUF - PRODUCT MANAGER AND COMMUNITY BUILDER

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

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

The concept of Phase Space was introduced by "Dynamical Systems Theory", which is an area of mathematics that describes the existence of complex systems and their behaviors in conjecture with each other and also independently.

A phase space is a space in which a system exists with all possible states. These states in their independent forms coexist and correspond with each other.

Consider a parameterized solar system and the objects existing within that system in conjecture with each other, there are dependencies, push, and pull of gravity, the wobbling effect of one object on another affecting the internal compositions of objects, and how one system in its existence keeps the objects bounded within the boundaries.

For software applications, the phase space represents system components bounded with finite time loops ("Initiation and termination of instances, functions, data, and APIs") containing visible and invisible boundaries.

The existence of these boundaries depends on the conditions and constraints coded for the software components or are determined by the extent to which the system interacts with its users and other entities in a physical universe.



Software Objects existing in the Phase Space:

A depiction of objects in a gravitational bond with each other in phase space, while some remain in the center as a mass influence, others reflecting a more volatile nature, and within this delicate balance, we can see a balanced ecosystem of objects in conjunction with each other.

The Movement:

As a universal principle, each object bounded by force will move in a chaotic yet predicted movement. The object may change its path, trajectory, or speed, but it will remain in a pattern that can be observed and then influenced by external forces. Same as any software application component.



The Force And Attractor's Relationship

The oracle that drives the force and attractor's relationship is based on the principle that if an object creates attraction to another object by a certain force, then there should be an opposite amount of moving force to create that motion for the attractor. In terms of software applications, the greatest attractor to move objects is the data, the combination of users, and their behaviors.



The Wobbling Effect:

The concept of wobbling is related to object instances interacting with each other. This interaction can be via internal APIs, data, or external excursions by the users.



The effect can be also caused by a different domain interacting with core system objects just because a user or the API has become a medium for that interaction.



The example application exists in the phase space with object universal properties of wobbling, force, push, and pulls from other objects and applications.

Testers' Perception:

Testing embeds itself as an encapsulated activity. If we see testing in a three-dimensional format, ideally it forms the shape of a sphere and the core of this sphere will always be Specifications and Requirements.

The "context" puts an intense amount of its gravitational force on an application sphere. The changes to requirements concerning time are directly proportional to the force and effect of context on the application.

This changes everything because once the application is not part of the developers' domain and is basically in the open (in the hands of the testers and users), the strong stimulus of user experience, business rules, cognitive behavior, test results, boundary analysis, and technology will shift the perception of application as a structured line of codes to some kind of object with wobbling boundaries and is extremely sensitive to human interaction. Eventually, the bugs are raised this way.

No matter how many test cases we write in the limited time we have, we will never make sure of the coverage we need. This de facto effect of context inflation changes the rules of the game.

The key thing here is to make something sturdy out of the first set of requirements. Something flexible enough to take on the pressure of inflation. For example, consider the following scenario of the ride-sharing application:

The Occurrence of inevitable:

A commuter wishes to book a ride.

The application browses the list of available rides.

The commuter, in haste, clicks the ride, just started to pick up the passengers, and was on the move.

The ride is booked successfully.

Commuter receives a booking number and can now track the coming vehicle on the application, alongside vehicle and driver information.

Booking is successful

The commuter reaches the designated station.

The vehicle approaches but never makes the stop to pick up our passengers.

What happened here?

The scenario for the driver application was not to book any rides for the vehicle when the ride is initiated by the driver.

The scenario for the commuter application was never intended to stop or deny bookings for a ride that is already started.

In the real murky world context, the business loses a customer.

A human condition, triggers software conditions, resulting in users' dissatisfaction and operational delays.

Customers' Perception

Human behavior in using software applications changes concerning the time they spend on that application.

month, the number is around 32 times considering 4 weeks of traveling.

These several interactions (as above) with a real-world user and the software can trigger an unlimited number of contextual events and behavioral decisions not written as part of code as test cases and neither mentioned in the UATs nor highlighted in the User Journey interaction and focused group session.

checkout, and payment test cases.



Consider very carefully that these types of tests cannot be carried out on an unlimited timescale. It's the opposite. These are carried out with a written script, using a bounded timeline, and a minimal audience.

The result for such activities is usually driven by "undeclared assumptions" where the developers and testers think of the problem as "too obvious" to occur and treat them as trivial or say "we will deal with them if and when they occur."

- For the example above, the commuter may use the application at least twice a day from going to reaching a certain destination, so every
- For example, in terms of commuter application, not all scenarios can be tested without the actual use of a vehicle, travel, onboarding,



Unlock the Highest Level of Efficiency and Reliability of your Apps with the Avo **Test Automation Platform**

Learn More

WEBINAR BY ASSOCIATION FOR SOFTWARE TESTING.. SIGN UP FOR FREE!



RECOMMENDED EVENTS AND YOUR CHANCE TO MEET TEAM TEA-TIME WITH TESTERS IN 2023





EUROSTAR 2023 JUNE 13-16, 2023

CHECK IT OUT!

SEASTARCONF 2023 OCT 15-19, 2023

CHECK IT OUT!



CAST BY AST, 2023 NOV 06-07, 2023

CHECK IT OUT!

QUALITY-CONSCIOUS SOFTWARE DELIVER

CAST ASSOCIATION FOR



ISSUE 02/2023 COMMUNITY

Lalit Bhamare



WRITE WITHOUT FEAR.

SEND IT TO

editor@teatimewithtesters.com



JOURNAL FOR NEXT GENERATION TESTERS

CONTENT PREVIEW : ISSUE 03/2023

MORE AWESOMENESS IS ON YOUR WAY THIS SEASON!

YOUR ARTICLE HERE

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



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

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

TEA-TIME WITH TESTERS ISSUE #02/2023



023 AY THIS SEASON



TEA-TIME WITH TESTERS

THE SOFTWARE TESTING AND QUALITY MAGAZINE



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

