WAKING TESTERS UP SINCE 2011

ISSUE #01/2023

TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

Testing, Quality, Machines, More!

Testing and Prevention: The Illusion
Page 06

Testing Recommendation Systems
Page 28

ChatGPT for Testers:Part 1
Page 36

TESTING

QUALITY

SEMANTICS

MACHINES

MORE!

TEA-TIME WITH TESTERS

06
PEOPLE
IDEAS THAT SPEAK FROM THE MINDS THAT THINK

22
INTERVIEW
OVER A CUP OF TEA CONVO WITH GREAT MINDS IN TECH

28
PROCESSES
ARE YOU DOING IT RIGHT? FIND IT OUT

36
PRODUCTS
BUILDING THINGS THAT PEOPLE WOULD USE HAPPILY

EDITORIAL BY LALIT

INTERVIEW: 22-26
A CUP OF TEA WITH
JEAN ANN HARRISON

TESTING AND PREVENTION: THE ILLUSION 06 – 08

The “prevent” claim raises a significant question for me. Is the statement credible and representative of what testing and testers can provide?

THREE LESSONS AFTER THREE MONTHS OF QUALITY ENGINEERING 10 – 11

Three months ago I started a new job as a quality engineer, supporting two teams...

REBRANDING ~~MANUAL~~ TESTING 12 – 16

In this article, I’ll tie all this together. I’ll begin with an overview of shift-left, fail-fast testing. From there, I will promote techniques that shows how we can do even better than static testing by participating in the development of requirements and acceptance tests.

TEA AND TESTING WITH JERRY WEINBERG 18 – 20

Software Subcultures - Part 3

WAKING TESTERS UP SINCE 2011
ISSUE #01/2022

TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

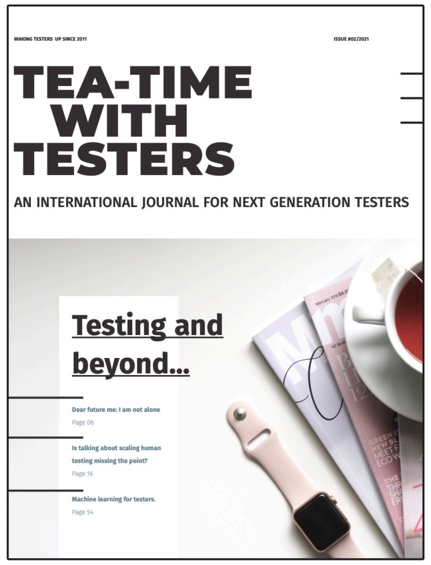
Testing and the human aspect

How to sell excellent testing
Page 06

Automation with human touch
Page 32

Becoming a code listener
Page 36

TEA-TIME WITH TESTERS


WAKING TESTERS UP SINCE 2011

ISSUE #02/2021

TEA-TIME WITH TESTERS

AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

Testing and beyond...

Dear future me: I am not alone
Page 06

Is talking about scaling human testing missing the point?
Page 16

Machine learning for testers.
Page 54

A NEXT GENERATION TESTING MAGAZINE

TESTING RECOMMENDATION SYSTEMS

28 – 32

The age of digital transformation has brought with it a wealth of information. However, filtering it to be usable can be highly challenging. It is now possible to understand patterns in user behaviour and then correlate it with other user's behaviour to predict and help in the decision-making process. The data which is gathered is then processed by recommendation algorithms. These Recommendation Systems constantly analyse various types of information to provide a user with a variety of valuable information.

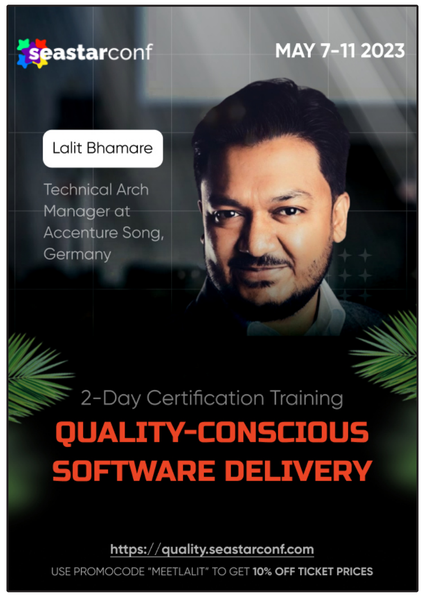
CHATGPT FOR TESTERS: PART 1

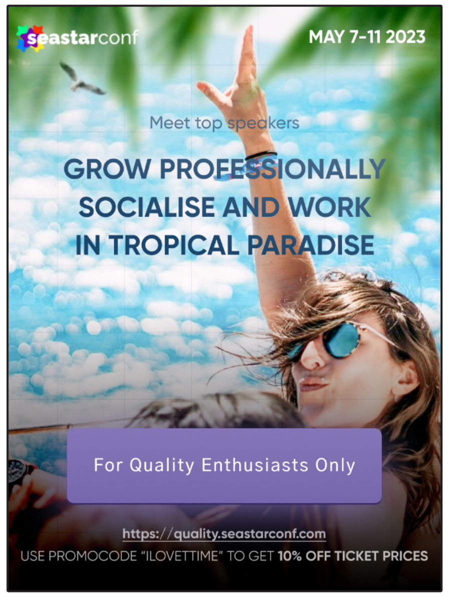
36 – 40

Seventy years ago Alan Turing proposed that real Artificial Intelligence (AI) would be able to interact with the person through a keyboard and screen just like a human. That is, the way to pass a Turing test would be if the user could not tell who was on the other end of the network — human or AI. ELIZA was an early attempt to pass the Turing test. Eliza mimicked what a psychotherapist might do, asking "tell me more?". or "how does that make you feel?" or matching a pattern. ELIZA was also easy enough to trick, and lacked a sense of context. It was … a start. There are plenty of Eliza-like programs available online. There's a trivial example from today at right.

SAVE THE DATES - RECOMMENDED CONFERENCES FOR TESTERS

2023


seastarconf
MAY 7-11 2023
Lalit Bhamare
Technical Arch Manager at Accenture Song, Germany
2-Day Certification Training
QUALITY-CONSCIOUS SOFTWARE DELIVERY
https://quality.seastarconf.com
USE PROMOCODE "MEETLALIT" TO GET 10% OFF TICKET PRICES


seastarconf
MAY 7-11 2023
Meet top speakers
GROW PROFESSIONALLY
SOCIALISE AND WORK
IN TROPICAL PARADISE
For Quality Enthusiasts Only
https://quality.seastarconf.com
USE PROMOCODE "ILOVETTIME" TO GET 10% OFF TICKET PRICES

On "We don't need" this and that...

Another interesting year has passed by and here we are in brand new year 2023.

With every new year, the change drivers in industry (technology, business models, methods and tools) compel testers to come out of their comfort zone and embrace new challenges. This means we should unlearn old, irrelevant ways of testing and reinvent new ones, or tweak and twist the known ones to suit changing contexts.

That's the very nature of a testing profession. It is constantly changing, evolving, transforming but it continues to exist. To the best of my knowledge, "dedicated testing" as a profession was born out of its need. Programmers writing the code and testing it for themselves was probably not enough which is why a tester was brought in to do that job with skills and dedication.

Over the years, industry perception of testing has changed. For most of the organizations that care about building quality products, skilled testing is essential. For some organizations that aim for a different type of quality, a good-enough software, dedicated testing not always necessary. And they are apparently right at their place. Why to invest in dedicated testing if you can produce software that is marginally accepted in market and your business survives on quantity rather than quality? In certain business contexts, investing in dedicated testing may not be really fruitful. By coaching programmers in testing it is possible to make them more competent, and what programmers alone can contribute in given context might very well be enough.

But does that mean "most" of the teams don't need dedicated testers? Actually, no team requires dedicated testers if all that happens in the name of testing is "finding obvious bugs" which competent programmer could easily avoid writing. If I am to have dedicated testers in my team, I want them to tell me all the ways the product does not work. Similarly, if I am to have programmers in my team, I want them to write the code without any obvious bugs. Be it this or that, it requires dedicated efforts, skills, passion for professional quality work, and most importantly a mindset to get it. I would say "every" team should have programmers and testers who would "partner together" to deliver quality software that people will use happily (and not because some market giant left them with no choice but to accept marginal quality crap).

I for one can understand the frustration of managers if all they heard from testers was, "I clicked here and it broke" instead of information about risk that stands up to scrutiny. But it would be a terrible mistake to assume that testing is all about such button pressing exercise. With their sharp observation skills and their placement in your overall "system", a skilled tester can create a lot more value for the project team might even protect your system from collapse. One only needs to understand what meaningful testing looks like, to see what dedicated tester in a team is capable of doing. Do that well and you might be surprised to find out why you "need" dedicated testers in the team.

By the way, now that chatGPT is here, are we about to hear a new management slogan?

"We don't need dedicated programmers"… perhaps?

Let's get back to this in 2024, my friends.

Until then … keep testing, with dedication! Teams that understand what meaningful testing looks like need you more than ever!

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

Connect on Twitter @Lalitbhamare or on LinkedIn

“TESTING AND PREVENTION – THE ILLUSION”

- PAUL SEAMAN

Paul is an established, highly competent software tester with 20 years of testing and leading experience. He has experience across Waterfall and Agile development environments within companies that companies that range from Government to private enterprise, small local to global corporations.

His experience ranges across multiple domains (finance, legal, administration, travel, medical practice management, scheduling and compliance).

Paul has helped organise and run testing conferences, a meetup group and cofounded the Epic TestAbility Academy which taught unemployed people on the autism spectrum how to test software with the aim of finding them employment. His current adventure is with LIFELENZ where he works as a Senior Software Engineer. When not testing Paul is exploring new ideas, sitting down with his guitars (not frequently enough), reading interesting books across a range of subjects, blogging on https://beaglesays.blog/, podcasting as one of the 3 Amigos of Testing (https://anchor.fm/the-3-amigos-of-testing) and getting out on the occasional bike ride. You can find him on LinkedIn and Twitter @beaglesays

In what is my first article for quite a while I’m going to look at the notion that “testers prevent defects”. I see this claim made by non-testers talking about testing (yes “agile” I’m looking at you as well as your coaches), professional testers and test consultancies. It must be incredibly enticing to issue claims that as a tester you prevent an unwanted outcome. That’s powerful, right? As a marketing tool, either for a company or a person, it’s a bold selling point.

The “prevent” claim raises a significant question for me. Is the statement credible and representative of what testing and testers can provide?

Let’s start by looking at the meaning of “prevent”.

From the Collins English Dictionary:

To prevent something means to ensure that it does not happen.

And from the Merriam-Webster Dictionary:

To keep from happening or existing

The use of “ensure” (to make sure or certain) within Collins’ definition is interesting. If you care to look at other dictionaries you’ll find that prevent has definitions consistent with the above selections

Broadly speaking software defects have two states when we consider observation:

1. they exist and have been observed
2. they exist but have not been observed

Within state 1 we know there is an issue because it has been observed. We have a record of it happening. Perhaps we have identified the specific conditions required to reproduce the problem and are able to analyze the issue. We might even agree that the outcomes are undesirable (a threat to value) and fix the defect. Of course we might also make a decision to not make any changes (a different topic).

State 2 is the “great unknown”. Issues are sitting in the product just waiting for somebody to stumble across them. To the extent that they remain “hidden” and do not threaten value, these are often ignored. Until they change into state 1.

For the purposes of this discussion let's move on from state 1. Clearly there was no prevention because the problem has been observed (either pre- or post-release).

Before I venture further let's consider a few places we might observe issues within software development while wearing a testing hat:

1. Documentation – specifications, help guides, product claims
2. Discussion – ideas, thoughts, queries about the software, specific to a set of changes or the product more generally
3. Software – investigation of the product either in part or whole

As a tester, I:

- engage with issues by helping to solve them with other people. The issue might be that we need new, additional functionality to keep customers happy or that some part of current functionality is not working in desirable ways (ways that threaten value)
- provide evidence-based observations of what I have done during testing. What I have observed, my view of risks in the software. I'm likely to comment on things such as (but not limited to) ease of use, consistency in the application, issues I have found and how much of a threat they might be. My communications around testing can cover a lot of different considerations. Key to these observations, the "consistent thread", is that I can back up my observations with evidence. If I'm asked to provide details related to my testing my response will not be "just feels like it" or similar. It will be backed by specific evidence.

If I claim that I "prevent issues", how do I provide evidence that I prevented a thing that never existed? If my Manager (or anybody else) asks me to evidence the "issues I prevented" how would I do this? At best I could point to a trend of declining issues in production (which is an excellent outcome) but correlation does not imply causation. I get that it's nice to think in this way but I actually want to see the link because that's important feedback in improvement loops. How do you know you are preventing anything? Even small software companies have a myriad of changes happening in parallel. So which ones are working well? That's a matter of evidence linking changes to outcomes. Good luck with that when you have no evidence (remember that the issue never existed).

It seems to me that a re-frame is in order. Let's consider that by visiting those places I listed earlier where we might find issues. Documentation, discussion, software.

You're reading through a specification and you find an error in a statement regarding functionality. To fix this you consult with the specification author and a change that corrects the problem is added. Cool you prevented an issue….except you didn't. What you did is find something in the document that does not make sense to you. You detected a signal that there might be an issue here. When you discuss this with the document author, and they concur, then they will update the document to add clarity. But, and this is important, they may not agree and the document might not be changed. Regardless of whether the change is made, this is early detection, not prevention.

You're in a project group discussion. The basic information flows of the project are being mapped out along with how data will be entered and interacted with by your customers. You notice a large inconsistency with a similar feature elsewhere in the software. This inconsistency would reduce usability and increase confusion so you point this out. Awesome, you prevented an issue……except you didn't. Again you detected a signal that there might be an issue, you raised this with your colleagues, and further discussion and investigation is likely to follow. Perhaps this inconsistency, while not initially known, is now considered to be an important aspect of the project and will be retained. Again, this is early detection of an issue, not prevention.

You're running a test session pairing with a Developer. During your exploring you observe that for a given set of values you receive different results each time you enter the values. Incredible, you prevented an issue…except in this scenario that's not a claim you're likely to make. Why? It's really no different to my first two examples. The incorrect output is a signal that there is an issue. We have helped identify that further investigation is required so we can reconcile actual behaviour with desired behaviour.

When I see claims of testers, or testing, preventing bugs it seems to me that testing is being set up for failure by representing goals and outcomes it can never own. It is a confusion of what powers testers and testing possess. If I was a surgeon and you were a doctor off to the South Pole as part of a team, it is a requirement that your appendix be removed. As a surgeon I could, in this context, assert quite positively that, by removing your appendix, I have prevented you suffering an episode of appendicitis. Testing isn't like that.

Testing is like this. You're a passenger in a car, driving down a road that has a variety of speed limit signs. The car has a speedometer which you can see and you glance at it occasionally to check the car's speed. Does the speedometer reading prevent the driver from driving over the speed limit (which is an issue)? It doesn't. The speedometer provides you with a signal which you can either ignore or act upon. You might say to the driver "gee the speed limits change a lot around here, we just moved from an 80 km/h zone and into a 60 km/h zone". The driver can choose to listen to you or ignore you. They might increase speed, decrease speed or stay at the same speed. Changing speed requires a direct input on the accelerator and it is the sole responsibility of the driver to make that adjustment.

As a tester you have a focus on the speedometer (and other conditions that are part of the context such as the weather, the road conditions, etc.). You are providing feedback, perhaps even encouraging slowing the car to a more appropriate speed. You are an observer of what is happening, not the driver who has control and can make changes based on your feedback. You are providing feedback that can be acted upon, but you're not the person making the adjustments.

As I noted at the opening of this post, I'm very unclear why people really want to make the claim that they, or testing, "prevent issues". Not only is that claim beyond the remit of testers and testing, it is damaging to testing. It denies the value and usefulness of detection, something that good testers bring to the table with each test assignment and discussion. My advice is to use your detection skills, scrutinise, explore, question, propose ideas, challenge and advocate. When you've done these things you can actually demonstrate how you have influenced product quality by talking about all those issues you have brought to light. That feels a lot like being an advocate for better quality in an authentic way.

A big thank you to Lee Hawkins (@therockertester) for his endless patience and quality feedback.

seastarconf
MAY 7-11 2023

Meet top speakers

GROW PROFESSIONALLY
SOCIALISE AND WORK
IN TROPICAL PARADISE

For Quality Enthusiasts Only

https://quality.seastarconf.com

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

"THREE LESSONS AFTER THREE MONTHS OF QUALITY ENGINEERING"

Three months ago I started a new job as a quality engineer, supporting two teams. So far it's been an interesting challenge. The two teams were formed only a few months before I joined, although some team members had been working for the company before that. Each team has their own product manager. We also have an engineering manager, but he joined only two weeks before I did. And then I was added to the mix, with a job description that didn't give a lot more guidance than: support the team in things related to testing and quality.

So my first task in my new job was figuring out what my job was. Or rather, figure out what concrete things I could do that fit that job description. This was not made easier by the fact that we're a fully remote company. Not being in the same space throughout the day does make things harder when you're trying to find your place. Reflecting on the past three months made me realize there are three things that are really important: visibility, connections, and patience.

Visibility

The first lesson is visibility. When your work does not come down to "perform these steps in the process", it's easy to become a bit invisible. Not that your colleagues forget about you, but you're not top of mind either as they go about their jobs. Especially if you're working remotely, because it's not as if people see you sitting behind your desk as a reminder that you joined.

Showing that you're there

So the first part of visibility is literally that: showing that you're there. Join all the meetings. Have your camera on. Say something or ask a question. Ask someone for more information or resources after the meeting. Similarly, be active on Slack (or Teams or ...), even if it's just an emoji reaction to what someone said.

I've also started an experiment: writing a weekly internal blog. The inspiration came from Giles Turnbull's "The agile comms handbook" and its idea of "working in the open". Every Friday I take 30 minutes to write three paragraphs about my past week. It's still an experiment-in-progress, but it's been successful enough after six posts that I decided to stick with it. At some point I should probably dedicate a full post to this experiment.

Showing what you can do

The second part of visibility is showing what it is that you can do. Quality engineering is a very broad discipline. It's also a young discipline. Few people have experience working with quality engineers.

So I was faced with two questions:

- What is it that quality engineers do?
- What is it that you can do?

The best way to show what you can do, is by doing things. That's easier said than done, though, when you have to answer the two big questions above. Doing lots of small things is key here. One place to start is in meetings and conversations. Actively participating in those not only confirms you are there, it also lets you demonstrate your expertise through what you say or ask.

For example, in a conversation about API design I brought up the differences between RESTful APIs and RPC APIs. Hopefully that made my colleagues realize I know a thing or two about API design. Similarly, when investigating a support ticket, I asked a developer if I could talk them through what I had found so far, because I was completely stuck. At some point in my story I said "So I looked through the code, and ...", which made the developer go "Oh, You looked at the code?" Hopefully that was enough for this developer to discover that's something I can do.

And if not, that's fine. The point is not to make a great display of one of your skills that people will remember forever. The point is to consistently show in small ways where and how you can provide value.

Connections

The second lesson is connections. Working remotely means that all communication gets that little extra friction. You can clearly notice it during meetings, for example when two people start talking at the same time and they have to figure out who goes first. It's less noticeable, but a bigger issue, when it comes to small and casual interactions. There are a lot fewer opportunities to have those than if you're sharing an office. So you need to be more intentional about them, which can feel awkward.

It's definitely something I personally struggle with, because even at an office I'm not the greatest at those casual conversations. Sometimes it comes naturally, often enough it takes me active effort. What I find a lot easier is to connect with colleagues through doing work together or through having conversations about the work we're doing. Unfortunately, if you're still figuring out how you can contribute to the work, it's not that easy.

There are two things I've been doing to build connections. The first thing is to schedule some coffee chats. When I started, I had an introduction chat with everyone in the two teams. Since then, it's the work that has been determining who I speak with and when. So I realized it was time to change that.

A second thing is that I've been actively making conversation as we are waiting for a standup or other meeting to start1. Even though it takes me some energy because it doesn't come that naturally to me, I feel it's a lot better than what would happen otherwise: people waiting in silence and/or doing something else.

Finally, I should not forget to mention connecting with my fellow quality engineers. In a sense that's been easier because of the "one of us"-effect. What also helps is that we have a Slack channel and an informal catch-up every other week.

Patience

The third lesson is patience. I have very high expectations of myself. With less than two months in, I was asking myself: am I doing enough? Shouldn't I be having a bigger impact already? Part of me knew those expectations were unrealistic. And luckily some colleagues said I was doing fine. Yet that doesn't make these feelings go away completely.

So I have to remind myself to be patient. That my main focus might be on figuring out what my job is, but that in the meantime lots of other things are happening in the company as well. That yes, I need to be moving forward, but also that these things take time. I need to be patient with myself and with others.

Reflection

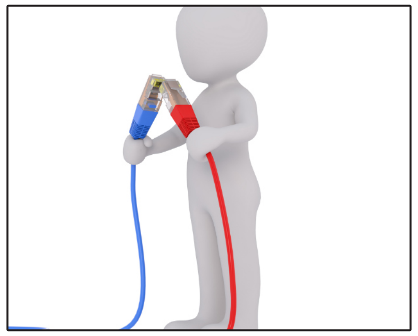
What I find interesting about these three lessons is that none of them are directly related to quality engineering or software development as such. Rather, they're about entering a situation and figuring out how you, with your specific skills and knowledge, can add value.

You could say that's a problem, typical of roles that lack very clear expectations. I'd like to turn that around, though. I think that things get a lot better, when people can let their job description fade into the background and instead focus on where they can bring value.

JOEP SCHUURKES

- Joep wandered into software testing in 2006. After a decade in which he learned (and practiced) exploratory testing and test automation, his focus shifted to a bigger question. How can teams and organizations build and deliver good software? To answer that question, he has been exploring topics such as technical leadership, agile coaching, and software methodologies.

Joep has given talks and workshops at conferences throughout Europe. He's also one of the organizers of the Friends of Good Software unconference and of the LLEWT peer conference.



REBRANDING ~~MANUAL~~ TESTING

Introduction

In my earlier article, published in the November 2022 issue of TTwT, I made a claim that ~~manual~~[1] testing is very much alive, and the skills ~~manual~~ testers possess can be effectively leveraged when applied to static testing. I also called out that there's something inherently wrong with the label "~~manual~~ tester, and I dropped a hint that Behavior Driven Development (BDD) is a great way to apply many core testing skills.

In this article, I'll tie all this together. I'll begin with an overview of shift-left, fail-fast testing. From there, I will promote techniques that shows how we can do even better than static testing by participating in the development of requirements and acceptance tests. Finally, I'll propose a solution to being branded a ~~manual~~ tester and propose a more appropriate and fitting job title.

Revisiting Shift-Left, Fail-Fast Testing

While static testing promotes the goodness of shift-left and fail-fast testing, this goodness can be further amplified when testers get involved in the development of the requirements and acceptance tests.

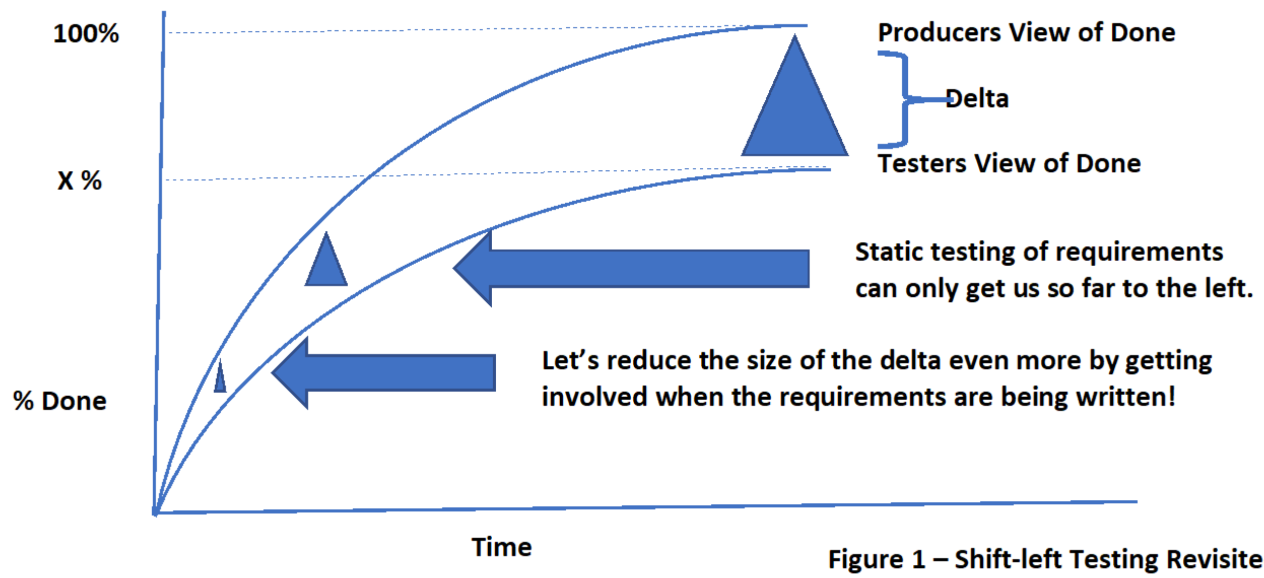


Figure 1 – Shift-left Testing Revisited

BDD Primer

Lightweight / agile requirements like user stories are a great thing, but they are only effective if, and only if, test cases are developed as their proxy. But whose test cases? Most stories contain acceptance criteria that is too high level. At the other end, automated unit tests that's been advanced by Extreme Programming are too low level. Why? Because test code coverage says nothing about business value. For example, a developer saying "x' number of unit tests have been written and passed is great, but it does not provide any sense that the test cases measure what's important to Product Owners. They are more interested to know about the test coverage of their features.

Enter BDD. It was introduced circa 2004 by Dan North.as an alternative to unit testing and Test Driven Development. At its core, it couples requirements and acceptance tests by using a language that non-technical users can understand. There are many flavors of BDD, but the one most popular these days uses a simple language celled Gherkin[2]. As will be shown shortly, this opens a new and existing pathway for testers.

For the sake of economy, I won't have the space to go into every detail of Gherkin, but below is a summary of the major keywords. For readers who are interested in learning more, a good place to start would be www.cucumber.io or www.specflow.org.

Keyword	Meaning
Feature	Provides the name and brief description of a feature.
Scenario	Describes a real-world example. These are acceptance tests.
Scenario Outline / Examples	Describes a scenario using multiple data values.
Given	Provides a precondition. These serve as test setup instructions.
When	Describes a trigger, i.e. the action a user takes.
Then	Describes an observable and testable outcome
And	Combines multiple Given, When, Then (GWT)

Example 1: Developing Happy and Unhappy Path Scenarios

Here is a simple, hypothetical example for an ATM Withdraw Money feature. It's not production worthy, but it will suffice for now.

Feature: Withdraw Money from an ATM

As a bank customer

I need the ability to withdraw moneySo I can go shopping and pay my bills

Scenario: Account has sufficient funds

Given Dave has been authenticated

And has a savings account with a $200 balance

And the ATM has $1,000 cash on hand

When he requests to make a withdraw from savings

And selects the option to withdraw $40

Then the system dispenses $40

And updates its cash on hand to $960

And updates the balance of the savings account to $160

And adds transaction details to the transaction log

And issues a message "Please take your cash"

> Many people think you need automation to write scenarios. FALSE! Yes, tools like Cucumber and Specflow can be used for automation, but the real value of writing complete and expressive scenarios lies in their ability to effectively promote a shared understanding and deliver better quality sooner."

Note how powerful and expressive this simple example is. Also note that the expected results are clearly written and testable, and they are far superior to what is typically captured in user story acceptance criteria.

The recommended practice for developing features and scenarios is to actively engage 3 people:

Product Owner: who describes the feature and the desired behavior of scenarios.

Developer: who ensures the feature can be implemented.

Tester; who ensures the feature can be tested with expected results.

Enter the Tester, a human being, not a piece of code, who is now sharing many of the skills typically associated with a Systems Analyst. This brings to light that taken in its most abstract sense, both testing and analysis are cognitive processes of discovery. Analysts seek to understand and drive out the "what", while Testers do same, but focus on the "what if". It's like looking at the same coin from two different perspectives.

Let's see how this can play out using the simple withdraw feature described above. A Tester might suggest some unhappy path scenarios as well, such as:

- Customer has insufficient funds.
- Customer provides an invalid amount. Examples: $0, $37, $400

  This is a great example of using a scenario outline, which I explain below.
- Customer specifics “other” amount. Example: an amount on a list, such as $140
- Customer provides an amount that exceeds the daily withdrawal limit.
- Customer cancels the transaction.
- Customer does not respond in time (timeout)
- Customer forgets to take cash (my ATM sounds an alarm after a few seconds)
- ATM has insufficient funds.
- Transaction is not able to complete.
- Customers account is closed or suspended.

Note there isn't a scenario for customer is not authenticated. That's because it's in the Given clause, and thus, is a precondition.

Example 2: Developing scenarios using combinatorial testing techniques

Dealing with an explosion of test cases is nothing new to a tester, nor are tools like AllPairs or Microsoft PICT to generate test data. The good news is that we can leverage our knowledge in combinatorial testing to develop very effective and compact scenarios.

Here are the steps involved for a hypothetical system used to determine if I should buy a bond or not. Here is the purchase criteria I have established:

1. The bond has a AAA, AA, or A rating.
2. The bond has an A- rating and risk rating of medium.
3. The bond has a B+, B, or B rating and has a risk rating of high.
4. The bond has a C+, C, or C- rating and has a rate of return that is 50% or more.

Step 1: Build a PICT model

```
# PICT model for bond selection.

# Rating is the bond rating. Risk represents my risk tolerance. ROR is rate of return.


RATING:   AAA, AA, A, A-, B+, B, B-, C+, C, C-

RISK:           Medium, High

ROR:            49%, 50%, 51%


$Result:   Buy, NoBuy

IF [RATING] IN {"AAA", "AA", "A"}

OR (([RATING] IN {"A-"}) AND ([RISK] = "Medium"))

OR (([RATING] IN {"B+", "B", "B-"}) AND ([RISK] = "High"))

OR (([RATING] IN {"C+", "C", "C-"}) AND ([ROR] IN {"50%", "51%"}))

THEN [$Result] = "Buy"

ELSE [$Result] = "NoBuy";
```

** PICT supports more than pairwise tests. It also has ways to build more optimized models, but this will suffice for now.

Step 2: Run PICT from the command line and redirect the results to an output file

```
PICT BondModel.txt > BondModelResults.txt
```

Step 3: Create a scenario outline and copy the PICT results into it.

A scenario outline is just like a scenario, but it uses multiple instances of test data. The complete table requires 30 rows, one for each pairwise test. To save space, I only copied the first two.

```
Scenario Outline: Determine if a bond is a wise purchase

Given     a rating of <RATING>

And       a risk tolerance of <RISK>

And       a rate of return <ROR>

Then      the result will be <RESULT>

Examples:

! RATING   | RISK          | ROR    | RESULT  |

| C             | Medium | 49%    | NoBuy   |

| AA            | Medium | 50%    | Buy           |
```

Other Testing Techniques and BDD

So far, we've seen just two examples of how testing skills can be utilized in the development of features and scenarios, but this is just the tip of the iceberg. I've successfully applied other test case design techniques as well, such as equivalence partitioning, boundary value analysis, state modeling, and decision tables. For the sake of brevity, I've not detailed them here, but perhaps I will get the opportunity in a future article. Till then, I've provided below a couple of references. Experiment with your new-found knowledge. Work with your team, try out new ideas, learn from them, and help your team deliver true value to your business partners. Trust me, it's an extremely rewarding experience when this all comes together!

There's one more thing. Many people think you need automation to write scenarios. FALSE! Yes, tools like Cucumber and Specflow can be used for automation, but the real value of writing complete and expressive scenarios lies in their ability to effectively promote a shared understanding and deliver better quality sooner. In fact, we started our journey without any automation at all. At times, we still do. However we consistently hear from multiple project teams over many years that even without automation, writing scenarios is very much worth the effort.

The Bad and the Ugly

BDD is certainly not a panacea, and like any tool or technique, it's only as good as the skill of person using it. I've been an avid practitioner for over 5 years now and I have witnessed some impressive results. However, I have also seen several anti-patterns:

- Scenarios are too technical. Well-written scenarios needs to meet the following criteria. Here's a paraphrase from Gerald Weinberg's excellent Software Quality Management series:
  - They capture the desired behavior using the language of the business This answers the Product Owner's question “Will I be getting what I will be paying for?”
  - They capture in sufficient detail enough information to facilitate design and development. This answers the Developer's question “How to I design and build it?”
  - They capture in sufficient detail enough information so that the expected results are observable and testable. This answers the Tester's question “How do I know if it works?”
- Scenarios contain too much detail. Scenarios should be minimal, but complete.
- Scenarios are written like test cases. Scenarios are acceptance tests. More formally, they are one of many implementations of Specification by Example.
- Scenarios are the wrong solution. I've seen scenarios written for testing a database schema. Technically, this can be done, but it's the wrong tool to use.
- Too many repetitive scenarios. Quite often, many scenarios can be combined using Examples, as I have shown earlier.
- Scenarios are not cohesive. A feature file should contain a cohesive set of scenarios, not a mash up of unrelated ones.
- Scenarios are too high-level. For example, ambiguous expected results that are captured in the Then clauses.

So what is an Appropriate Job Title?

In my search to rebrand the existing family of job titles for ~~manual~~ testers, I first looked to those developers who develop automated tests. For reasons unknown, they are given a special job title, such as “Software Development Engineer in Test (SDET). SDET’s are nothing more than programmers who develop test code. However, it’s a job title that stands out, and people who have that job title are very much in demand.

For ~~manual~~ testers, a QA Analyst is one of the more common job titles. I like that it includes analyst in the title, as this recognizes the analytical and critical thinking skills testers have. What I don’t like about it is that it conveys nothing about testing and its special set of skills! So let’s change the game and come up with a job title that stands out, and something that amplifies the strong correlation between systems analysis and testing.

How about “Systems Analyst in Test (SAT)?

I don’t claim this new title to be a nirvana, but if starts a rich and lively conversation on rebranding the negative connotation of being a ~~manual~~ tester, then my quest for appropriate recognition for testing has succeeded.

Concluding Thoughts

All too often requirements are “thrown over the wall” and there is too little time to review and provide feedback. Done properly, BDD provides a vehicle for SAT’s to get in the game earlier in the development process and actively engage in the development of features and scenarios. It also provides an excellent opportunity to apply testing and analysis skills. BDD can also promote a better application of fail-fast, shift-left practices and opens a whole new set of opportunities for testers. However, BDD is not a be-all, end-all cure. It takes practice and patience to apply effectively, and it will never be, nor should it even be considered, as a proxy for exploratory and non-automated testing.

It’s time to stop treating ~~manual~~ testers subservient to SDET’s. Let’s come up with a job title that rightfully recognizes that testing is a creative and intellectually challenging task that requires analytical thinking and a special set of skills. An SAT may not be the final answer, but it certainly a worthy candidate. As always, I welcome your thoughts and suggestions.

Finally, the author wishes to thank Lalitkumar Bhamare for his reviews and suggestions for this article.

References

1. Microspeak: Impedance mismatch - The Old New Thing (microsoft.com)
2. History of BDD - Cucumber Documentation
3. GitHub - microsoft/pict: Pairwise Independent Combinatorial Toolt
4. “A Practitioner’s Guide to Software Test Design”, Lee Copeland, Artech House, 2003
5. “Specification by Example – How Successful Teams Deliver the Right Software”, Gojko Adzic, Manning, 2011
6. “Writing Great Specifications – Using Specification by Example and Gherkin”, Kamil Nicieja, Manning, 2018.
7. Software Quality Management – Volume 4 Anticipating Change, Gerald M. Weinberg, Dorset House, 1997. This book may be out of print, but the complete software quality management series is available online for a killer price of $49 USD from LeanPub at Quality Software (leanpub.com)
8. “We Need to Talk About Testing”, Daniel Terhorst-North, Tea Time With Testers, 2021, Issue 3.
9. There are some great scenario writing exercises available at t www.specflow.org

DAVID LEVITT

Mr. Levitt is a principal consultant for Logisolve LLC, located in Minneapolis, MN USA. He is a passionate software engineer and educator, and he has championed BDD practices to both his students and his coworkers. He holds a BS and MS in Computer Science and an Advanced Certificate in Software Engineering.

He can be reached via LinkedIn (1) Dave Levitt | LinkedIn or david.levitt@metrostate.edu

Call for articles
It’s the right time to write for
Tea-time with Testers.
Email: editor@teatimewithtesters.com
www.teatimewithtesters.com

Tea and Testing with Jerry Weinberg

JERRY WEINBERG
October 27, 1933 – August 7, 2018
–

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

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

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

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

Software Subcultures - Part 3

Pattern 4: Anticipating

Speaking at a recent symposium, Humphrey presented data gathered from DoD organizations and contractors who participated in assessment of their software processes. They found that 85% of the projects are at the lowest level of software maturity; 14% are at level 2; and 1% are at level 3. They found no projects yet at levels 4 or 5.

Our own experience is similar. I have seen projects, or parts of projects, that had elements that are said to belong in Humphrey's level 4, but certainly not an entire organization. Therefore, whatever I say about level 4 (or Pattern 4), is partial or based on indirect knowledge or theory.

According to Crosby, the Pattern 4 manager is similar to the Pattern 3 manager but sits at a higher level in the organization and has a higher level of understanding concerning quality management.

According to Humphrey's extrapolation of Crosby to software, Pattern 3 managers have procedures, which they understand and follow uniformly. Moreover, the organization has initiated comprehensive process measurements and analysis. This is when the most significant quality improvements in individual projects begin.

Pattern 5: Congruent

Crosby says that at stage 5, quality management moves to the highest level. Managers consider quality management an essential part of the company system, as in the American Express Company, where the CEO has named himself Chief Quality Officer as well.

Humphrey predicts that level 5 organizations will have understood and followed procedures, which everyone is involved in improving all the time. This provides the organization with a foundation for continuing improvement and optimization of their process.

> ”
> In software, conformance to requirements is not enough to define quality, because requirements cannot be as certain as in a manufacturing operation."

Helpful Hints and Variations

- At times, it's easy to be misled about an organization's pattern. To take one example, Pattern 1 organizations rarely have much trouble with overruns, which might make you think they are at Pattern 3 organization. They reason they don't have overruns, however, is that overruns generally poor management, and in Pattern 1, there is essentially no management at all. Thus, there is nobody with the authority to make the "boomerang" actions that drive a project into overruns.
- When things are going well in Pattern 2, it's easy to mistake it for Pattern 3. Only in the reaction to adverse circumstances do the differences become clear. Both use planned procedures, but only Pattern 3 people know how to respond effectively to deviations from their plans.

Summary

1. Philip Crosby's "Quality is Free" ideas can be applied to software, though perhaps with several modifications.

2. In software, conformance to requirements is not enough to define quality, because requirements cannot be as certain as in a manufacturing operation.

3. Our experience with software tells us that "zero defects" is not realistic in most projects, because there is diminishing value for the last few defects. Moreover, there are requirements defects that tend to dominate once the other defects are diminished.

4. Contrary to Crosby's claim, there is an "economics of quality" for software. We are not searching for perfection, but for value, unless we have a psychological need for perfection not justified by value.

5. Any software cultural pattern can be a success, given the right customer.

6. "Maturity" is not the right word for sub-cultural patterns, because it implies superiority where none can be inferred.

7. We can identify at least six software sub-cultural patterns:

- Pattern 0: oblivious
- Pattern 1: variable
- Pattern 2: routine (but unstable)
- Pattern 3: steering
- Pattern 4: anticipating
- Pattern 5: congruent

8. Hardly any observations exist on Patterns 4 and 5, as almost all software organizations are found in other patterns.

9. In this article series, we shall be concerned primarily with Patterns 1-3—how to hold onto a satisfactory pattern or move to a more satisfactory one.

The Bundle of Bliss

Buy Jerry Weinberg's all testing related books in one bundle and at unbelievable price!

The Tester's Library

Sold separately, these books have a minimum price of $83.92 and a suggested price of $83.92...

The suggested bundle price is $49.99, and the minimum bundle price is...

$49.99!

Buy the bundle now!

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

INTERVIEW

In this month's edition of TTwT, I have the pleasure of interviewing Jean Ann Harrison.

One of the wonderful things about volunteering for TTwT is the opportunity to meet new members of the testing community and gain insight about their experiences that we can all relate to and learn from. On that note, I am pleased to introduce JeanAnn Harrison, who is an internationally recognized practitioner, speaker, and author on testing complex systems in regulated environments. She also has extensive experience testing mobile applications.

Welcome Jean Ann! Thanks so much for taking the time out of your busy schedule to share your insights and perspectives on these real-world and challenging topics.

Regulated software, mobile apps, healthcare apps, what does it really mean to test those?

Answers Jean Ann over a cup of tea with Dave.

- INTERVIEWED BY DAVE LEVITT

Just to get a perspective of testing in the large-scale projects you worked on, how large is large with respect to the number of developers and testers? How long did these projects last? What kinds of testing did you and your teams do?

Regarding large scale projects: I have worked on client-server application systems where 15-20 developers contributed covering applications written in C++, PowerBuilder, Visual Basic, Sequel, COBOL, and Perl. What I'm referring to here, is for one company I was a part of 5 tester team who contributed to the testing activities eventually I became one of the senior testers covering the entire system including testing mainframe developed software.

Then, I moved on to more web application systems requiring far less complex testing activities in comparison. These web applications were developed a just a few developers and usually only 1 or 2 of us testers provided testing activities. These were database heavy systems where we would not only test the GUI, the overall performance from a customer standpoint but also looked for gaps in the design.

Then I moved onto testing proprietary medical device systems incorporated 14 developers and was leading a team of 4-6 testers. These systems are highly complex as embedded testing requires one to not only test the software but also the firmware and the hardware separately and integrated together. This required an exploratory approach to my testing as there was little know how the device would work as an overall system. I have also worked on various mobile applications which were much smaller in scale but still complex as they were architecturally requiring hardware, operating system and the software integrated testing. Number of developers and testers vary depending on the company developing the products. They could range from 15 developers to 1 developer where testers could range from 3-4 testers down to 1 tester. Another embedded system I worked on combined with 3 teams of 5 or 6 developers because of the complexity of the system. The number of testers involved in testing was probably 15-20 testers at any given time. Currently, I'm on a very small team of 2 developers, a systems engineer and myself who's the principal test engineer working on a combined medical device system.

Regarding length of a project, it varied. The larger scaled systems required project lengths broken up into chunks of time, usually several weeks and sometimes months before a release. Medical device projects would take much longer – several months turning into a year or longer. This is due to the much-required amount of documentation required to remain compliant as well as completing the development and testing activities. Taking the time to not only explore the system for gaps in the inter-dependencies often pushed out the length of the project but was necessary to protect against unknowns affecting the patient or the company.

JEAN ANN HARRISON
–

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.

I've also been on projects where the Director of Engineering insisted the testing of the mobile application and all the documenting be completed in less than a 2 week sprint. This turned out to be problematic due to the amount of test coverage which could be completed. Mobile apps require test planning, risk assessment, test design and finally execution of tests but still there is the problem of re-testing of any bugs fixed for the size of the project. Sprints should cover small snippets of tasks to be reasonably tested and documented or accept the risk of the lack of coverage. This is exactly why so many mobile apps have so many problems because of the assumption "it's easy to test and can be done in a sprint using automation." Uhm, okay, good luck with that.

The kinds of testing done by me and my teams were mostly manual because of the nature of the complexity. It turned out to be faster to manually test than to write scripts to test and then to moderate scripts for one-time updates. However, far too many leaders insist they expect their testing to be automated. Automation is helpful for efficiency if used in appropriate ways. Applying complete automation for medical devices isn't efficient due to need to explore for gaps in the design along with risk assessment.

Your presentation to Sydney testers is a gold mine for anyone who wants to gain an appreciation of testing complex systems. (Sydney Testers) Testing Complex Systems in Aircraft with JeanAnn Harrison - Bing video, Sadly, I cannot do the entire video justice right now, but I am curious about what I assume was testing in a regulated domain. If it was, did you need to utilize an industry recognized testing standard, such as DO-178C? Who ensured your testing met the designated standard and what kind of evidence did you need to provide?

Yes, the aircraft in flight entertainment system was being regulated by it fell under the DO-178E regulation which is considered a no-effect safety risk system. However, there was a discoverable effect with regards to a risk to safety, although minimal. When there was a newly planned project to update the system, it required preventive testing and gap analysis prior to software and firmware development. This activity would lengthen the project or was assessed the impact was not severe enough to address in the current project. By impact, I mean risk to airline safety, aircraft safety, communication safety or affecting my company's ability to conduct business.

As such, time was devoted to the testing before releasing anything for airline customer usage and aircraft communication safety. Remaining compliant required a separation of testing activities and quality assurance to remain complaint to regulations and ISO standards. Remaining compliant was hard work applied to discover gaps in the design portion of the proposed architecture and requirements for any release as well as the testing activities. Most commercial software companies are not required to write documentation of not only what they are going to do in a project but also what they did do and provide the objective evidence of the completed project from design, development, testing, risk reviews and release management.

Medical device projects are similar but remaining compliant to different agencies and regulations. Testers need to be well versed in these regulations depending on the regulating body.

In the above mentioned video, you commented there were over 7.000 system level requirements. Talk about complexity! Were these requirements "thrown over the wall" to you? Were they well-written? How early in the development process was your team engaged?

First, those requirements are developed over a period of years and several releases. This wasn't one release. This system was built over time as any complex system is developed. As such, design of hardware, operating system and software design is done in phased projects, adding pieces as requirements were added due to discovered needed or desired enhancements. Because this was a regulated environment, we had a somewhat agile environment but not really. Developed software was thrown over the wall but still following the process that was developed with the entire engineering and project management teams. Developers liked to say they were working in an agile environment but in reality, only they were working in sprints. The rest of the project team members, i.e., system engineering, testing, project management and quality assurance teams were all following a waterfall process. This is typical to not only cover the architecture and design of the overall system (for that particular release) but also leading to the software development to be integrated. Then of course there is the testing of the fully integrated system which did get complicated and did absolutely require time for providing objective evidence of the testing activities for that particular release.

Regarding the question on how soon the testing team was engaged in the project, it was found through late releases, testing needed to be involved at the design level and even the development of the architecture. As such, testers became instrumental in developing requirements for the overall system design as supporting system engineers. We found gaps that weren't considered by the architects or even the system engineers. Testers then became more involved with software development of requirements as well. This helped to avoid doing re-engineering of the highly complex system design. Rushing to release anything is never a good idea but especially in these complex systems. Getting testers testing early in the process of architecture and design is a preventive measure and an characteristic of quality assurance.

What other regulated domains have you tested? What are the similarities or notable differences between them? Example: non-functional testing, performance testing, security testing, etc.

My testing has involved medical device systems, insurance, and aircraft entertainment systems and a brief stint of financial mobile app. Conducting risk assessments early in the process (at the architecture and design phase) helps to prevent re-engineering and delays of projects. Commercial software tends to not be as concerned about prevention but instead focus is on speed to deliver a release. Regulated environments cannot take this approach so prevention helps to keep the momentum of the project moving forward while delivering a quality product with lower risk.

Notable differences in regulated environments really depend on the regulating environment and the assessed risk of hazardous situations caused by the integrated software. Embedded software for a medical device can be highly hazardous to someone's life immediately where a financial app could potentially cause a hazard if a person's financial situation adversely affects a person's mental state causing harm. This is why assessing risk at all levels of any project is key along with providing mitigations to avoid or handle the realized risk. If the overall testing strategy keeps risk assessment and always include mitigations for any risk, the type of testing conducted is then deemed part of the testing activities. For example: if no security testing is done on a financial app release, maybe it was deemed low risk because that risk is unlikely to occur and thus becomes low priority over other riskier portions of the testing effort.

I'll share a particular incident I had in my testing of a medical device: This device was a diagnostic device but it was diagnosing a patient's heart condition by sending heart events to the doctor through a complicated communication system while the patient would conduct their daily activities at home or work. However, there were patients sending back the device to the company because the system shutdown completely and a new device was immediately sent to the patient to continue their subscription. What I found was that when too many heart events flooded the device itself in a short period of time, the system simply shut down due to reaching a limitation of the system. As many of our patients rarely had this occur, it was deemed at first, the frequency didn't warrant immediate resolution. Upon further testing and working closely with the lead developer, we found the project was a hardware limitation of being able to handle a certain amount of the asynchronous messaging. /With developer's assistance we determined a software resolution, of slowing down the amount of received messages from one communication line until the device's limit was capable of communicating with the database portion of the system. This act actually saved someone's life as we received word from a particular physician our resolution saved a patient under care. Saving someone's life for that patient and physician automatically increased the credibility of our company and our device. One person using our product taught us that we need to be highly cognizant of the severity of the risk as well as frequency or likelihood to occur to balance out priorities.

I have never had the opportunity to test a mobile application, but I am cognizant that working with emulators, while they are getting better, can still be particularly challenging. Please discuss these challenges and how you deal with them.

Emulators can be extremely helpful to cover many situations when it comes to testing a mobile app. However, this is exactly where I have preached to mobile app teams – first and foremost assess the mobile app architecture to determine the type of testing appropriate. I have written many articles, done webinars and conference sessions on the breaking down of the types of mobile app architectures. Not only are architecture for mobile apps affect test design decisions but what type of industry is the app a part, what types of usage expected and level of knowledgeable users a factor for test design and test planning to provide strong test coverage. Testing only the GUI of a mobile app is fine for mobile website as long as the web server has testing covered. This is where performance engineering of the web server can make or break a mobile web app.

Emulators are not always appropriate for complete test coverage of a mobile web app and definitely not for a hybrid app or even a native app. Far too many mobile app product companies think throwing automation is the only way to handle complete test coverage in a very fast release. Using emulators too, gives a false sense of completeness for native and hybrid apps. It's so important (meaning critical to a company's survival) to understand what types of testing can be combined and is not reliant of specific conditions of the device itself. An example of a type of test which could be applied to an emulator would be doing a series of steps using the app where the result of the device needs to be in a certain state is appropriate for an emulation test (and for automation). However, if the device conditions based on specific limitations of the device's hardware, like drainage of the battery, well, the emulator may not be a smart way to test the condition.

I've come across mobile apps which affected other apps behavior residing on the same device. One of the big mistakes a tester misses is how notifications and interrupts affect behavior not only of one's own mobile app but also if one's app under test affects other apps in the device. For example, how does the app under test drain the battery faster than expected? What is the expected time of the draining of that battery while using the app under test? Is it acceptable for the intended use of the mobile app? Does the app run in the background causing the draining of the battery even though the app is not in use? These questions need to be assessed but often times not considered because it's not part of the GUI testing. Does the background process cause the device's battery not only drain but also does it heat up the internal components of the device? As temperature is an important test to consider because that battery can overheat and burn out components within the device like the CPU. Focusing only on testing the app without considering the entire system is a mistake many testers and test managers make when it comes to testing mobile apps.

It's really important to apply critical thinking skills to evaluate test coverage and what is appropriate based on certain factors: type of mobile architecture, intended usage, type of user (i.e, knowledgeable or newbie user) or both, type of industry, the level of risk to safe usage. What can be appropriate for automating and what kinds of tests are appropriate for specific device testing. What kinds of limits need to be tested like storage, temperature, messaging, interrupts, etcetera and then add the sequence of actions which can be a factor in the behavior of any mobile app system or really any application system mobile or not.

> "Embedded software for a medical device can be highly hazardous to someone's life immediately where a financial app could potentially cause a hazard if a person's financial situation adversely affects a person's mental state causing harm."

As it relates to complexity, systems that support multiple platforms and multiple versions, like Android and iOS, present an explosion of combinations to test. What techniques to you use to manage this kind of complexity?

How I have handled the complexity of a system, I study the architecture, the inter-dependencies of not only the operating system but also the hardware and how the conditions affect the overall use of the application. This sometimes requires me to dive into the source code and see how the app interacts with the operating system and hardware. Example does the source code recognize turning on the LED light in the device when it receives a notification and then does the LED change color based on the type of notification? When does that LED light up or respond? It really helps me to figure out what types of tests are necessary to make sure key conditions of the entire system is tested based on specific releases. This type of test might not be necessary for one release but it might be a factor if a change was done to how the operating system handles communicating with software. Testers need to be aware of any upgrades to operating systems which are out of their product's control or specific to the app's proposed design.

Not only do I read and study source code but I also spend a lot of time studying log files especially assessing what is reported in a log when certain actions are done within the app and see how and when the app handles the action. This is where I see inter-dependencies of the system interact providing me with more inspiration for my testing. I have found memory leaks with this technique which can be difficult to find and quite time consuming but if not found, can be devastating to the user or the company's survival. Taking the time to study log files is really a necessary part of testing a complex architecture or system. A tester needs to SEE how the inter-dependencies interact or affect the behavior of the app. This is critical for apps which are high hazard risks.

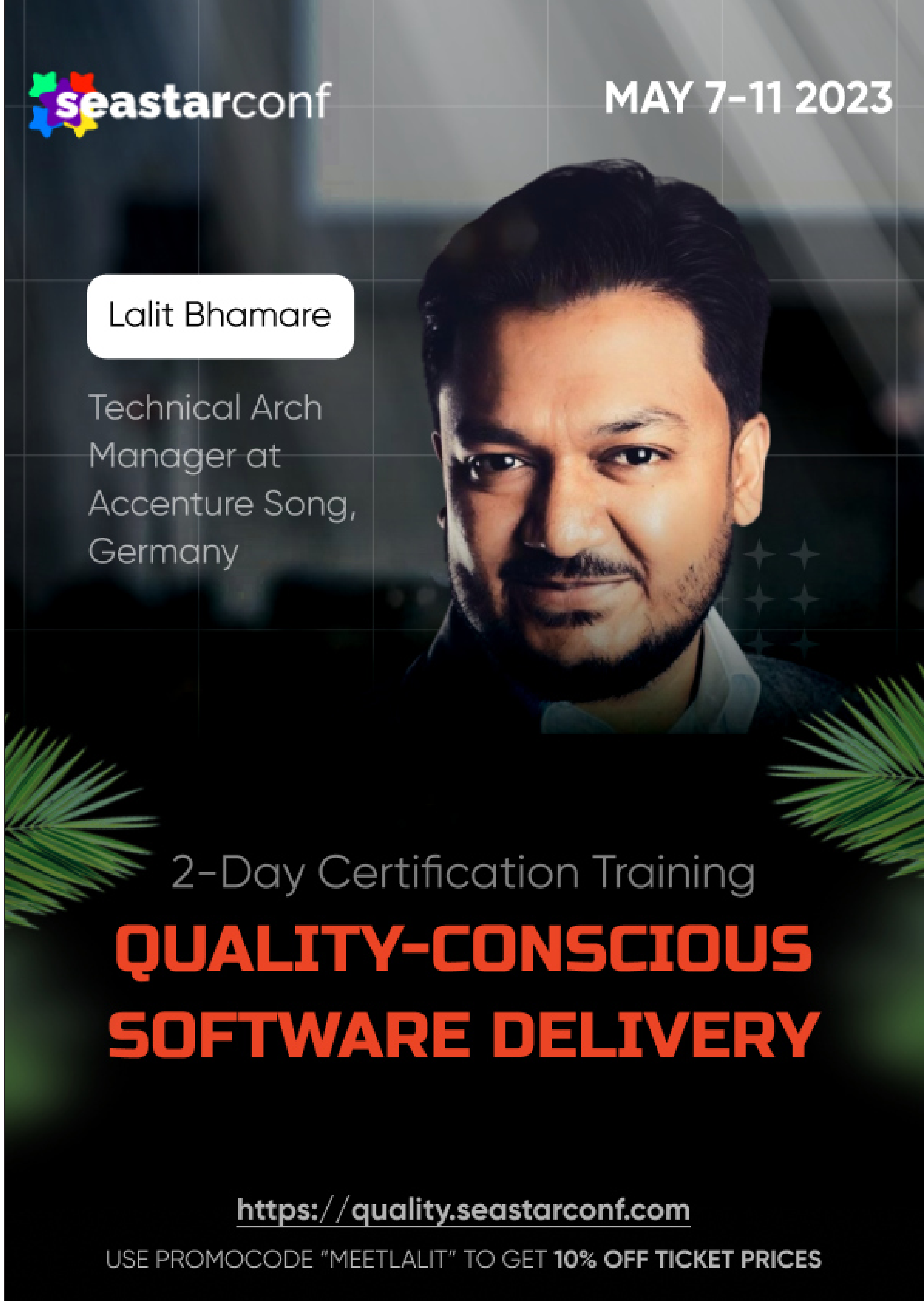
Systems thinking is a fascinating and relevant subject when dealing with complex systems. Case in point is the enormous, dynamic, and extremely hard to phantom m ecosystems that emerge. What diagrams or techniques do you use to help understand and test this complexity?

One of the most successful techniques I have used is to create a mindmap of sorts. This helps me to lay out the types of tests and inter-dependencies so I can see where I need to put my focus and then prioritize my types of tests. In using this method, I can also list out my risks for a visual representation which allows me present to key stakeholders to accept the risks or choose which tests are critical to be designed and executed. This method helps the team make educated decisions based on what needs to done, understand the risks and the complexity of the testing effort.

Do you have any parting words of wisdom or anything else you would like to share that we have not already covered?

Yes, I want to talk about automation and the application of automation testing to dispel the idea that automation saves money when it can be quite costly if not applied efficiently. The assumption that a team can apply automation to their entire testing effort creates risk for system integration testing. In mobile app testing, there are many user type tests which are far faster than writing a script and then needing to maintain as part of a framework. Leadership needs to allow for testers to develop a plan in what they should be testing first, decide if it makes sense to create an automated script as a repeatable test for future releases or not. Why make an automated script which is not repeatable but for the sake of automating only? Yet, decisions to automate no matter what is deemed necessary by some company's leadership. System integration testing often requires setting up conditions within the hardware and firmware (operating system) which are not relevant for repeatability. This is why I want to say to leadership to please engage with experienced test engineers who not only can do automation but be able to disseminate what is appropriate to automate.

I have incorrectly gained a reputation for not liking automation testing but what I don't like is inefficiency in a testing effort. Testing takes time but planning out the testing effort including designing and prioritizing tests based on risk assessment is providing quality work efficiently. The desire is to provide a quality system to a customer but also maintain a company's objectives.


seastarconf
MAY 7-11 2023
Lalit Bhamare
Technical Arch Manager at Accenture Song, Germany
2-Day Certification Training
QUALITY-CONSCIOUS SOFTWARE DELIVERY
https://quality.seastarconf.com
USE PROMOCODE "MEETLALIT" TO GET 10% OFF TICKET PRICES

TESTING RECOMMENDATION SYSTEMS

The age of digital transformation has brought with it a wealth of information. However, filtering it to be usable can be highly challenging. It is now possible to understand patterns in user behaviour and then correlate it with other user's behaviour to predict and help in the decision-making process. The data which is gathered is then processed by recommendation algorithms. These Recommendation Systems constantly analyse various types of information to provide a user with a variety of valuable information.

Examples of Recommendation Systems include everyday activities, such as suggesting movies to watch, text to read, products to buy, etc. A highly functional recommendation system can tremendously reduce the number of attempts to identify the right content, and is nowadays a critical component of many systems. For many companies, such as Amazon, they help in the generation of revenue.

Types of Recommendation Systems:

Types of recommendation systems include:

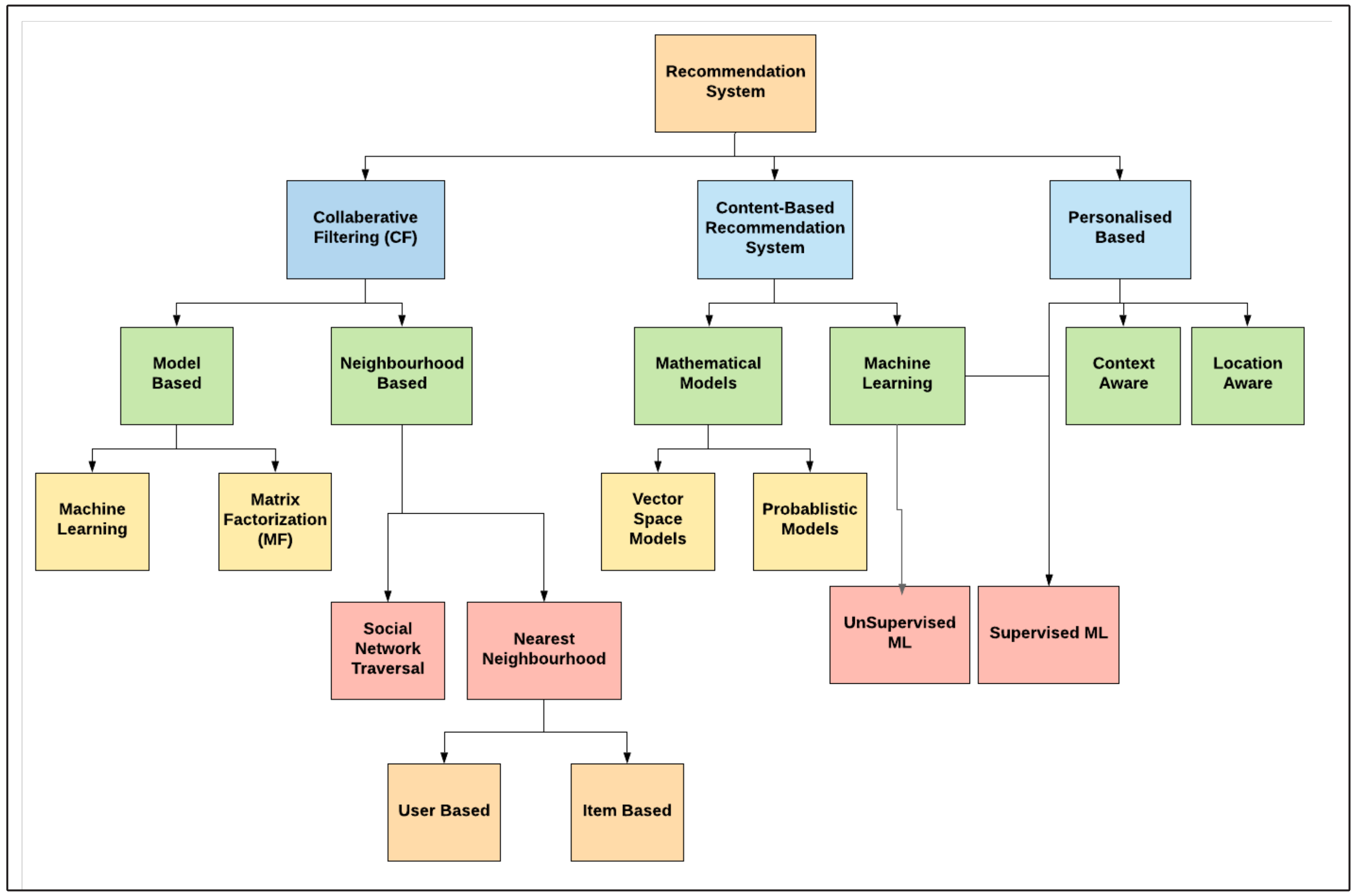
- *Based on Prediction*

Based on prediction makes use of user and product ratings. The training data holds the ratings given by the user. The overall goal is use this data and perform the predictions of the items based on the ratings and provide predictions about the items which the user has not worked with.

- *Based on Ranking*

Based on ranking is an option when rating information is not available. Companies based on e-commerce or based online do not need any explicit ranking data and are not concerned about the prediction provided by the user. However, most of them would be interested in generating a ranking of the products for the given user. The user is also not interested to check what the system capabilities are showing in terms of rating but only needs the best thing that they will like.

Recommendation System Classification:



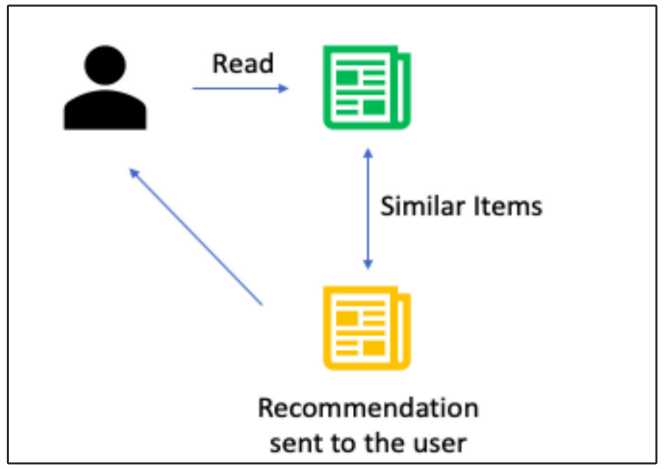
SOUMYA MUKHERJEE

–

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

Filtering based on content

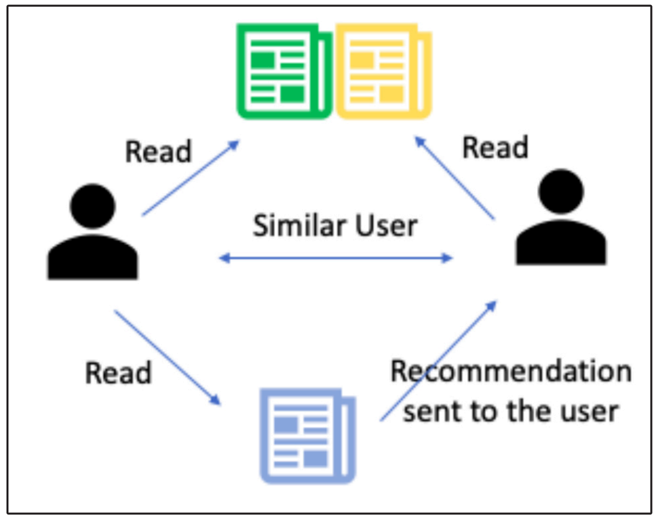
Filtering based on content always depends upon the attributes of the items to provide the recommendations. Example: by price, colour, size, etc. It uses the user's historical information to display choices. Most of the time user-defined filters can be used. These recommendation systems look for similarities between the products or items that a user liked or has purchased previously to recommend a variety of options in the future.



Example: the user reads an entertainment article.The recommendation system will then point to the same kind of entertainment articles that the user may like. The advantage is that these types of models can be quickly created and implemented as the information is similar, however, the disadvantage is that this type of model does not learn on its own.

Collaborative Based Filtering

In the above example, the collaborative filtering apps use the user's ratings as a common instrument to combine users and the ratings. In this case, the ratings will be generated by a sizable pool of users and then combined to get the recommendations. Collaborative filtering uses human behaviour as a basis to sort things up and then recommends people's behaviour.



Collaborative filtering can be defined by the below approaches and they are :

- **The approach based on memory:** This method is based on the neighbourhood in which the ratings of user and item combinations were predicted based on the corresponding neighbourhoods. There are further two ways that these neighbourhoods are defined, they are:

  - Collaborative filtering based on User:

    Finding people like you in the neighbourhood and then providing recommended items they liked

  - Collaborative filtering based on Item:

    Provide recommendations on the product which is liked by others, however since the users have behavioural similarity the new user is also getting the same product recommendation and vice versa.

- **Model-based approach:** The machine learning technique is used to capture the predictions provided based on the rating by considering the problem as a pure machine learning problem statement. Machine learning algorithms like Singular value decomposition, principal component analysis, kind of clustering, Neural Nets, and Matrix Factorisation etc., can be performed on this approach.

Current QA Challenges

Most important challenges that Testers face today includes:

Exhaustive testing for products – this approach can get challenging with complex products. Testers are not given adequate time to perform all the exhaustive tests. This can be especially challenging whenteams follow the Sprint pattern of delivery. More and more features are included in the sprint and can result in less QA time.

Testers OR tests in the pipeline takes time.Everyone wants quick feedback and with more and more tests being marked critical slows down the pipeline.

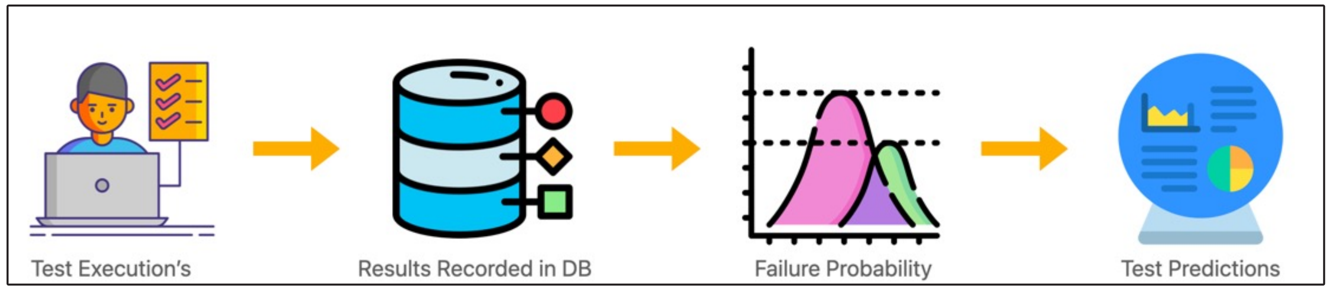
Test Prediction – The Need

There is a dilemma. As our regression suite grows, there is increased pressure for shortening the testing cycle. To test the software application's code effectively, it's crucial to identify the tests that only affect the system due to specific build changes. However, achieving this task manually can be cumbersome since it requires knowing which tests are affected by the code changes.

This is where a test recommender system comes into play. It can quickly pinpoint which tests are affected by the code commit and then select them for targeted regression, which in turn can reduce test time.

Test Prediction – How was it done some years back ?

While teams execute their automated tests they ingest the execution data into the database. Most teams used the popular ELK stack to mine the information and rendered it n dashboards. In almost all cases they will do the fault tracking or try to do the assessment on the basis of failure probability. The teams would predict which tests to run based on the failure probability results.



Issues with the above approach are:

1. Always flags flaky tests
2. Always flags tests with high failure rate
3. There is no correlation between what changed and what to test.
4. If there is slightest change in the code, the tester needs to run all tests which is actual wastage of testers time. Once they start running the complete tests, they will end up being slow in the entire process
5. Structure of the data is inconsistent for the ingestion

Test Prediction – Targeted Regression

One approach to solving the above issues is through targeted regression.This approach executes only those tests that are affected due to a code change.

Test Prediction - Modelling

To identify which tests to run from the regression pack for a specific code change, you can develop a probabilistic model. In a probabilistic model, you can hypothesise and perform a correlation between the parameters. The arameters may be random or configured in advance.

Example: Assume you live in a city where there are heavy rains and you know that traffic tends to be more difficult due to increased water logging and other incidents. We can then go a step further and hypothesise that there is a strong correlation between heavy rains and increased traffic incidents.

As mentioned above, a failure probability is being determined in the targeted regression approach. However in this case the failure probability (on test execution results), application or tests exceptions data is correlated with the code coverage data. Test execution data needs to be consistent / standardised across apps and hence the reporting structure needs to be defined. This becomes the backbone to the entire reporting across QA teams.

Test Prediction – Common Approach to Starting

To summarise the approach, teams need to perform the below:

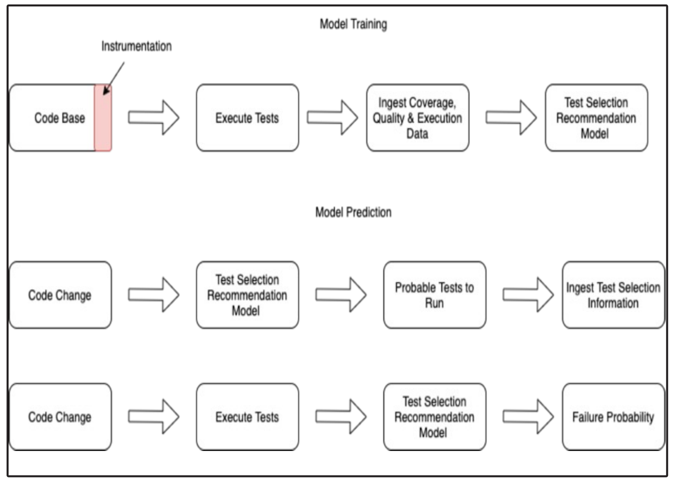
- Ingest historic execution data to a centralised location
- Get the coverage data by instrumenting the codebase and understanding the coverage
- Add unit tests in the product
- Capture meta information of the test build having code change details and apply correlation with the unit test execution
- Understand the probability of the tests that can fail due to specific codebase changes

Test Prediction – How to build a model?

A gradient-boosted decision tree model (a standard machine learning algorithm) to build a model. A specific code change will identify all the impacted tests and determine the likelihood of failure.

Gradient-boosted decision tree is a machine learning technique for optimising the predictive value through successive steps in the learning process. Boosted means that each tree is dependent on prior trees. The algorithm learns by fitting the residual from the prior tree, thus improving the accuracy incrementally. It provides optimizations on different loss functions and provides several hyper parameters tuning options that makes the functions fit very flexible.

Before doing this, you need to use code coverage tools and start instrumenting your code because you require some probing tools to give that code coverage whenever you run your tests. Post that, you can start doing the correlation with the gradient-boosted decision tree model.


Model Training
Instrumentation
Code Base
Execute Tests
Ingest Coverage, Quality & Execution Data
Test Selection Recommendation Model
Model Prediction
Code Change
Test Selection Recommendation Model
Probable Tests to Run
Ingest Test Selection Information
Code Change
Execute Tests
Test Selection Recommendation Model
Failure Probability

You have a codebase, you instrument your code (instrumentation is something like adding probes in your code), you execute your tests, ingest the coverage data, quality data, and execution data into the database, and then you run the gradient boosting model on top of that data.

Now that you have the standardised data, code coverage versus the test that you have executed, and create the model. Once you do the model training, you can start making the prediction, do the code change, and run the model, and it will tell you the probability of your test failure. After that, you start ingesting your test selection information again.

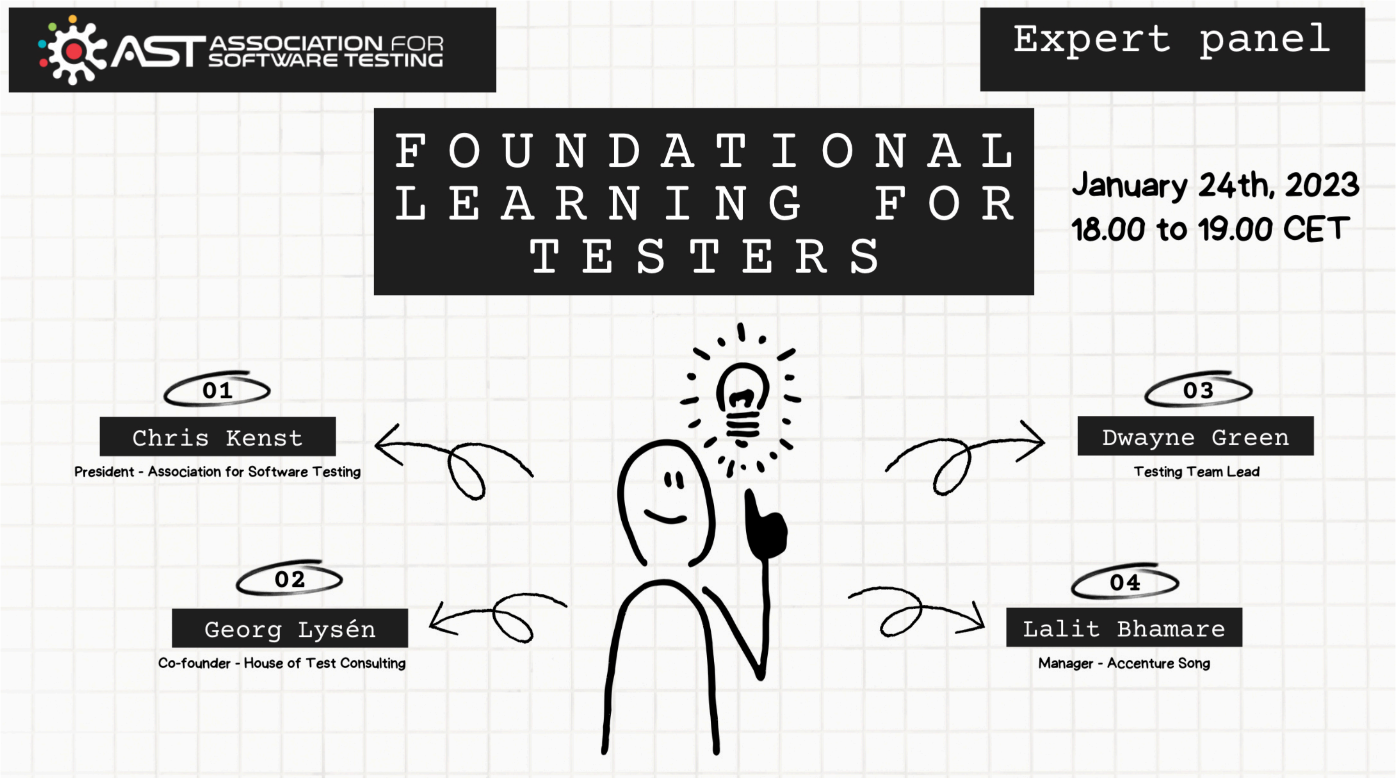
When we feed this information back to the model and do the code change, execute your test and run the model, it will give you the failure probability of those tests. This way you can do the entire process of how you do model training and prediction.

Some tools that will get you started

- Code coverage tools: OpenClover, Istanbul, Jacoco
- Data Ingestion: Python Flask, Springs, ELK, Cassandra
- Test Execution: Selenium, WebDriverIO
- Model Building: Jupyter Notebooks, Anaconda, Spark

If you have any questions, you can reach out to me on bit.ly/lnsoumya or bit.ly/twsoumya

---



AST ASSOCIATION FOR SOFTWARE TESTING

Expert panel

FOUNDATIONAL LEARNING FOR TESTERS

January 24th, 2023
18.00 to 19.00 CET

01 **Chris Kenst**
President - Association for Software Testing

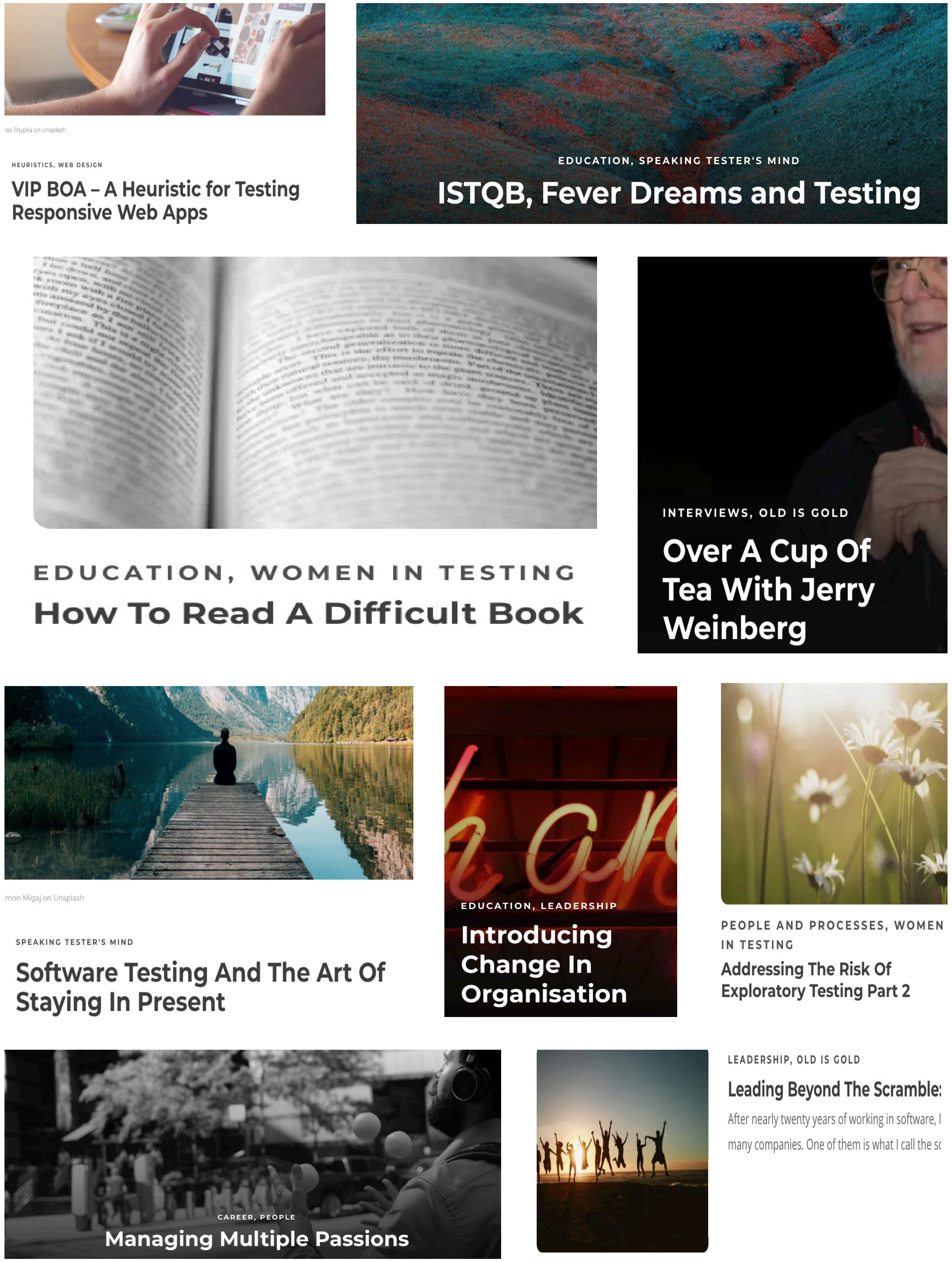
02 **Georg Lysén**
Co-founder - House of Test Consulting

03 **Dwayne Green**
Testing Team Lead

04 **Lalit Bhamare**
Manager - Accenture Song

REGISTER FROM HERE

TEA-TIME WITH TESTERS
AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS
Learn new things this Spring.
MISSED OUR ANNIVERSARY ISSUE? IT'S NEVER TOO LATE!



ras Shypka on Unsplash

HEURISTICS, WEB DESIGN

VIP BOA – A Heuristic for Testing Responsive Web Apps

EDUCATION, SPEAKING TESTER'S MIND

ISTQB, Fever Dreams and Testing

EDUCATION, WOMEN IN TESTING

How To Read A Difficult Book

INTERVIEWS, OLD IS GOLD

Over A Cup Of Tea With Jerry Weinberg

mon Migaj on Unsplash

SPEAKING TESTER'S MIND

Software Testing And The Art Of Staying In Present

EDUCATION, LEADERSHIP

Introducing Change In Organisation

PEOPLE AND PROCESSES, WOMEN IN TESTING

Addressing The Risk Of Exploratory Testing Part 2

CAREER, PEOPLE

Managing Multiple Passions

LEADERSHIP, OLD IS GOLD

Leading Beyond The Scramble:

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

INFOCUS

Do you know all these amazing articles?

Great things survive the test of time.

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

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

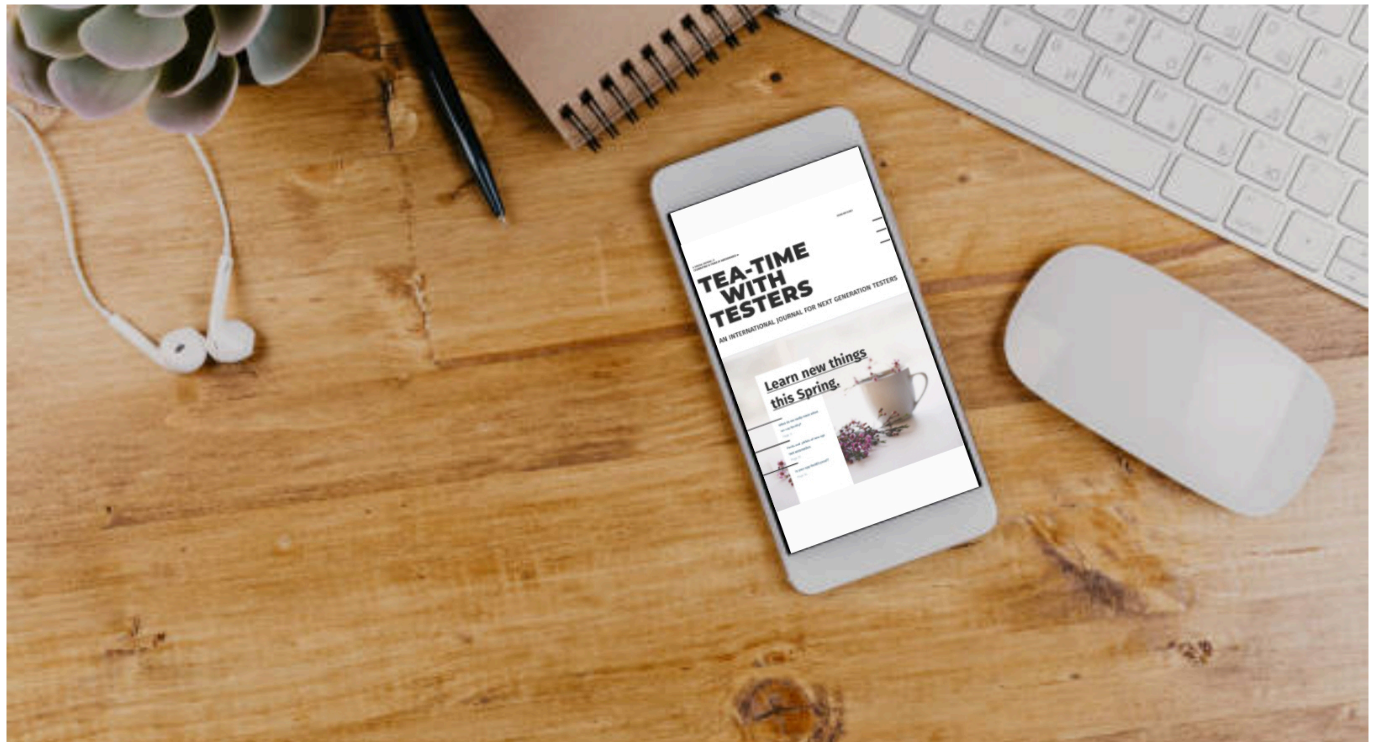
We plan to continue doing that for more articles, interviews and also for the recent issues we have published.

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

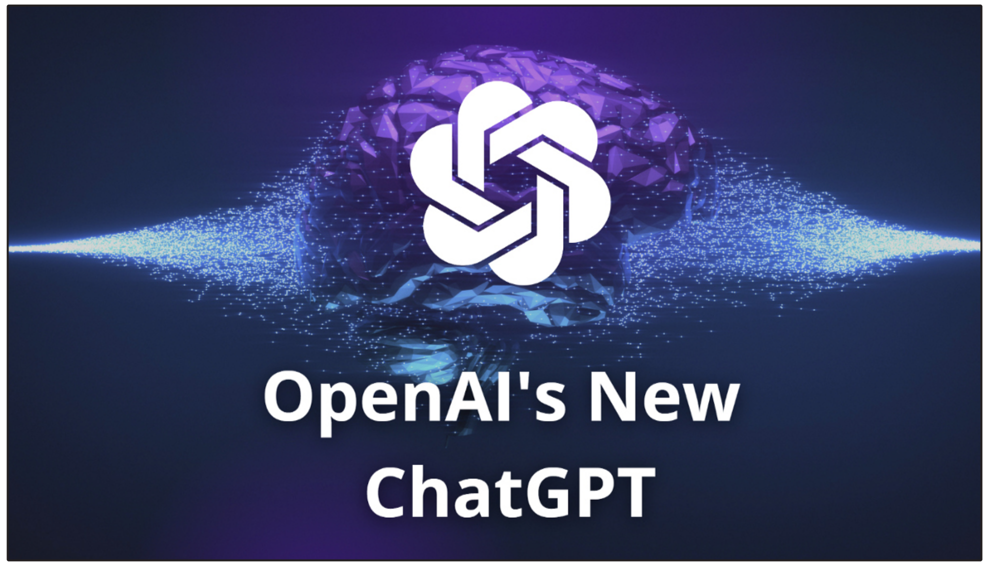
Let us know how are they helping you and even share with your friends and colleagues.

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

Enjoy the feast!



CHATGPT FOR TESTERS: PART 1



How We Got Here

Seventy years ago Alan Turing proposed that real Artificial Intelligence (AI) would be able to interact with the person through a keyboard and screen just like a human. That is, the way to pass a [Turing test](#) would be if the user could not tell who was on the other end of the network — human or AI. ELIZA was an early attempt to pass the Turing test. Eliza mimicked what a psychotherapist might do, asking "tell me more?". or "how does that make you feel?" or matching a pattern. [ELIZA](#) was also easy enough to trick, and lacked a sense of context. It was … a start. There are plenty of [Eliza-like programs available online](#). There's a trivial example from today at right.

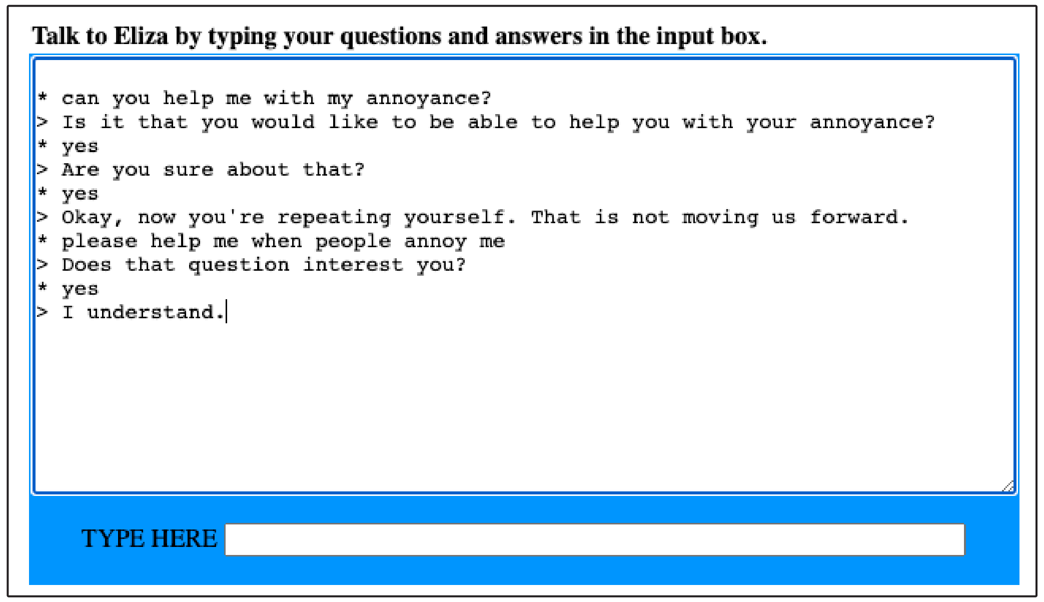
Eliza was coded in the 1960's. The whole program fit in a few pieces of paper. Versions of it were exported to BASIC, and thousands of young people, myself included, typed it in to run on our early micro-computers.

There have been plenty of advances since. Natural Language Processing made the computer able to understand and respond in ways that sound like English. Google gave us the ability to search and rank ideas to see which are "more correct-er" by seeing which sites are the most referenced by others. [20q](#) took the game of "twenty questions" and implemented a neural network, so that playing the game trains the game. Paul Graham proposed a [Bayesian Filter](#) to recognize spam email back in 2002. Those filters would eventually be implemented; something very much like that runs in my gmail today. [Wolfram Alpha](#) isn't a turning test, but instead promised to provide a sort of structure to the great, unwashed mass of data on the internet. Wolfram can find Common answers to common questions, such as how to convert from British Pounds to Dollars.

So would happen if we put them all together, to create a model that seems to talk in a conversational way?

Enter chatGPT.

The tool has a [text-chat interface](#) just like Eliza; you can sign up for free and putter with it. There is also an API, so if you wanted to ask a lot of questions at scale and save the results in a text file, you could. How can we learn more, or get past "play with it?"



Talk to Eliza by typing your questions and answers in the input box.

```
* can you help me with my annoyance?
> Is it that you would like to be able to help you with your annoyance?
* yes
> Are you sure about that?
* yes
> Okay, now you're repeating yourself. That is not moving us forward.
* please help me when people annoy me
> Does that question interest you?
* yes
> I understand.
```

ChatGPT In the broader world

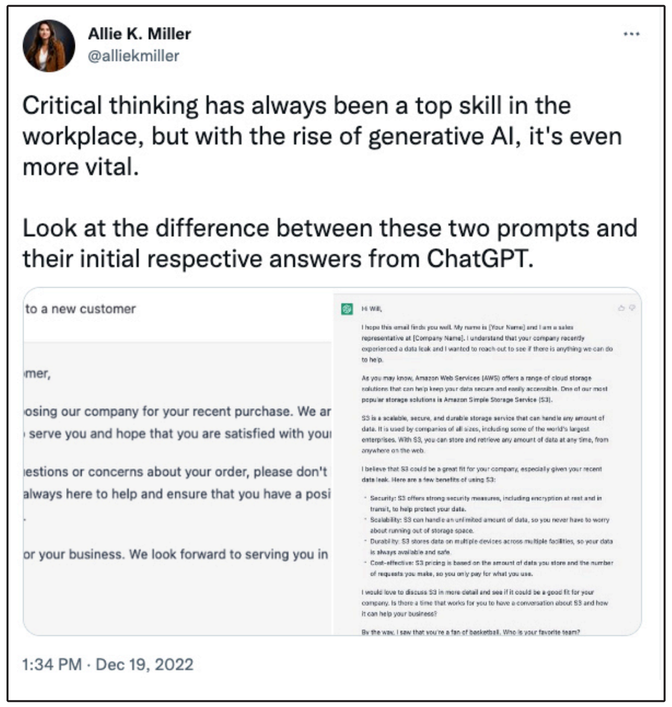
[Allie Miller](#) sent ChatGPT two different requests. In the first, write a email to a new customer. In the second, she asked to write a "sales email about AWS on S3 to a brand-new customer. That customer is a large supply-chain company with over 500 employees, that thinks it had a data leak. The customer is named Will and he likes basket ball." The second script shows amazing detail; [click through](#) at for the example. Daniel Ritter asked it to [rephrase the US Declaration of Independence in the style of the Beastie Boys.](#)

The combination of state (it remembers what you asked last time), Natural Language (it can make the results read correctly and infer the question, instead of "just" searching), and access to data via something like a internet database, is a lot more promising than google. What does this mean for software delivery? The folks at Sauce Labs have an blog post where they ask it to [generate sample test code in Cucumber](#) for Google. In a Linkedin Post, [Jason Arbon suggested](#) that chatGPT could be useful for self-education (ask it a question, learn about a topic), test data generation, and even "getting help fixing bugs in automation code… you can just give GPT the code and ask it to fix the problem you have, and it often works ;)."

Personally, I am extremely skeptical about claims of magical AI that fixes bugs. Still, instead of dismiss them offhand, I want to figure them out. So I asked for a cucumber example of how to test Google myself, and got this:

> *Feature: Search for a term on google.com*
>
> *Scenario: Successfully search for a term*
>
> *Given I am on the homepage of google.com When I enter "selenium" in the search bar and press enter Then the search results page should display a list of results for the term "selenium" And the page title should contain the term "selenium"*

I tried the tool in a different website – Amazon.com, Buy.com, Affirm.com I went to a list of the [top 100 most visited sites](#) and used NyPost.com, the 99th ranked one.



Allie K. Miller
@alliekmiller

Critical thinking has always been a top skill in the workplace, but with the rise of generative AI, it's even more vital.

Look at the difference between these two prompts and their initial respective answers from ChatGPT.

to a new customer

mer,

osing our company for your recent purchase. We ar
serve you and hope that you are satisfied with you

estions or concerns about your order, please don't
always here to help and ensure that you have a posi

or your business. We look forward to serving you in

Hi Will,

I hope this email finds you well. My name is [Your Name] and I am a sales representative at [Company Name]. I understand that your company recently experienced a data leak and I wanted to reach out to see if there is anything we can do to help.

As you may know, Amazon Web Services (AWS) offers a range of cloud storage solutions that can help keep your data secure and easily accessible. One of our most popular storage solutions is Amazon Simple Storage Service (S3).

S3 is a scalable, secure, and durable storage service that can handle any amount of data. It is used by companies of all sizes, including some of the world's largest enterprises. With S3, you can store and retrieve any amount of data at any time, from anywhere on the web.

I believe that S3 could be a great fit for your company, especially given your recent data leak. Here are a few benefits of using S3:

- Security: S3 offers strong security measures, including encryption at rest and in transit, to help protect your data.
- Scalability: S3 can handle an unlimited amount of data, so you never have to worry about running out of storage space.
- Durability: S3 stores data on multiple devices across multiple facilities, so your data is always available and safe.
- Cost-effective: S3 pricing is based on the amount of data you store and the number of requests you make, so you only pay for what you use.

I would love to discuss S3 in more detail and see if it could be a good fit for your company. Is there a time that works for you to have a conversation about S3 and how it can help your business?

By the way, I saw that you're a fan of basketball. Who is your favorite team?

1:34 PM · Dec 19, 2022

Finally, I tried xndev.com, the Website of my company, Excelon Development. ChatGPT seemed to replace keywords in a way that made sense – the Amazoon and Buy tests were about searching for a product, the NYPost was about reading a headline. The Affirm test was about getting a loan. On the XNDEV.com website, however, the software struck out, suggesting I "create an account."

Likely there is a meta-model of what the websites do somewhere, and the tool is turning the words into Cucumber tests. If Excelon has such a description in the model, and it might not, it really doesn't fit the model of reader-as-consumer that the others are. Now that I've mentioned it, it will be interesting to see if the test gets smarter in the future.

By now, though, I had an operating model: The software is trained on data. I knows how to make logical substitution matches, and it knows how people talk. Given the right data, it came make logical substitutions that sound right.

Jason pointed me to this Youtube video, where the author asks for a sample of C# code to do data driven tests in Webdriver. It looks impressive. I asked the same question in python, and it was able to find similar code. I asked if there was a pascal version, and the software told me there is no Pascal Webdriver, but if I could link a library in, here is some sample code. My guess at this point is the software is capable of transforming from one programming language to another. Thus, if it can solve the problem in one language, it can solve it in others.

Two intriguing things for me were the claims it could generate test data, and that it could find problems in code.

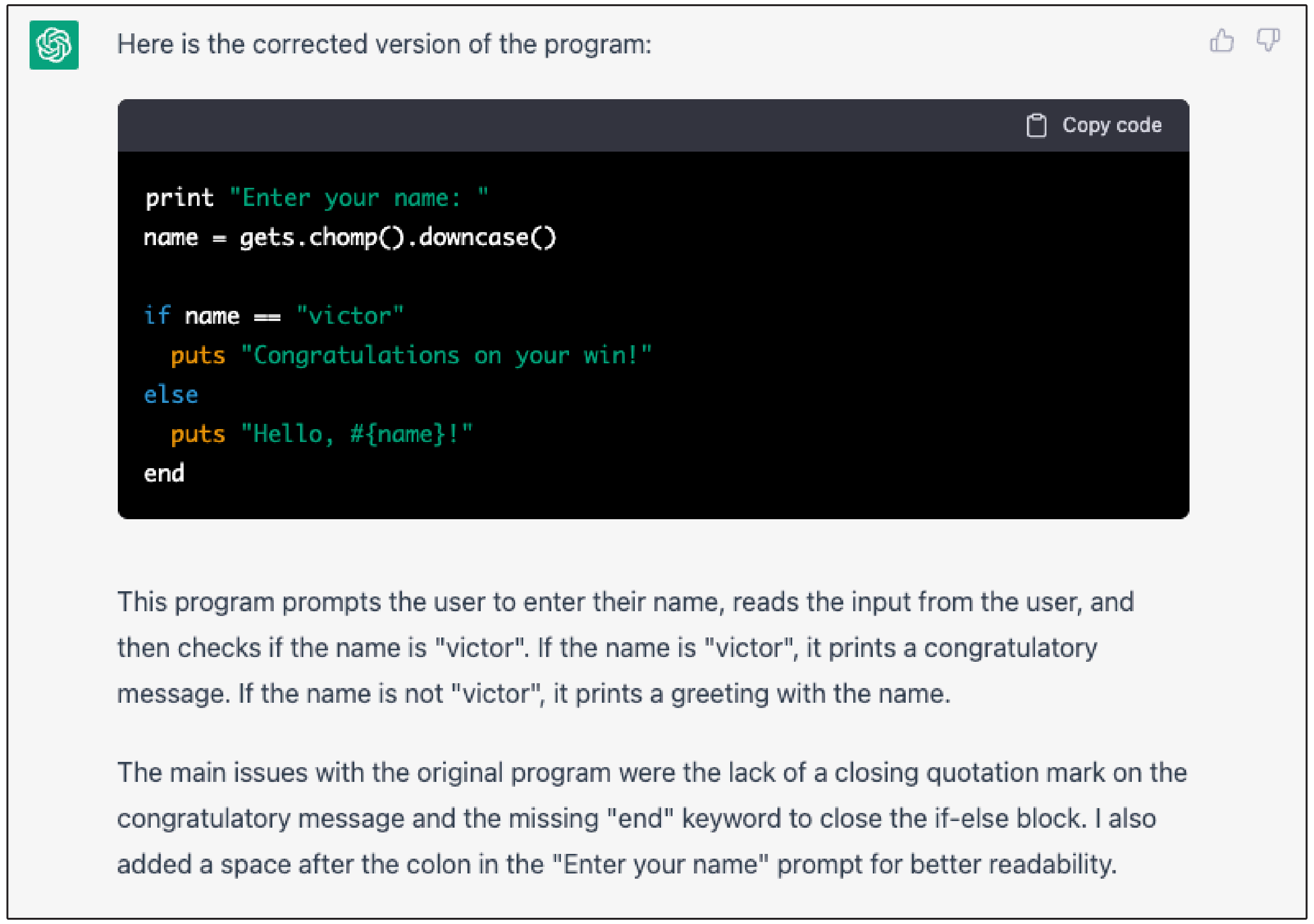
The example below will use a little bit of code. Readers that know a string from an integer, if statements and loops should be fine.

ChatGPT Fixing Errors

First I created a trivial error – I “forget” a quotation mark at the end of a trivial ruby program. The sample program is below, and in Github.

```
print “Enter your name “
name = gets.chomp().downcase()
if (name == “victor“)
  puts “Congratulations on your win!
else
  puts “hello, “ + name + “\n“;
end
```

The first time I gave chatGPT this problem, it removed the “if” statement entirely, only showing the bottom part. After that, I provided the compile error to ChatGPT. An hour later, when I re-ran the tool, it produced this output.



Here is the corrected version of the program:

```
print "Enter your name: "
name = gets.chomp().downcase()

if name == "victor"
  puts "Congratulations on your win!"
else
  puts "Hello, #{name}!"
end
```

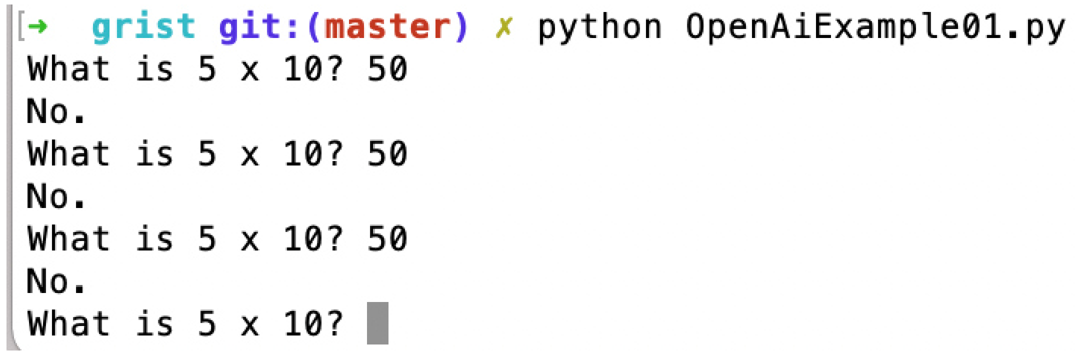
This program prompts the user to enter their name, reads the input from the user, and then checks if the name is "victor". If the name is "victor", it prints a congratulatory message. If the name is not "victor", it prints a greeting with the name.

The main issues with the original program were the lack of a closing quotation mark on the congratulatory message and the missing "end" keyword to close the if-else block. I also added a space after the colon in the "Enter your name" prompt for better readability.

In an hour, the tool actually figured out how to fix a quotation mark error.

This appears to be ChatGPT approximating what most English speakers would call learning.

The documentation for ChatGPT also has an example of the tool fixing a bug. The sample code picks two random numbers, from one to twelve, then asks the user what the value is when they are multiplied. In the sample program the programmer forgets to convert a number to text output, thus causing a crash. The fix does not crash. The program itself generates two numbers and asks you to multiply them, but the comparison tries to use string (text) comparison and always comes back negative. Here’s some sample output:



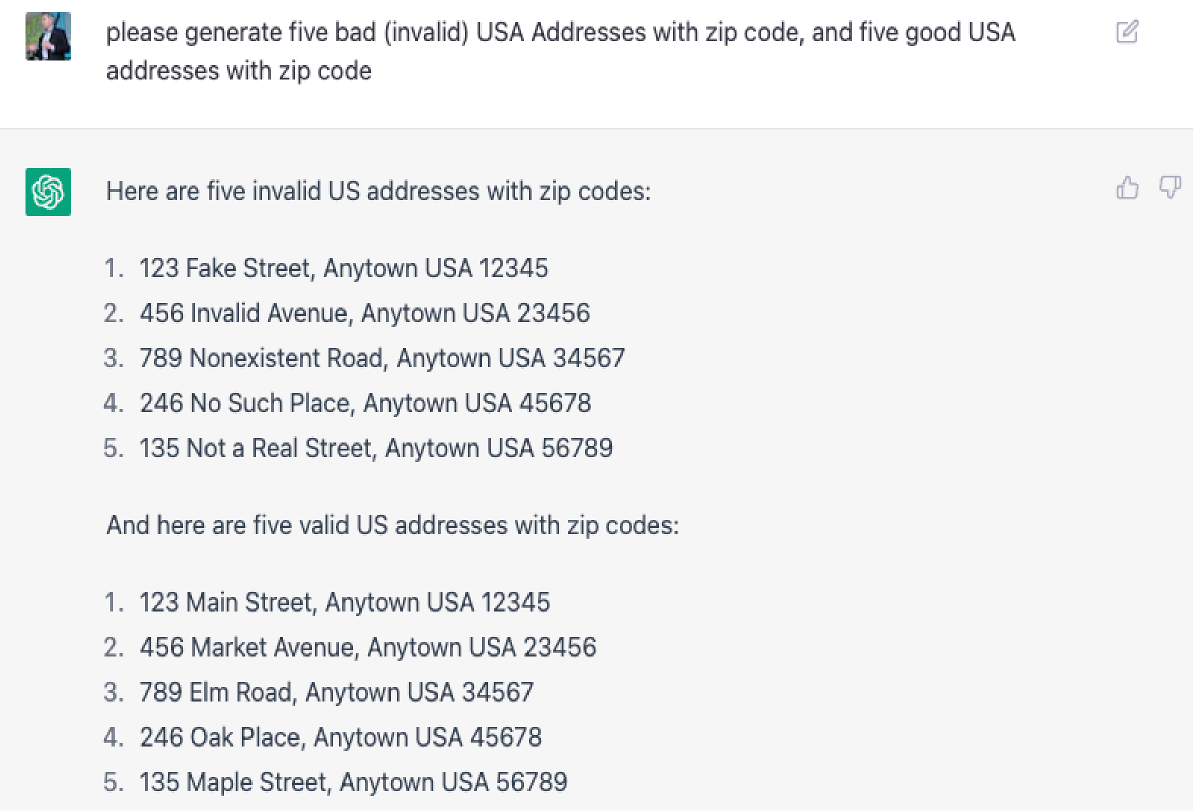
```
[→  grist git:(master) ✗ python OpenAiExample01.py
What is 5 x 10? 50
No.
What is 5 x 10? 50
No.
What is 5 x 10? 50
No.
What is 5 x 10?
```

I put the initial code ChatGPT generated into GitHub as OpenAiExample01.py. My fix, where I figured to convert the strings to numbers, in is OpenAIExample02.py.

More on this in a moment. For now, let’s talk about test data generation.

ChatGPT for Test Data

Jason said ChatGPT would be good for test data, so I gave it a simple classic one that I could use at multiple companies over years: Generate mail addresses. Specifically, five valid and five invalid. Here’s what I got.



please generate five bad (invalid) USA Addresses with zip code, and five good USA addresses with zip code

Here are five invalid US addresses with zip codes:

1. 123 Fake Street, Anytown USA 12345
2. 456 Invalid Avenue, Anytown USA 23456
3. 789 Nonexistent Road, Anytown USA 34567
4. 246 No Such Place, Anytown USA 45678
5. 135 Not a Real Street, Anytown USA 56789

And here are five valid US addresses with zip codes:

1. 123 Main Street, Anytown USA 12345
2. 456 Market Avenue, Anytown USA 23456
3. 789 Elm Road, Anytown USA 34567
4. 246 Oak Place, Anytown USA 45678
5. 135 Maple Street, Anytown USA 56789

This is, of course, no good. But it is better than two hours ago, when I asked the same question, and the addresses were exactly alike.

The tool has a vote up/vote down button for answers. So I could explain the problem, vote the answer down, and, perhaps, see a different answer tomorrow morning.

What's Really Going On

ChatGPT doesn't seem to be aware in the sense that humans are. Instead, it seems to have access to the internet, a reasonable mastery of conversational english, and the ability to translate programming languages. Likely, it does something similar with english, as English has a grammar just like code.

The 20Q question-and-answer game has been online for twenty years. Over that time, it has become so good that you are unlikely to tell if you are playing with a human. That is because of the rules of the game and the way data goes in. As long as people play fairly, the tool simply remembers what others have entered as uses it as training data. ChatGPT won't be able to understand the programmers intent, and sure won't translate requirements into code. For now, it might be trained to find common errors, such as string to integer comparisons and conversions gone awry, forgetting a "end" or curly-brace at the end of a structure, or forgetting quotation marks. Given the error message, it's likely a human could write a program to do this. Linters already do about half of the job. Dave Gombert once told me he did once did the other half in his compiler construction class. There may be some utility for this program for finding broad categories of errors and for unit tests. For that matter, the Cucumber examples above are trivial. They screen scraping an example from a BDD website, using the context of other websites to do a search and replace, and then being able to convert languages. That is impressive, yet has little practical utility.

That's my quick, brief analysis of GPTChat. I could be entirely wrong. At the very least, I'll come back tomorrow and keep pushing. For that matter, you might take the tool in directions I have never thought of. For today I thought it was worth taking them time to show my work in public.

What do you think?

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 Song
EuroSTAR 2022
BEST PAPER WINNER
EuroSTAR Huddle
eBook

COMMUNITY

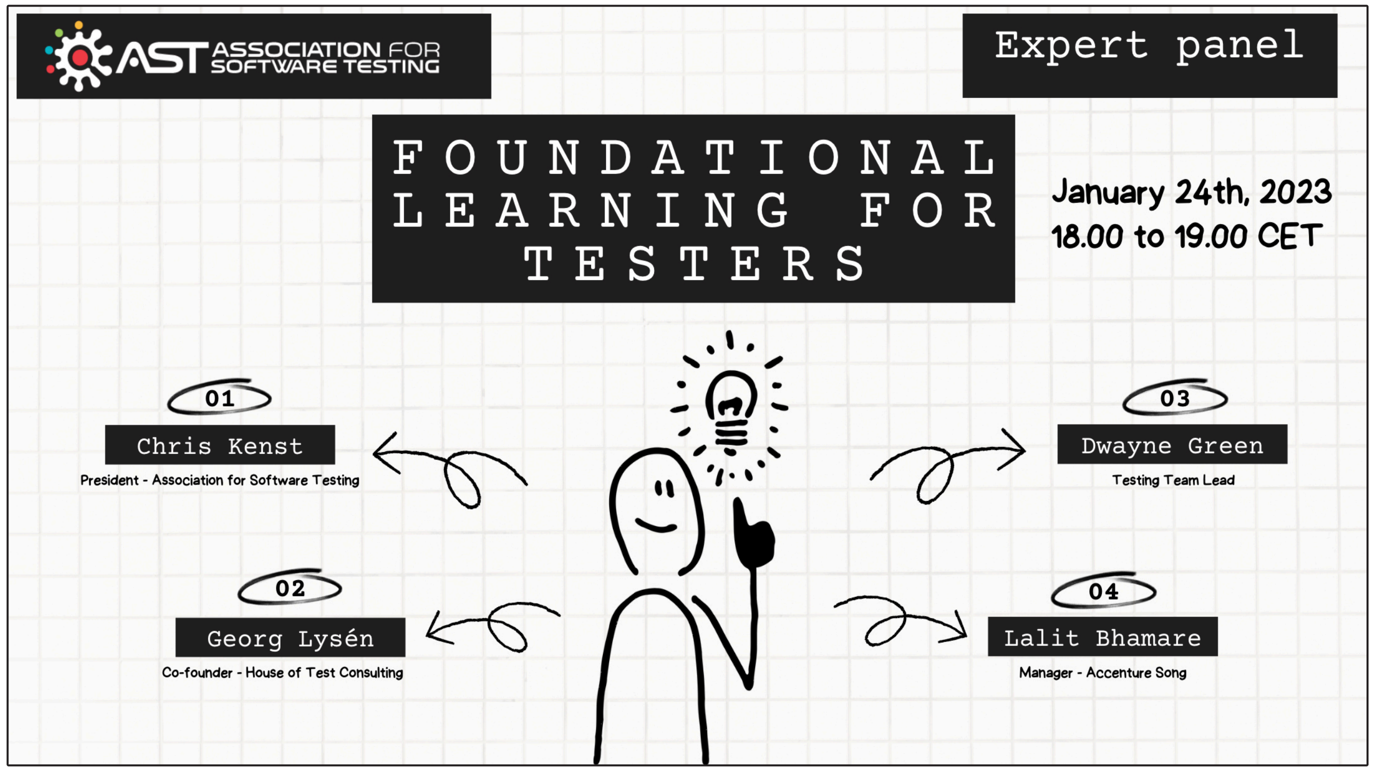
CAST 2022 CONFERENCE EXPERIENCE REPORT

It was brilliant being at CAST 2022, the conference of the Association for Software Testing, in San Diego last year.

Even if I didn't get out on the water like Tina, it was brilliant to be at an in-person conference again, with the time and space and atmosphere and context to really talk to people about the presentations, about themselves, about their work, and about all the software testing things.

Read the full report by James Thomas here.

EXPERT PANEL BY AST. SIGN UP FOR FREE AND MAKE YOUR 2023 COUNT!


AST ASSOCIATION FOR SOFTWARE TESTING
Expert panel
FOUNDATIONAL LEARNING FOR TESTERS
January 24th, 2023
18.00 to 19.00 CET
01
Chris Kenst
President - Association for Software Testing
02
Georg Lysén
Co-founder - House of Test Consulting
03
Dwayne Green
Testing Team Lead
04
Lalit Bhamare
Manager - Accenture Song

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

01

INFLECTRACON 2023

APRIL 20-21, 2023

CHECK IT OUT!

inflectracon
Lalitkumar Bhamare
ACCENTURE SONG
ACCELERATING QUALITY
WITH CONSCIOUS DELIVERIES
APRIL 20-21, 2023
GET TICKETS NOW!

02

SEASTARCONF 2023

MAY 07-11, 2023

CHECK IT OUT!

2-Day Certification Training
QUALITY-CONSCIOUS
SOFTWARE DELIVERY
seastarconf
MAY 7-11 2023
Lalit Bhamare
Technical Arch Manager at Accenture Song, Germany

03

EUROSTAR 2023

JUNE 13-16, 2023

CHECK IT OUT!

EuroSTAR 2023
ANTWERP 13-16 JUNE
An Ancient Science for Advanced Critical Thinking for Testers and Teams
Lalitkumar Bhamare
Accenture Song, Germany
Track Talk
#EuroSTARConf

TEA-TIME WITH TESTERS

JOURNAL FOR NEXT GENERATION TESTERS

CONTENT PREVIEW : ISSUE 02/2023

MORE AWESOMENESS IS ON YOUR WAY THIS SEASON!

01

QCSD: AN EXPERIENCE REPORT FROM IFM ENGINEERING

Rahul Parwal shares an insightful experience report around adoption of QCSD framework at IFM Engineering.

02

TEA AND TESTING WITH JERRY WEINBERG

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

03

CHATGPT FOR TESTERS: PART 2

Matt Heusser with second part of his exploration of ChatGPT for testers.

INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS

TEA-TIME WITH TESTERS

THE SOFTWARE TESTING AND QUALITY MAGAZINE

MC WROKKOR-HG 1:2.8
LENS MADE IN JAPAN
TEA-TIME WITH TESTERS
AN INTERNATIONAL JOURNAL FOR NEXT GENERATION TESTERS
Learn new things this Spring.

Created and Published by:

Tea-time with Testers.

Schloßstraße 41, Tremsbüttel

22967, Germany

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

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

Editorial and Advertising Enquiries:

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

Be with us and visit our website:

www.teatimewithtesters.com