



Proceedings

41st Annual
Pacific Northwest Software Quality Conference

a Hybrid Conference
held in Portland, Oregon and via Zoom

October 9-11, 2023



Permission to copy without fee all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.



This page is intentionally left blank.

Table of Contents

	<i>Page</i>
Forward	5
Conference Papers	
<i>in alphabetical order of first author's surname</i>	
1. Nimble, not Agile: Creating a flexible quality assurance framework for your company's unique SDLC - by Zenzi Ali	9
2. Data Quality Practices and Measurements - by Nick Bonnichsen	25
3. Revamp Software Quality for Android Robots through Advanced Development and AI Software Bugs Segregation System: DSF - by Ooi Mei Chen and Liu Keping	39
4. Software Continuous Integration with Hardware - by Brent Clausner	49
5. Software Cost Estimating – Friend or Foe (to Agilists) - by Carol Dekkers	59
6. Testing Warehouse Automation with - Autonomous Mobile Robots - by Mesut Durukal	67
7. Harness the Power of Debugging - by Kiruthika Ganesan	76
8. Improving Enterprise Scale Test Automation with ML-Based Predictive Analytics - by Dmitriy Gumeniuk & Tqriq King	87
9. Revamp Software Quality for Android Robots through Advanced Development and Deployment Methodologies - by Theodore Guo , Monica Bao, Sophia Lee, Kyle Zhou, and Bo Li	99
10. Break Silos, Build Chapters - by Ashok Kumar	115
11. A11y Advocacy: Why We Need to do More Than Just Test for Accessibility - by Michael Larsen	127
12. Continuous Testing Integration with CI/CD Pipeline - by Junhe Liu	136
13. Reducing Ambiguity in Requirements through Multi-modal Communication - by Evan Masters	147
14. Quality Engineering Considered as a Helix of Semi-Precious Quadrants - by Vivek Sahai Mathur	156

15.	Designing and Creating Testing Datasets for Medical AI/ML End to End Testing - by Nancy McCormack	164
16.	Quality & Risk Management Challenges When Acquiring Enterprise Systems Revised in the Age of Cloud and AI - by Jack McDowell & Ying Ki Kwong	177
17.	Art in Code and Quality - by Sophia McKeever	189
18.	Common pitfalls in modern QA - by Leandro Melendez	200
19.	Real time failure point detection and operational health monitoring of Cloud Infra - by Vittalkumar Mirajkar and Srivatsan Parthasarathy	209
20.	AnaCov - A novel method for enhancing coverage analysis - by Mustafa Naeem, Ahmed Tahoon, Omar Ragi, and Reem El-Adawi	221
21.	Chatbots As Meeting Stand-In: Enhancing Remote Work Productivity - by Kavitha Naveen and Pawan Kumar Naveen	232
22.	Increase Organizational Learning and Business Agility with Strategic Goal Setting and Innovation in your Testing Program - by Kristine O'Connor	242
23.	Transformation path for Tests and CI/CD when moving applications from data center to public cloud - by Amol Patil	253
24.	Building a Culture of Accessibility: From Awareness to Action - by Dhara Pattani & Younus Poonawala	265
25.	Navigating Complex GraphQL Testing: Automating Integration Test Generation - by Nate Smith	277
26.	Optimize Your Testing Performance by Using Cypress.io - by Ryan Song	285
27.	Automation - We're Doing it Wrong - by Melissa Tondi	293
28.	Amping Your Success by Leading with Quality: Why "Shifting Left" Isn't Enough - by Heather M. Wilcox	299
29.	Quality Ownership - by Linda Wilkinson	312
30.	Defect Management System Powered by AI (Artificial Intelligence) - by Peh Wei Woon, Liu Keping, Ooi Mei Chen, and Felix Eu	323

Forward

This is the 41st anniversary of the Pacific Northwest Software Quality Conference (PNSQC). We celebrate another great year by acknowledging the entire PNSQC community: our attendees, authors / presenters, peer reviewers, volunteers, sponsors, and board members. Thank you for your continued support!

You will continue to find PNSQC to be one of the best conferences in terms of learning state-of-the-art and practical methods for improving software quality. PNSQC 2023 continues this tradition.

This year's conference featured speakers from around the globe in four tracks. During the three days of PNSQC 2023, our technical program consisted of keynote addresses, invited presentations, peer-reviewed papers & presentations, panel discussions, tutorials, workshops, and exhibits of our sponsors. For additional details, see www.pnsc.org.

This is the second year that PNSQC was held using a hybrid format; with a physical venue in Portland, Oregon and a simultaneous virtual venue via Zoom. We are pleased that 135 attendees participated in person, and another 135 participated virtually. There continue to be lessons learned that will enable us to improve the conference experience in the future, for both our in-person and virtual attendees.

This volume contains all papers that successfully went through the PNSQC peer review process. In this process, at least two reviewers commented on each paper for possible improvement by its authors – typically over three drafts. This approach assures quality of our contributed papers and is an important differentiation between PNSQC and other industry conferences on software quality. We hope you find these materials informative and useful.

Best Regards,

Ying Ki Kwong, PhD
Board Member
PNSQC

This page is intentionally left blank.

Conference Papers

This page is intentionally left blank.

Nimble, not Agile: Creating a flexible quality assurance framework for your company's unique SDLC

Zenzi Ali

zali@clarity-innovations.com

Abstract

Many software testing lifecycle models are designed to correlate with the most popular software development lifecycle models, like Scrum or Kanban. But, there are many ways to build and develop software outside of these standard models. How can teams create a workflow that is conducive to delivering high-quality software in an unorthodox environment? My team uses Basecamp's Shape Up method to deliver software. It provides a great deal of flexibility for developers to ship products, but it can be challenging to inject thorough test procedures into a method that centers developer productivity above team process. I have spent time dissecting the most popular testing methodologies in order to craft my own flexible framework.

Bio

Agility. Advocacy. Efficiency. Affinity. Amazing things happen when engineering and humanity come together. Zenzi understands the power of high-quality engineering products and their significant role in driving the modern world. But she also understands the need for clear communication when developing and implementing this technology so that it can better serve human needs.

Before joining Clarity, Zenzi managed a community of new technologists (providing code and assignment reviews); created, optimized, and automated unit tests; sharpened her communication skills while working as an Executive Assistant; used Agile methodologies to plan, iterate, and execute tasks within cross-functional teams; and implemented testing processes with startups and large organizations alike. She even contributed to The Code for Philly Front End Fellowship, supporting technology to help Philadelphia eliminate food waste.

Zenzi merges her troubleshooting experience and test craftsmanship with her passion for refining processes and collaboration, to ensure that every edtech solution Clarity creates is as streamlined, secure, and useable as possible. It's all part of her desire to help solve human challenges—such as those facing education—through the latest, most efficient means.

Section 1 Introduction

When I joined Clarity Innovations in 2021, I knew that the team was extraordinary and that I could thrive in the environment. I have a tendency to over-analyze and over-prepare and this team was action-oriented and with the experience necessary to make adjustments on the fly. I felt that their practical method of delivering software was the perfect pairing for the analytical way that I have thought about quality and delivery.

The method that my team uses for building software is called Shape Up. This development model, created by BaseCamp (a project management platform), is fast-paced and solution-oriented. Although the method focuses on strategic scoping and thoughtful problem-solving, there is little room or expectation for testing involved. Because of this most of our testing was built around product validation for our customers and did not have the depth of other testing models. There was no expectation the testing would start early or even before a product was ready to deploy.

At the outset, Shape Up does not seem to leave space for quality. QA is shown to be nearly unnecessary in their process and used to explore edge cases only. Box checking and rubberstamping a finished product is not satisfying testing work, in my opinion. It is my belief that quality can be injected into any process. I also do not believe that Shape Up is alone in its view of the tester. Many testing professionals find themselves feeling like second-class citizens on a team and wondering exactly where they fit in and how to advocate for quality.

Luckily, my team is devoted to building high-quality software and has an interest in more thorough testing and exploring techniques. This left me with a large task. How can I inject quality and testing methodology into my engineering team's well-oiled process? How can I add value and assess risk without becoming a bottleneck to a high-functioning team? When and where can I shift testing left while keeping our business objectives and procedures as the primary focus? How can I create a process that can meet the varied needs of our business, which both creates and maintains a variety of technical products?

These questions and discovering my own pain points with our procedures caused me to do my own research and develop methods to aid in upholding quality throughout our development process. After reading *Shape Up* and observing my team in action, I developed my own techniques for adjusting my testing lifecycle to the project and product being developed. I will share my journey and methodology here.

The goal of this writing is to provide testers with the tools that they need to think deeply about testing and create a maintainable, effective testing routine. I in no way seek to malign or discredit *Shape Up*, Agile, Scrum, or any other software development lifecycle method. In my opinion, the best framework for creating quality is the one that you use repeatedly and consistently! Removing barriers to your quality practice is equally important to adhering to any framework.

This method is perfect for teams that are like mine:

- Solo testers or testers that are part of a small team.
- Testers that work with teams that use an unorthodox software development lifecycle model.
- Teams that build and ship fast.
- Quickly expanding teams that have not had time to focus on Quality Control and quality management.
- Testers that value flexibility in their tooling.
- Teams that manage multiple products with varied contextual needs.

Context often considers project needs, goals, and constraints, but my model also considers organizational “appetite” [more on this later] and internal standards for quality. My method of creating a test strategy allows me to fulfill my intrinsic need to amplify quality within any product I touch while respecting my team’s culture and norms. Business and development needs are, of course, of high importance when one is the sole advocate for quality. My method focuses on prioritizing business needs while fulfilling my personal need for self-development and professional fulfillment.

In the rest of the document, I will provide insights gained by using my experience in managing my internal test procedure. I will explain how Shape Up works, the challenges the method produces, and how my team chooses to meet those challenges. I will outline how I align my test activities with Shape Up’s software development lifecycle. I will also outline some ideas on how to advocate for the team buying in your new testing procedures. Moreover, I will include some visual aids that review phases of software testing and how they relate to software development. Finally, I will review scenarios of different types of teams and address how I might approach creating a maintainable, complete, and flexible process within those structures.

Section 2 Analyse the Environment and Create Goals

A plan of action was not drafted until my team’s processes were analyzed and understood. My company is relatively small, but engineering makes up nearly 1/3 of our employees. We are an ed-tech firm that performs consulting work for K through 12 educators. Although our area of expertise is specific, our products and projects range in type and scope. Some products arrive “complete” and are in production but in need of updating or scaling. Others are developed in-house for our clients from inception to completion. We use the technology that suits their needs and user base. This can range from Flutter mobile development to Drupal web applications or any tech stack that the client has used previously. This means that our team must be highly adaptable and prepared to support a wide range of projects that we do not necessarily own.

Testing falls within the domain of our development team. Our testing team is small; a team of not quite two. I am the sole full-time tester with our senior quality assurance professional of the team dividing time equally between testing and DevOps. Our project work can start anywhere within the software development lifecycle but, no matter where we start or what

the status of the project may be, we use Basecamp's product development model, Shape Up. Shape Up is a development model that is not well known but has worked for Basecamp for over a decade.

Basecamp's approach features a six-week build cycle that is shaped by senior project owners. There are no sprints, kanbans, or backlogs. The goal of this method is to speed development by keeping requirements relatively vague and focusing on solutions to a rigidly scoped problem. When it comes to Shape Up, the key is proper scoping and an understanding of the problem you would like to solve. This helps engineers tame complexity for faster development and separate concerns for better strategic planning. "Projects are defined at the right level of abstraction: concrete enough that the teams know what to do, yet abstract enough that they have room to work out the interesting details themselves." (Friedman 18)

The Shape Up model operates in three distinct phases: Shaping, Betting, and Building. During **shaping** a product owner focuses on how to deliver a solution for a whole product or feature set. The designs and ideas are rough, but the goal is to solve a problem. In this phase, shapers and trackers cooperate to create shaping documents for a pitch (a meeting where shaper(s) advocate for their project). The goal is to define the scope of this project, which should last no longer than its six-week cycle. This phase should include extensive discussions involving the project goals (what need or problem will this project address), possible rabbit holes, clear parameters, and project constraints. The project shaping document provides a development team with enough information to create without constraint. Setting the project parameters and defining "done", also known as breadboarding, is accomplished by deciding on the size of stakeholder appetite, and how much time, energy, and resources are we willing to invest to solve this problem.

Key staff must decide if a sufficient solution, even a rough one, is created within the time outlined. This process is done in isolation, or with only staff with the necessary expertise for scoping and shaping.

After shaping, the **betting** begins. Betting is the stage where shapers share a pitch with the larger group—placing their "cards on the table". They discuss the details of what they have shaped and the team "bets" on which project they will be working on for this time. If a shaper's project is not chosen, there is no backlog. The theory is that an important idea will show up again in the future. If a project needs refinement or if a solution is inadequate, the shaper can pitch at a later time. Betting is also when additional context to the project is added. Is this an existing project or a new one? Is research and development needed? Is this product already in production? Has it reached a maintenance phase or is it nearing retirement? All of these questions serve to add context to designing a project and betting on the most impactful and necessary project for this round of development.

Finally, **building**, the final phase, occurs. The building phase starts with a kick-off meeting between stakeholders and the development team, as well as any other required team members. At that time, stakeholders hand the project to our development team with the goal of developing slices of complete work that can be integrated into a larger system if

necessary. Developers are expected to be assigned projects, not tasks. There are no Kanban boards or Issue tickets to assign. Developers should have a complete scenario or issue that they are programming a solution for and they should be programming just enough to complete the next building step.

During this stage, there is also scope mapping to confirm that the imagined tasks can be replaced or updated with discovered tasks. The team can continue to evaluate if their appetites match the size of the effort needed for the solution. Sunk costs are examined at every turn. Developers are expected to show progress, not discuss the process. Once our basic solution has been created, the developer stops and moves on. There is no list of future To-Do's. We have solved our issue and now we provide feedback to the team at the end of the 6 weeks. Is there an additional appetite for a more complex or elegant solution? This can be decided at that time.

After observing my company's software development lifecycle and engaging in software testing, I began to think strategically about how we test, what can be improved, and how I can affect software quality without substantive changes to my team's SDLC, which they were happy with. I also felt that the SDLC was outside of my domain as a tester. Whether my team uses an iterative model or an incremental one, my goal is to be adaptable in my process. One tool that I used to assess our quality management ecosystem was a SWOT analysis. A SWOT analysis is an exercise that helps organizations (or individuals) identify their Strengths, Weaknesses, Opportunities, and Threats. It is essentially a risk assessment exercise. SWOT helps the user differentiate what environmental elements they can control and how to allocate team resources. I time-boxed one hour to review our procedures and another to consider what should be placed in each SWOT section. The analysis matrix looks like this:

Figure 1: SWOT Analysis of My Team's Approach to Development and Quality:

SWOT ANALYSIS

S

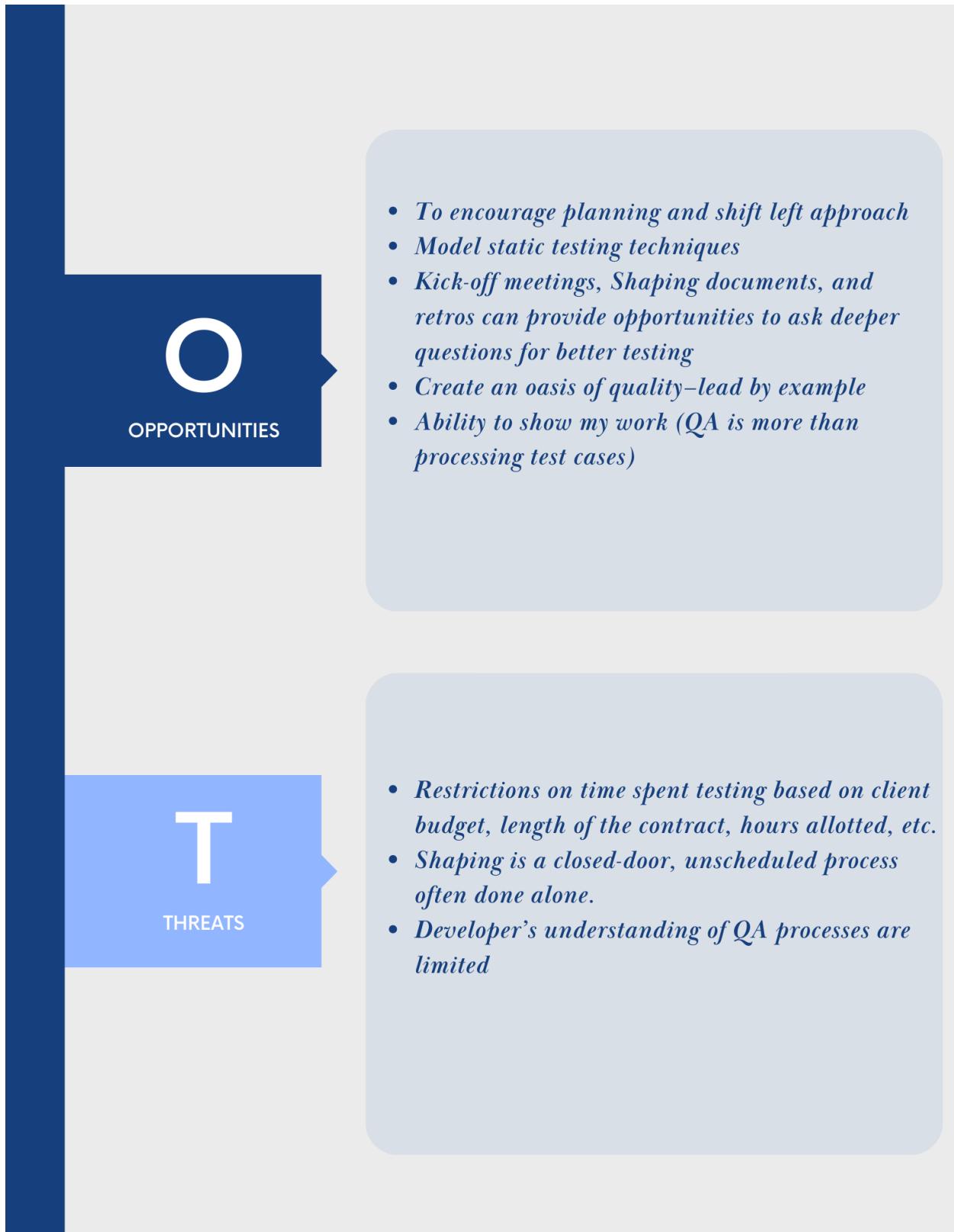
STRENGTHS

- Team open to new testing procedures
- Team values testing expertise
- Project variety allows for learning new test techniques
- Shape-Up features rigid cycles - "done" is defined
- Contractual obligations create well-defined UAT parameters

W

WEAKNESSES

- Shape Up risk assessment prioritizes "scope bloat" above other quality-related risks (security, performance, UX, etc.)
- User feedback is rarely received/explored in this model
- Fat Marker sketches are purposely abstract - no detailed wireframes for UI visual testing



Section 3 Reviewing the SDLC and Common Test Activities

The software development life cycle is a methodical approach to planning, building, and shipping software. While the life cycles that are well-known tend to focus on these things independent of test techniques, it is our job to create quality practices that correspond to each phase of the cycle. The most common types of models are sequential, incremental, and iterative. See the key below with key aspects of each type of SDLC. Using an established model increases the likelihood of a successful project by establishing phases, activities, and opportunities for collaboration across job functions.

Our 3 most common SDLCs have corresponding software testing lifecycles (STLCs). The software testing life cycle features test activities and test levels that correspond to the development phases of its development model. For example, the traditional sequential model begins with documentation and planning. One may begin by testing using static testing techniques, reviewing documents for understanding, potential contradiction, or the need for clarification of features/functionality. Shape Up is not a standard technique or life cycle model. Because of this, they have their own approach and goals for QA.

For many years, Basecamp had no QA personnel. Eventually, they decided to hire a single QA professional for their development team of 12. With the developer-to-QA ratio being so high, the team must think differently about quality and who is responsible for it. Traditionally at Basecamp, quality falls on the design team and the developers.

Risk assessment and analysis is the domain of Product Owners and management. In the Basecamp context, QA is done at the end of the build to consider edge cases and approach the project with a tester's mindset. QA is considered valuable and essential, but siloed and separate from development. This approach has been successful for Basecamp but, through trial and error, my team has decided to tailor their approach to allow for more thorough QA testing in each iteration. In section 4, I will discuss how things stood at the beginning of my SWOT analysis (see Figure 1) and the changes I have made.

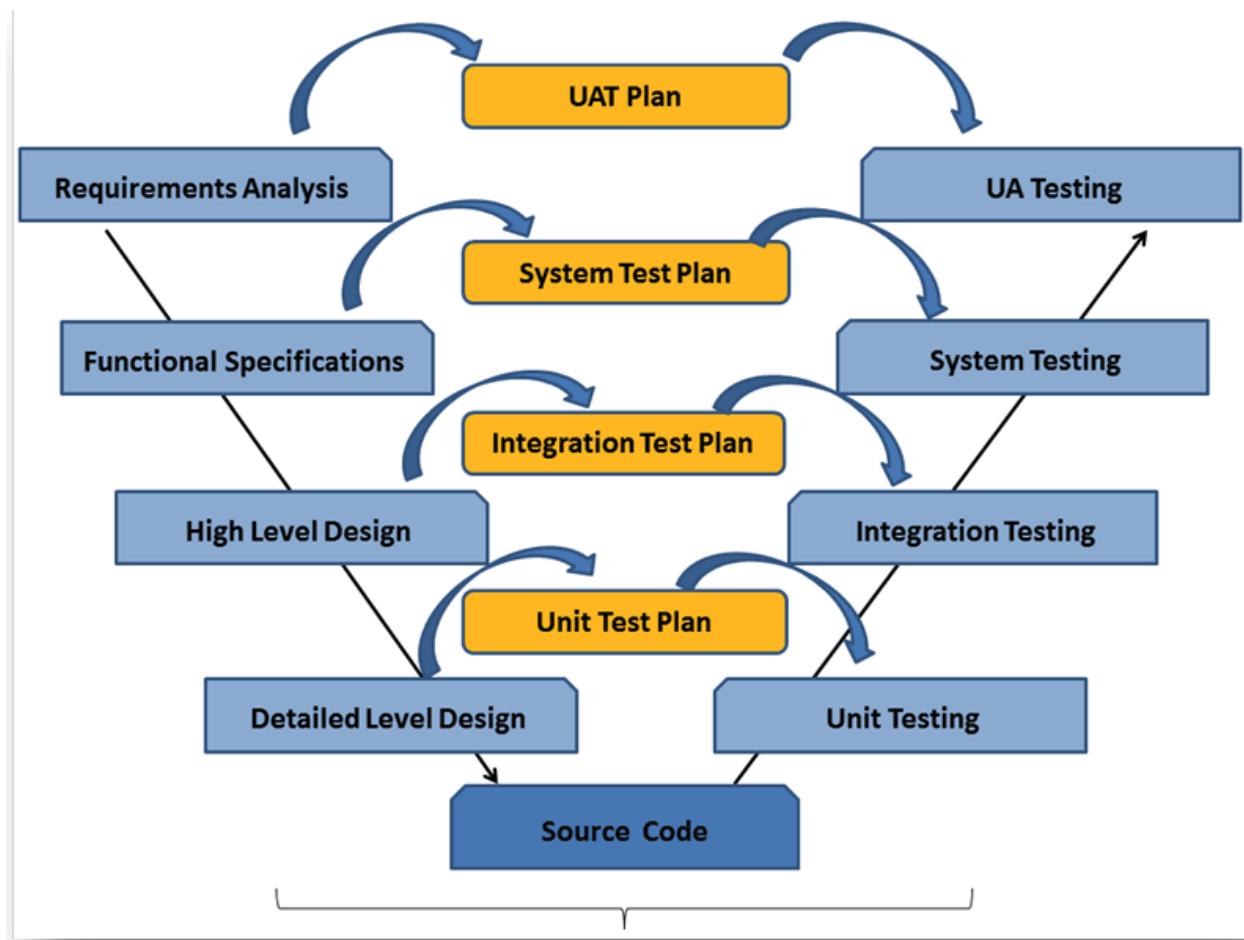
Figure 2: Comparison of SDLC Models:

	Sequential (ex: Waterfall, V)	Incremental (ex: Unified Process)	Iterative (ex: Spiral, Prototyping)	Shape Up
Model Attributes	Sequential approach. Begins at the system level and progresses through analysis,	Used to build large, complex systems. Based on the idea of adding new features, or "increments," to	Initial development work is carried out based on well-stated basic requirements, and successive	Solution-centered approach focused on building a working product or feature to completion by the end of a cycle

	design, coding, testing , and support.	an existing system instead of a sequential build	enhancements are added to this base piece of software through iterations until complete.	
Testing Scope	Full system considered	Testing can be performed at each level and increment.	During early iterations, QA focus is on unit and integration testing, while later on, system testing and user acceptance testing.	Little to no QA expected
Test Timeline	Static testing at the beginning, dynamic at the end	Testing at the end of each increment	Testing at the end of each iteration	Must happen within the cycle/scope
Documentation Level	Heavy documentation, documentation is often the basis for testing	Light documentation	System requirements are heavily documented, but features and approaches may evolve as the iterations continue	Documentation after the cycle and often handled by other teams, Siloed collaboration means few details are captured in writing, no backlog means no todo list and light issue tracking
Test Approach	<ol style="list-style-type: none"> 1. Examine requirements 2. Review Designs 3. Integration testing 4. System Testing 5. Acceptance Testing 	Regression and integration-focused testing, fast feedback due to shortened development cycles, focus on experience-based testing techniques	Regression-heavy testing	Just edge case testing for QA, focuses on product use and approaching the project with a fresh eye
Automation	Not practical as the entire system should be built for pre-release testing	Automation preferred for faster regression testing	Automation preferred for faster regression testing	NA-not practical for this test approach

Tester Responsibilities	Requirement reviews, test analysis, test design, system maintenance	Testing incrementally developed features, providing prompt feedback to developers	Collaborate on exit criteria, test planning, bug tracking, test execution, data, and environment prep	Edge case testing, designers and developers are responsible for quality. QA can generate UX tasks that can be added to this cycle or pitched in a later cycle
-------------------------	---	---	---	---

Figure 3: Sequential SDLC with Corresponding Test Activities:



Source: <https://qainsights.com/improving-the-test-process/>

Section 4 Modularize the Test Cycle and Create a Framework

Prior to my team's SWOT analysis, our testing relied heavily on a checklist approach. The key to finding defects effectively in that approach is having long-term experience with the product. Experience-based testing can be highly subjective. When testing, I found myself reaching out to developers often to confirm functionality because it did not meet my personal expectations. With no test oracle or test management software, communication of this sort happened throughout the testing process. This worked for my team when everyone was based in Portland but, since the pandemic, we have become a remote team with members in every US time zone. We needed test techniques that improved our communication and made the most of what documentation we chose to maintain in the future.

Although Shape Up does not prioritize QA, it does realize the position's importance. I did find the Shape Up method's testing process to feel isolated and a bit tacked on to development. My remote East Coast position heightened this feeling. I wanted to improve the testing process and feel more integral to the product delivery experience. This required me to decide on my testing principles and discover how I could use those to meet testing and business needs. This can be done via brainstorming, mind mapping, or an additional SWOT analysis.

I decided to brainstorm current blockers and perform a solo retro to assess process failures. This was challenging in a team that does not center metrics. How could I assess my value and decide if incremental changes that I made in the process were impactful? I decided to gather any data that I had access to and that could be compiled quickly. Test completion estimates are very important to my team, so I could not increase the amount of time spent testing by any large amount.

My professional values centered on product ownership, communication, and remote-friendly processes. I wanted to incorporate my professional values into the framework to garner a more personal sense of accomplishment from my work. Because I work in consulting, I'm a few steps removed from product ownership. Owning the process increases the satisfaction that I feel in my work. I also decided to use testing activities as a way to communicate with stakeholders and report early and often to my team without affecting their process. I used the tools that were available to me to complete testing that was not being performed in the past.

After my initial SWOT, we made a series of small process changes that improved testing needs:

- Transition from text document test cases to Test Management Software
- Remove testing discussions from Slack (where they could get lost) to their proper merge request or to a document in a shared drive
- Implemented automation to free QA time from regression testing
- Attending product meetings to observe (understanding context)
- Create Test Plans for new projects
- Re-organize tests to shift to a user story-focused test strategy
- Ensemble testing sessions and test case reviews with Developers

Each activity matches a Weakness or Threat discovered during the SWOT Analysis and/or strategically uses a team Strength to promote quality.

Figure 4: Test Activities by Development Phase:



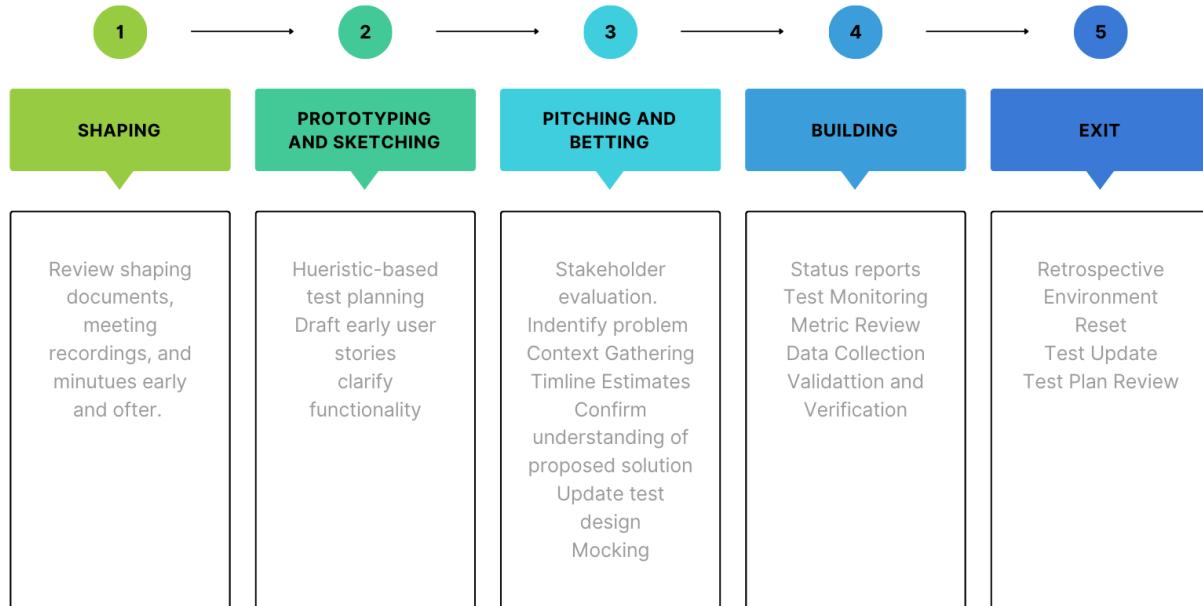
Figure 5 shows the most common testing strategy that I use. I was able to shift testing left because my team does not use the pitching and betting phases as designed by Basecamp. We are consultants, so our clients “pitch” a problem, our client lead performs betting activities, and then the client lead will Shape with a lead engineer.

I have chosen to display the phases linearly, but often the first 4 phases occur concurrently. This is beneficial to my testing process. I can start testing requirements via the documents because I do not need to wait for my team to bet on a project. Another benefit is most of my static testing occurs with no interruption to my team’s process. Shaping and design products are drafted as a part of our business processes. I can use these underutilized assets to my advantage without any extra time commitment from the team.

One area that is not given an official phase is completion. Basecamp completes its projects on time with little ceremony. Feedback is left to review for later pitches if significant negative feedback or product bugs are reported. For my team, there is a formal retro that allows me to solicit feedback from the team.

This is our general process, but it also varies by project. Choosing techniques and activities is easily done by using the ideas in Figure 4 as a starting point. Time is of the essence, so I like to time box creating a one-page test plan. Continuously adding techniques to my toolbox and discussing testing and product development with my co-workers and testing communities keeps my process fluid and nimble for changes that may occur during the building process.

Figure 5: Personalized Shape Up SDLC-STLC diagram



I had many opportunities to propose ideas and discuss changes that I would like to see on our team. I often used weekly one-on-one meetings to discuss ideas for professional development, a not-too-often discussed part of a good test strategy. Because of my team's

unorthodox approach to testing, I did not have a basis for common test strategies and spent much of my first year reading books like *Explore It! Reduce Risk and Increase Confidence with Exploratory Testing* by Elisabeth Hendrickson, *Agile Testing Condensed: A Brief Introduction* by Janet Gregory and Lisa Crispin, and *Foundations of Software Testing ISTQB Certification* by D. Graham, R. Black, and E. van Veenendaal. I was encouraged to share our learnings at weekly engineering meetings. My team loves to talk theory, so it was never difficult to gather feedback for testing ideas.

As I mentioned in Section 3, my team does not review metrics often. However, I found them to be useful as I implemented these changes to our software testing process. What I found was that I spent an increased amount of time on test planning and test design. Test analysis led to a more robust risk assessment for our products. We also used the findings of our assessment to increase test coverage of our products. Reading shaping documents, dissecting analytics, and discussing user stories led to updates in our cross-browser and cross-platform testing that have increased reliability for our clients that use older hardware or browser versions.

With increased planning and design time, I noticed execution time was speeded. I was able to provide more accurate estimates for test completion. The use of test management software also allowed for more standardized test reporting (and gave stakeholders the ability to see testing status). I did not see an increase in the number of bugs, but I have noticed an increase in the finding of high-priority bugs much earlier in the testing process. Traceability is also improved as I can more easily add software, operating systems, and hardware versions to test runs. Overall, each step that I have taken to create this framework has led to greater success. Because I discuss and display my work more often, the team has a deeper understanding of the many activities involved in quality management. Executing test cases is a small part of my process.

Section 5 Some Scenarios with Matching Frameworks

The Lone Tester Problem

"I work with a team of developers and I am the only tester on my team. The devs tend to get excited about new ideas and testing is often an afterthought. I often only find out about bugs or new features when they are sent to me for QA. I am super busy and I work on a metaphorical island of testing. To top it off, my team is always asking me about the status of their individual branches."

The Lone Tester Proposed Strategy

The lone tester (LT) needs strategies that keep their team informed so that they can focus on the work. If the team tends to forget to invite them to meetings or feels that their presence is not necessary, request access to meeting minutes or recordings. If the lone tester is commonly surprised with new testing work, a long-term strategy is not practical for them. Daily time boxing must be a part of their framework.

Each day, the lone tester could check for new branches in the team's version control system so that LT could be aware of what their pipeline contains. Reviewing Merge Requests provides the opportunity for preliminary static testing. Automation should be added to LT's toolbox. Setting up automation triggers using Zapier will eliminate busy work. In the mornings, automate a Slack or Teams message that explains where you are in the testing process for each assigned project or feature. "Zaps" can also trigger alerts to your team

when you update the status of QA in a Jira ticket or Trello board. If there is a way to automate status reports via messaging, do so. LT can also set up their own testing pipeline Trello board and create a “Zap” that creates a Trello card each time a new project branch is created in version control.

If automation is not available, timebox this, too. I recommend starting with 30-minute intervals for any task and adjusting to fit the day. Thirty minutes is more than enough time to create a quick test strategy or test plan that can be shared with your team. The most important thing that a busy tester can do is to advocate for professional development of some sort. This can take the form of class work, joining special interest groups, ensemble coding or testing, or meeting with product managers to learn more about your product. Professional development and access to product information inform your testing process. Share what you have learned, thoughts, and ideas during regular meetings, retros, or asynchronously.

Team Just Ship It

“My team knows that perfection is the enemy of good. They are creative and aren’t afraid to build their parachutes on the way down. This means testing in production is very common. I’ll spend a lot of my time waiting for the worst.”

The Just Ship It Solution

This QA team should not wait for the worst, they should anticipate it. This means emergency preparedness and risk assessment should be a key part of their framework. Stress testing and forced error testing can alleviate the stress of production surprises. While this is commonly done during the planning and control phases, a team that tests in production must integrate system assessment into all phases of testing as part of the team's new risk strategy.

Additionally, the QA team should be aware of the most common and critical user stories so that, if testing time is short, these are never neglected. Because there will not be a large time commitment to pre-release testing, regular bug bashes must be scheduled and anticipated. This is an excellent opportunity to pair with the developers and assess the most common problem areas. Bugs tend to cluster together, so this can also be a part of risk analysis for future test rounds.

Conclusion

Many software teams feel that their primary goal is to ship software. Of course, that is important. It is the keystone of their business model! But, we must remember that software is more than that. Our development teams are solving human problems using technical expertise. And the testing team is the bridge between customer expectations and our business demands. Upholding quality is a challenging task! However, creating your own system can provide you with a sense of satisfaction and a process for analyzing and reflecting on business impact. Taking a thoughtful approach to this work allows you to promote a culture of quality and nearly any type of team.

References

Shape Up Stop Running in Circles and Ship Work that Matters Singer, Ryan 2019
<https://basecamp.com/shapeup>

Data Quality Practices and Measurements

Nick Bonnichsen

nick.bonnichsen@hmhco.com

Abstract

If your business and customers depend on data you need data you can depend on. Storage is cheap, processing is flexible, and data is a hidden trove of value increasingly used to drive business decisions and priorities. We generate it by the terabyte. We copy it from database to database, transform it from NoSQL to Relational data to graphs and charts. We ingest it from customers and reflect it back to them with augmentations and additions our Sales departments have promised are just what is needed to propel customers to the next level. More than ever data drives our customers' business decisions as well as our own. Poor quality data can mean poor decision making, or worse, customer outrage.

After some hard lessons in what poor Data Quality can cost, we determined to find ways to measure and assess Data Quality outside of functional testing requirements. For Data Quality to be measured we needed methods to compare datasets together, either by comparing a dataset to an idealized dataset or comparing two datasets against each other. Independent Data Quality dimensions could be defined, agreed upon, and measured. Further we extended quality analysis work to practices and cultures that encourage Data Quality or that enable poor quality to sneak in when absent. Data Stewardship was identified as a starting point for data ownership and continual improvement work. Data standards enable development teams to build in Data Quality from the get-go with practices like rigorous input validation and periodic data surveys.

Biography

Nick Bonnichsen is a Staff Quality Assurance Engineer with 20 years of experience in testing and quality initiatives. Currently he is working on adjusting company culture and processes in pursuit of higher quality development and testing practices, like Agile and DevSecOps. Outside of work he enjoys glamping, backpacking, travel, role playing games, video games, and board games. Occasionally he mauls wood in a futile attempt to create furniture. He is married to an Australian, has an adopted daughter, and a one-year-old grandson.

Copyright Nick Bonnichsen 2023

1 Introduction

On a completely normal cold wet miserable day in February the CTO called a meeting with me and some teammates to discuss some recent problems with our data. He was justifiably nervous. An update in the recent past caused many of the reports that we supply to our customers to have faulty information. Other problems had been reported with customer-facing APIs where data was missing even after a year-long project to overhaul the data pipelines that fed the service. More pressing some recently signed contracts with high value customers included punitive payments should certain Data Quality issues arise. His ask from me and my team of Senior QA fellows was to “improve Data Quality” and ensure that we didn’t continue to have these issues.

As I had the most experience in testing our data centric applications I was quickly nominated to lead this effort. Our CTO hinted that what we wanted was a silver bullet of QA mastery that would settle the matter quickly and with minimal extra work or disruption to the yearly schedule. Having neither a tricorder or a wand of elm and unicorn hair to measure our datas’ output of qualitons, the fundamental quantum quality particle, I was forced to resort to existing practices, the vast quantity of Data Quality(DQ) related information on the internet, and my own experience to craft a set of Data Quality recommendations.

2 Definitions for Data Quality

The first step was aligning on a solid definition for Data Quality that we could work from. As we began researching the problem in depth we found that Data Quality was a widespread term used by different industries with differing focuses. Search engine results quickly informed me of the top selling Data Quality tools that purported to do the job for me at exorbitant price and without a lot of detail on how they performed the task. Other sites wrote intelligently about the business challenges of Data Quality without offering a real definition of what that meant, as if the audience just knew what quality was or was not. Quality data is “fit for intended uses in operations, decision making and planning” (Redman, 2008) or “fit for purpose” were common phrases. “No one complaining about your data” or data that “exceeds customer expectations” as an indicator of quality was also mentioned frequently.

While those sentiments captured the important essence and expectations of high-quality data, they did not directly lend themselves to an easy method of measurement. Like many descriptions of quality, they fall back to a “you know it when you see it” sort of description which I felt was lacking specificity. I was looking for something more proactive and concrete that could be reduced to actual measurements that can be regularly taken, tracked, and reported on. I needed data on Data Quality.

In desperation I turned to the Wikipedia article on Data Quality which listed similar statements as those above and concluded that what they all had in common was that “Data Quality is a comparison of the actual state of a particular set of data to a desired state” (Data Quality, 2021). Therein lay the foundation for a practical set of metrics that I had been looking for. To measure Data Quality, you must *compare* a set of data to some other set of data. How nearly it approaches the idealized or actual set used for comparison in various dimensions can, overall, be taken as a measure of general quality.

This definition fit well with the other nearly universally described practice of Data Profiling. Data Profiling is the process of measuring different aspects or Dimensions of Data Quality. Data Profiling in action involves the periodic scanning and aggregations of data to understand its structure, content, and adherence to business rules. It’s the most frequently cited technique used to determine Data Quality. Many of the tools that I initially found when searching for a Data Quality definition were focused on setting up regular data profiling jobs and displaying the results of those scans. Such a tool is nice but not required as data profiling can also be achieved with simple data scripts to count records or fields within a dataset that meet particular criteria and collect that data.

Now that I defined Data Quality to be measurable by comparison to some other data I needed some other data to compare to. Our frequent data related defects immediately suggested that comparing a source dataset to one that had been copied from it, a target dataset, as a great place to start. This is applicable

whether your target dataset is immediately downstream of your source dataset or if you want to compare a target dataset that had been copied several times through different transitions from some blessed and trusted source dataset, a source of truth. If all software were operating correctly your target dataset should exactly match your source dataset. I referred to this kind of comparison as Data Fidelity.

A second less obvious comparison can be made between a given dataset and an idealized version of that dataset that perfectly meets some business or real-world thing which the data is meant to describe. A dataset of Persons, for instance, could be idealized as having a surname, firstname, date of birth, and identifier for each record within the dataset. That can be compared to a real dataset that may be missing one or more of those attributes. I call this Intrinsic Data Quality.

Someone looking to understand their Data Quality by looking at Data Fidelity or Intrinsic Quality will find that there are a number of different comparisons and measures that can be made. Any given aspect of comparison between two datasets can be thought of as a Data Dimension, each of which should have a commensurate measurement, though some resist easy measuring. The key idea behind focusing on measurements was that we wanted some way to judge whether any changes we made to processes or programs had a positive, negative, or neutral impact on Data Quality. By taking actual measurements periodically, by gathering data on the data itself, we had a metaphorical yardstick we could use to judge whether our Data Quality improvement efforts were successful.

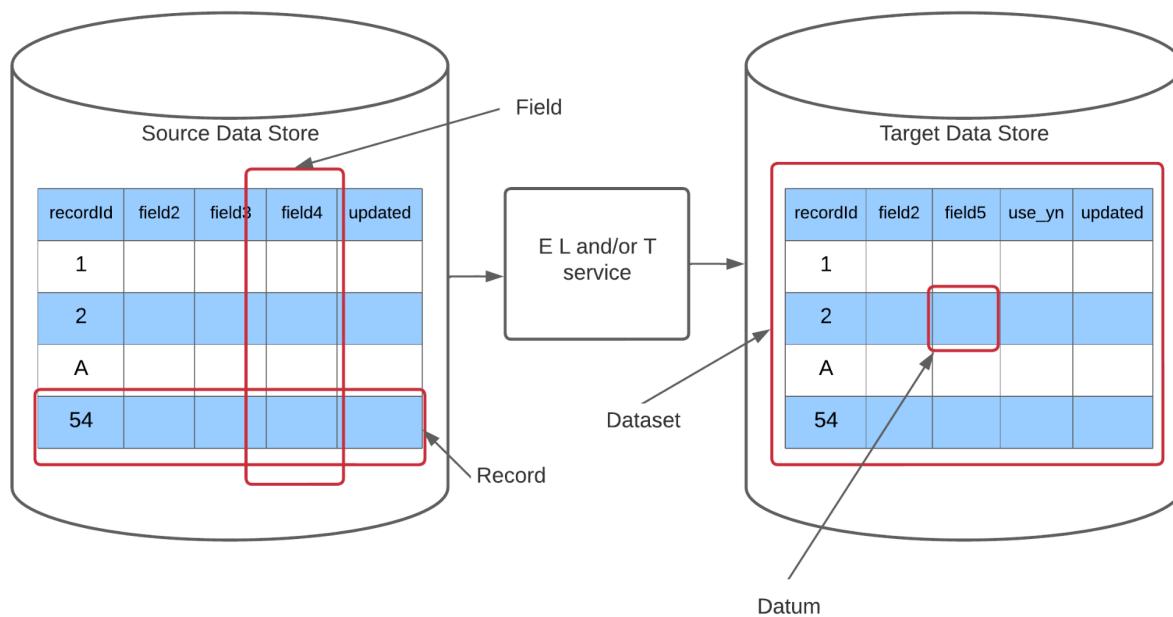


Figure 1: Data Quality terms

As we set up new processes to address Data Quality on a select few teams where troubles were prevalent we considered how to use data profiling and data comparisons to get to actual measurements, real data about our data. As we considered the best dimensions to explore we began by aligning on some basic terminology for the subject. For the purpose of this paper, and to avoid confusion or database specific terminology, I'll use the following terms

- Datum: a single piece of information that is necessary, useful, interesting, or important for operations, decision making, planning, tracking, or any other purpose as defined by business or technical needs
- Data: Multiple datum usually found collected together in some structured way.
- Dataset: A discrete set of data that is organized for a particular purpose or around a specific real-world concept that the data attempts to describe. Most often when measuring DQ a dataset will

- be considered equivalent to a single table of data from a relational database or a set of documents from a NoSQL database.
- Data store: A collection of datasets and the physical system that holds them. Equivalent to a database in most cases with some flavor of schema. Basically, a set of datasets.
 - Field: A named category of data that describes what the data represents and ties data to requirements regarding the data type, precision, format, and any applicable business rules about the data. In a relational database a field is equivalent to a column. In NoSQL databases it's usually equivalent to a single key-value pair wherein the key is the defined name for the field and the value is the datum. Fields include all the data in the dataset with that same field, in other words all the datum within that column of all of the values with the same key name.
 - Record: The set of data that contains all the information for one particular real-world or business object, entity, relationship, concept, measurement, or thing within one dataset. In relational databases a record would be one row of a particular table. In NoSQL data stores a record would be a single NoSQL document. Records should have either a unique identifier or a set of identifiers that collectively are unique which is usually called the primary key. Primary keys are not required but it's a real pain if they aren't there.
 - Source dataset: When data is copied between a dataset in data store A to a dataset in data store B the starting dataset in data store A is the source dataset
 - Target dataset: When data is copied between a dataset in data store A to a dataset in data store B the resulting dataset in data store B is the target dataset.

3 Data Fidelity

Among the most glaring problems that triggered our CTO to reach out to my team to build a broader foundation of Data Quality was a host of issues with missing data. Having adopted a microservice approach to architecture we had a number of stand-alone APIs and single page web applications that were engineered to rely on a local copy of data in order to achieve the performance desired by our customers and Product Owners. During functional and automated testing these applications performed very well and consistently passed QA checks. In Production however these services generated a continual stream of reports from customers of missing data. Sometimes as small as a single record, sometimes thousands. Root cause remained hard to identify or outside of the resources of the responsible teams to address. Teams were reduced to solving problems with direct data editing or some brute force tools to reload data from a trusted source dataset when customers reported problems.

To address this problem we needed to measure the fidelity of the various target datasets that resulted from a trusted source of dataset. We began doing so with some custom made tools that would query both the trusted source data store, a postgres database, and target stores that could be NoSQL or relational databases. The tool relied on universal unique identifier fields which were consistent across all data stores that some wise designer had previously established in our data. Thus we were able to perform record to record and field to field comparisons on data that we would otherwise have not been able to match due to data store dependent identifiers. When measuring the datasets we also needed to keep in mind any and all rules that may have intentionally omitted data from the copy, transformed the data during the move, or other factors that would legitimately change the data during the process.

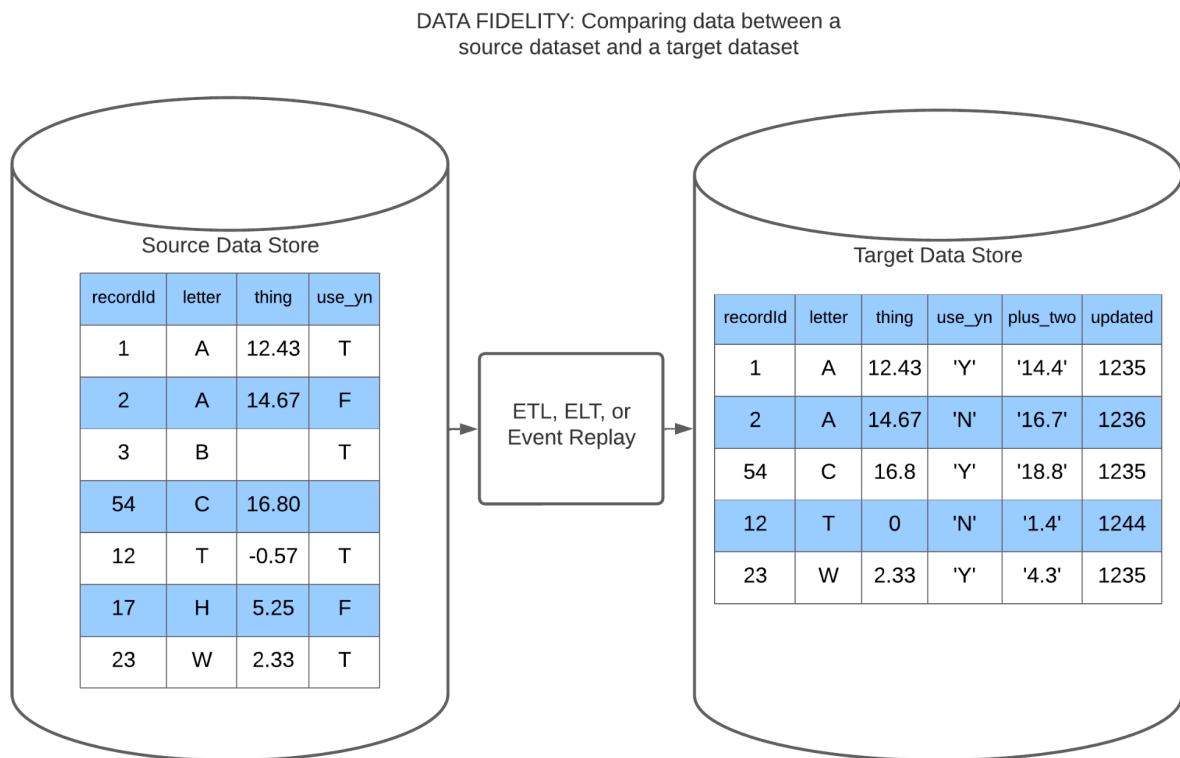


Figure 2: Dimensions of Data Fidelity

3.1 Measures of Data Fidelity Dimensions

3.1.1 Set-Completeness

Set-Completeness, also just called completeness, is a measure of the number of records in the source dataset that should have been copied to the target dataset compared to the count of records in the target dataset. In *Figure 2* above the Source dataset has 7 records while the target dataset has 5. Compare them together for each dataset to understand how many records have been transferred successfully. If the business rules of the copy are that all records in the source dataset should be in the target dataset then we have detected a quality issue.

This measure can be made difficult if there are business rules involved in the data transfer operation that omit some records from transfer. In such a situation you are required to understand the business rules and incorporate that logic into the programming doing the counting. In our *Figure 2* example if a business rule is that no records where use-yn is False (F) should be transferred then the record count on the Source dataset would be 5, the same as in the target dataset. No quality issue detected, even though the eagle-eyed will notice that it is the records with use-yn set to False that have missed being transferred. Thus Set-Consistency is not the sole measure of Data Quality you should use but it is often the easiest and quickest measurement to make frequently to determine if quality is slipping enough to warrant more extensive measurement. .

3.1.2 Consistency

Data between two datasets is "consistent" if the data for each record in the source dataset, or a selection of fields in each record, matches the record and field in the target dataset exactly. This is a high value measurement but also an expensive one to execute as it requires a record to record and field to field comparison of the data between the two datasets. When the datasets are in two different data stores, in possibly wildly different styles, the comparison is even more onerous to perform. In *Figure 2* a consistency comparison would compare each record in the source dataset with each record in the target dataset, matching on the *recordId*. Assuming the *plus_two* and *updated* fields are not directly copied we can omit those and just compare the *letter*, *thing*, and *use_yn* fields. We can see that the *letter* field is transferred with high consistency, having a 1:1 match with all records in the target dataset. The Consistency on *thing* is less ideal as the value in record 12 is 0, not -0.57. So for that field only 4 out of 5 records are consistent, maybe even 3 of 5 if you notice the 16.80 became 16.8. Further a deep comparison would also pick up either that records 3 and 17 are missing in the target dataset or that record 2 was included when it maybe shouldn't be.

Consistency is probably the most thorough measurement that can be made to assess Data Fidelity. However with large complicated data sets compounded by complicated business rules it can become time consuming and expensive to run frequently.

3.1.3 Validity/Precision

Data between two sets is "valid" if the data in each field matches the requirements for numerical precision (number of decimal places as set by business rules), data type, and format. For calculated values it matches the expected result from independent calculation given the same inputs and determined by business rules. In our example datasets in *Figure 2* we can assess the rounding rules and data type for the *thing* column and the calculated *plus_two* column. The *thing* column in the target seems to match the values and precision of the source dataset exactly, except for the 0 for reportId 12. The *plus_two* field however may have some issues. Despite being a numerical calculation it is being stored as text. Secondly the resulting value is rounded to one decimal place, not two like that addition of 2 to the source *thing* data would suggest. Depending on your business rules all of those values may be invalid. A Validation/Precision measurement can be expressed as the number of invalid records over the number of records measured.

3.1.4 Transformational Integrity

Data that has been transformed during movement has been transformed correctly. Data is transformed for a huge number of reasons and those transformations should be checked to make sure they were executed correctly. In our example data there has been a clear transformation of the values in the source dataset *use_yn* field from boolean True and False values to the odious 'Y' and 'N' text values. Comparing the source and target datasets will involve performing the transformation being done by code again in your profiling tools and validating that the transformation of the profiling tool does based on the source data matches what is stored in the target data. The overall transformational integrity can be expressed as the number of records correctly transformed over the total number of records that should have been transformed. We could score our example in *Figure 2* as having 4 out of 7 records transformed correctly.

3.1.5 Exactness

The data in the target dataset has the same meaning and names as the data in the source dataset. This is a more manual measure to make than the others as it requires a human (for now) understanding of the meaning of the fields and the data within them between the source and the target data. Our example data may have high Exactness as the field named between the two datasets closely match and seem to describe the same data. However we may have a Data Quality issue if the *use_yn* field in the source dataset is meant to express whether that record is useful in the target dataset and the *use_yn* field in the target is expressing that the data for that record is useful in a particular report or application. In that case

those fields would fail an Exactness test and you could say that only 3 of 4 fields in the target data set were Exact.

Exactness is a lesser used dimension that might come up as data proliferates to independent data stores, each of which can have their own business rules and naming conventions. It is important to keep all data consistently named with what it means. Duplicate names that mean different things within the same are a fantastic way to make poor Quality Data when one developer or team makes the wrong assumption.

3.2 Other Data Fidelity Dimensions

There are a couple of Data Fidelity Dimensions worth mentioning that are difficult to apply direct measurement to but are worth keeping in mind.

3.2.1 Cross data connections

Data that is related between datasets. For instance, if one field is dependent on the values of other fields, or even on an aggregation of data in other datasets, you'll need to develop bespoke profiling rules to validate the data. Many of the measurements you can make in pursuit of Data Quality fall into this category.

3.2.2 Timeliness

Timeliness is a measure of the time it took to update data in a target source dataset from when it was updated or created in a source dataset. This usually requires a comparison of explicitly added fields from the source dataset that record when it was last updated and similar fields in the target dataset for the same record. The gap between the two is your timeliness for that record. An overall accounting of the timeliness of each record in the target dataset should identify situations when data is taking longer than normal to propagate or even when the copy has failed if the time the source was updated is later than the time the target was updated by more than an acceptable amount..

3.2.3 Data Legacy or Data Lineage

A record of how the data got from the source dataset to the target dataset that you are testing from the first time that it was entered or generated by your software and through any and all transformation steps. A data legacy or lineage can be generated from a variety of tools and can be extremely useful in troubleshooting data problems. Data Lineage is difficult to measure as it usually requires programming the collection of data on how records moved through the system which creates yet more data that needs QAed. The updated field in our example is a rudimentary kind of data lineage that tracks when a record was last updated in the target dataset. More complicated ones could include timestamps for when data was modified, what process modified it, and almost anything else that is deemed relevant.

4 Intrinsic Data Quality

Once progress was underway to measure our Data Fidelity issues we turned our attention to finding ways to validate a dataset when we could not simply compare it to some trusted source data. As with our Data Fidelity problems there were a number of high profile defects that pointed to weaknesses in our understanding of our data and its consumption. In one instance a number of frequently used reports were displaying data that did not agree with each other even though they were using the same source data when generated. In other cases we had reports from data analysts that they were spending up to "80%" of their time scrubbing and cleaning data from our production data stores before beginning analysis. For these we needed to examine qualities within our dataset not simply compare them to data from which they were copied and assume our sources were flawless.

In the examined literature the most frequent dataset for which quality was a concern was a defined dataset that someone needed for a technical or business operation, a purpose. In the case of our company, that was data gathered from school districts detailing student, school, teacher, and class information, relationship data about which students were attending which schools, taught by which teachers, and other details of school district operations that needed to be reflected back to the customers and augmented with our own data. Whether it is school data, or online order details, or bank transactions the data overall needs to be of high quality to allow for operations to run smoothly and for later data analysis to aggregate and sift in the hopes of finding patterns for further business opportunities. According to our definition of Data Quality in order to determine the quality of an independent dataset, even indirectly, we must compare its actual state with a desired state. In other words some other dataset. a comparison of the actual state of a particular set of data to a desired stateWhen you want to measure the quality of an independent data set you compare it to an idealized version of that same data.

The Idealized data need not be actual. It's often sufficient to have a description of what the idealized data will look like and go from there. For instance, identifying and putting limits on which fields in a dataset could be NULL or empty and identifying what percentage of empty datum in a given field across the dataset gives a good indication of whether or not you are missing data. A common approach to defining what goes into an idealized dataset is to establish a data dictionary that explicitly lays out, in technical, business, and common language, what the meaning and limits for all fields in the dataset should be. Each method of idealization is a Dimension which could potentially be measured by doing full scans on a dataset and collecting aggregate values for which records or fields fall outside of the ideal state..

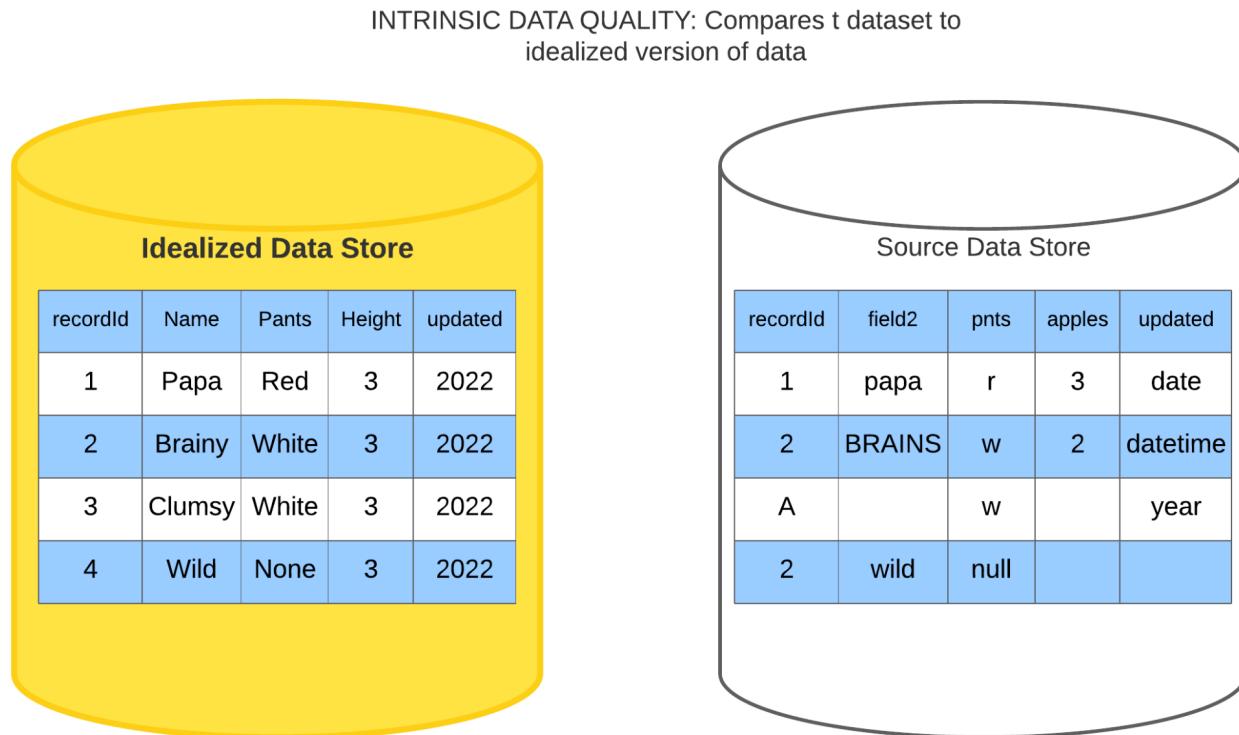


Figure 3: Dimensions of Intrinsic Data Quality

4.1 Measures of Intrinsic Data Quality Dimensions

As mentioned there are many data dimensions. In fact some of the more useful ones are probably defined more by the business specific nature of the data that is being analyzed than by any standard set. I wanted

to focus on specific and somewhat generic Dimensions that can be easily measured with automated software tools. Note that not all of these dimensions apply to all datasets.

4.1.1 Completeness

A "complete" record would have values in every field. Incomplete records are missing data.

Completeness overall is a measure for each field of how many records are missing data in that field. A usual example would be US addresses. Each record is presumed to have a street number, street name, city, state or province, zip code and some optional fields like apartment number. A dataset of addresses could be measured for completeness in every field and compared to expected results. All records should have a zip code, city, and state or province record, so any missing those values would be considered incomplete. Other fields are usually, but not always, required (yes, even street names and numbers can be omitted in strange situations) so your dataset may have 99% of records with street names and still be considered within quality bounds. Apartment numbers would be lower still. If 30% of your records have apartment numbers that might be high quality or low quality depending on the nature of the data.

In the example shown in *Figure 3* a complete record would be expected to have data in every field. Comparing that to the dataset that we're testing we can see that a number of the fields are missing. Under field2 we find 1 out of 4 records are missing. Thus we can say that that field for this dataset is only 75% complete.

4.1.2 Uniqueness

Records in a table are unique if they have identifiers and values which are not duplicated anywhere else in the dataset. Identifiers generated and used by data stores often enforce uniqueness and will not allow the storage of non-unique values in a field that is an identifier. Other fields are not so lucky. In either case it is still recommended to count unique values from your dataset and report a number of unique values over the total number of records. If those values don't match then some values are not unique. Most fields in a dataset will not be expected to have unique values, but those that are defined as unique usually MUST be so and having duplicates can cause a number of high profile data defects.

4.1.3 Validity

We can say a datum is valid if it is of the correct data type, format, value range, and precision as would be expected for what the datum defines. A price, for instance, would be valid if it was expressed in a legitimate currency (US Dollars), was a number data type and had two decimal points of precision. A negative price would also be invalid, as would a price with two decimal points, or a non-numeric symbol. In *Figure 3* the data in the updated column would all be invalid as they are the wrong type, being descriptions of dates, not years. Measure the validity field by field in a dataset by seeing if it compares with the established pattern for values of that field as established by convention and business rules and documented in a data dictionary. The number of valid fields divided by the number of fields tested is a validity value for that field.

4.1.4 Orderliness

Data is "orderly" if it conforms to the required structure defined for the data. This is a great dimension to use for higher order structured data like JSON or XML fields. Similar to Validity, high quality data that is Orderly would match a pattern of how the data is supposed to be structured according to a data dictionary or other business rules. A simple example would be a 16 digit credit card number that should appear as 4 sets of 4 numbers. Orderly data would match that pattern while less orderly data might omit the spaces between the data sets at times, string the numbers together, or have less or fewer numbers.

4.2 Other Dimensions not as easily measured

There are a number of important Dimensions of a dataset that are useful to consider but hard to measure in an automated or manual fashion. Nevertheless the presence of these dimensions in a dataset can enhance its quality.

4.2.1 Auditability and Legacy

Data that is “auditable” contains fields and data in each record that track when the record was last changed and by what user or service changed it. More sophisticated systems can even keep and continually update a list of all changes to a record from its initial creation until the current time, which is referred to as a legacy. While it is more in the realm of functional testing to validate that the audit fields are updated correctly when records change, their presence in a dataset enhances the ability to troubleshoot data issues and determine how poor quality data entered the system and what user or service is responsible.

4.2.2 Privacy

Data is private if it is only accessible by authorized users and services and is otherwise protected. One could measure the effectiveness of Privacy conditions on fields by attempting access during Profiling with both authorized and unauthorized users.

4.2.3 Accuracy

Accuracy has a variety of definitions. A common one is to say that data is Accurate if it exactly describes the real world object or process which it represents. That is extremely difficult to measure with any sort of automated process, or manual ones for that matter. In some cases Accuracy can be measured if an established source of absolute truth can be used instead of an idealized dataset, for instance postal addresses can now easily be validated against the list of all valid addresses kept by the USPS. That does not tell you if that's the correct address for a particular person though.

4.2.4 Name Correctness (Aptronymity)

Names of fields describe the data the fields contain. Names of datasets should describe what the individual records within the dataset collectively describe. When either the data in the field changes to reflect something other than the field or dataset name describes or when business changes the names of terms and the datasets don't keep up with those changes then the names in your data become misleading. At best they are quirks of your data model that must be consistently kept in mind when working with it or when consuming that data. At worst they can become active points of confusion that consistently cause defects.

5 Meta-Data Quality thinking

Once we had build some data profiling tools and set teams loose on making data quality measurements we needed to start thinking of ways we could improve our development and data handling practices to improve quality. The first method, to make actual measurements so we can make quality more visible, was finding more defects and resolving more issues than years of customer complaints had. Having a set of measurements taken over time created a baseline of Data Quality that we could use to determine if any other initiatives to improve Data Quality were actually working. That, after all, was the main reason to focus on measurements.

So we began looking more closely at the list of defects that had caused issues and tried to get a deeper understanding of causes. In many cases we found that development teams had made mistakes in programming based on faulty assumptions of what data within the system meant or was used for. In others we found situations where business rules restricting data movement were established some time in

the past, poorly documented, and then forgotten completely by which time the rule looked more like a defect than an intentional omission. Specific defects were caused by incorrect assumptions made about the data they were ingesting which then became bad test cases based on those assumptions. Our customers knew better.

I began putting together a list of better practices that would hopefully lead to better Data Quality, a kind of meta-Data Quality practice. We are still refining the practices and in the end these better practices may not be better. We can only tell by leaning into the measures that we're gathering from the various dimensions as well as the ultimate lagging indicators of quality, Production defects and customer complaints, to tell if any particular practice is providing higher quality data or not.

5.1 Hopefully Better Data Practices

5.1.1 Data Stewardship

Data Stewardship is the practice of assigning a subject matter expert, called the Data Steward, to be responsible for the Data Quality and lowest level Data Governance of particular datasets. The Data Steward needs to be knowledgeable about the data they are responsible for, understand what it represents in the real world, understand what it means for the business, how it's been transformed, and how other services are using the data. A Data Steward can help a development team understand their data and what it means which I hope will prevent the kinds of defects we saw that were caused by developers blindly processing stories to transform data without really understanding why. A Data Steward can also help run periodic profiles on data they are responsible for and process the results for broader consumption..

5.1.2 Data Dictionary

Build and maintain a Data Dictionary which details the names, meaning, and business and technical rules that apply to each data field and dataset. Keep it up to date. This is a great way for the Data Steward to communicate to teams about what the data means and how it's being used. It is also a key component in several Data Quality dimensions like Validity, Orderliness, or Uniqueness.

5.1.3 Track your data defects

Prior to our Data Quality initiative we did not categorize our defects based on anything other than the service that caused the problem and the team responsible for fixing it. Now we have implemented a field on our defects which allows us to track whether a defect is related to data issues, software functionality, UI, or other categories. This allows us to better analyze our data periodically to see if our Data Quality efforts are paying off in a reduction in data related defects in total or as a percentage of overall defects.

Since we have started tracking our data defects, which coincided with our other efforts around data profiling and measurements we have seen a marked reduction in the number of data defects being reported and have had no high profile defects..

5.1.4 Implement Data Quality checks within your data pipelines

One of our teams found great success in building their data profiling into the build pipelines they have to deploy new code to production. Running each time they deploy new code and on a nightly basis their tests measure for Completeness, Validity, Uniqueness, and Consistency of a presentation layer of data that one of our reports depends on. They have then taken that data on their Data Quality (talk about meta-data) and created a dashboard compiling it and showing recent and past results, trends, and other automated tests to get an overall view of the project and data pipeline health.

5.1.5 External profiling

Run external profiling on your data. Do not just depend on the QA testing that went into testing the services that move or manipulate the data. Especially with microservice architectures weird things can happen that cannot be tested or predicted from the QA of individual services alone. You need independent testing of your data itself to validate its quality.

Once you have set up profiling to run on a regular basis, capture the results and routinely analyze them. Compare profiles across time as the data changes to get an idea of what your stable baseline is in measurements like Completeness or Timeliness. Further your Data Quality journey by combining your profiling data with the release dates of data impacting services, process changes meant to improve Data Quality, or anything that may alter Data Quality to try and quantify its impact, positive or negative. Combined with an easy reporting dashboard it's a great way to understand where efforts to improve Data Quality are working and where they are not.

5.1.6 Build and ship manifests

When moving a lot of data in bulk include additional meta-data like the total number of records being sent and aggregate values for that set such as a MD5 hash of all values in a particular field. Receiving services in data pipelines can then check that all data was received and appropriately processed by comparing what data they received against the meta-data..

5.2 Practices to avoid

In addition to practices we hope will avoid some of the problems we saw that caused issues with our Data Quality we found a number of practices that seemed to directly contribute to reducing Data Quality. These are to be avoided, if possible.

5.2.1 Multiple tools to ETL/ELT the same data

A practice that we've seen repeatedly was for a team to develop a data pipeline to ETL/ELT data from a source data store, the parent system where data originated, to a target data store, a new system that needed data to operate, on a periodic basis varying from 5 minutes to daily. The data pipeline was rigorously tested and exercised to validate that it was capturing all data additions and updates in a source data store that occurred in the data since the last time the pipeline had run and that the pipeline was able to build up the data in the target data store to be an accurate copy of the source.

Teams then determined that the data pipeline, as designed, was insufficient to move the existing mass of data in the production source data store to the target. So they proceeded to develop a separate tool to load all historic data from the source into the target as an "one time" initial data load.

Inevitably, when Data Quality issue arose this one-time tool was called into action to realign data that was missing or flawed by forcing all data for a particular customer from the source to be refreshed in the target data store. Now any changes to the business logic had to be applied to both tools, both tools needed testing routinely, and any time a Data Quality problem arose with the data in the target data stores we would have to determine which tool was causing the problem.

We recommend that this is a practice to avoid. Rather than have multiple tools to copy the same data it would be preferable to have one tool that applies the same transformations to all the data that it is moving and can be configured to switch between updating with newly entered data or historic datasets.

5.2.2 Copying partial datasets between data stores

On several projects over the years we made decisions when developing ETL/ELT data pipelines to only move part of a dataset between two data stores. The usual logic at the time was that we wouldn't ever need the data left behind in the new data stores. Inevitably as use cases changed for what the new

service was doing and people forgot the original constraints the data that was being left behind was wanted, or worse, was just assumed to be there and defects filed when it was found to be missing. Development that could have easily handled the missing records when they were originally omitted now had to be forced into newer logic that never anticipated the ways the left behind data was different. .

For instance if you have a source dataset of Orders which are in a variety of states or possibly even Invalid do not build an ETL service that only pulls the Valid orders for reporting purposes into a new reporting focused data store. Bring all the orders, similarly transformed, and filter them at the source when generating reports or build a more ELT style service that loads all the Orders into a raw staging data store that is then used to generate more useful data from which Invalid Orders can be removed.

5.2.3 Records lacking a cross data store unique identifier

If you do any ETL/ELT operations between data stores, make a unique identifier for all records of data that is agnostic to the data store. This enables you to very easily and concretely track that data as it moves through different data stored by unique id. This is in addition to any data store specific identifiers that you might need. We call this unique identifier the “business identifier”, or BID, other places refer to this as a master ID. Not having a business identifier in some of our data has resulted in confusion and time wasted on troubleshooting that could have easily been avoided if a BID were present. Further we recommend that if you have a BID make it UUID type as it is less prone to be duplicated accidentally.

5.2.4 Keeping existing database field names when business changes the name

Time rolls on and businesses love to change their nomenclature. A company may rebrand or may change the names for the data that it provides to customers for better clarity. For whatever reason if you have a data field, a dataset, or even a whole data store that is named for something that made sense years ago but no longer reflects the business operations that the object is used for or describes- change the name, and track the old names in the data dictionary. To not do so builds a continual debt of historical understanding that all new and existing employees must be taught and keep in mind when working with the data.

6 Conclusion

Data is an essential component of business operations. To ensure that data is of high quality, reliable, and fit for purpose it is necessary to test your data apart from the services that create and manipulate it. Testing data to determine its overall quality requires techniques unique to the task by comparing your data to known or idealized datasets and making specific and regular measurements of Data Quality dimensions independent of other QA efforts to ensure that high quality software is developed. Further, you can support good measurements with good governance of data by Data Stewards and Data Dictionaries. Once a baseline of Data Quality is established you are well positioned to try better data handling practices or retire poor practices and see how your Data Quality is impacted.

Data Quality, all the cool kids are doing it. We implemented data profiling and measurements in a number of teams with problematic data and saw a marked improvement in Data Quality as measured by a decline in Data related defects and improvements in the measurements that we were taking over time. Meta-data practices to help team members know where their data comes from, to know where it's going, and to know what it means have helped them build in better quality during development and avoid common pitfalls.

Data Quality issues are an overall detractor from business efficiency. Conversely high Data Quality, proven by routine measurement, gives leadership confidence in their data and data driven decisions, customers confidence in your product, and developers confidence in their software's data handling abilities.

References

- Data Quality.* (2021). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Data_quality
- Redman, T. C. (2008). *Data Driven: Profiting from Your Most Important Business Asset*. Cambridge, MA: Harvard Business Review.

AI Software Bugs Segregation System: DSF

Ooi Mei Chen; Liu Keping

mei.chen.ooi@intel.com; keping.liu@intel.com

Abstract

Software bugs fixing always will be expensive. A complex software solution usually consists of multiple sub-components or combinations of multiple software solutions. The more software sub-components integrated; the more Software bugs occur. Most of the time it is exceedingly difficult to identify which sub-component breaks the end-to-end functionality.

Data-Search-Fix (DSF) is an AI software bugs segregation system that search and analyze the historical data in a smart way. Recommend a solution to narrow down debugging scope and reduce debugging cost. DSF consists of 3 major blocks - "Data Collection", "Search Algorithm", and "Fix's recommendation" on Dot net and MSSQL (Microsoft SQL). The data and search algorithm are designed in a form that can be easily swapped with other data or algorithms to fulfill the specific needs for different software communities.

This paper will explain DSF software bugs segregation system detail end-to-end flow with a deep dive example. With the implementation of DSF shows productivity improvement by demonstrating quick turnaround time to identify the issue owner and root cause the issue.

Biography

Ooi Mei Chen has been in software engineering for over 12 years and has held many roles spanning code development, design, and project management. She currently serves as a senior System Software Quality Engineer at Intel Corporation based in Penang, Malaysia. She holds a Degree in Computer Science from University Tunku Abdul Rahman, Malaysia.

Liu Keping is a Technical Leader in Software Quality Assurance at Intel Corporation based in Shanghai, China. She is a certified CMMI assessor, ISO internal assessor, ASPIICE internal assessor, CSQE (Certified Software Quality Engineer), and gained 6 Sigma Orange Belt and CPMP certification since 2009. She holds a master's degree in computer science and technology from Central South University in China.

1 What is the Problem?

Software development is a creative work based on a large amount of people's knowledge and skills. If people do not catch their mistakes during code implementation, it will lead to bugs in the products. And software bug fixing will always be expensive. A complex software solution usually consists of multiple sub-components or combinations of multiple software solutions. The more software sub-components integrated; the more Software bugs occur. Most of the time it is exceedingly difficult to identify which sub-component broke the end-to-end functionality.

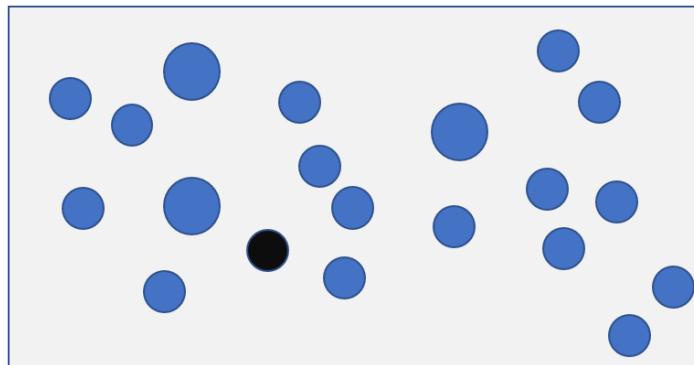


Figure-1 Where is the bug?

The cost spent on bug fixing is increasing as the development phase evolved, straightly burst out after shipping to customer. See Figure-2 the research result from Jones, Capers in "Applied Software Measurement: Global Analysis of Productivity and Quality."

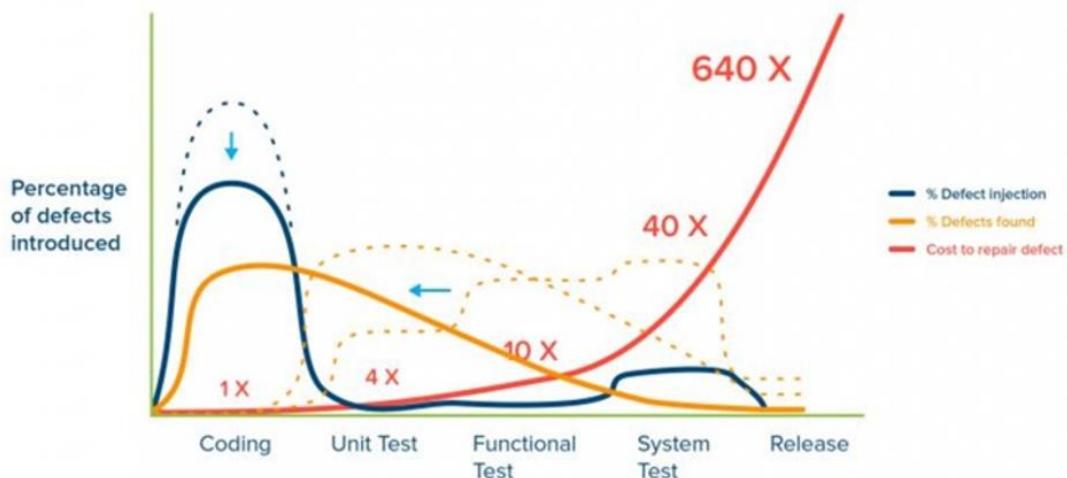


Figure-2 Bug Fixing vs. Cost

Thus, an effective approach to find the root cause of a bug in a quick manner and finds the right injection point which really caused the problem is extremely important. But it is always difficult!

2 Why is this difficult?

Unlike software development which intends to construct a system according to a well-known specifically defined product requirement, bugs fixing is working on an opposite way that aims to find the root cause

and fix potential hidden problems that have occurred in the code database. It is unpredictable and not as planned.

Complex dependency between sub-components is a main factor. A typical example is that: the subcomponents passed the component level unit test but failed in the end-to-end integration test. It is extremely hard to identify which component caused the real problem. This happened frequently, especially in large collaborative product development. Though the team has identified component dependencies in the beginning, with new features/function added in, new dependencies are brought into the system which are impossible to uncover thoroughly. Even if all the dependencies were obvious, it still takes time to figure out the exact "suspect."

Many software development engineers lack of the overall end-to-end system knowledge is another main factor. When bugs occur, the sub-component development team does not have the end-to-end knowledge to debug the issue.

Some other factors which contribute to the difficulty include

- Critical quality assurance process steps are skipped occasionally when packaging in release phase. Multiple check-in requests submitted at the same time within a short duration. Some files are skipped or overridden which is easy to overlook.
- Responsibilities are not clear which requires skillful communication capability to get the debug work to move ahead. For example, there is a disagreement between the subcomponent teams on where the bug resides and no one willing to dig it out first.

3 Approach

The Data-Search-Fix (DSF) is an AI software bugs segregation system that searches and analyzes the historical data in a smart way by recommend a solution to narrow down the debugging scope and reduce the debugging cost. The DSF consists of 3 major blocks - "Data Collection", "Search Algorithm", and "Fix recommendations" on .NET and MSSQL (Microsoft SQL). The data and search algorithm are designed in a form that can be easily swapped with other data or algorithms to fulfill the specific needs for different software communities. The main goal is not to eliminate the debugging work but to identify the potential sub-components and suggest the potential root cause and debug direction with the statistic of the bugs and root cause occurrence.

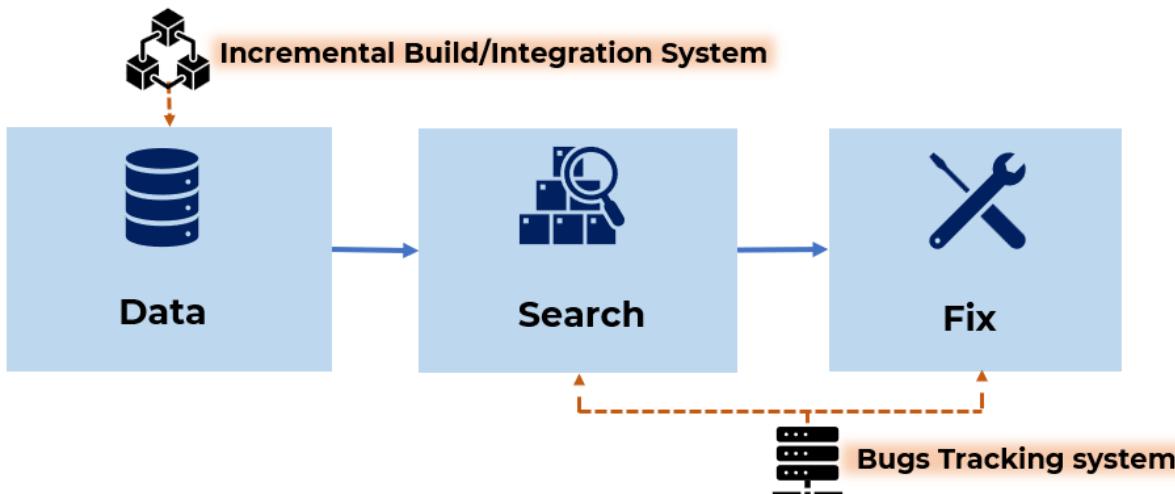


Figure-3 Data-Search-Fix (DSF) AI software bugs segregation system

3.1 Data

Collecting high quality data base on integrated software behaviors is the foundation of building effective data searching and analysis. The DSF has demonstrated an example to store the data in MSSQL and sort the data based on the sub-component functionality complexity, keyword, and injection phase. Data collecting and sorting needs to be considered carefully by the DSF designer with integration, build and software development teams to make sure it corresponds to the search algorithm and is flexible and scalable when the data is used in “Search Algorithm” and “Fix Recommendation”.

3.1.1 Data Storing and Sorting

The “Data Collection” block of the DSF determines what types of data need to be collected. Below are the recommended data that can be used for analysis and segregation.

- How many sub-components or solutions have been integrated into the software
- Which sub-components have more code change or new functionality
- Clear software architecture design, incremental build sequence and dependency of each of the sub-components
- Group the component that have the strong dependency
- Consider the historical data like “root cause,” “injection phase” and other Root Cause Analysis entries from existing root caused bugs captured in bugs tracking system

One more thing to remember is standardize the bug fields like “key word,” “issue log” and “error log” which will benefit the search algorithms. The reason these data are collected and how they will be used will be explained below in [section3.2](#) and [section3.3..](#)

3.2 Search

There are many search algorithms that can be used for bug segregation, such as binary search, linear search, interval search, interpolation search etc. Considering the data structure being searched and prior knowledge about the data, DSF decided to use the binary recursive search algorithm to narrow down the sub-components that have the failure test case.

3.2.1 Identify most possibly went-wrong sub-component with Binary Search

To isolate sub-component which breaks the end-to-end functionality, the DSF when supported by an incremental build system also automates the sub-component segregation and end-to-end integration test together with the selected search algorithm. The example below explained how DSF uses the binary recursive search algorithm on a complex software solution that has 20 sub-components about 4 times, and finally narrowed down the issue most possibly went-wrong in component11 to 15.

- **Arrange and find the Mid**

Based on the data collected in 3.1.1, “Which sub-components have more code change or new functionality” this data will be used to sort and define which component is less or more complicated. Secondly, “How many sub-components have been integrated into the software” this data will be used when DSF search divides the total components into two halves and finding the middle index “Mid.” This example we choose to measure the complexity by line of code change in each sub-component. Then the arrangement of a less complicated component on the left and a more complicated component on the right. In the example, in total, 20 components and component10 is the middle index. Note that the component number is not fixed but determined by actual project complexity.

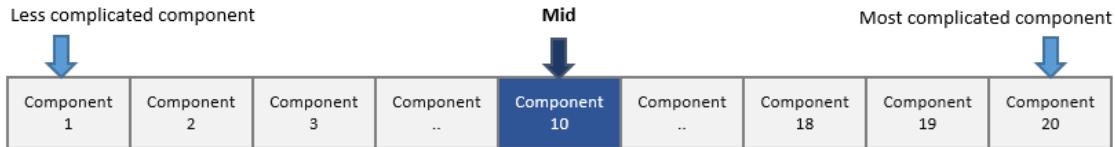


Figure-4 Arrange and find the Mid

- **Pinpoint the bug via integration test**

To isolate sub-component which broke the end-to-end functionality, the DSF when supported by an incremental build system, also automates the sub-component segregation and end-to-end integration test together with the selected search algorithm. The binary recursive search will start with the first half components from 1 to "Mid" (1 to 10 in this case). Continuously integrate and test to see if the bug existed in the first half.

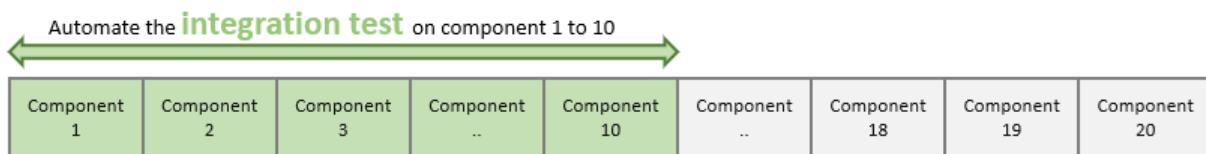


Figure-5 Automate the integration test

If the bug was not observed in component1 to component10, the binary recursive search will proceed to execute the integration and automate testing on the second half (11 to 20 in this case).

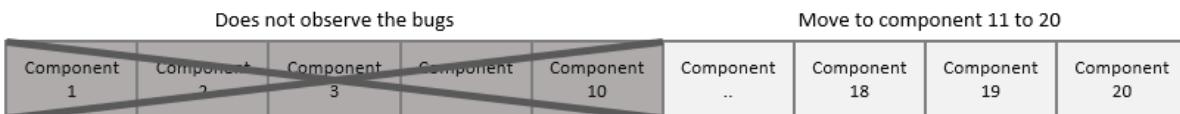


Figure-6 Exclude and move to next block if no bug identified

- **Identify 5 most possibly went-wrong sub-component**

Same as above, the components 11 to 20 are arranged with less complicated components at left and more complicated components at right. By catching the test case failure, finally we observed the bug potential happen in components 11 to 15 - the 5 most possibly went-wrong sub-components. For cross check purposes, the integration and validation can be extended to execute on the second half of components (16 to 20 in this case). This search algorithm can continue until the last sub-component or stop at certain number of sub-components based on the goals that the DSF designer wants to achieve,

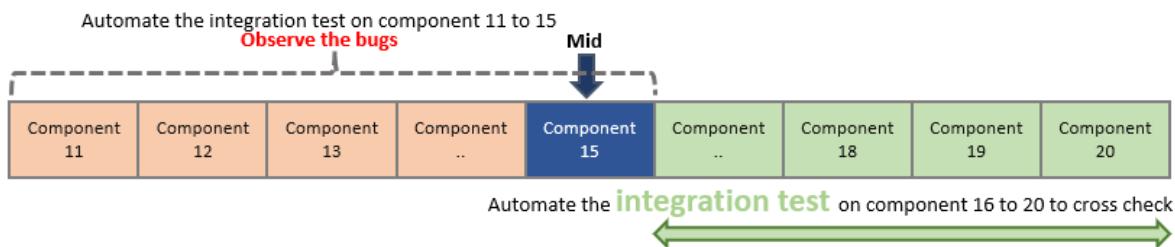


Figure-7 Identify 5 most possibly went-wrong sub-component

```

5 int binarySearch(int component[], int size) {
6     int low = 0;
7     int high = size - 1;
8     int mid;
9     bool found = false;
10    int start_component;
11    int end_component;
12
13    //Recursive check to look for fail integration test
14    while ((low < high) && (found == false))
15    {
16        //Find the mid component by dividing the component list into half
17        mid = (low + high) / 2;
18
19        found = LookForFailIntegrationTest(low, mid);
20        if (found == false)
21            low = mid;
22    }
23
24    if (found == true)
25    {
26        start_component = low;
27        end_component = mid;
28    }
29
30    return -1;
31}

```

Figure-8 code snippet for component binary search

According to the code snippet in figure-8, once the found is “true” then it will suggest the most possibly went-wrong sub-components that have the failure test case, the next step is to use the “Fixes Recommendation” function to find the recommended solution based on historical data.

3.3 Fixes Recommendation

To construct the accurate fixing or debugging direction, analysis and filtering needs to be done before the recommendation can be made. Continuing with the example above, for the most possibly went-wrong sub-components, DSF will use the decision tree machine learning algorithm to further process based on historical data like “keyword/issue log/error log,” “root cause,” and “injection phase.”

3.3.1 Analyze the bug’s historical data with Decision Trees in Machine Learning

Decision tree learning is a supervised learning approach usually used in statistics, data mining and machine learning. In this proof of concept, classification trees are used to draw the conclusion with “Yes” or “No” outcome. In the example, DSF will search for the bugs occurrence in the historical data based on the bugs description, similar keyword, injection phase and other Root Cause Analysis entries that been tracked in the existing bugs system.

- **Step 1: Check if the bug happened before**

Based on the most possibly went-wrong sub-components identified, DSF will proceed with the last block “Fixes Recommendation” and reporting. The decision trees start at the (root) node with a question “is the bug happening before?” The answer will lead to branches that hold potential answers “Yes” or “No.”

If there was a similar bug description found in the historical data, the path will proceed to “Yes,” DSF will generate the report and recommend the root cause or debugging direction.

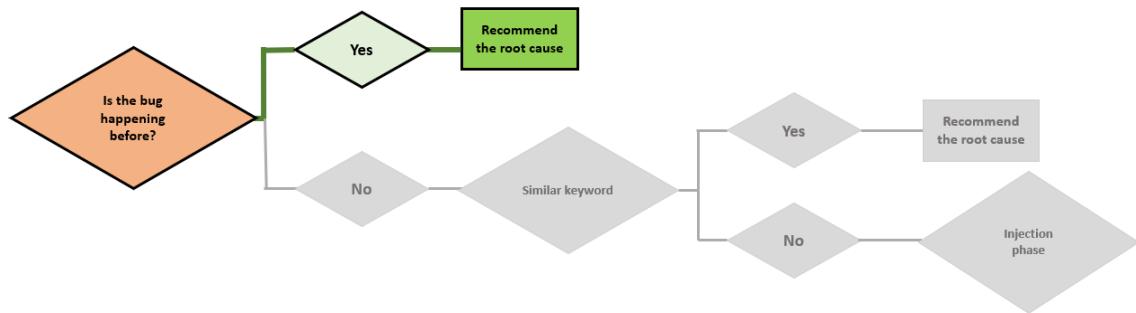


Figure-9 Check if the bug happened before

- Step 2: Check if there was any similar keyword**

If there was “No” similar bug description, the decision tree will proceed to the next decision criteria (branch) “Similar Keyword.” In this step, the decision tree will use the data collected earlier in section 3.1.1 to filter and find the error log or bug's keyword.

If the answer was “Yes” with similar keyword, DSF will generate the report and recommend the root cause or debugging direction.

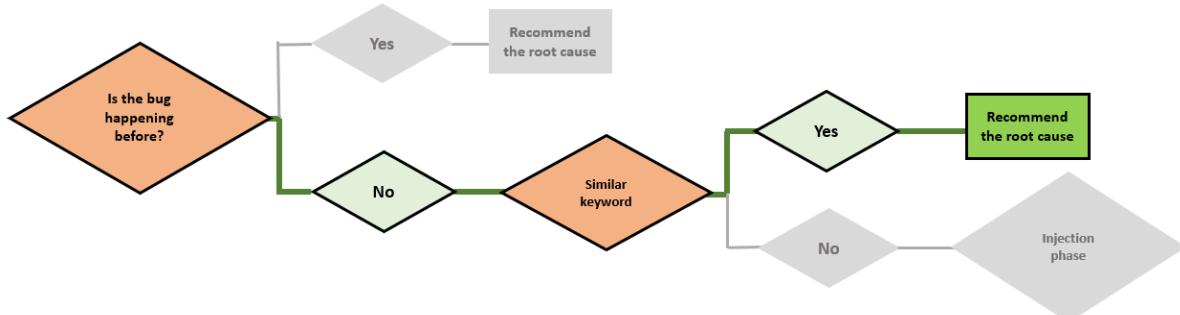


Figure-10 Check if there is any similar keyword

- Step 3: Check the sub-component injection phase**

If the answer was “No,” the decision tree will go on to the next query “injection phase” until the “Yes” reaches or terminal (leaf) node meets checking criteria.

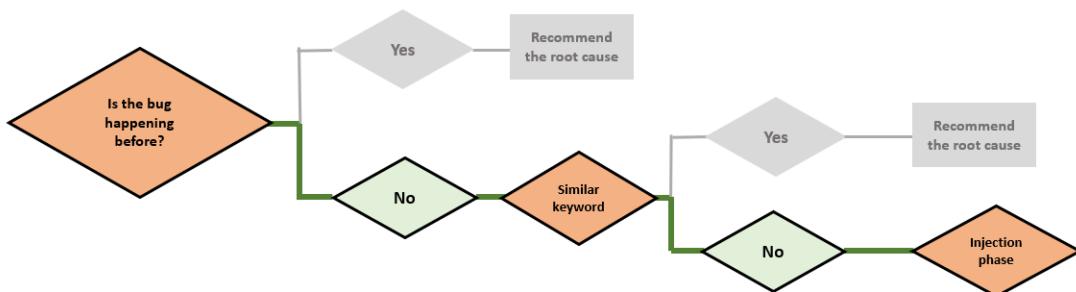


Figure-11 Check the sub-component injection phase

```

1 DecisionTree* createDecisionTree(vector<string> attributes, vector<vector<string>> issues) {
2     if (issues.size() == 0) {
3         return nullptr;
4     }
5
6     // Find best attribute to split the issues
7     int bestAttributeIndex = FindBestAttribute(attributes, issues);
8
9     // Create a new decision tree node with the best attribute
10    DecisionTree* node = new DecisionTree();
11    node->attribute = attributes[bestAttributeIndex];
12
13    // Recursively create child nodes for each issue with the best root cause
14    for (int index = 0; index < issues[0].size(); index++) {
15        string value = issues[0][index];
16        vector<vector<string>> filteredIssue = filterIssue(issues, bestAttributeIndex, value);
17        node->children.push_back(createDecisionTree(attributes, filteredIssue));
18    }
19
20    return node;
21}

```

Figure-12 code snippet for recommendation decision tree

As stated in figure-12 the decision tree scope can be controlled by the list of the vectors. The decision tree can continue until it recommends the desired number of root causes.

In this example, the final DSF report consists of the most possibly went-wrong sub-components, and the potential root cause and debug direction of each sub-component, with the statistic of the bugs and root cause occurrences.

4 Learning and Challenges

While AI based DSF presents an incredible efficiency in bug segregation, there are some limitations we must face. Learning and solving challenges is a continuous effort to improve the maturity of a system. DSF has gone through the POC (Proof of Concept) and fine tune stage to ensure the end-to-end system is working fine and the result is trustworthy. Below are some of the key elements that need to be considered to make DSF successful:

- Easier coupling and decoupling:
The software architecture is suggested to be designed with easy module coupling and decoupling. The dependency between sub-components needs to be clearly captured as well.
- Standardized issue description:
When the testing is done by different validation teams, it is difficult to ensure that issues are described in the same way. It is suggested to create a template to standardize the issue description to make search easier.
- Scalable and flexible criteria:
The search algorithm and the fixes recommendation decision tree criteria should be designed in a scalable and flexible block so the algorithm and criteria can be easily swapped in and out based on the project need.
- Sufficient training data:
Make sure the data is meaningful and enough to be analyzed. When training the system, we need lots of data, and long existing software projects work better. Once trained, it can be used for new data analysis. Currently in POC, DSF works better for long existing software projects compared to new software projects - as lack of historical data for training the system. Measures of the successful rate of correct prediction and improvement need to be done from time to time.

5 Conclusions

This paper explained DSF software bugs segregation system detail end-to-end flow with a deep dive example:

- Starting from what data need to be collected
- Then analyze the data by using one of the algorithms, binary search
- And finally report with fixes recommendations

With this implementation, DSF software bugs segregation system shows productivity improvement by demonstrating quick turnaround time to identify the issue owner and issue root cause reports within a complicated software. Lastly, the learning and challenges are also captured for future improvement.

References

- Capers Jones, "Applied Software Measurement: Global Analysis of Productivity and Quality." Third Edition
- Implementing a Decision Tree from scratch using C++
<https://towardsdatascience.com/implementing-a-decision-tree-from-scratch-using-c-57be8377156c>
- Decision tree learning https://en.wikipedia.org/wiki/Decision_tree_learning
- Binary search algorithm https://en.wikipedia.org/wiki/Binary_search_algorithm

Software Continuous Integration with Hardware

Brent Clausner
 beclausner@sei.cmu.edu

Abstract

Validation and maintaining a stable code base are vital for software to be regarded as a quality product. An open question is how to ensure that the software you are developing will not be broken by another team within your organization making their own bug fixes? With limited hardware and growing business, how do we ensure the product works as expected with thousands of features being developed or already existing while new development is being done? Keeping costs down and software quality up it is necessary to test frequently and often. The overwhelming parallel development of new code along with feature enhancements, bug fixes and other modifications to existing code impedes the task of ensuring a product will function as expected. Business models may not fully support the hardware needs of teams to test a codebase appropriately to sustain high software quality. One good solution is to test periodically, use emulation, and seek out offending “bad” code.

Biography

Brent Clausner is a DevOps Engineer who has been working in the Software Development Life Cycle for 16 years. Within this time, he has worked as a System Administrator, Software Engineer, Software Engineer in Test, Tools Developer, and DevOps Engineer.

Brent is currently a DevOps Engineer at The Software Engineering Institute, Carnegie Mellon University. Here he has worked with new languages to develop prototype applications, web services, and pipelines to secure code bases for various languages. Previously he has developed detailed functional test plans, wrote code to split a program into multiple processes for producing multiple reports, developed tools for migrating data between systems, and setup multiple development pipelines.

Brent currently resides in Trafford, Pennsylvania along with his wife and child. Brent studied computer programming at Pittsburgh Technical Institute and computer science at Point Park University. He is an avid survival crafting, video game player.

1. Introduction

Executing tests on a regular basis and doing so upon every code submission is a very important part of ensuring an application works as expected. Doing this with software that is tied to a specific hardware can be difficult depending on the size of the workforce for the product. If 100 submissions occur in a short span of time, to keep moving quickly would require either a large amount of equipment or can become a long sequential queue for executing testing. Executing continuous integration in this fashion can miss some cross functional incompatibilities.

Having a lot of equipment is not cost effective. As the product changes new hardware will be required and updating the testing equipment can mean downtime for testing. At this time if code changes are allowed to go in, untested, it can result in code dependencies on “bad” code making the removal of those changes, difficult if not impossible.

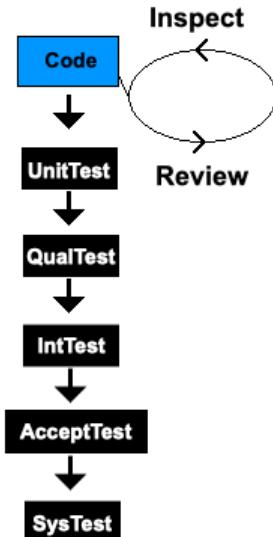
Many functional teams working on an application may have dependencies on each other that would not show any problems with alterations to feature specific functionality until later in the release development process. Because of this, it is important to execute tests including all current code changes at once. Also targeting a specific feature area in testing may work most of the time, however dependent functional areas should also be tested. In this document I'll go over some processes and tools that I've experienced use of and will share some of the benefits and pitfalls of doing things in this way.

2. Current DevOps Integration Testing

Developing an application that serves a single purpose makes for easier design and focus. Exposing serviceable endpoints, via REST for example, need to be tested for functional completeness and needs to be robust. A single endpoint can touch multiple functional areas. Authentication should be implemented and used by every endpoint. The endpoint itself then may interact with another area, you can think of a creation of an object that gets stored in a database or memory. A few different feature areas will then have been touched, authentication and database.

Most software is designed to work with a specific hardware architecture. This makes development easier as you can implement a DevOps pipeline that covers what the application is expected to do. Cross platform building is also possible and can add a layer of complexity but generally can be virtualized to allow parallel building and testing.

After code is developed, it needs to build. This is the very first step towards quality. Within this phase unit testing can be done within the same language being used. Most languages have testing frameworks that allow this to take place. When an engineer looks to have code submitted, the normal phase of code review and inspection can help to alleviate issues with normal flow and typical pitfalls. Within configuration management, pipelines can be used to automatically run tests to ensure that code meets standards necessary put forth by the organization. This is a location which can include things like static code analysis or enforcement of coding standards can take place.



Development Stages. [1]

When coding is typically being done, the normal phased approach for this allows us to have our code inspected and reviewed by peers if the team size allows it. This is then followed by unit, quality, integration, acceptance, and system testing. Unit tests that are written either beforehand or during the coding phase to lock in codes functionality. Quality testing is done to ensure that the code is functioning as expected. Then testing is normally accomplished by other teams where it is worked for integration with the rest of the code base. Acceptance testing allows it to be used by other teams. Finally, system level testing can stress the test with more complex scenarios. [1]

3. Software for Specific Hardware

Developing software for a specific piece of custom hardware does have its own challenges. There are multiple layers to what is needed to test a version of software. The software needs loaded unto the system. Configurations will need to be applied after it's initialized. Testing can then begin. At this point, several different functional areas have already been touched. Each of those need to work for testing of different feature areas to begin.

Getting to a point where the system is ready for testing, can take a long time. The more complex the hardware is the longer it can take and more opportunities for things to go wrong. Using a product like rconsole can allow connections to take place without manually being at the keyboard. This is good for being able to access things like the BIOS. If you must issue any special commands to have the system load in a debug mode can be done this way too. Most automation looks for output to then proceed to the next step. Most times, initialization can have issues with timing. Order of processes coming up need to make sure that there are no race conditions that would cause any failures.

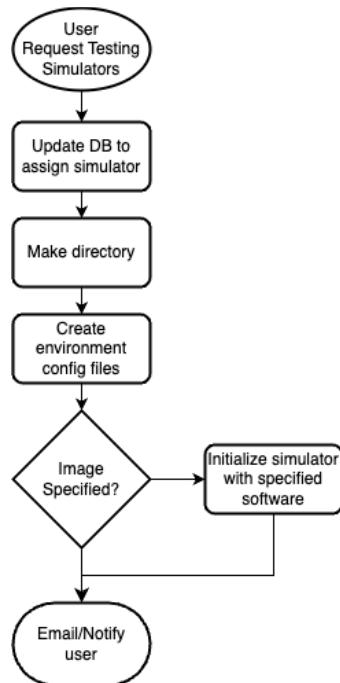
Errors with the hardware may be displayed to the console only and need to be tracked as well as any standard output. Logging the information as you execute automation is very important to be able to determine what occurred with a given system. This also can interrupt automation as the output may look garbled up with characters overwriting each other. Special care is needed when writing automation that handles checking the output with a console connection.

Timing of commands and the results from those may not be instantly complete. Most automation works such that when the output is seen, it immediately executes a command. A good example of this would be the “login:” prompt. When you see that, the system is ready for a username to be inputted. This can be used for ensuring that the system has loaded a specific module necessary for testing. If you work with an asynchronous system and you issue a command that makes a call, your result of the command may not be seen as fast as the automation can check. A lot of time this means it’s necessary to loop over a command to verify that what you did, resulted in the expected output. As an example, if your system takes a backup of the configuration, this job typically happens in the background. The job can include taking a snapshot of an internal database, then compressing it, and finally updating the internal database to indicate the status of the backup for the user interface. This can require any test that issues the creation of the backup to loop over a command to see the created objects, that waits for the desired object to show up. The system should still be available when this job is running. Automation should not wait forever in this case and a reasonable time should be listed in the design documentation for the maximum. This factor plays into the usability of a system that you want it to be responsive. New work may cause additional delays in commands from running and if automation is relaxed to much that when you go to manually use it, it feels that it has too much lag in responding to your inputs.

Integrating hardware into a git pipeline directly can be troubling with this type of product. Typically, it requires having additional server processes running. Having another process required to execute commands on a system can lead to performance degradation and issues like running out of file descriptors. With specific hardware it’s important to keep the product as close to a deliverable as possible. If there is another machine in between the product and the test execution gives another point for failure. The more that we add in between the higher the risk for failure outside of the product and relates to the infrastructure.

4. Emulation is Important

While executing tests on the actual product, having an emulator drastically cuts down on the cost and time to test. Having a product that can be loaded as a Virtual Machine makes it easy to load up a system on the fly. This can cut down time to test at any phase.

*Sample testbed creation flow.*

Emulated hardware allows you to have a farm of bare metal machines that can be used by any team to execute automation or manual testing. When you combine that with continuous integration testing, it can be very powerful. This lets you submit code in and execute tests from any team. Having the proper code coverage is necessary to ensure that every feature area is protected. Implementing a common flow like the figure above, you can automate these to look for available systems and submit test jobs periodically. With the setup being a directory for housing the environment settings it allows users or automation to run from a specific location with everything setup to execute tests or see the connection information to allow manual or hybrid manual automation tests.

A common template for setting the emulator makes developing testing suites for this environment quick as well. Common methods can also be analyzed for speeding up the timing on getting it ready. Shared templates allow that to be done once and every testing suite benefits. As an example, during initialization it is good to look for multiple asynchronous commands that are used to be called one after another and letting the simulated system handle multiple processes at once. Have the automation check at the end for all created objects and command completion rather than issuing a single command and wait for completion.

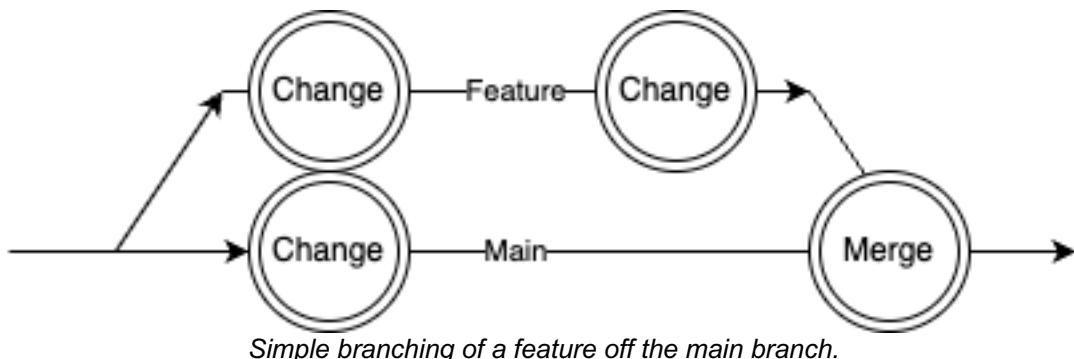
Having an infrastructure setup like this allows the number of tests that the system can handle to be a lot larger than just a single machine. With a test framework that allows you to split your process into many threads can help to speed up testing a bigger set of tests. Sub-processing out the testing portions of the automation, you can execute tests that do not touch the same functional areas within the software to allow more to go on at once safely. Negative tests that crash any processes mostly should be avoided with shared resource testing, as it can negatively impact other functional areas. When the automation test run is complete, it is important to update the database to allow the emulator or hardware to be provisioned again. This allows for multiple teams to add to the testing automation a lot easier. Scaling up the number of bare metal machines and emulators is very easy with a setup like this. It also does not limit you on only using emulators, but you can add actual hardware to the infrastructure if there are specific things that can only be validated there.

Generally having a bare metal machine like an ESX server offers a good solution in this space. Having a machine that hosts multiple virtual machines like this can offer benefits for hosting as it cuts out unnecessary resources. This eliminates running the operating system under the virtual machine. [2]

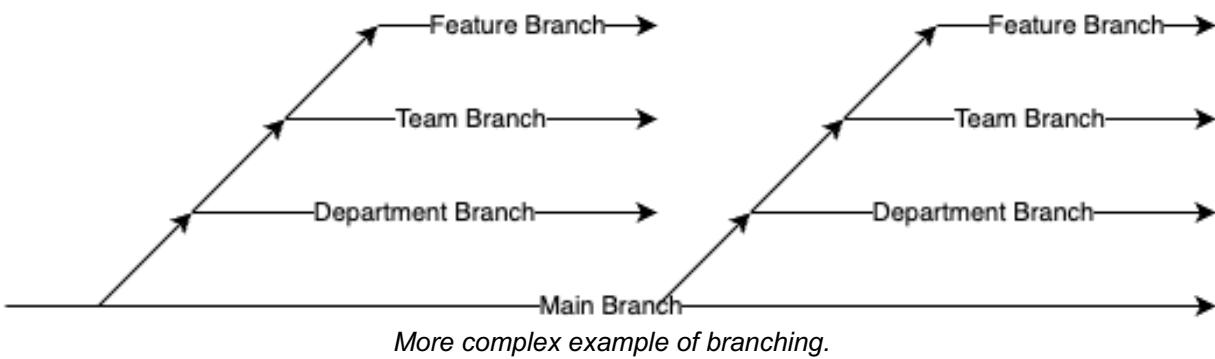
5. Single Main Branch for Developing

For a small software project, with a single main branch and developing on a branch from that main is a good way to keep the code clean. Having product release branches that are versioned are necessary for supporting a product for the long term. If the configuration management system is setup in a way that each functional area has its own branch, this can lead to a huge delay in changes being propagated throughout the code tree.

Looking at the hardware initialization phase, if a change has been committed that breaks the flow for users having things configured in a normal manner, no testing of other functional areas can take place. Using a single main branch for code submissions can have a large impact to several teams in this case. Having multiple branches not shared between teams can cause long delays in seeing integration problems. With specific tests that look only at the functional area may pass or have specific requirements that other areas do not typically use.



Branching from main, while working on a feature area may take time and can also require multiple branch synchronization. This can cause issues for a team working on the same areas of code, so it is a better practice to submit smaller incremental code and have it merge in. The code can have longer soak time and will be touched by multiple users.



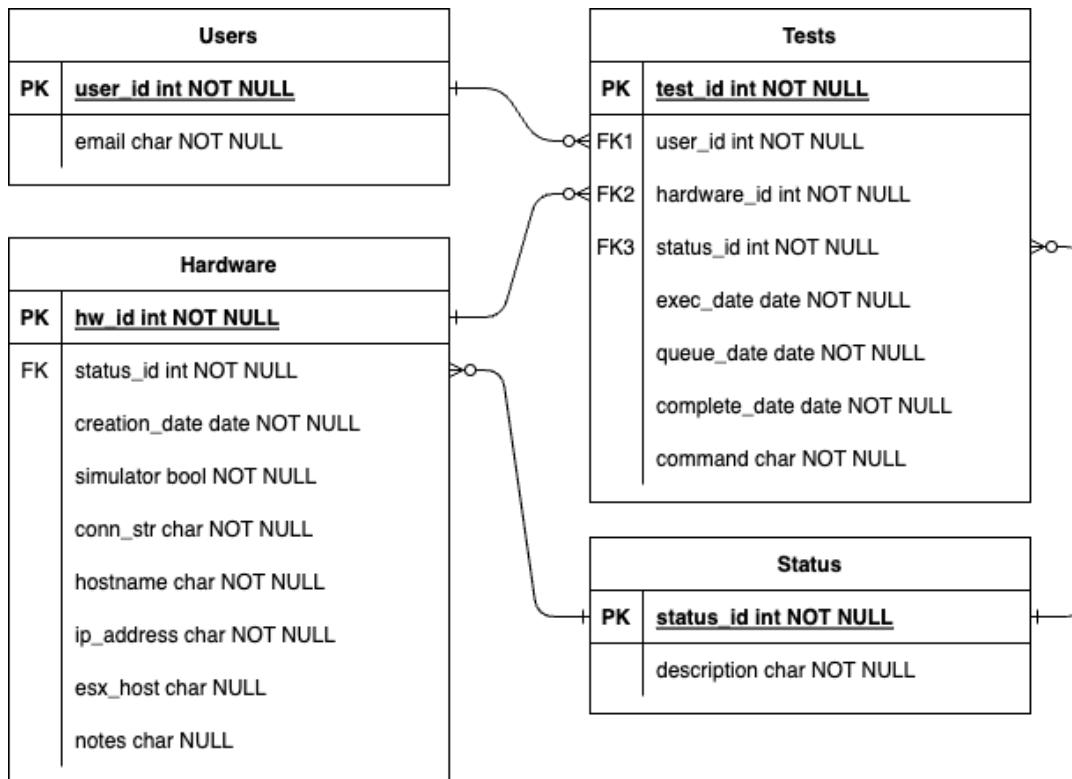
With a more complex layout of branches as an example looking at the figure above, at each merge between branches testing would be necessary to take place to ensure that each level of automation passes. That will take a long time for it to go from one branch to another, I have seen it take over the course of months to propagate changes in between feature branches like this. Cherry picking changes

between branches can result in incomplete code as a feature may rely on other code that would be needed. This can be an option, however if the code changes drastically in between picking and developing the feature work it can be problematic with syncing and merging. This way didn't work well from my experience.

If we have a hundred teams working off the main line at any given time, there can be a lot of changes being made. Each submission if it has testing being done upon submission will flood the integration testing infrastructure can result in major delays. Using a scheduled integration testing system can combine all the active changes at once for testing.

When you test periodically and not on every submission, you can get a better picture of quality to the product if the testing being done covers your feature area. You will get failures, but if you have an emulator in use this can allow you to sort out failures. If you have a simple numeric based commit system, any number of sorting algorithms can be used. This lets you try different ones to see what works best for you.

Doing continuous integration testing like this requires multiple tools to get things to work well. For instance, using git normally will fire things off immediately and if you want to schedule you need another system. Something like Jenkins will allow you to schedule things as a cron. Periodically testing like this would need to access hardware and its handling of the hardware with software running on it as well, so it suffers from similar issues as a gitlab-runner. That requires additional software to be running on a system to execute properly [3]. Any additional process resources may impact how testing is done.



I haven't found any good product that handles hardware management that doesn't require a database to be setup and managed by hand or written proprietary software. In the above diagram you can see an

example of how a database scheme can be setup to allow automated testing or manually allocated emulator or hardware. The last time that I've looked it was the best practice to have the database manage the hardware with a state machine that can manage if the hardware is in use or not. There are various states that need to be taken into consideration like this such as online, offline, in-use, and in-error. Managing emulators in this way also allows you to store proper connection information with it. This still requires an interface that allows the database to be updated for requesting the simulators and it can be expanded to allow reservation of hardware or for that to happen when testing fails.

Integration testing like this need to still run every test suite to validate all changes. When the offending change has been found, it can be rejected from the system to keep the main branch clean. This is why it is so important for a feature developing team to write good tests to ensure their area will work as designed.

A benefit of doing things this way is that you can write code and submit it after doing your due diligence of validating the functional area the code was written for, you do not need to worry about breaking other teams. This is because the system does the testing for you, and you need only worry about the areas that you are aware of affecting. When it comes to areas that have multiple dependencies on, it can be good to execute the integration tests prior to submitting the code into the main line. That should be done prior to having the code reviewed.

Having new testing suites being developed, you can include them with the normal runs and not allow them to be considered for code failures. It is important to have several passing runs along with a lot of clean intermittent runs being done prior to being allowed to reject code. This gets easier as the templates for the infrastructure and testing suites are developed.

This does not mean you cannot use something like on code submission for executing and it makes sense to have static code analyzers to scan your code for any problems. These types of tools are still a good thing to implement.

6. Drawbacks

Intermittent failures can be devastating and lead to false positives for valid code changes. Executing testing on known good changes, with a lot of the same testing going on can help to identify problems with the tests that are being executed and specific infrastructure pieces. That can be done to help make the integration testing system work well. Sometimes it is necessary to mark test suites as not reliable and to allow code to be submitted that causes failures in those cases.

Delays can occur because of a system like this. Having multiple "bad" code changes that break the same area can cause problems with finding them. It may be necessary to stop allowing code submissions in to sort out the problems. For instance, if a developer submits code that alters an internal replicated database table layout that causes a backup database object to fail creation and another developer submits code in that breaks creation of files for a backup, that testing area will take a while to find the bad changes. It may be necessary to lock the main branch to have time to find and remove all the bad code submissions. Locking the main branch and preventing submissions delays everyone from getting work done but finding the problem change(s) is more important for the sake of the product. This can be minor though, as developers use the system, they learn to execute tests for other areas that are depending on their code prior to merging.

The requirement of custom automation to accomplish this will have drawbacks of its own. The owning organization may need to set a team itself to handle common automation techniques to have the system run reliably.

Overall time to get code submitted and marked for approval can be a while. Working up to the last minute is not a good idea. With scheduling periodic runs of testing, it can make it difficult for submitting code prior to going on vacation. If the change is rejected, you may leave people depending on your code to wait on testing or integrating with it.

7. Conclusion

Using a configuration management infrastructure with a single main line and having timed periodic integration tests has led me to have a passion in software quality control. With having some awareness of pitfalls such as timing of commands and commands executing very fast, knowing ahead of time of what to do with polling for objects really helps to produce quality automation quicker. Reducing costs for businesses and using emulation for the product should be a priority for any organization that produces hardware. When it comes to working with many teams on the same product, I've seen it have many benefits with product quality. Writing test automation that protects the feature area that I am concerned about and seeing failures for code submitted from other teams when their changes break an area that gets removed is wonderful. Preventing code breaking changes from going into a product and being able to test ad-hoc has led to a faster time to test with as much code changes from an entire product line.

There is a lot of data that can also be captured using systems like this. Organizations can make their own determination of what data is valuable. It can show things like teams that prevent the most breaking code from being submitted into the main branch. Fixing intermittent problems results in a major impact as those are normally the most devastating. For me it is very satisfying to fix issues like this as they are the most difficult to find.

References

1. Tim Menzies, William Nichols, Forrest Shull, & Lucas Layman (Retrieved on 2023, May 29) Are delayed issues harder to resolve? Revisiting cost-to-fix of defects throughout the lifecycle.
Retrieved from: <https://link.springer.com/article/10.1007/s10664-016-9469-x>
2. David Davis (Retrieved on 2023, May 30) What is VMware ESXi Server and Why do I Need It?
Retrieved from: <https://www.pluralsight.com/blog/it-ops/what-is-vmware-esx-server-and-why-you-need-it>
3. Install GitLab Runner (Retrieved on 2023, May 30) Retrieved from:
<https://docs.gitlab.com/runner/install/index.html>

Legal Markings

The following markings MUST be included in work product when attached to this form and when it is published. For purposes of blind peer review, markings may be temporarily omitted to ensure anonymity of the author(s). Copyright 2023 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM23-0633

Software Cost Estimating – Friend or Foe (to Agilists)

Carol Dekkers, PMP, CFPS (Fellow), CSM, P.Eng.
President, Quality Plus Technologies, Inc.
dekkers@qualityplustech.com

Abstract

Software cost estimating – the mere mention of the term elicits a myriad of polarized opinions, and like most other topics associated with software development today, it has become a maelstrom of self-appointed experts who weigh in with commentary. As a software engineer with years of development, project management and software measurement experience, I had the good fortune to be contracted as the Lead Author for the Software Cost Estimating Body of Knowledge (CEBoK-S) from 2019-2020 and I worked with some of the worlds leading experts in software cost estimation. I anticipated the role to be challenging, however, looking back at the project, it was a “drinking from the firehose” experience that illuminated that I had no idea how involved and complex professional cost estimating really is. I didn’t know what I didn’t know.

In my role to develop CEBoK-S I was armed with experience and expertise as a world-recognized software measurement expert, and project manager. I thought I knew what software cost estimating was all about, at least the basics, and in the beginning, I held a similar opinion to that of one of the reviewers of my first drafts of this paper: “I do think Estimates are a very basic discipline that you simply can't do without.”

I no longer hold that opinion and I realize it was based on a lack of knowledge about professional software cost estimating. If you gain one message from this paper, I hope it is that “software cost estimating is a profession worthy of a seat at the management table and deserving of the same (or better) respect and status of equally experienced software development and project management professionals.” In May of 2023, ICEAA released the first version of the official CEBoK-S together with an associated Software Cost Estimating Certification (SCEC.) I am extremely proud of the product and of the role I played in its development. Software Cost Estimating is both necessary and integral at the beginning and throughout software development – not as a single pie-in-the-sky activity to be performed when management initiates a project. Software cost estimating, done right, involves skills in statistical analysis, data mining, business analysis, formal methods, identification, and quantification of areas of risk and uncertainty, developing appropriate cost estimating relationships (CERs) based on normalization and data analysis to piece together often inconsistent and incomplete historical data. To the team of professional cost estimators who patiently educated and work with me to develop CEBoK-S, I am grateful and hold the utmost regard. Hopefully after reading this paper and attending my presentation, you’ll gain an appreciation for the professional and necessary role that professional software cost estimators play in our software development industry.

Biography

Ms. Carol Dekkers is a Certified Scrum Master (CSM), a Certified Function Point Specialist (CFPS-Fellow), a Professional Engineer (P.Eng-Canada) and a Certified Software Cost Estimator (SCEC). She is also the lead author of the International Cost Estimating and Analysis Association (ICEAA)'s new Cost Estimating Body of Knowledge for Software (CEBoK-S), and a long-standing member of the U.S. delegation to the International Organization for Standardization (ISO) subcommittee for writing Software and Systems Engineering standards. Ms. Dekkers expertise spans software development, software measurement, quality engineering, project management, and software cost estimating and has shared her insights with technical and non-technical professionals worldwide through keynotes/presentations, textbooks, and articles published in industry journals. Ms. Dekkers has received numerous awards for her industry contributions and thought leadership including Computing Magazine Global Leader in Consulting – Pro Bono (2023), ICEAA Educator of the Year (2022), IFPUG Honorary Fellow (2022), Brazil and Korean metrics associations special awards (circa 2010) and was named one of the 21 New Faces of Quality for the 21st Century by the American Society for Quality, ASQ (2000.)

Copyright <Dekkers> <5-Sept-2023>

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

1. Introduction

Software projects are different from hardware or construction projects in that they are human-centric, creative, and technical endeavors that bring software ideas to life, and for which there are many uncertainties – especially early in the development process. At the time when early estimates are required, software requirements may not yet be articulated or understood. Software, while often considered an intangible product, follows a development path involving stakeholders (many), processes, rework, incomplete requirements, and other changing factors that complicate creating accurate estimates. Dr Barry Boehm, the inventor of the Constructive Cost Model (COCOMO) at the University of Southern California introduced the concept of a “cone of uncertainty” to illustrate how early software estimates can be inaccurate by up to +/-4X depending how early in the development life cycle they are created.

With agile software development, projects commence with only minimal requirements specificity, and the range of estimating uncertainty could be even greater. Software estimating approaches typically include adjustments to account for anticipated project variations, but agile software developers believe these variations are so marked that cost estimating should be abandoned altogether. Such was the impetus for the “no estimates” movement. Today, cost estimators and agilists often remain at odds. On one side, estimators are tasked to create software cost estimates from early requirements (documents lack specificity,) while on the other side, agile software developers hesitate to participate in preparing estimates that they feel will become budgets, schedules, and targets.

This paper acknowledges that agile software development introduces added uncertainties and challenges to software cost estimating and suggests recommendations to formalize the role of software cost estimators in creating more realistic estimates. More realism in cost estimates should lead to more realistic budgets and timelines for software development contracts and hopefully position all software development projects for greater success.

In 2020 when I signed the contract as Lead Author for the CEBoK-S for the International Cost Estimating and Analysis Association (ICEAA), I envisaged my task as fairly routine – I’d be starting off with an established curriculum from the 5-day US Department of Defense Acquisition University (DoD DAU) software cost estimating curriculum and “simply” remove DoD references, and update five year old content to current best practices. During the process, I worked with some of the top professional software cost estimators in the US, as well as participants from abroad. I am humbled that my original opinion of software cost estimation actually mirrors the shared opinion with one of the reviewers I do think Estimates are a very basic discipline that you simply can't do without of this paper who stated: “.” I was so wrong!

And so, it is with the majority of software-savvy developers and project managers who share my original opinion.

Whether or not you agree with cost estimates and the myriad of opinions about their relative merit or damage, or that accompany Cost estimates are a necessary first step in the acquisition process for funding large scale software development efforts, regardless of the development approach. Software is pervasive in every aspect of our lives from Smart homes with remote controlled appliances to Self-driving cars to Smart highways. Even on projects where software is not dominant (such as a toll-highway construction project,) delays in the software development can have a major impact. In addition, the volume of software “code” now involved in major programs such as the F-15 military aircraft program has increased so exponentially that software (once a minor component of the product) is now as costly and dominant as hardware and other product components.¹

In 2001, ***The Agile Manifesto***², written by seventeen prominent software developers, revolutionized the approach to developing software and presented guiding principles to both streamline software development *and* increase the value of delivered software, among others. The change to how software is developed minimize the upfront, protracted requirements definition of the earlier waterfall approach, and allows the software scope to evolve iteratively instead of monolithically. This means that software product development today does not come about using a fixed scope of software requirements, but rather through incremental “deep dives” into the software requirements as prioritized by the product owners (the business.) What has not changed, however, are the needs for advanced, high-level cost and schedule estimates for software development often prepared as much as 2-3 years in advance. As such, software estimates remain a non-negotiable requirement for agile and non-agile projects alike. This creates friction within the

¹ Software Cost Estimating Body of Knowledge (CEBoK-S) Lesson 1: Motivation for Software Cost Estimating, 2020.

² **The Agile Manifesto**, aka **The Manifesto for Agile Software Development**, Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, et al, 2001, <https://agilemanifesto.org/>

agile development community and resistance to the entire concept of estimates by agile software developers who argue that estimates of product development are not possible based on the creative nature of software development, couple with a lack of scope or detailed requirements.

At the same time, the ongoing definition of project success according to the annual Standish Group CHAOS³ report continues to be those projects delivered “on-time,” “on-budget,” and meeting the user needs (based on estimates.) The degree of success for agile software development projects has improved over the traditional waterfall (linear, fixed) approach to building software, however, despite the approach, less than 50% of projects meet the criteria to be deemed successful. There are many reasons that software projects exceed their budgets and schedules including management issues, development delays and handoffs to multiple teams, development process inconsistencies, supply chain delays, over-optimism, and others.

In the 25+ years of CHAOS reports, improvements to overcome and improve project success have focused on the software development processes and the management-related reasons for project “failure” assuming that the project budget and schedule were solid. (The reasons most frequently addressed include a lack of customer-centric processes and tools, management support, and user engagement.) The assumption that the budget and schedule were appropriate in the first place assumes that the underlying project estimates were sound from the get-go, yet in this author’s opinion, poor software cost estimates may be a fundamental cause of under-budgeting and overly optimistic schedules.

Focusing on the appropriateness of the contracted budget and schedule implies that the underlying estimates were sound, yet during this author’s research as lead author for the international software cost estimating body of knowledge highlight that overly optimistic estimates and immature estimating processes may be at least partially at fault for at least some project failures.

This paper and the accompanying presentation answer such questions as:

- Why do software projects never seem to finish on time and on budget? (And why it is not the agile developer’s fault.)
- How can developers help to create better cost estimates?
- Why do cost and schedule estimates take so long to prepare when all the data are right there in their tools?
- Why are formal cost estimating approaches beneficial in today’s development environment?

For software development to succeed in meeting project goals of on-time and on-budget delivery, the software cost estimates on which they were based should be reliable, realistic, and data-founded. An exploration of how to professionalize and formalize software cost estimating follows, with simple steps for agile software development teams to productively engage with cost estimating teams. The time is now for software developers and customers to embrace software cost estimating as a critically important profession and engage to guarantee project success for the future.

2. Background

Software development is young compared to other product development industries such as construction. In the 1950’s software was designed to automate manual business practices and to automate the processing of large volumes of data using written computer “code” or programs, but without standardized software development life cycle processes. Over the decades, technological advancements, and the application of automation to more industries added complexity to software development and a variety of sequential, formal approaches were used.

In 2001, a group of developers who realized that software requirements often changed after the design phase, wrote the Agile Manifesto which espoused a dozen customer-centric concepts such as valuing working software over documentation, and people over processes. Thus, the Agile Software Development movement was born. Today, software development is a multi-billion-dollar industry touching every aspect of human life from deep sea exploration and space travel to self-driving cars and smart highways. The software development landscape today includes both waterfall (sequential, adverse to change) and agile (evolving scope) approaches, with the majority of large scale government agency software development embracing a hybrid approach to software product development that can include Commercial Off The Shelf (COTS) software packages, custom software development, glue code (to put the pieces together), software as a service (think rented software) and configured enterprise resource planning (ERP) implementations. Regardless of the specific project, two major cost drivers often dominate software development costs: the size of

³ CHAOS Reports, Standish Group, annually 1994-2020 https://www.standishgroup.com/sample_research_files/chaos_report_1994

the software and the productivity (complexity of the software, skills of the team, and tools) to complete the project. Software and its sustainment are now a major cost center for businesses worldwide.

As a case in point, during the winter of 2022, the US-based carrier Southwest Airlines suffered major business loss due to flight disruptions caused by out-of-date software (a business decision.) The software itself was not SWA's major business, however, it supported (and then did not) their business of getting people from point a to point b during a busy holiday season. Countless other businesses are disrupted or seriously challenged by software issues. This is peripheral to the point of software cost estimates; however, it serves to demonstrate how integral software is to our daily life.

The status quo of software development

The biannual Standish Group CHAOS reports tracked IT project success, challenges, and failures by defining SUCCESS as on-time and on-budget deliveries that met customer requirements. Since the first report in the early 1990's, the success rate has fluctuated between 30 and 42%, the latter in the last year of the formal report in 2020 for agile projects. Challenged projects were those that were either over-budget or behind schedule, while outright failures included canceled projects or projects completed but over-budget, late and/or not meeting customer requirements.

Additionally, the average cost growth of software development projects hovers around 40-50% and occurs on 80% of development projects, while the average schedule delay for the same projects is between 60-80% and occurs 90% of the time, according to Dr Christian Smart in his 2021 book *Solving for Risk Management: Understanding the Critical Role of Uncertainty in Project Management*. It is plain to see how overly optimistic and unrealistic (wishful thinking) software cost and schedule estimates a major root cause of software development "failure" can be regardless of the development approach.

Why do projects never seem to finish on time or on budget (and why it is not the development team's fault)

Even though project success or failure is based on on-time and on-budget project performance, most of the attention for process improvement of challenged or failed projects was focused on process and management causes. Software process maturity, development tools and methods, and management practices are all amongst the improvement areas, however, the research assumed that the estimates were realistic and accurate in the first place. In actuality, late projects and cost overruns may be attributable to poor estimates produced by immature estimating practices, over-optimism, and a lack of or ignorance of good historical data. Added to the known "Cone of Uncertainty" where preliminary software development estimates can vary by as much as +/- 400% with actuals, it is surprising that software projects hit their targets even 30% of the time.

According to the Standish Group, unrealistic estimates resulted in \$81B USD in cancelled software projects, and \$59B USD in budget overruns (2015 Standish Group CHAOS report, one of the last publicly available reports.) There are many reasons that software projects overrun their budgets and schedules and are worthy of discussion, however, this paper focuses on issues related to improving the underlying estimates on project success or failure are based. .

The good and bad news about status quo software cost estimating

The good news if you are a member of an agile team on a project that is over-budget and/or behind schedule, if the original project estimates were unrealistic from the beginning, the fact that the project is "challenged" is not your fault.

The bad news is that until software cost estimating is embraced as a bona-fide and formal professional endeavor, and the processes to collect historical data (actuals) are improved, software development is trapped in a tornado type cycle similar to a dog chasing its tail:

1. **Few strong inputs & low estimating maturity.** Poor requirements documents and apathy (or downright disdain for estimating) on the part of the software development team are poor inputs to the estimating process. This leads to
2. **Weak, and unreliable estimates.** Immature software cost estimating practices (ad hoc, non-data-founded, non-standard processes) together with lack of good data lead to overly optimistic and unrealistic software size and effort estimates. This leads to
3. **Unrealistic plans (cost, effort, duration).** Lack of standardized estimating process (ground rules & assumptions, cross-checking, verification by the development team) leads to

4. **Impossible contracts / internal projects.** Development begins based on flawed contract values (and undocumented) assumptions and both the customer and supplier seek to make changes/clarifications to lower their risk, which leads to
5. **Out of control projects.** Both sides end up at odds (there is no win-lose in software development – only win-win or lose-lose) and
6. **Adverse project outcomes:** uncontrolled project “growth,” failed & cancelled projects, litigation, Wasted investment. Project is cancelled, restructured, descoped, or failed. Then the actuals data are not collected consistently. Few lessons learned, until...
7. **A new project emerges /or cancelled project restarts at step 1,** with minor change except for the promises/optimism to do better. The cycle continues with the same well-intentioned, but unrealistic estimates based on flawed assumptions and poorly documented software requirements.

Certainly, just improving the software cost estimating process will not guarantee project success – estimates are simply best-guesses of how the project could go if all inputs and assumptions come true. Good estimates at least position a software development effort in the running for success.

Software cost estimating is the subject of at least a dozen major US Federal Government guidebooks and courseware including works by the Government Accountability Office, GAO (2021) who produced two guides, one on cost estimating (with a section devoted to software costing) and one on agile software development (featuring sections on cost estimating), the Department of the Navy, Homeland Security, Defense Acquisition University (DAU) and others, and of commercial consulting organizations. In addition, there are several parametric cost estimating tools used successfully worldwide to estimate cost and schedules for software intensive systems development. GAO’s 2021 manual outlined a series of challenges to cost estimating and recommended some mitigating factors, including a lack of high-quality software development data.

At this point with the lack of requirements specificity (software size) and the number of uncertainties involved, especially in agile development, one might wonder why bother spending time creating an estimate (that will be wrong) in the first place? The answer is that there is often no choice about whether to estimate or not to estimate. For US Government software acquisition at least, in order to advance any sort of software development program, some form of cost and schedule estimates are a requirement. While the type of estimate and the specificity of content varies based on the type, timing and size of the project, there are few programs that do not require a software cost estimate. The same is true for commercial enterprises – software cost estimates are often needed two to three years ahead of the budget cycle in order to be included in potential funding. As such, the question is not so much about Why both estimating, but rather, how can we create credible and realistic software cost estimates that will be beneficial to the software development projects?

3. What I learned about Software Cost Estimating (that I didn’t know I didn’t know)

The following subsections comprise the author's top ten lessons learned about software cost estimating: (Note, these points are all outlined in the PowerPoint® presentation version of this paper.)

1. **Data-founded estimates** are more defensible than theory (or expert opinion.) If you don't have historical data, seek out publicly available data (such as those found in commercial estimating tools or in such databases as the International Software Benchmarking Standards Group's (ISBSG) development and enhancement repository), use a Wide Band Delphi expert opinion approach (whereby subject matter experience/expertise are relied on in lieu of data), remember the cone of uncertainty and check for published industry rules of thumb. A defensible estimate is like a “line in the sand” and provides a backdrop on which the estimate has merit. In Agile development, opt for T&M (time and materials) contracts with flexibility for change. (Refer to the *Acquisition & Management Concerns for Agile Use in Government Series: Estimating in Agile Acquisition by the Software Engineering Institute.*)
2. There is a published **Cost Estimating Maturity Model** illustrating that Software Cost Estimating is a professional endeavor (and is non-trivial.) Most organizations (worldwide) start out at Level 1 and do not have a formal approach to software cost estimating.
3. **Estimation scope** is critical to the context of the estimate
4. There is **no “One Size Fits All”** software cost estimating approach. The estimating approach depends on the purpose, timing, and type of estimate as well as available data. Estimating involves more than estimating a size and then multiplying by a magic or historical “productivity” rate.

5. **Quantifying Software Size** is fundamental to a good parametric Cost Estimating Relationship (CER). This may be argued by supporters of the bottom-up-WBS task oriented estimating approaches. Not to disregard the value of task or activity-based estimating, but size is a known cost driver. Consider the construction of a building – the cost per square foot should be a consideration at least as much as a general contractor's estimate to frame a building based on how quickly crews can work. Cross-checks with multiple estimating methods improves the estimate.
6. **Software size can be estimated using standard units of measure (UOM)** and based on ConOps/EPICS/prelim backlog or other early requirement documents
7. **Software development cost is non-linear (S-curve) and subject to Diseconomies of Scale (EXP > 1)** Note: This is different from hardware estimating and other product development where volume of widgets comes into play. Analogous (linear) estimates are possible only within a given range of application size.
8. The software development approach (waterfall or agile) brings in **different cost considerations**, as well as whether the development is done in-house or by third-party vendor(s).
9. Most software development is a mix of **hybrid solutions, necessitating multiple estimates**. The type of development, availability of technical baseline data, historical data, level of quality required in delivered software, etc. should all be considered when preparing and presenting the estimate. Beyond software development, consider the sustainment, integration, deployment, and other factors to create and provide the context for a realistic estimate. Hybrid software solutions often necessitate multiple software cost estimates.
10. The **software development versus procurement continuum is often overlooked** and can impact the development of a realistic estimate. Additionally, the level of leadership required can impact costs (bigger programs need more leadership - e.g., running a large software development program that involves systems of systems is impossible without a strong leadership team to guarantee the delivery of quality software.)

4. Ideas and next steps (based on today's environment)

As an agile tester or agile developer, collaboration with the software cost estimating team is needed to ensure that the software estimates (4 parts) are reasonable, realistic, inclusive, and presented in context. This must be a joint effort that starts with the project initiation documents (such as a Concept of Operations: ConOps) through to estimating through to development release strategies, contracting, development and change management, and culmination in the software release(s.) Data collection and maintenance of a repository are also collaborative efforts where the quality team and project management organization (PMO) should play a lead role.

4.1 Consider the benefits of better software cost estimates on your project success and how to build the competency in your company

There is a certification: SCEC, Software Cost Estimating Certification, and a complete set of formal documentation for the ICEAA Cost Estimating Body of Knowledge for Software (CEBoK-S.) The documentation covers the aforementioned lessons learned and also the following major concepts:

- Types of software cost estimates and considerations (Lifecycle cost estimate, ROM, software development estimate, software sustainment, risk factors, etc.)
- How to select the best approach for doing a software cost estimate
- Important concepts: estimating maturity model, size, productivity, ground rules & assumptions. Data normalization and analysis
- Steps to preparing a good estimate
- Cross-checking, context, presentation to management
- Ten modules of materials plus backup slides (over eight hundred slides)

5. How can Cost Estimators and Agile Software Teams work together to achieve success

Given that software cost estimates remain a requirement for acquisition and software development funding, it is critical that the best possible, data-founded, and realistic software estimates are developed.

While it might seem that a software cost estimate would be a range of numerical values representing the estimated software development size, effort, cost, and schedule (duration), a good software cost estimate should consist of much more.

As a guideline, a software cost estimate should contain the following information:

1. **Contextual information** about the software project/release/phases included and:

- Program/project/release identification (name);
- Description of same
- Type(s) of cost estimate(s) and scope of each;
- Source documents and subject matter experts included in the formulation of the estimate;
- Basis of estimate;
- Ground rules & assumptions;
- Technical baseline;
- Scope of the estimate (e.g., software development, procurement, sustainment, maintenance, and any other inclusions). Explicitly state exclusions from the estimate.

2. **Software Size estimate:**

- Size(s), unit(s) of measure and sizing method used (e.g., International Function Point Users Group Standard or Simple Function Points or other);
- Growth factor(s) and/or adjustments applied;
- Source document(s);
- List of included software requirements;
- Templates or shortcuts used.

3. **Software Effort, Cost and Schedule estimates:**

- Estimating approach(es) (e.g., analogy, published parametric equation, derived CER);
- Assumptions not covered in #1 above
- Effort estimate and unit(s) of measure (person hours);
- Cost Estimating Relationships (CERs) /Schedule Estimating Relationship s (SERs) used and range of applicability
- Productivity assumptions
- Historical data used (as applicable)
- Cross-checks and assumptions
- Growth factor(s), risk, and uncertainty considerations (and confidence levels)

6. Conclusions

As outlined in this article, software cost estimates prepared without care or knowledge can contribute to underbudgeting of software development projects, thereby reducing a project's chances of success. While estimating software costs will always involve subjectivity and assumptions about the development which may or may not occur, an estimate based on formal estimating processes, normalized historical data, and participation by subject matter experts is a first step to creating a project environment where success is possible. While better software cost and schedule estimates will not guarantee success, better estimating processes coupled with better data and collaboration between developer and estimators will help. In addition, a recognition that software cost estimation is a professional endeavor to be practiced, formalized, and standardized can provide projects with contracting environments and funding to increase successful on-time and on-budget software deliveries. While estimates are the best guess of what the project should cost and how much effort it should take given ideal circumstances, they are still estimates and should be taken as such. There are no guarantees of actual project performance, especially when historical data prove otherwise. Estimates based on realistic historical data and data-based cost estimating relationships (CERs) give projects a baseline on which to base changes, and a fighting chance for success.

References

CHAOS Reports, Standish Group, various years biannually 1994-2020
https://www.standishgroup.com/sample_research_files/chaos_report_1994

Cost Estimating Body of Knowledge for Software (CEBoK-S), International Cost Estimating and Analysis Association: ICEAA, Dekkers, Carol- Leader Author, 2023

Solving for Risk Management: Understanding the Critical Role of Uncertainty in Project Management, Smart, Dr. Christian, 2021

Software Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement, Bundeschuh, Manfred and Carol Dekkers, Springer Publishing, Germany, 2008

The Agile Manifesto, aka **The Manifesto for Agile Software Development**, Beck, Kent, Mike Beedle, Aire van Bennekum, Alistair Cockburn, et al, 2001, <https://agilemanifesto.org/>

Websites:

- International Cost Estimating and Analysis Association (ICEAA) www.iceaa-online.com
- International Function Point Users Group (IFPUG) www.ifpug.org
- Quality Plus Technologies, Inc. www.qualityplustech.com

Testing Warehouse Automation with Autonomous Mobile Robots

Mesut Durukal

durukalmesut@gmail.com

● Abstract

Warehouse automation is an evolving market with bleeding-edge technology. Autonomous mobile robots assist human operators in picking items from the shelves in warehouses to increase efficiency in these systems. In this way, human operators make a shorter distance and don't have to carry heavy boxes.

Testing such a system has various challenges. Firstly, ensuring that the most fundamental functionality is one of the biggest difficulties. The expected result of the algorithm in each picking sequence is the calculated path that has to be followed. This is a minimum distance problem [1]. Testing this and all the other functionalities with physical robots is not easy when the environmental conditions are considered.

Quality is not only functionality. We can consider several aspects such as operability, maintainability, recoverability, performance, usability, and efficiency. These are important for all products, but especially for the Warehouse Automation systems, even more, since the main idea is increasing productivity. Recoverability is also very significant since all the operations may be stuck in case of a fatal error.

Finally, as far as test automation is taken into consideration, there are several challenges coming into place. Along with the hardware testing, every customer has a different configuration. Various options like the number of robots moving around, the number of items in the warehouse, the map of the area, and the priority of the orders should be tested. The reusability of tests is important in terms of duplication and maintenance.

These challenges can be coped with by applying several solutions such as:

- Performing longevity and endurance tests along with benchmarks
- Test Simulation
- Evidence Collection and ROS Bag
- Improving Code Quality in the test automation framework

By improving the testing approach and minimizing drawbacks, a system of Warehouse Automation and Autonomous Mobile Robots development can be verified and validated. By applying the proposed solutions, quality issues are reduced, and customer satisfaction is ensured.

● Biography

Mesut Durukal is a Quality Assurance and test automation enthusiast with experience in several domains. Along with having proficiency in CMMI and experience in Agile practices under his belt, he has taken various roles like Quality Owner, Hiring Manager and Chapter Lead in the organization, leading multiple QA squads in multinational projects.

He has expertise in test automation and integration to CI/CD platforms supporting continuous testing with logging, reporting and root cause analysis packages from scratch. Besides, he has been facilitating test processes and building test lifecycles in the projects.

1 Introduction

Warehouse Automation and Autonomous Mobile Robots (AMR) development is a hot topic. Nowadays, in an online shopping and delivery world, the biggest companies are investing in developing these systems. Companies doing online delivery to either external individual or corporate customers or internal sites, have warehouses to keep their items. These items are categorized on the shelves on the racks in warehouses. Racks are aligned on the isles in a warehouse and items on the shelves have unique barcode numbers. Deliveries are managed by orders. Whenever a delivery is to be done, an order is created with the list of items and products to be packed and sent. Operators collect those items from the relevant locations and leave them at the delivery points. After an order is fulfilled, the next is accepted and processed.

Warehouses are of different sizes and accordingly working operators and shift times are decided by the warehouse operation managers. Optimizing costs and allocated resources are some of the key success metrics. In terms of the operation performance, the completed number of orders and the average order completion pace are other metrics to track. The order of picking is very important since the visiting points will constitute the travel path. Selecting the path, which will result in minimum walking distance, is the main problem to be solved. Figure 1 illustrates how picking order affects the walking distance. Besides, since some of the orders are more urgent than others, order priorities should be managed properly.

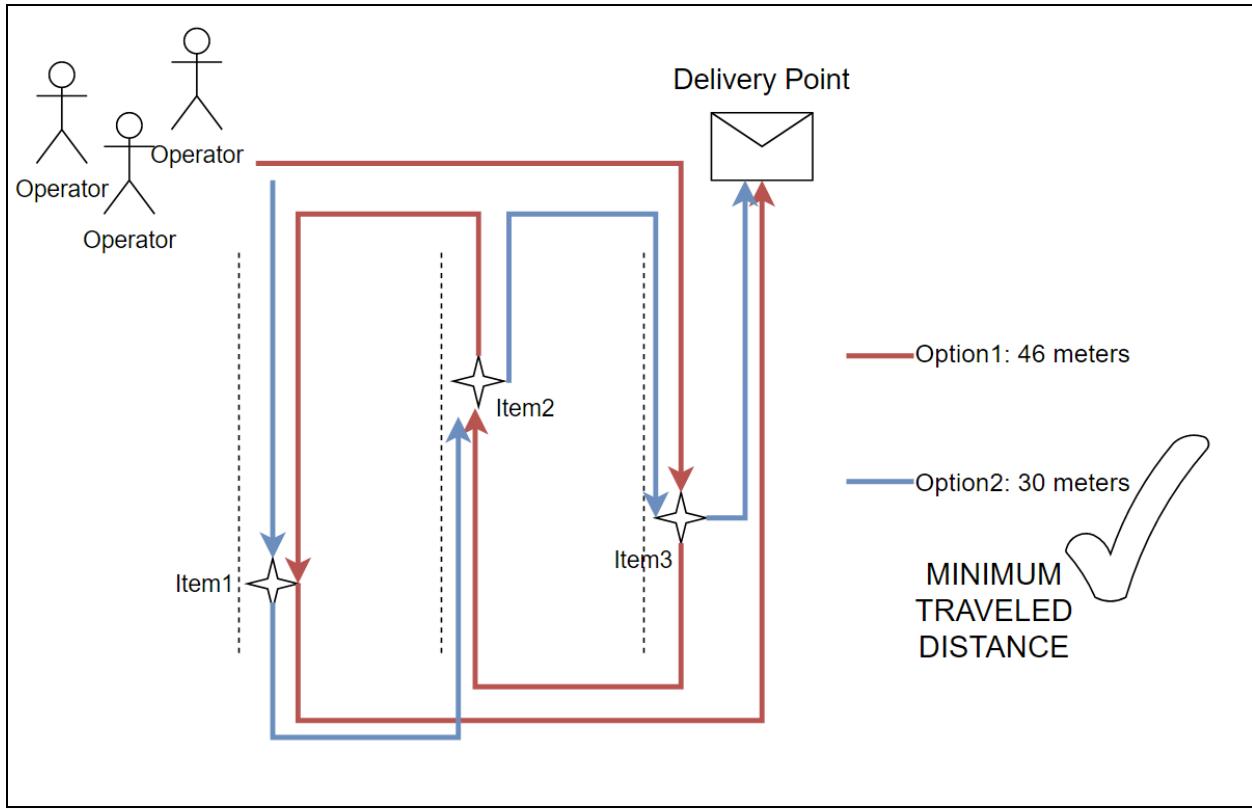


Figure 1: Effect of order of items picking in walking distance.

This is a very basic illustration, shown in Figure 1, where there are only 3 items and 1 operator picking them. But in real-life cases, there are more complex situations with more than 1 operator. Even in the illustrated case, it can be seen that picking items with the order (3,2,1) and the order (1,2,3) results in almost a 50% efficiency difference.

Warehouse automation systems are developed to cope with these challenges. Autonomous robots are deployed in the warehouse to assist the operators. In this way, orders are managed properly with the correct priority. To fulfill each order, the minimum distance is calculated and operators are notified about the next location to proceed. Apart from these benefits, after picking the items, operators have the chance to place them in the trays on the robots to avoid carrying heavy products.

Testing such a system has various challenges. Firstly, ensuring that the proposed travel path by the system is the minimum possible alternative. Additionally, hardware components like barcode scanners, cameras, radar, and all the others introduce more interfaces. To be able to automate the test executions, several simulation environments should be prepared. Finally, developing reusable test cases is needed for managing several warehouses. Each warehouse has a different configuration such as the number of robots moving around, the number of items in the warehouse, the map of the area, and the priority of the orders.

1.1 System Under Test

In this section, the components of the system are described in detail to give a better understanding.

1.1.1 Autonomous Mobile Robots (AMR)

Autonomous Mobile Robots are installed in warehouses, and they move around to assist human operators in picking operations. In the warehouses, there are racks placed in numbered isles and on the racks, there are items to be packed for the deliveries. Whenever there are orders to be processed, human operators go to the relevant locations to pick up the items.

1.1.2 Robot UI

On robots, there is a touchscreen. On the screen, the instructions including the order progress, next picking location, and other operational information are shown. But users are not only notified but also interact with the system using the touchscreen. By these means, they click the buttons on the screen to acknowledge that the current picking is completed, and they are ready to go to the next item. So, basically, it is a web application running in the robotics operating system in full-screen mode. The general overview of a robot with trays, a base, and a screen can be seen in Figure 2.



Figure 2: An AMR [2].

1.1.3 Inventory Management System (IMS)

Items are managed by the Inventory Managed System (IMS). All the details including dimensions (length, height, width, weight), barcode number, name, and location in the warehouse are found in each entity. IMS has an API to respond to the queries made by Central PC and the other components.

1.1.4 Central PC

In this system, a main application is controlling the whole operation. The PC where this software is installed is called the central PC. Whenever an order is created, the following steps are performed by central PC to make decisions:

- Items to be picked within the order are listed
- Location of each item is defined
- Dimensions of items are defined
- Based on dimensions and the tray sizes on the robots, needed tray number is calculated
- According to the calculated tray number, the number of needed robots is defined
- Current number of available and busy robots are checked
- Depending on all the collected information, robot allocation is calculated
- All individual tasks are sent to the relevant robot.

There is a web app, which runs on the central PC and has a UI, that shows order statuses, robot tasks, and some other graphs to the operation managers.

1.2 Testing Strategy

Several customers using this system have different warehouses. The challenge in this regard is, verifying the functionalities of each warehouse configuration. Their size, racks, map, items, and operators are customized. The following test activities are performed when the product is developed for any of those warehouses:

- Unit tests
- Map of the warehouse is generated and simulation is executed
- System tests are executed with simulation
- Tests with real robots in the test area
- Tests with real robots in the warehouse
- Acceptance tests

For managing different levels of testing, separate frameworks are used. Where gtest and C++ based code is developed for unit tests, Python and E2E frameworks like Selenium are used for system-level tests.

Some warehouses are far away from the development offices. Going to the customer environment before seeing the developed software with real robots in in-house environments would be risky. By performing tests in the designated areas inside the company first, trivial bugs are found in first place before going to real execution environments.

Figure 3 shows the test automation framework. System-level tests are implemented with Selenium [3] and Requests [4] library. The API and UI interactions are automated in this framework.

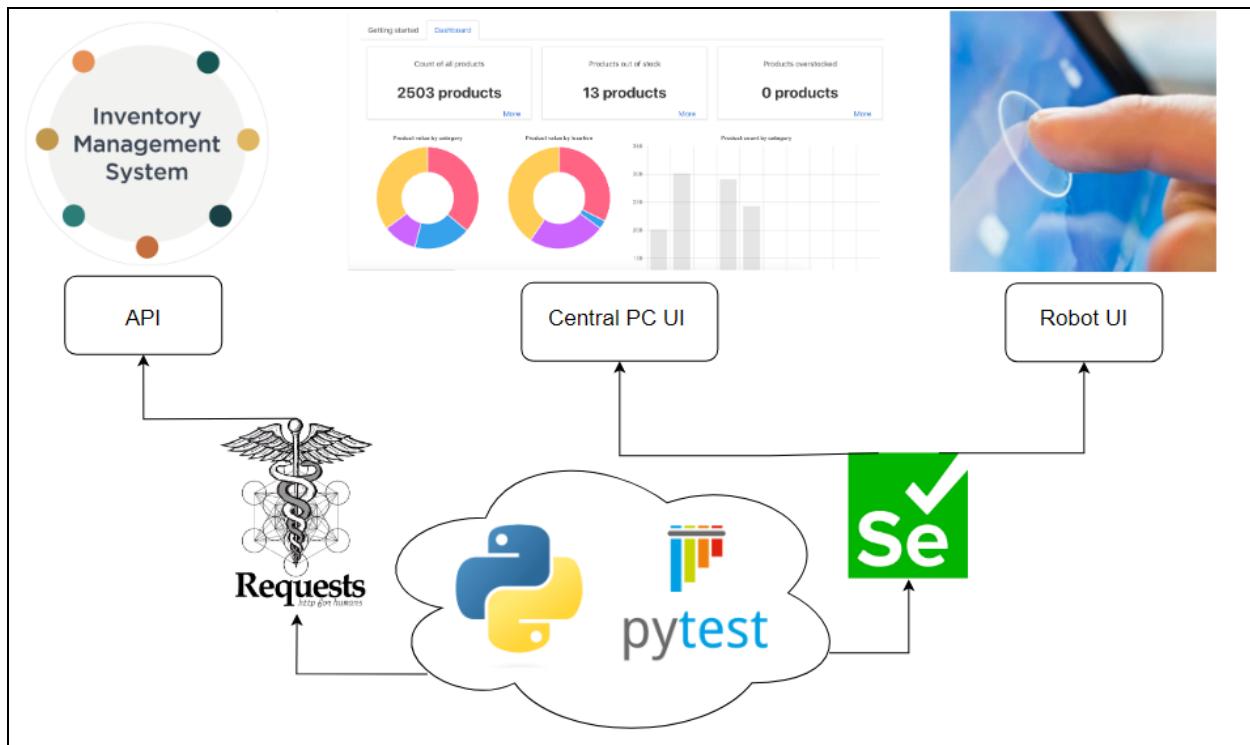


Figure 3: Test automation environment.

2 Testing Challenges and Solutions

2.1 Ensuring Efficiency

The main purpose of the system is to reduce the traveling distance. But this is a complex and complicated problem that can be solved with mathematical optimization. With functional testing, it is not easy to verify that operators are traveling the minimum distance.

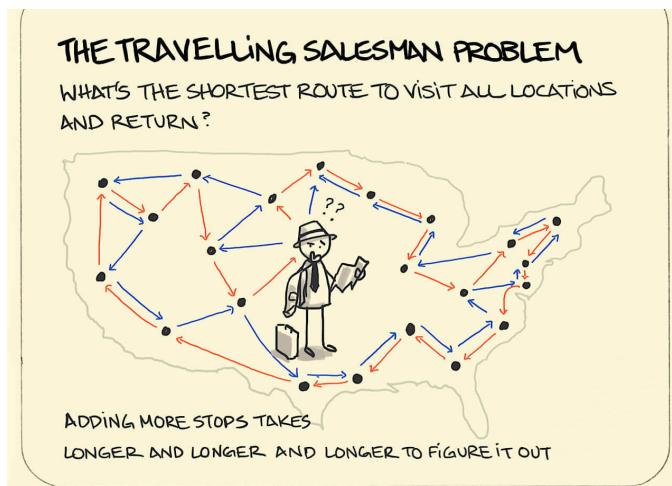


Figure 4: The Traveling Salesman Problem [5]

To prove the benefit of the system, what we do is the benchmarks. We calculate key metrics after a longevity test on both an operation with only human operators without robots and an operation with robots. The performance metrics that we collect are:

- Order Completion Time
- Walking Distance
- Human Operator Needed

At the end of the operations, demonstrating that they are completed in a smaller time with robots is already convincing for the customers. Running similar comparisons with lots of versions paves the way for finding the best-performing algorithm.

2.2 Hardware Components

Since there are hardware modules in the system, test automation is not straightforward. The sensors, camera, or other hardware modules like the brake and emergency button should be integrated into the testing environment in order to simulate the scenarios. Manually triggered operations like brake, or camera detecting an obstacle in front of the robot can be simulated by generated signals. They can be mocked in the simulation, which is developed on top of the Robot Operating System (ROS) [6].

In real life, the robot is developed as a network composed of nodes communicating with each other through ROS. Sensors and processors are the nodes on the robot, forming a network. All nodes subscribe to the messaging channel to receive generated messages by other nodes. Publishing a message in the relevant messaging channels by mocking the hardware modules enables automating the relevant test cases.

2.3 Non-functional quality aspects

Not only functionality but also several other aspects of quality contribute to customer satisfaction. One of the most important non-functional aspects is usability. Especially in this system, the ease of use is crucial to assist the operation. If the operators struggle to use the system, they lose time to complete the interactions, and the waste of time will increase.

For the sake of efficiency, usability should be ensured. One thing which can be done to collect feedback about the usability of the system and figure out its weaknesses is customer surveys. After revealing what the customers are struggling with most or what they request, they can be implemented in the next versions.

Besides, in warehouse automation systems, recoverability is very significant. In warehouses, some orders might not be completely fulfilled, which means they might just fail. For instance, if an item to be picked inside the order is missing, the order can not be completed. Due to a similar reason, if an order fails, the next orders should be able to proceed. Otherwise, the whole chain would be blocked, and all orders would be pending. The system should be designed flexibly enough to manage these kinds of failures. In an unexpected failure, the order can be marked as failed and the system proceeds with the next pending order.

The recoverability of the system can be tested by chaos testing activities. Some parts of the systems can be made down intentionally to observe the other parts. Similarly, intentional order failures are triggered to observe the system's capabilities to handle the failures.

2.4 Reproduction of Issues

In test automation, reproduction of the issues is an important challenge. Tests are executed automatically in the pipelines and the results are analyzed after the execution is done. In case of failed tests, the root cause should be understood to take action items. If they are false alarms, tests should be fixed. On the other hand, if they are real bugs, they should be fixed from the product side.

But to be able to perform the root cause analysis, the issues should be reproducible. Sometimes it is not easy, because the conditions under which the issue happened may be arbitrary. Considering the moving robots and the environmental conditions like humans around or temperature, humidity, and other factors which may affect sensors, it is highly possible to struggle to reproduce the failure.

What can be done to understand what happened during the issues is, recording all the ROS messages and playing with the ROS bag. Figure 5 shows a depiction of the ROS bag. You can replay the recorded messages and in this way better understanding of the incident can be had.

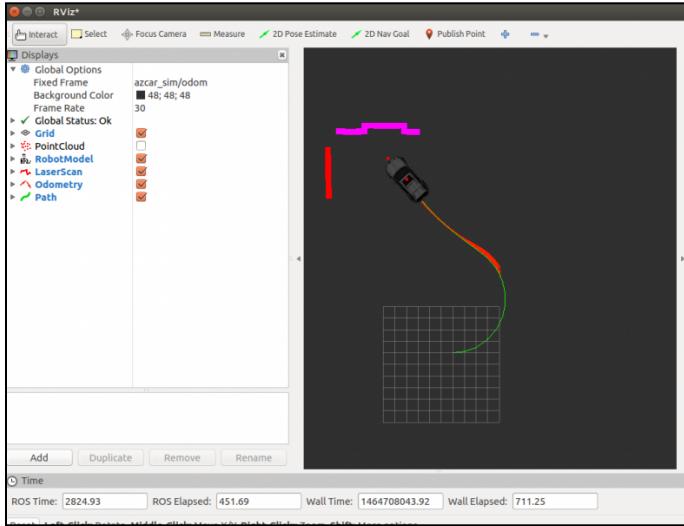


Figure 5: Rosbag [7]

2.5 Maintainability of Tests

Some warehouses are smaller, while others have very wide areas. In other words, in some warehouses, a few robots are enough for the operation but for some others, up to 30 robots are installed. Tests should be flexible and scalable enough to cover various configurations. Otherwise, each time the same scenarios should be implemented over and over again which results in duplication.

Figure 6 shows an example warehouse. To reduce duplication and improve scalability, tests are implemented in a flexible way, and environmental variables are separated from the test code. They are managed as a configuration project, providing the necessary flexibility to the tests.



Figure 6: Warehouse view [8]

Maintainability of the test automation framework is not only related to the warehouse automation systems but any test automation project. Test code should be kept up-to-date covering the behaviors, flakiness should be removed and reliability should be continuously maintained.

Code quality can be maintained via either static code analysis tools or peer review activities. Potential considerations to improve code quality:

- Test smells such as flakiness, fragile tests
- Reliability
- Robustness and stability
- Understandability
- Execution duration
- Resource consumption
- Efficiency
- Compatibility (cross-browser testing)

3 Conclusion

Warehouse automation testing is not easy and there are various challenges. However, there are ways to cope with these challenges on the other hand. All aspects of quality can not be verified by functional testing. They should be supported by various non-functional testing activities like performance, longevity, resilience and chaos testing.

The structure of the system with moving robots and other environmental factors like obstacles in the robots' paths, and human operators, cause some difficulties in testing activities. Perception modules and sensors are connecting robots with the real world.

A summary of challenges and proposed solutions is shown in Table 1.

Challenge	Proposal
Ensuring efficiency	Longevity testing
Hardware components	Simulation
Usability, Recoverability	Customer communication, Chaos testing
Reproduction of issues	Evidence collection
Maintainability of Tests	Reusable tests, Code quality

Table 1: Challenges and Solutions in WA Testing

Since the operators are the direct users of the system, their opinion and their satisfaction with the usability of the system is very important. End User surveys and warehouse visits can give a chance to collect feedback including the potential improvements to the product.

Finally, as for all the other test automation projects, code quality is one of the most impactful dimensions of the automation framework. The quality of the test code affects the reliability, robustness and accuracy of the tests. Furthermore, maintenance and analysis efforts can be reduced by improving the quality.

In short, there are several challenges to performing verification and validation of a warehouse automation system. But with the correct adaptations, it is possible to achieve quality goals.

● References

- [1] Chih-Ming Hsu, Kai-Ying Chen, Mu-Chen Chen, Batching orders in warehouses by minimizing travel distance with genetic algorithms, Computers in Industry, Volume 56, Issue 2, 2005, Pages 169-178, ISSN 0166-3615, <https://doi.org/10.1016/j.compind.2004.06.001>.
- [2] <https://www.rapyuta-robotics.com/ja/> (accessed June 10, 2023)
- [3] <https://www.selenium.dev> (accessed June 10, 2023)
- [4] 'Requests: Http for Humans', <https://requests.readthedocs.io/> (accessed June 10, 2023)
- [5] 'Traveling Salesman Problem (TSP) with Miller-Tucker-Zemlin (MTZ) in CPLEX/OPL', <https://co-enzyme.fr> (accessed June 10, 2023)
- [6] <https://www.ros.org/> (accessed June 10, 2023)
- [7] <https://cps-vo.org/node/26614> (accessed June 10, 2023)
- [8] <https://www.rapyuta-robotics.com/ja/> (accessed June 10, 2023)

Harness the Power of Debugging

Kiruthika Ganesan

Kiruthika.ganesan@gmail.com

Abstract

Debugging is a superpower which will elevate anyone in tech to the next level and for a tester the benefits increase multifold. With this superpower in their arsenal any ordinary tester could transform into an extraordinary one. This talk is a reflection on how debugging has changed my life and empowered me to navigate the various challenges in my career. I started debugging as part of analyzing bugs, then used it to understand and read code. Later, I got involved in writing automated checks, which in turn gave me the tools to expand my testing and look at software implementations with a different perspective. Following the positive results, I have tried to employ the same routine in other problem areas. From my experiences, I have found that one could debug a production issue to do root cause analysis, debug a process to identify gaps, debug an idea/problem to understand the scope, debug an initiative to explore the possibilities and many more. The whole journey takes a different turn when done alone, when done with someone or a group, when different tools and techniques are used.

Biography

Kika is passionate about testing and approaches testing with a holistic view. She has over 17 years of experience in the IT industry, working as a tester, developer, and a trainer. Kika believes in the power of people and collaborating to build a safe environment where the teams can thrive and work on creating great software. She enjoys teaching and getting involved in community activities like speaking at conferences and delivering workshops. She is also one of the tutors at the Coders Guild and a core member of Synapse QA. Also, a keen advocate of Women in Tech initiatives and a global ambassador. In her spare time, Kika loves spending time with her family and writing short stories.

Copyright Kiruthika Ganesan 21/06/2023

1. Introduction

What is debugging?

According to Wikipedia: "In computer programming and software development, debugging is the process of finding and resolving bugs (defects or problems that prevent correct operation) within computer programs, software, or systems."

A little bit on etymology. The terms "bug" and "debugging" are popularly attributed to Admiral Grace Hopper in the 1940s. While she was working on a Mark II computer at Harvard University, her associates discovered a moth stuck in a relay and thereby impeding operation, whereupon she remarked that they were "debugging" the system. However, the term "bug", in the sense of "technical error", dates back at least to 1878 and Thomas Edison who describes the "little faults and difficulties" of mechanical engineering as "Bugs". Below is the first bug report ever written and it was an entry in Grace's diary.

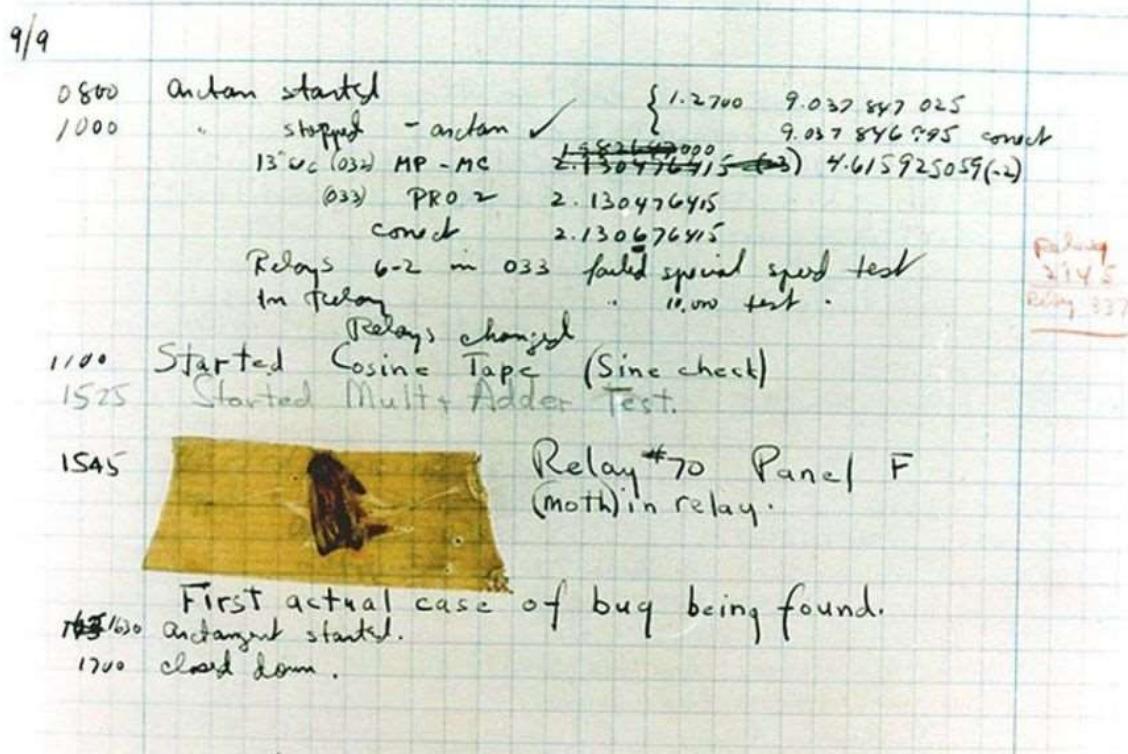


Figure 1 A computer log entry from the Mark II, with a moth taped to the page

However, I would describe debugging as a creative process which uses both sides of the brain. The left hemisphere is more logical & analytical while the right is intuitive and imaginative. Traits like curiosity, problem solving, and synergy come from the right brain while the left makes use of our rational processes of analysis and logic. When both work in harmony, any problem can be solved. So, debugging also means problem solving.

2. Stages of Debugging

When we talk about bugs, we can find them in any phase of the Software Development Life Cycle (SDLC). Bugs don't just exist on the code level. We can find them in requirements, design, architectural diagrams, UI wireframes, code, documentation and even processes. If we ever have a bug or a problem, we can use these four stages to explore and fix them.

The four stages are;

1. Identify
2. Isolate
3. Fix
4. Review

The initial stage is to Identify the bug/problem and gather information on what we know. It could be from previous experience handling a similar problem, talking to people who know more context on the issue, from the logs, monitoring tools, or bug reports. This phase is primarily to identify what the is bug and understand why it is happening.

Next is to Isolate so it can be reproduced. Where is this bug - frontend, backend, database, API, message queue, etc. Form a hypothesis and experiment with it – test the hypothesis. In the case of an opaque process (something you don't have the code for), follow the same approach and try to isolate the part of the process which is the pain point and try different options.

When we are happy with the result, make the relevant change. Fix the code or fix the problem.

The last stage is to test the changes, check whether the bug is no more and, without fail, run the existing regression suite , so we don't introduce any new issues. It might also be a good idea to assess whether this scenario can be added to the regression suite to ensure the bug doesn't resurface again.

3. My Experiences with Debugging

I want to share my top three experiences with debugging in this paper and highlight how it has benefitted my career.

3.1 Debugging a production bug

Finding out what has happened and trying to reproduce a production bug in a test environment is what got me started on debugging. It is fascinating when you go down this path because you are critically thinking about why it happened, what did we miss?



Figure 2 Online payment flow showcasing the different parties

Once upon a time, I was working on a payment gateway team, who was the middleman between the merchant and the banks. As you can see in the diagram, a user would go onto a merchant website like John Lewis and order a sofa. They will pay for the purchase on the John Lewis site. This transaction would go through the payment gateway (our team) to the acquirer who is the merchant's bank and then to the issuing bank which is the card holder's bank. Lots of checks are done and when both banks approve, the payment is taken by the merchant. As the payment gateway team, we capture all the transactions for the merchant and at the end of the day we would process settlements for them, too. Settlement is a very crucial step in the process as that is when the merchant gets paid. One day, this process was broken. We got angry calls from merchants saying their account doesn't have the money in it. Some merchants need this money to open shops for the week. It was serious and our team was tasked to find out and fix the problem.

3.1.1 Fishbone Diagram

We used a Fishbone diagram (also known as an Ishikawa diagram) for the analysis. This has three parts to it. We have a head depicting the problem. Next are the categories which are the blue boxes representing the fish bones. These are the main contributors to the problem. Finally, the categories are further subdivided into causes.

The whole approach is quite visual and organizes the cause-and-effect relationships.

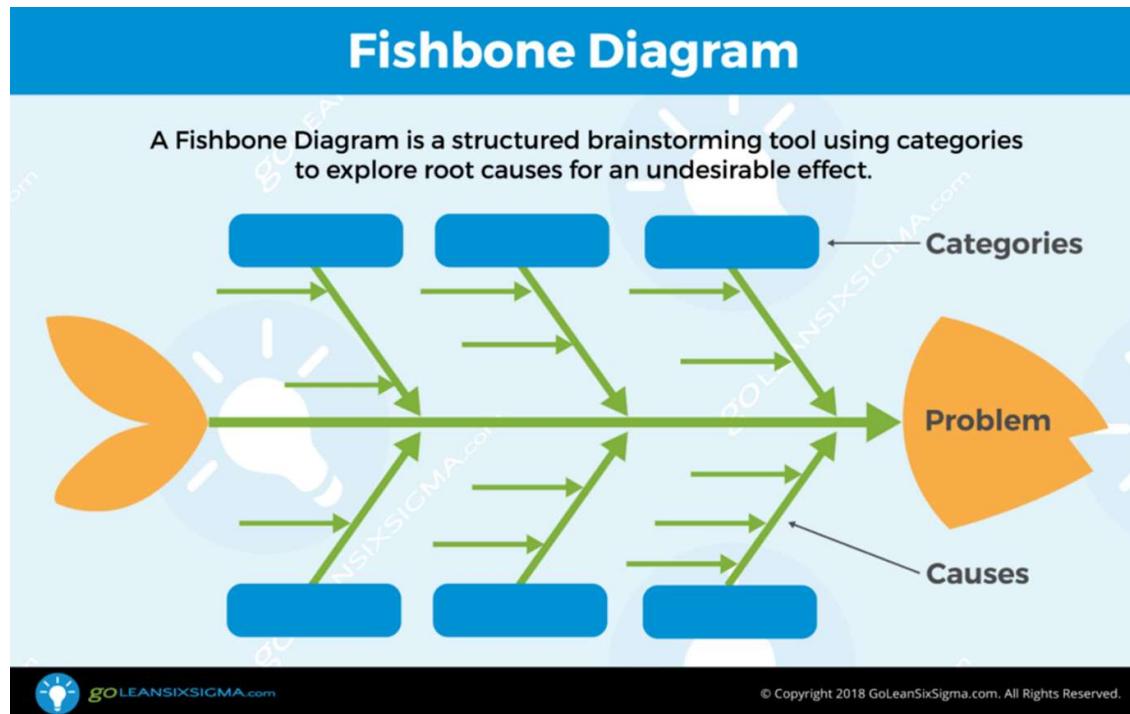


Figure 3 Fishbone diagram (also known as an Ishikawa diagram)

We used the fishbone method to debug the problem of broken settlements. Our categories were Requirements, Deployment, Development, and Testing as we felt these would be the main contributors. Then we started breaking down the categories further and asking questions: Did we build it against latest specs? Was the release deployed? Did we use feature toggles? Did it work in the test environment?

After answering these questions, we were able to discount the categories Requirement and Deployment as they seemed to be in order. We used a different technique to deep dive and brainstorm the other problem areas.

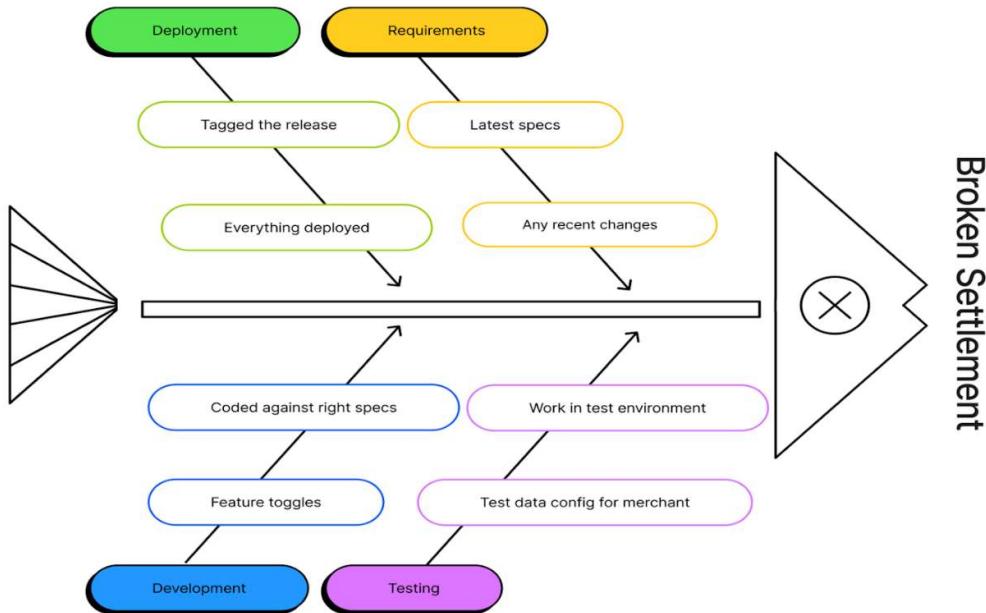


Figure 4 Debugging settlement bug using Fishbone diagram

3.1.2.5 Whys Technique

This technique is all about asking the ‘Why’ question and, for each answer, ask ‘Why’ a total of five times. With every ‘Why’ you move from one symptom to its underlying cause. We were still trying to reproduce the problem in our test environment, so we started with that as our first question: Why can’t we reproduce the issue in our test environment? The answer was, “Prod Merchant config was different.” Following which we changed our test setup to match production, tried again, and still couldn’t reproduce. Then we asked the next question: “Why can’t we still reproduce it?” The answer was, “Maybe it is the merchant’s setup causing the issue.” We needed to know what was happening on the bank’s side and they were eager to solve this issue, too. We set up a joint call to investigate. Still, we couldn’t reproduce the issue and asked another “Why?” as the problem was not us, but the bank. So, who else was in the equation? The answer was: Maybe a 3rd party. Following which we started comparing their test and prod environments. They had a 3rd party API integration which they were simulating, so we switched it off and tried again. Bingo! That was the issue. The 3rd party did some code changes, which started adding an extra character in the responses to the bank. The bank was sending us the same response and it somehow changed the order of the transactions in our settlement file. This was the issue causing the settlement process to fail. We got the 3rd party to fix their changes, the bank updated their mocks, and we updated our tests to additionally validate the order of the transactions.

We were feeling like Sherlock Holmes at this point, and I have been using the Fishbone diagram and 5 Whys for any sort of root cause analysis ever since. I have used this technique & framework to debug problems in requirements, designs and even processes. Understanding the problem well is a job half done. So, do spend the time to investigate and isolate before fixing anything.

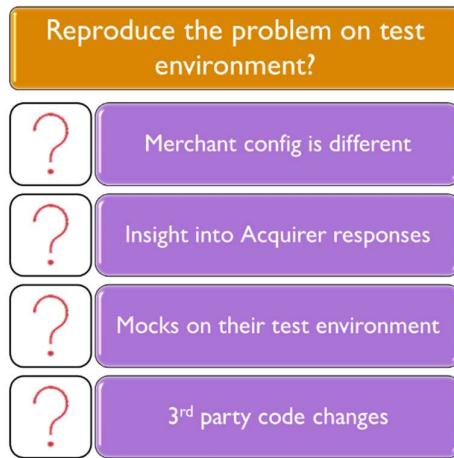


Figure 5 Debugging settlement bug using 5 Whys technique

4. Becoming technically competent

The first experience helped me understand how important those investigation skills were. The next experience got me curious to look beyond the black box. As a tester I became very good at asking questions, exploring the product, understanding logs, and analyzing bugs. Never really ventured inside the box or looked though the code. This posed a gap which held me at bay during technical conversations with the developers. So, I took baby steps to read code and find out how the software works.

4.1 Debugging failed builds

Debugging failed builds was the next steppingstone in my journey. A failing test is an opportunity to improve your understanding and your code. This is also when I fell in love with the debuggers because it gives you an opportunity to be transported into the code. Let's look at Chrome Dev Tools which I frequently use.



Demo_ Get Started
Debugging JavaScript

Figure 6 A video of using Chrome Dev Tools to debug javascript

I found a Twitter post from Julia Evans explaining the amazing debugger features (Figure 7 . Most debuggers have conditional breakpoints, the ability to edit and continue, set watchpoints, backtraces, do time travel, and establish dependencies between breakpoints. It is worth investing some time to get to know the debuggers for your languages.

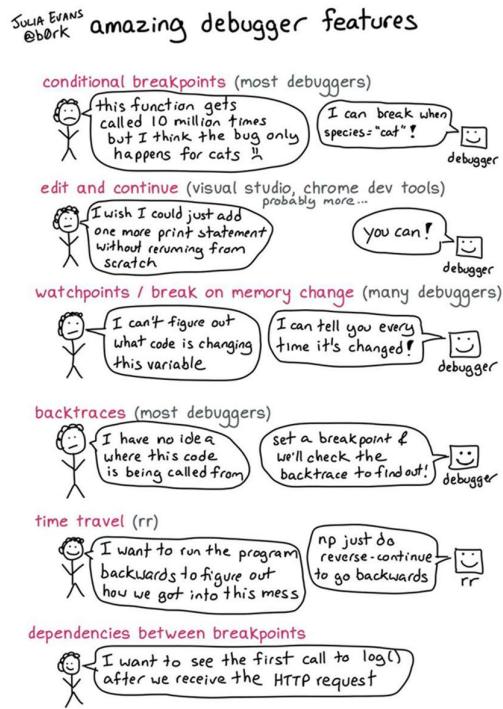


Figure 7 Twitter post from Julia Evans explaining amazing debugger features

One of my personal favorites is IntelliJ for its versatility. With Postman for APIs, you can use the API client, mock server, and console. I use Datadog for logging, monitoring options and session replay to view real time user journeys. And then there's Charles, a reverse proxy tool which records all the traffic between your machine and the Internet. Now-a -days there is an abundance of tools based on your tech stack and needs.

It is easier to learn or upskill on the job instead of through books or courses since you can practice and implement it straightaway. It is easier to relate to and it will stick with you when you use it a lot. So, find a problem worth solving and debug it. It is a rewarding experience. If possible, find an accountability partner. Once a fellow tester and I spent a couple of hours every week mapping out all the backend hops for our main customer journeys by referencing the UI wireframes designs and the architectural diagrams. For example, when a user searches something on the website, what is the microservice or API being called, what DB populates the data, what does the logging look like, and what is the ID which ties all the hops together. This was super handy for all of us for future debugging and to understand the traceability.

4.2 Wolf Fence Algorithm

What is this algorithm? We will answer it through a question: There's one wolf in Alaska, how do you find it?

You would first build a fence down the middle of the state, wait for the wolf to howl, and determine which side of the fence it is on. In debugging, this means you add some logging/print statements.

Repeat the process on that side only, until you get to the point where you can see the wolf. This is kind of a divide and conquer solution and it works well when you are trying to find a needle in a haystack.

4.3 Bug Magic

Bugs are very good at hiding. While debugging, if you have a complex piece of code, try the different paths and test out different combinations as bugs change form and shape. Heisenbug is the name

given to those bugs that only happens when you're not debugging / disappears when you are. Sometimes these are time sensitive bugs such as race conditions. Have you noticed the same test works fine locally on your machine but fails on Jenkins? Be conscious of those bugs and switch your debugging tool and technique for it. Regression testing is very important. We don't want to debug and fix one problem then introduce more problems.

4.4 Rubber Ducking

Debugging is very intense. Sometimes when you hit a wall and want a fresh perspective, Rubber ducking can help. It is about explaining your approach to another person as the process of explaining gives you clarity on the problem in turn allowing you to fix it. If you can't find a person nearby, you can also explain it to an object like a toy duck. Testers can provide valuable insights in this process as we are good at asking the question Why? I personally have benefitted from this process by being on both sides.

4.5 Over Thinking

While debugging the production bug, we understood the power of analysis and thinking. But sometimes we might go into overthinking mode. For those times, we need to quit thinking, dive into the problem and start looking.

4.6 Remote Debugging

Our team used to do a lot of pairing and mobbing sessions. This all came to stand still and not so effective sessions during the COVID pandemic due to remote working. So, we had to find something to help us work collaboratively.

The "code with me" option in some IDEs was a game changer to bring back pairing and mobbing with a remote setup. On IntelliJ, we could invite people for a session by giving them full/restricted access. Full access means they will be able to do whatever you can on your IDE: write code, debug, and run tests. This is very handy for mobbing as in a traditional setup where a single machine would be used to code and the drivers and navigators would take turns. You could also combine the session with a video/audio call. There are options to share screen, chat, raise hand, and add emojis. Once done with the session you can disconnect and turn off the access. This option is available on the free version too but there is a restriction on the session limits. We have a licensed version of the IDE and it is super useful.

5. Writing code and preventing bugs early

The final part of the experience was attempting to code. After learning all these new skills and taking part in these pairing and mobbing sessions, I was charged up to try it out myself.

5.1 Test Driven Development (TDD)

My respect for coding went to another level when I got the opportunity to pair with an awesome dev who was obsessed with clean code practices from Uncle Bob. If you are serious about coding, I would suggest taking Uncle Bob's courses or reading his books. He introduced me to the concept of Test Driven Development (TDD). It essentially means you start by writing a failing test. Then, you write just enough code to pass the test. Next, refactor/improve the code with tests intact. TDD is a type of preemptive debugging. This approach is far more effective than just coding as it will reduce bugs because the code is simpler to understand.

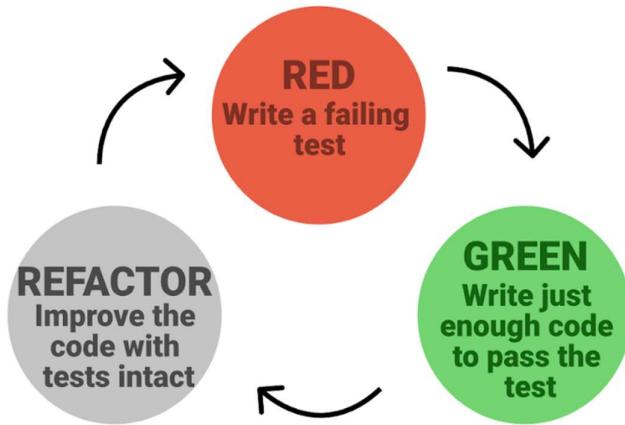


Figure 8 A picture showing the TDD concept

5.2 Preemptive debugging & Defensive Programming

The debugging type we used for production bug and failed bugs are reactive debugging. We didn't anticipate the problem while building it, but we dealt with it after it happened. Preemptive debugging on the other hand, involves writing code that doesn't impact the functionality of the program but helps developers either catch bugs sooner or debug the source code more easily when a bug occurs.

Preemptive debugging means adding better logs, retry mechanisms, and exception handling.

Defensive Programming is another aspect of it where you don't let an error happen. It is about making the software behave in a predictable manner despite unexpected inputs or user actions. It is about writing foolproof code. For example, have you sent a mail saying there is an attachment and forgot to attach it? Defensive programming in this case would be to have a pop-up message saying "You have mentioned attachment in the mail, but there are no files attached. Do you want to send anyway?"

5.3 Code refactoring

Code refactoring is an integral part of TDD as it improves the overall code quality by reducing the complexity and maintenance. Anyone can easily read, understand, and work on it. In the long run, it also saves time and cost. My favorite refactoring option on IntelliJ is renaming. No need to find and replace, just use the renaming option. The Extract option can be used on multiple items like method, number, field, variables, parameter. If you have a big method, you can use this option to split that method into smaller methods. The Inline option is the reverse mechanism of Extract. If you want to combine two or more methods into one, this option can be used. Instead of reinventing the wheel, use the options these IDEs provide.

To sum up my experiences, first was the Sherlock Holmes mode. Next was being technically competent and understanding code mode. And the 3rd one was writing code mode.

6. Summary and Takeaways

1. The four steps in debugging are: Identify, Isolate, Fix and Review. You can debug all sorts of problems, not just code related ones.

2. Bugs and debugging make you better. Have the right attitude and take it as an opportunity to improve.
3. Know the problem well before solving it. Take the time to investigate and understand.
4. Sometimes you need to quit thinking and start looking at the logs or just get into it.
5. The best way to learn something new is to find the problem you are passionate about solving and then your learning becomes easy and focused.
6. Debugging is intense. It's mentally challenging. Take breaks, talk to people, and practice rubber ducking.
7. Don't panic. Ask for help and have fun! You are not alone.
8. Celebrate the wins and share stories. Do this to motivate yourself and others.

7. References

Wikipedia reference: <https://en.wikipedia.org/wiki/Debugging>

Website:

Figure1: Debugging: <https://en.wikipedia.org/wiki/Debugging>

Figure2: Online payment flow: <https://stripe.com/in/guides/introduction-to-online-payments>

Figure3: Fishbone diagram: <https://goleansixsigma.com/cause-and-effect-diagram/>

Figure7: Julia Evan's Twitter post: <https://twitter.com/b0rk/status/1144011000208863239>

Figure8: Test Driven Development: https://goodfangsm.life/product_details/70761037.html

Improving Enterprise Scale Test Automation with ML-Based Predictive Analytics

Gumeniuk D., King, T. M.

Dmitriy_Gumeniuk@epam.com

Abstract

AI and machine learning continue to make considerable progress in advancing state of the art computing. Recently, the focus has been on generative AI systems like ChatGPT. However, these models encompass a fundamental aspect of ML, the ability to observe and analyze historical data and make predictions. Automation is a core component of any testing strategy but is especially important in enterprise projects with hundreds of thousands of tests. Collecting, coalescing, and analyzing the results of large-scale test automation projects across multiple levels and types of testing is challenging. However, incorporating AI and ML into the testing process, including leveraging it to enhance test reporting and analytics, is an effective way to improve large scale test automation. This paper describes some of the key challenges faced in large scale test automation projects and provides a summary of how AI and ML are being applied in practice to overcome those challenges. Emphasis is placed on the stability of executing automated tests at scale, and providing meaningful, actionable insights from the results. The findings and experiences from enterprise case studies are summarized, including one that leverages an open-source, AI-powered test automation dashboard.

Biography

Dmitriy Gumeniuk is the Head of Testing Products at EPAM Systems, a leading global provider of digital platform engineering and software development services. With over 17 years of experience in software development, Dmitriy has been at the forefront of innovation in test automation. He has led the development of multiply solution and accelerators at EPAM's Test Competency Center, focusing on the application of Machine Learning and Neural Networks in test automation. Dmitriy is a frequent presenter at conferences, has notable contributions into open-source community and DevTestOps community by organizing the annual DelEx conference.

Tariq King is the Vice President of Product-Service Systems at EPAM, where he manages a portfolio that lies at the intersection of software products and services, and supports the business through technology consulting. He has over 15 years' experience in software engineering and testing and has formerly held positions as Chief Scientist, Head of Quality, Director of Quality Engineering, Manager of Software Engineering and Test Architect. Tariq holds Ph.D. and M.S. degrees in Computer Science from Florida International University, and a B.S. in Computer Science from Florida Tech. He has published over 40 research articles in peer-reviewed IEEE and ACM journals, conferences, and workshops, and has been an international keynote speaker at leading software conferences in industry and academia.

1 Introduction

In the rapidly evolving landscape of software development, test automation has emerged as a cornerstone for ensuring software quality, particularly in large-scale enterprise projects. While automation has significantly improved the efficiency and coverage of testing, it also brings forth a set of challenges, especially when hundreds of thousands of tests are involved. The complexity of managing and analyzing test results at this scale necessitates innovative solutions.

Artificial Intelligence (AI) and Machine Learning (ML) have shown promise in various domains, including natural language processing, computer vision, and more recently, in software testing. This paper focuses on the application of ML-based predictive analytics to improve enterprise-scale test automation. Specifically, we explore the use of k-Nearest Neighbors (kNN) models, enhanced by Boosting Trees algorithms, for the triage of failed test reports. The primary objective of this work is to automate the categorization of failed test reports into distinct groups: Product Bugs, Automation Issues, and System Issues. Such categorization aids in streamlining the workflow for engineers and provides actionable insights for decision-makers.

This paper outlines the key challenges associated with large-scale test automation and offers an AI-driven solution to overcome some of these challenges. The proposed methodology incorporates text normalization techniques to reduce noise in logs and gives a higher weight to recent test results to produce a more relevant triage. The insights are grounded in real-world case studies, highlighting the tangible benefits of integrating AI and ML into test management systems. The paper is organized as follows: Section 2 provides the necessary background and outlining the problem statement, and Section 3 summarizes the key challenges that motivate this work. Section 4 describes our approach and the methodology employed, followed by a detailed explanation of the technical aspects of the ML models leveraged in the solution. Section 5 presents real-world case studies, including an analysis applied improvements and application to validate the effectiveness of our approach. In the last section, the paper concludes with a discussion, future work, and acknowledgments.

By the end of this paper, the reader will understand the challenges in large-scale test automation and how ML-based predictive analytics can offer practical solutions to overcome them.

2 Background

Continuous Testing and Quality Gates

Test automation has been an integral part of software development for years, enabling teams to execute a large number of tests efficiently. However, as software systems grow in complexity and scale, traditional test automation methods often fall short. This is especially true as full transparency into the reasons for test failures has become integral to establishing Continuous Testing (CT) practices with automated decisions on Quality Gates (QG). The ability to fully triage all failed test reports is crucial for making GO or No-Go decisions at Quality Gates — points in the delivery pipeline where the product's quality is assessed. Fully triaged failed reports provide clarity on the number of existing product issues versus test automation issues, thereby influencing critical decisions in the CT process.

AI and ML in Test Automation

Artificial Intelligence (AI) and Machine Learning (ML) have made significant strides in various domains, including software testing. While much of the focus has been on generative models and natural language processing, predictive analytics has received less attention. Predictive analytics in software testing involves using historical data to make future predictions, such as the likelihood of a test failing or the probable cause of a failure. Enterprise-level projects often involve hundreds of thousands of tests, making the management and analysis of test results a daunting task. The challenges include, but are not limited

to, the triage of failed tests, identifying flaky tests, and providing actionable insights for decision-making. The triage of failed test reports is a critical aspect of test automation. Traditionally, this has been a manual process, requiring significant effort from engineers. The application of k-Nearest Neighbors (kNN) models, enhanced by Boosting Trees algorithms, offers a promising automated solution. These algorithms use logs and failed stack-traces to form predictive models that can categorize failed tests into groups like Product bugs, Automation issues, and System issues. The initial categorization of a failed test report often falls under a "To Investigate" state, requiring manual intervention. ML algorithms can automate this initial triage, thereby streamlining the workflow for engineers and reducing the manual effort involved.

3 Challenges in Enterprise Scale Test Automation

Automated testing is now a necessity in enterprise software development, especially for agile or DevOps environments that prioritize speed and continuous delivery. Yet, while automation can streamline many aspects of testing, it also brings forth a unique set of challenges that are amplified at an enterprise scale. In this section, we explore some of these obstacles that organizations often encounter in implementing and sustaining automated testing at a large scale.

Complexity of Test Scenarios. One of the primary challenges lies in the complexity of enterprise-scale software. Large-scale applications often have myriad functionalities and subsystems that need rigorous testing. Creating automated test scenarios that accurately mimic real-world use cases becomes exponentially more difficult as the application grows in complexity.

Management of Test Data. Data is the backbone of any testing regime. In an enterprise setting, handling vast amounts of test data across different databases, services, and environments becomes a laborious task. Ensuring that the data is consistent, secure, and up-to-date for every test cycle is often easier said than done.

Integration with Existing Systems. Enterprises usually have a blend of modern and legacy systems. The automated testing tools need to be compatible with different architectures, APIs, and even other testing tools. The integration often demands extensive customization and fine-tuning, making it a time-consuming endeavor.

Resource Constraints. Resource allocation is often constrained by budgetary limits, existing project commitments, and availability of skilled personnel. Building and maintaining an automated testing suite can be resource-intensive, often requiring investment in specialized expertise and ongoing training.

Maintainability of Test Scripts. Over time, as the software evolves, the automated test scripts can become outdated or irrelevant. Updating these scripts to adapt to new features, changes in UI, or alterations in workflow can be an overwhelming task, especially for large codebases.

Noise and False Positives. Automated tests are not immune to generating noise in the form of false positives or inconclusive results. Filtering out these anomalies to focus on genuine issues requires additional layers of analysis, which can sometimes offset the time saved through automation.

Evolving Technologies. The technology landscape is ever-changing. The introduction of new development methodologies, languages, and tools can make existing automation frameworks obsolete. Organizations need to be agile enough to adapt to these changes without disrupting their existing testing pipelines.

Understanding the 'Why' Behind Failures. Traditional automated systems are excellent at pointing out 'what' has failed but are usually not designed to diagnose 'why' it has failed. This requires human intervention, further complicating the automation process.

By understanding these challenges in depth, we can better appreciate the value that intelligent, adaptable auto-analysis brings to the table. In the next section, we will examine how this tool addresses some of these challenges by leveraging machine learning and advanced analytics.

4 Machine Learning in Test Automation: A Paradigm Shift

The challenges presented by enterprise-scale test automation are not just hurdles; they are opportunities for innovation. One promising avenue is the application of machine learning algorithms to the automation process, a shift that significantly redefines our approach to testing. This section aims to elucidate the role that machine learning plays in modern test automation.

Intelligent Test Data Management. Traditional test data management systems struggle with the complexity and volume of enterprise data. Machine learning algorithms can predict what kinds of data are most likely to induce failure, allowing for more targeted testing scenarios. ReportPortal, for instance, uses analytics to understand the past behavior of test data, helping in creating more realistic test cases.

Self-Healing and Adaptive Test Scripts. One of the most significant challenges in maintaining automated test suites is the requirement for continual updates. Self-healing mechanisms can dynamically identify the correct locator at the moment of test execution. When a script encounters a missing or changed element, the self-healing algorithm kicks in to search for the most likely alternative locator based on historical data or predefined heuristics. This not only minimizes test flakiness but also reduces the manual effort required to update test scripts, further enhancing the adaptability and resilience of automated test suites. For the adaptive test scripts Machine Learning models can learn from test outcomes to automatically adjust test scripts, all this dramatically reducing the overhead involved in maintenance.

Enhanced Failure Diagnosis. While traditional test automation tools can identify a failure, they usually can't explain why it failed. Advanced machine learning models can analyze a multitude of variables in real-time to offer not just a 'what' but a 'why,' thereby significantly aiding in troubleshooting.

False Positive Identification. Machine learning algorithms can provide an additional layer of analysis to distinguish between actual defects and false positives, a common issue in automated testing. This significantly reduces the time required for manual oversight.

Customizable and Scalable. The use of machine learning in testing allows for more personalized, project-specific testing scenarios. It adapts to the idiosyncrasies of a given software ecosystem, thereby offering a higher level of customization without sacrificing scalability.

Real-World Applications: Test Failure Categorization. ReportPortal has integrated machine learning to automatically categorize defects and suggest similar historical defects. Its auto-analyzer feature includes retraining capabilities that take into account human-made decisions over time. The tool leverages machine learning to analyze failure reports and identify current patterns in stack traces, thereby making the tool more intelligent and adaptable over time.

By incorporating machine learning into test automation, enterprises can not only overcome many of the challenges previously discussed but also open up new avenues for optimization and quality assurance.

5 Case Study

ReportPortal is an open-source solution that offers a practical method of how ML can be applied to test automation. The platform uses advanced algorithms for auto-analysis and provides options for manual analysis, thereby offering a balanced approach.

5.1 Experimental Set Up

To gauge the efficacy of machine learning algorithms in a structured methodology was applied. The study adopted a longitudinal analysis approach, evaluating 20 large-scale enterprise projects over a span of 6 months. The projects selected for this study were diverse in nature, ranging from API testing to end-to-end UI testing, including BDD, and showcased a diverse mix of programming languages such as Java, Python, JavaScript, and C#. Additionally, different test runners, including TestNG, Junit, NUnit, Pytest, Jasmine, and MochaJS were utilized. Another layer of complexity was added by the various speaking languages used in logs to elucidate causes of failures (English, Russian, Chinese).

Data Sources

For study, we used and obfuscated production database with total data amassed as 2.7Tb.

The primary types of data were collected:

1. **Automated Testing Results:** Test execution logs were collected, including logs of test automation execution, specifically all error logs and stack traces of the failed test cases, test case IDs, parent test suites, test parameters and attributes.
2. **Manual Reviews:** To serve as a benchmark, manual categorizations were also conducted for a subset of tests, focusing on the areas where machine learning-based recommendations were provided.

Metrics for Evaluation

The performance of machine learning in automated testing was evaluated based on the following metrics:

- **Accuracy:** The ratio of correctly classified defects to the total number of defects.
- **Precision:** How many of the flagged issues were indeed defects.
- **Recall:** Of all the defects, how many were correctly identified by the machine learning algorithms.
- **F1-Score:** A balanced metric that considers both precision and recall.
- **Time-Saved:** The amount of manual effort saved by automating defect categorization and recommendation processes.

Tools and Technology

ReportPortal: This was the primary tool used for this study, since it serves a role of a single entry point for the test reporting in our organization and consolidate a variety of testing results.

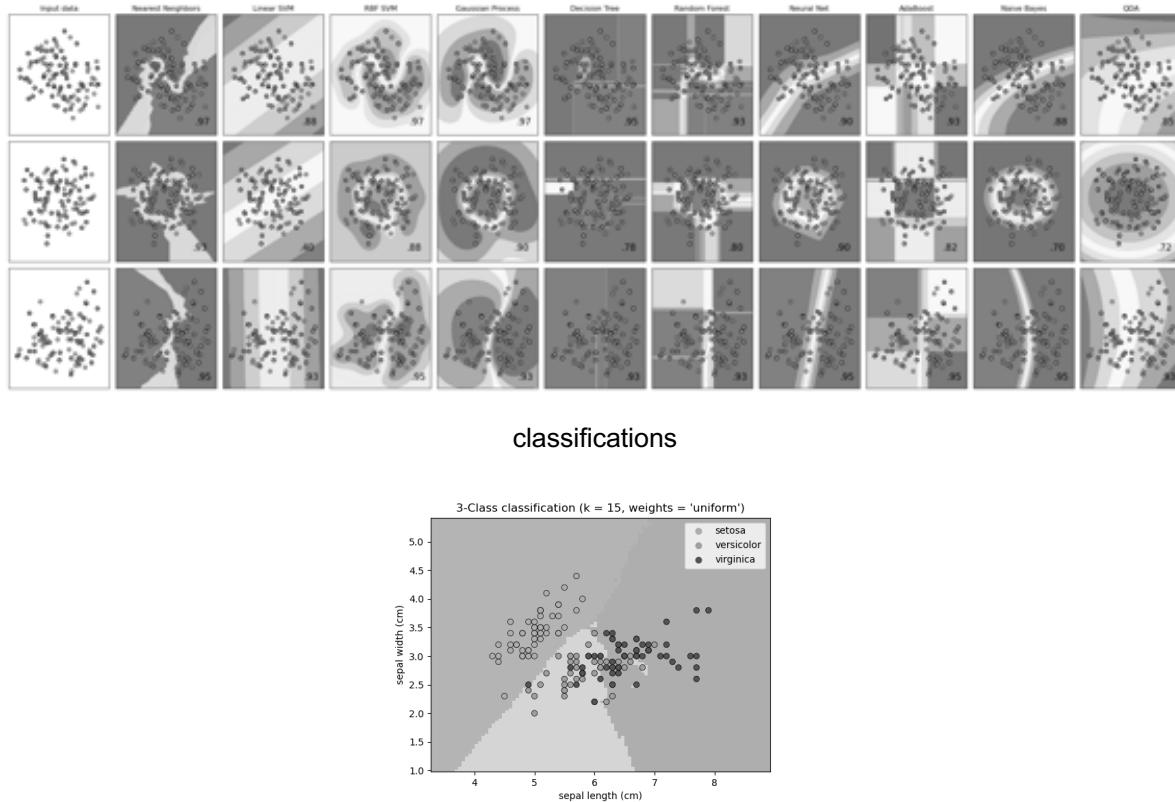
Python for Data Analysis: Python libraries such as Pandas and Scikit-learn were used for data pre-processing and statistical analysis.

Tableau for Visualization: Key findings were represented graphically through Tableau dashboards.

Statistical Methods

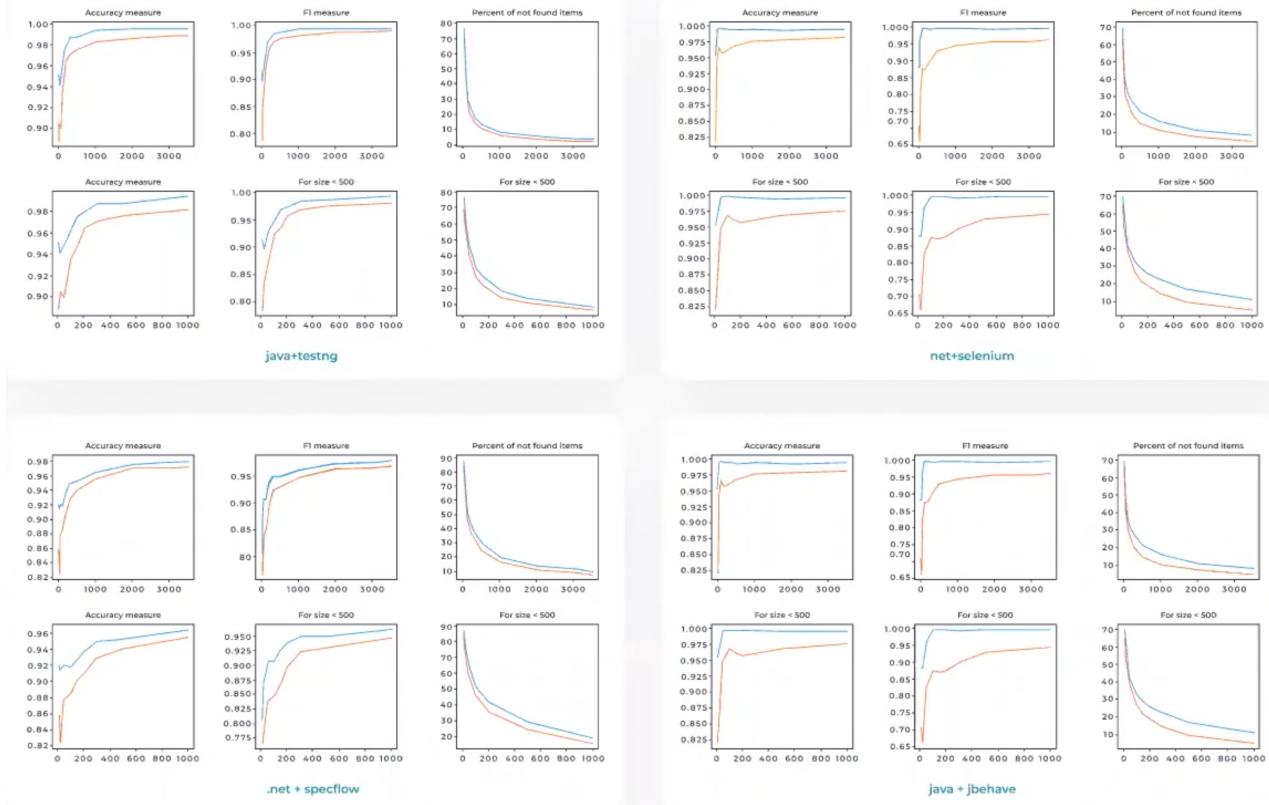
To ensure the robustness of our findings, statistical tests like chi-square tests for independence, t-tests for comparing means, and logistic regression for understanding variable impact were employed.

By adhering to this comprehensive methodology, we aimed to ensure the reliability and validity of our findings. The results are presented in the subsequent section.



5.2 Results and Analysis

Through an exhaustive examination of 20 large-scale enterprise projects, this study sought to demystify the complexities and challenges of test automation at an enterprise level. The data amassed was comprehensive, offering a rich dataset for in-depth analysis. Here are the pivotal findings:



Log Quality and the Human Factor

- The quality of logs plays a crucial role in the outcomes of machine learning algorithms. Generic error logs or assertion error messages, such as "expected <true>, but was <false>" or "failed", hinder algorithms from deducing meaningful insights.
- A mere enrichment of error descriptions in assertions or exceptions managed by try-catch led to an 80% surge in categorization accuracy.
- The principle is clear: If a QA engineer unfamiliar with the failing test can't discern the issue based on the log description, machine learning will likely struggle as well.

The Human Factor: "Friday Bug". Human uncertainty in categorizing test failures often leads to the selection of inappropriate categories. These categories often undergo transformation after 2-3 test runs. The "Friday Bug" anomaly was identified, where engineers, in a bid to wrap up their week, hastily marked a segment of failed tests as defect-free. This behavior significantly blurred clustering and reduced precision.

Incremental Learning is essential due to changes in the project landscape, such as the introduction of new tests, application evolution, and enhanced coverage. The machine learning model should reuse newly marked data with the latest categorizations made both by human and machine input.

Decisions Based on Recency. For the ML model it does make sense to prioritize latest categorization decision over the old ones via boosting the scoring. Decisions related to tests were heavily influenced by the recency of the decision timestamp. As teams collectively unearth details, decisions about functionalities evolve.

Engineer Decisions versus Machine Decisions should always have a priority. However, the over-reliance on human decisions can sometimes adversely impact accuracy, especially when considering the human element discussed in the "Friday Bug" anomaly.

Traditional automated systems are excellent at pointing out 'what' has failed but are usually not designed to diagnose 'why' it has failed, necessitating human intervention.

5.3 Stability and Actionable Insights

In the realm of test automation, especially at the enterprise scale, stability and the ability to derive actionable insights are paramount. The integration of machine learning into this domain has brought forth a new dimension of stability and has enhanced the depth and breadth of insights that can be extracted. Here's a detailed exploration:

5.3.1 Stability in AI/ML-Driven Solutions:

- **Consistency in Predictions:** Machine learning models, once trained with a substantial amount of data, tend to provide consistent predictions. This consistency is vital for test automation, ensuring that the same input will yield the same categorization or recommendation, barring any model retraining.
- **Adaptability to Changes:** One of the hallmarks of machine learning models is their ability to adapt. With incremental learning, as discussed in the previous section, models can adjust to the evolving nature of applications and their testing scenarios. This adaptability ensures sustained stability even in dynamic environments.
- **Reduced Human Error:** By automating certain decision-making processes, the potential for human errors, such as the "Friday Bug" anomaly, is significantly reduced. This leads to a more stable and reliable testing process.

5.3.2 Deriving Actionable Insights:

Root Cause Analysis: Advanced machine learning models can delve deeper into test failures, not just identifying the 'what' but also illuminating the 'why'. This deeper understanding aids in troubleshooting and can guide corrective actions more effectively.

Predictive Analysis: Beyond just analyzing current test results, machine learning models can predict future outcomes based on historical data. For instance, predicting which tests are more likely to fail in the next cycle or identifying potential bottlenecks in the testing pipeline.

Optimization Recommendations: By analyzing patterns over time, machine learning can suggest optimizations in the testing process. This could range from recommending changes in test sequences to optimize execution time or suggesting areas where test coverage could be enhanced.

Feedback Loop for Continuous Improvement: The integration of machine learning creates a feedback loop. As tests are executed and results are analyzed, the insights derived can be fed back into the system, leading to continuous refinement and improvement of both the testing process and the machine learning models.

5.3.3 Real-World Application and Challenges

The ReportPortal solution exemplifies the practical application of deriving actionable insights. Its auto-analyzer feature not only categorizes defects but also suggests similar historical defects. This dual functionality provides immediate insights into potential defect patterns and historical context, aiding in faster resolution.

While the integration of machine learning offers enhanced stability and insights, it's essential to recognize that it's not a panacea. The quality of insights is heavily dependent on the quality of data fed into the system. As highlighted in the "Results and Analysis" section, generic or non-descriptive logs can hinder the depth of insights derived.

Additionally, while machine learning models offer stability, they need periodic retraining to maintain their accuracy and relevance, especially in rapidly evolving application landscapes.

In conclusion, the fusion of machine learning with test automation at the enterprise scale offers a promising avenue for enhanced stability and deeper, more actionable insights. However, its success hinges on the quality of data, the relevance of the models, and the continuous feedback loop that ensures both the testing process, and the models evolve in tandem.

6 Conclusion and Future Direction

The integration of machine learning into enterprise-scale test automation has already shown significant promise, but the journey is just beginning. As technology continues to evolve and the demands of software testing grow more complex, there are several potential directions that this fusion of AI/ML and test automation might take. Here's a glimpse into the possible future trajectories:

6.1 Concluding Remarks

The landscape of software testing, particularly at the enterprise scale, is undergoing a transformative shift. As the demands of software quality assurance grow in complexity and scale, traditional methods of test automation are increasingly proving inadequate. This research has delved deep into the challenges faced by large-scale test automation and has illuminated the potential of machine learning as a solution to many of these challenges.

Several key takeaways emerge:

1. **Human Factor in Automation:** While automation aims to reduce human intervention, the quality of human input, especially in the form of detailed logs and accurate categorization, remains paramount. The "garbage in - garbage out" principle holds true, emphasizing the need for quality data for effective machine learning outcomes.
2. **Machine Learning's Potential:** The integration of machine learning into test automation offers a plethora of benefits, from intelligent test data management and self-healing test scripts to generation of the test case scripts. Tools like ReportPortal showcase the tangible benefits of this integration, offering both auto-analysis and manual analysis capabilities.
3. **Expanding Data Sources:** The potential data sources for analysis and categorization are vast. Application logs, often overlooked, can provide deeper insights into system behavior. Additionally, the advent of Visual AI, leveraging image recognition techniques, can further enhance the depth and breadth of analysis.
4. **Continuous Evolution:** The technology landscape is ever-evolving. As new challenges emerge, the fusion of AI/ML and test automation will need to adapt. Continuous research, learning, and adaptation will be pivotal in ensuring that this integration remains relevant and effective.

In conclusion, the fusion of machine learning and test automation represents a promising frontier in the realm of software quality assurance. While challenges abound, the potential benefits in terms of efficiency, accuracy, and quality make this a journey worth undertaking. As technology continues to evolve, it is incumbent upon the software testing community to harness the power of machine learning, continually innovate, and set new benchmarks in software quality.

6.2 Future Directions

Advanced Machine Learning Models:

- **Deep Learning Integration:** While the current focus has been on predictive analytics using models like k-Nearest Neighbors (kNN) and Boosting Trees algorithms, there's potential to integrate deep learning models, especially for more complex pattern recognition in test logs and application under test logs.

- **Natural Language Processing:** Advanced NLP techniques can be employed to better understand and categorize error logs, especially those written in natural language or in multiple languages, as highlighted in the methodology section.

Application Logs as a Rich Data Source, often overlooked, can serve as a treasure trove of information. These logs can provide deeper insights into system behavior, user interactions, and potential error patterns. Machine learning models can be trained to analyze these logs, offering a more granular understanding of system performance and potential areas of concern.

Visual AI for Enhanced Analysis, as an Image recognition technique can be employed to detect error patterns or pop-ups in application screenshots. This "Visual AI" can be pivotal in identifying visual discrepancies that might be missed in traditional testing. Furthermore, pixel-perfect change detection can ensure that any visual modifications to the application, intentional or otherwise, are immediately flagged and reviewed.

Automated Test Generation with Generative Large Language Models, aimed at a language generation, not just analyze but also generate test cases based on software requirements, user behavior, and historical defect data. This can lead to more comprehensive and targeted test coverage.

Self-Optimizing Test Suites beyond self-healing test scripts, envision test suites that can self-optimize, reordering tests based on dependencies, historical fail rates, or execution time, ensuring the most efficient test run possible.

Enhanced Integration with Development and Operations, allowing for real-time feedback loops. This can lead to immediate defect resolution during the development phase itself, further streamlining the DevOps pipeline.

Augmented Reality (AR) and Virtual Reality (VR) Testing become more mainstream, there will be a growing need for automated testing in these domains. Machine learning can play a pivotal role in understanding and simulating human interactions in virtual environments.

1. Ethical Considerations and Bias Mitigation

- As machine learning models make more decisions, there will be a growing need to ensure that these decisions are free from biases, especially if they impact user experiences or business outcomes. Future

research might delve deeper into ensuring fairness, transparency, and accountability in AI-driven test automation.

2. Continuous Learning and Adaptation

- The models will need to be designed for continuous learning, adapting not just based on historical data but also real-time data streams, ensuring that the testing process remains agile and responsive to immediate software changes.

3. Collaborative AI in Testing

- Envision a future where AI-driven testing tools collaborate, sharing insights, models, and best practices across different platforms and organizations, leading to a collective improvement in software quality globally.

In essence, the future of machine learning in enterprise-scale test automation is brimming with possibilities. While the challenges are manifold, the potential benefits in terms of efficiency, coverage, and quality assurance make this a frontier worth exploring in depth. As with any technological evolution, continuous research, experimentation, and adaptation will be key to realizing the full potential of this integration.

7 Acknowledgments

We extend our deepest gratitude to the team members of ReportPortal for their invaluable contributions and relentless dedication to the project. Their expertise and commitment have been instrumental in the success of our research and the development of the platform.

Special thanks go to key figures in the machine learning community, whose pioneering work laid the foundation for our research. Their insights and innovations have been a constant source of inspiration.

We are particularly grateful to Maryia Ivanina and Andrei Varabyeu for their guidance, mentorship, and unwavering support throughout this project. Their expertise has been invaluable in navigating the complexities of machine learning in test automation.

Lastly, we would like to express our appreciation to the members of EPAM's software testing practice. Their contributions, both big and small, have enriched this work in countless ways. Their collective wisdom and experience have been indispensable in shaping the research and ensuring its relevance to the industry.

8 References

Briggs, K.A., King, T.M. "Semi-autonomous, site-wide A11Y testing using an intelligent agent." *Journal of Software Testing*, Vol. 12, No. 3, 2020.

Daye, Philip Jr. "Quality Guild: Creating a Culture." *Journal of Software Quality*, Vol. 10, No. 2, pp. 25-40, 2019.

Cover, T. and Hart, P. "Nearest neighbor pattern classification." IEEE Transactions on Information Theory, Vol. 13, 1967.

Shubert, E., Rousseeuw, P.J., "Faster k-medoids clustering: improving the PAM, CLARA, and CLARANS algorithms." <https://arxiv.org/abs/1810.05691> (accessed August 27, 2019).

Pawlak, M., Augsten, N. 2015. "Efficient Computation of Edit Tree Distance." ACM Transactions on Database Systems, vol. 40 (1), No. 3. <https://dl.acm.org/citation.cfm?id=2699485>, (accessed August 27, 2019).

Freund, Y. and Schapire, R.E. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." Journal of Computer and System Sciences, Vol. 55, 1997.

"ReportPortal: An Open-Source Reporting Platform with Machine Learning Capabilities." Official Documentation, 2022. <https://reportportal.io/docs/>

Pedregosa, F. et al. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, Vol. 12, 2011.

"Tableau Software." Official Documentation, 2022. https://help.tableau.com/current/pro/desktop/en-us/gettingstarted_overview.htm

Witten, I.H., Frank, E., Hall, M.A., and Pal, C.J. "Data Mining: Practical Machine Learning Tools and Techniques." Morgan Kaufmann, 2016.

"EPAM Systems: A Case Study in Large-Scale Test Automation." Internal Whitepaper, EPAM Systems, 2022.

McKinney, Wes. "Python for Data Analysis." O'Reilly Media, 2018.

"Challenges and Solutions in Large-Scale Test Automation: A Survey." Journal of Software Engineering Research and Development, Vol. 8, No. 1, pp. 1-20, 2020.

"Continuous Testing and Quality Gates: A Practical Guide." Journal of Software Testing, Vol. 14, 2019.

Revamp Software Quality for Android Robots through Advanced Development and Deployment Methodologies

Theodore Guo¹, Monica Bao², Sophia Lee³, Kyle Zhou⁴, and Bo Li⁵

revampedrobotics@gmail.com, bo.a.li@intel.com

Abstract

Robotics software quality is crucial to control robot hardware while interacting with the physical world to accomplish a set of complex tasks. Software is evolving to tackle sensor errors, robot software/hardware failures, human errors, and uncertainties from the external environment. This paper describes challenges to revamp the software quality for an Android platform robot based on the experience of the First Tech Challenge (FTC) robotics team, Revamped Robotics.

FTC is an annual competition for pre-college students with thousands of teams across the world competing to solve real world challenges through firsthand robotics experiences. FTC competitions begins with a two-month period between when the competition challenge is released and the first competition. The FTC season is roughly eight months long consisting of multiple periods of robot performance improvements between FTC competitions. To assess the quality of suggested robot changes, the team employed the SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis methods to enable a smooth coordination between the robot hardware and software. The team utilized agile development processes, the cloud-based GitHub software management platform, and Continuous Integration/Continuous Deployment (CI/CD) methods to enable frequent software improvement cycles. In addition, the use of deep learning TensorFlow with robot embedded sensors/actuators increased the effectiveness of software development.

Biography

Authors are high school students from Revamped Robotics, a seven-year veteran FIRST Tech Challenge (FTC) team from Portland, Oregon. They advanced to the FTC World Championships three times and received recognitions for their competitive robot, software programming and contribution to the local, national and international community. The team developed a fast and streamlined design and build process while ensuring software quality throughout the competition season. They also focused on outreach activities in educating young students in communities about Science, Technology, Engineering and Mathematics (STEM) through Tech Talks, community open houses, and mentoring newer robotics teams.

Bo Li works in software development and management for Technology Development Analytics and Automation at Intel. He joined Intel in 1999 after receiving Ph.D. from Georgia Institute of Technology. Since then, he has been focusing on software development, quality, performance, and testing for a variety of software applications. Outside of work, he volunteers to mentor robotics teams in the community for high school and middle school students to foster passion and knowledge in STEM.

¹ Jesuit High School, Portland, Oregon

² Beaverton Academy of Science and Engineering, Beaverton, Oregon

³ Sunset High School, Portland, Oregon

⁴ Sunset High School, Portland, Oregon

⁵ Technology Development Analytics and Technology Automation, Intel Corporation

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

1 Introduction

FIRST is an international youth organization to develop ways to inspire students in engineering and technology fields [1]. FIRST stands for "For Inspiration and Recognition of Science and Technology" and is among the broad spectrum of avenues to pursue robotics at the pre-college level and promote Science, Technology, Engineering, and Mathematics (STEM) around the world. The FIRST Tech Challenge (FTC) is designed for students in grades 7–12 to compete using a sports model. Teams are responsible for designing, building, and programming their robots to compete in a 2-team alliance format against other teams. Teams, including students, coaches, mentors and volunteers, are required to develop their strategy and build robots based on engineering principles. The ultimate goal of FTC is to reach young students with an accessible and low-cost opportunity to discover the excitement and rewards of STEM.

1.1 Background for FIRST Tech Challenge (FTC) Robot Programming

FTC game matches are played on a 12 foot by 12 foot playing field. Each match is played with 4 randomly selected teams, 2 teams per alliance. Four 18" x 18"x 18" robots must be able to navigate around each other without breaking when hit by another robot.

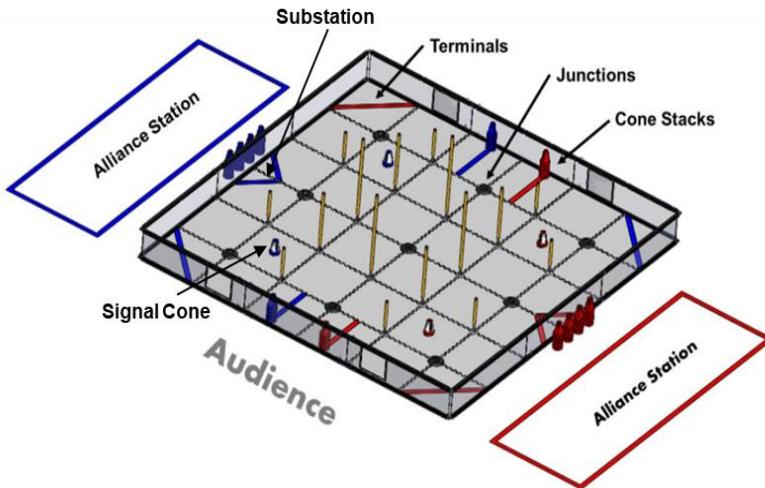


Figure 1. FTC field setup for the 2022-23 FTC season [2]

Figure 1 shows the field setup for the 2022-23 FTC season. Here is the high-level summary of 2022-2023 FTC robot game rules:

- The FTC robot game is played between two opposing alliances, Red alliance and Blue alliance.
- Each alliance is composed of two robots each built by a separate FTC team.
- The FTC robot game is composed of two phases: (1) the autonomous phase and (2) the driver controlled tele-operation phase.
- The referees at first randomize and determine the right robot parking zone # right before the match starts. The signal cone in the field will then be manually rotated so that it has the right picture on the signal cone facing toward the alliance station. Different pictures on the sides of the signal cone will match with the zone # that the Robot should park in at the end of the autonomous period.
- The goal of the game is to score the most points. Points are scored by doing the following:
 - Deliver a cone of your team's color to a junction earns 5, 4, 3, or 2 points for high, medium, low, or ground junctions, respectively.
 - 30 points for parking in the autonomous zone as determined by the signal cone.
- Teams begin in designated Stations and each robot must autonomously attempt to complete scoring tasks in the autonomous phase.

In FTC, the robot motions are controlled by a REV control hub on the robot programmed using Java. The REV control hub sends commands to different motors in order to make the robot move. The control hub also receives input from internal and external sensors. The Android device inside the control hub is programmed from a computer using Android Studio and the FTC software development kit (SDK), which adds functionality to control motors from the REV control hub. A diagram illustrating the interaction among different components of this robot system is shown in Figure 2. Both autonomous and driver control tele-operation programs are downloaded onto the robot, and then are run from the REV driver hub, connected to the control hub via a direct Wi-Fi connection. During the autonomous phase, the robot moves based solely on its autonomous program. During the driver control tele-operation phase, the REV driver hub, connected to two joysticks, transfers joystick input to the REV control hub on the robot. Thus, the drivers of the robot can use their joysticks to drive the robot. Teams use the Android software platform located both on the robot and with the driver to run their programs that control the robot. During the tele-operation period, one human player from each alliance will manually place the cone into the substation in the field one cone at a time. The robot needs to collect these cones on the field and stack them up onto the junction to score points.

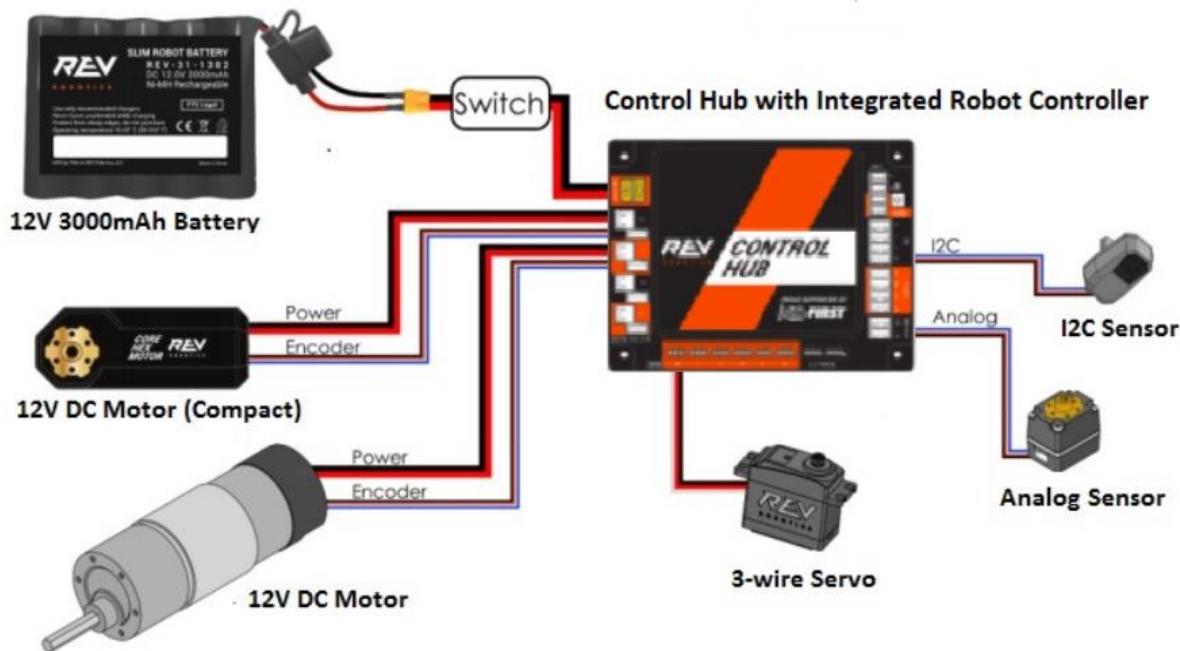


Figure 2. The control system of an FTC robot and its subsystems (e.g., external sensors, motors, power, etc.) to detect and control FTC robot [3]

1.2 Introduction to FTC Software Quality

FTC software quality is crucial due to the high demand of consistent competitive performance during FTC competitions. If the software runs into quality issues, it can significantly impact FTC competition results. The team revamped the software of the robot through advanced development and deployment methodologies leading to increased quality and high scoring capability during competitions for the FTC season.

This paper presents two main challenges in accomplishing high software quality for FTC robotics competitions. The first challenge is ensuring that the software changes can be adapted rapidly with high quality. FTC 2022-2023 season competition was released on September 10th, 2022, and the first competition was on November 12th, 2022. There was only a short two-month period to design, build, test, program and refine a competition-ready robot. To achieve this goal, the team employed the SWOT methodology, leveraged Github software storage management, and utilized CI/CD deployment methods to

meet the tight schedule. The second challenge is ensuring that the robot can utilize machine learning software to minimize the effect of the constantly changing physical world and make the right decision in different situations. The machine learning algorithms must be tested thoroughly to ensure high quality robot performance.

2 Revamp Software Quality in Android Robots

Throughout seven years of FTC experience, the team perfected a process of SWOT analysis and an industry-adapted design cycle [4] to effectively evaluate and optimize proposed solutions using the robot's software and hardware, allowing the team to achieve its maximum capability. When the complex FTC challenge is first presented to the team, each member comes up with an individual design to attempt to solve these main tasks at hand. Each design is evaluated using weighted metrics and SWOT analysis [5], based on criteria that the proposed solution must meet. Weighted metrics is a method of evaluating prototypes based on subjective criteria. Proposed solutions are assigned a numerical value based on this evaluation, and then ranked based on their relative sums. SWOT analysis evaluates the Strengths, Weaknesses, Opportunities, and Threats of a design. Strengths are internal factors that provide certain advantages to the robot's design. Weaknesses are internal factors that may limit certain aspects of the robot's performance. Opportunities are external factors that can be utilized to enhance the robot performance, and Threats are external factors that pose challenges to the capability of the robot. Utilizing weighted metrics with SWOT analysis at the beginning of the FTC season allows for preliminary robot designs to be synthesized. The result is a metaphorical uncut diamond; effective but unoptimized. Throughout the season, the team's continuous iterations with the industry-adapted design cycle allows for the robot to be optimized in order to accomplish the peak performance.

While utilizing the SWOT analysis model, the team employed the industry adapted design cycle consisting of four steps [6] — brainstorming, prototyping, fabricating, and testing (See Figure 3). During the brainstorming phase, the team focuses on creating criterion for the mechanism and laying out the initial thoughts as whiteboard drawings. The team scrutinized current bottlenecks on the robot and conceptualized more optimal solutions. Next, to test the validity of the proposed solutions, the team completed prototyping separately from the existing robot to ensure that a functioning robot is still intact in case the solution fails. The prototyping stage focuses on using PTC Creo and Fusion 360 CAD software to better visualize new or altered mechanisms, translating them into initial prototypes. The visualization then enables the team to fabricate the components on CNC and vertical milling machine tools. During the fabricating step, the team manufactured modules using industry-grade materials and machines with high precision. Lastly, to ensure the effectiveness of the solution, it is extensively tested in a variety of scenarios to ensure its efficacy in competitions. Through this process, SWOT analysis is continually applied to the robot's components to ensure that they are always at optimum performance. Similar to the weighted metrics for robot components, the robot's software and components are also assessed using identified criteria in the evaluation phase. This evaluation provides insights into the current state of the robot and helps identify areas for improvement. Then, based on the evaluation results, optimization techniques are applied to enhance the robot's performance. This evaluation-optimization process aims to maximize strengths, mitigate weaknesses, capitalize on opportunities, and address threats located in the SWOT analysis to revamp software quality.

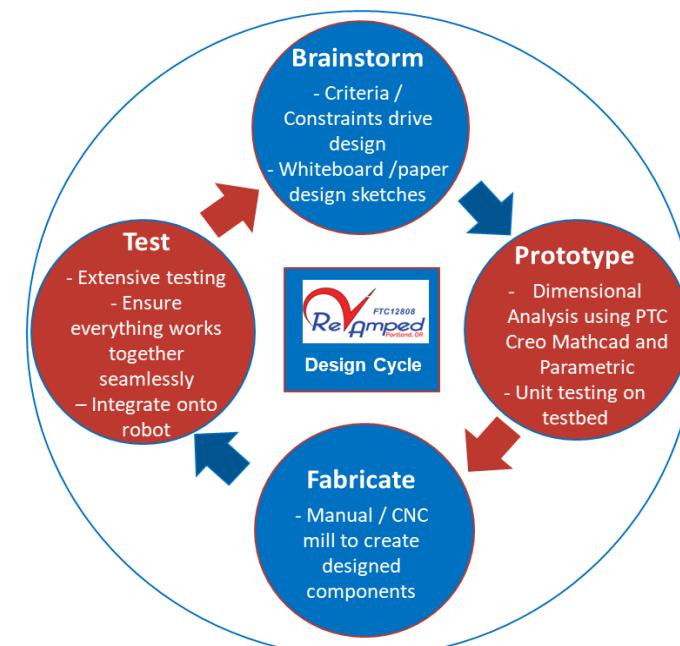


Figure 3. The Revamped industry adapted design cycle

3 Case Study: Advanced Development and Deployment Methodologies to Revamp Software Quality

The Revamped Design Cycle allows the team to make effective changes to the robot software and hardware in a fast-paced competition environment, allowing for on-field testing at the competition to ensure the high quality. After competition matches, the team further iterates upon the robot using this same cycle to produce high quality robot components while ensuring it fits with overall robot mechanisms.

3.1 Case Study: Robot Claw Optimization to Interact w/ External Physical World

During the 2022-2023 season, one critical robot component was the design of the robot claw. The claw was integral to the robot's scoring capabilities as it was the component that directly interacted with the external physical world to reliably grasp a cone and drop it off onto the junction. Thus the claw came under heavy scrutiny as the team continued to iterate and optimize the robot. At the beginning, the robot claw was a simple two-pronged grasping mechanism (Figure 4a) barely capable of retaining the scoring element, a circular plastic cone (Figure 4b).

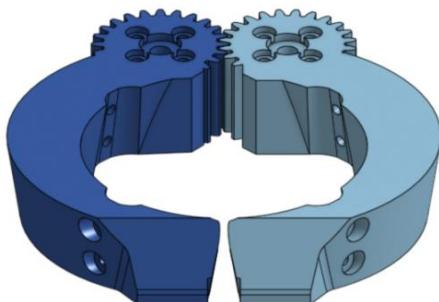


Figure 4. (a) The team's preliminary claw design #1 and (b) the aforementioned cone

The ability for the robot claw to hold a cone securely for a long time was very crucial to the robot's performance. After the evaluation-optimization process, the team realized the need for a friction generator lining the inside of the claw to create sufficient friction with the cone which would allow for an easier retention of it. Design #2 quickly emerged with more force to hold the cone (Figure 5a). Thus, the team began to use SWOT analysis and weighted metrics to evaluate each design to ascertain which design would hold more promise. The criteria that each design was evaluated on was simplicity, effectiveness, and size, as well as ease of maintenance and replacement for the claw design option. The weight value is determined by the magnitude of the impact and the probability of occurrences for these SWOT analysis line items in Strengths, Weaknesses, Opportunities and Threats. Eventually, the team selected the second design based on SWOT analysis results (see Table 1 for SWOT analysis process data). This design still had flaws and need to improve to securely hold the cone. The team iterated on the claw design to finally arrive at the current design (see Figure 5a and Figure 5b). Figure 6 shows that the claw mechanism is a critical part of the robot during FTC competitions.

Table 1: SWOT Analysis for the New Claw Design (#2) with Weighted Metrics

Strengths		Weight	Weaknesses		Weight
	Software can control effectively to grip, hold and release the cone	0.25		Securely hold cone during the robot movement in matches	0.2
	Simple claw design and easy for the software to control	0.1		More possible autonomous position control for the robot claw	0.1
	Right size to fit into the overall robot	0.1		Extra steps & complexity to build the claw	0.05
Opportunities		Weight	Threats		Weight
	Easy attachment of sensors to improve the accuracy of the claw movement	0.3		Need to adjust the claw shape for more effective software control when hit by another robot	0.15
	Adjust the physical connection with the robot to simplify install	0.1		Need to change the software to cover from sensor failures	0.15



(a) The CAD of the claw design #2



(b) The current claw

Figure 5. The CAD of the claw and the current claw gripping a cone

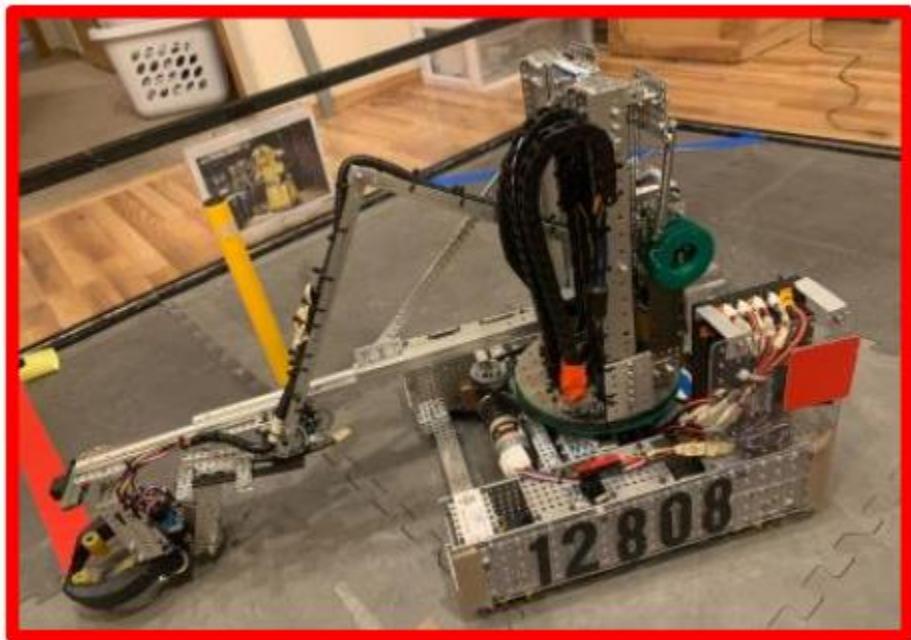
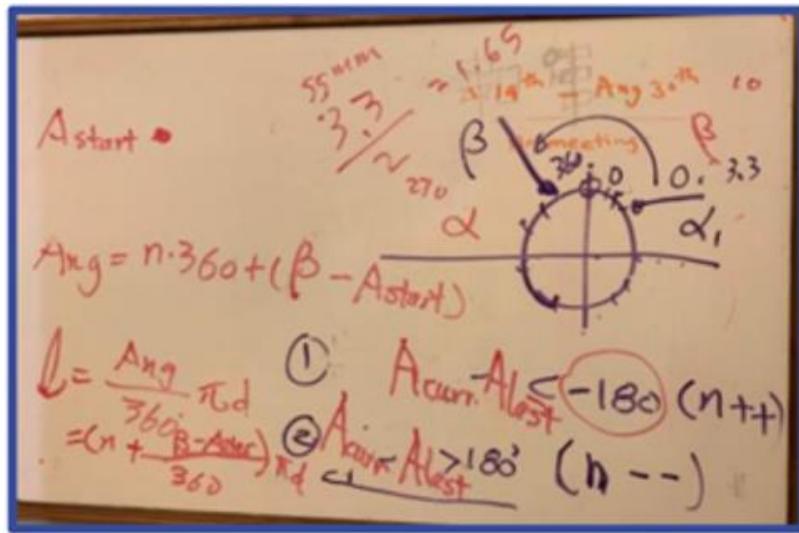


Figure 6. The Revamped Robotics FTC competition robot for FTC 2022-2023 season

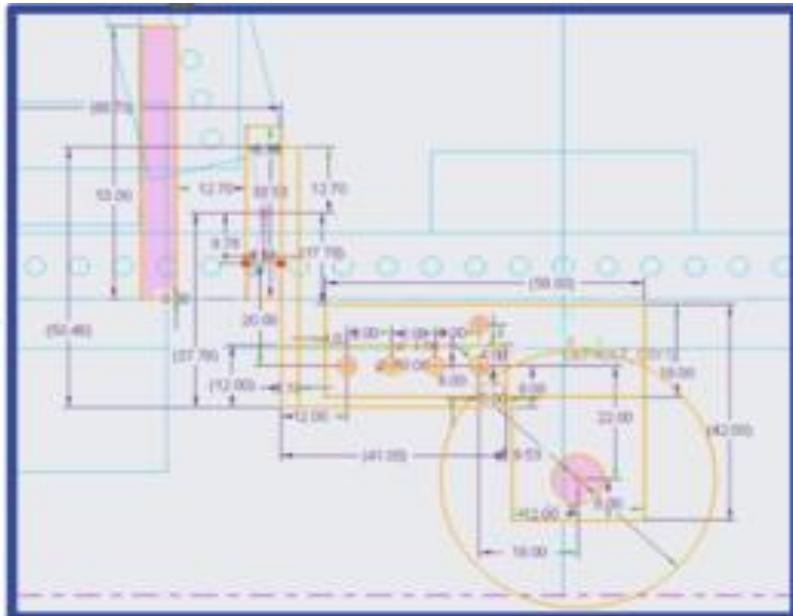
3.2 Case Study: FTC Robot Odometry Pod System Software Quality Improvement

Another case study can be constructed from the team's work on the odometry pod system, as it includes both software and hardware synthesis. The team at first identified the problem where the mecanum drive needed for accurate pathfinding in the field had reduced friction and higher drive error than was acceptable. Utilizing an external measurement of the robot position allows the robot to move with accuracy in the autonomous period so that the team can complete all autonomous goals with consistency. The team brainstormed spring-loaded carriages for constant contacts with the field and centered pods along the major axes of the robot for the highest accuracy. The system should be within 2% error on the x and y axes and must allow the sensor fusion with the external Nav X IMU for constant position tracking regardless of the robot heading direction.

The team then started the iteration of development and fabrication processes. The robot movement path and the corresponding robot wheel movement model were carefully calculated (See Figure 7a) to determine the distance and direction by converting encoder shaft outputs to the arc length traveled by a wheel. The team then CAD'ed the design (see Figure 7b) and used MA3 absolute encoders, iterating from a 1:1 to a 3:1 gear ratio and the lateral spring tensioning to prevent the revolution skipping by maintaining constant contacts with the field. At this point, the team also switched to 3 encoders. However, the testing revealed inaccuracies and complexities in fine-tuning and programming the absolute encoders, so the team switched to US Digital E4T encoders. While absolute encoders require a function to constantly convert the voltage into the distance, only a simple linear regression is needed to program the E4T encoders. Specifically, three E4T pods (2 in the y-axis and 1 in the x-axis) were used for a better spring tensioning and a more compact design. Gearing down is unnecessary because the encoder measures at ticks, resulting in an ultra-compact design. The team also switched the encoder wheel to smaller and thicker plastic wheels to address movement inconsistency issues (Figure 7c and 7d).



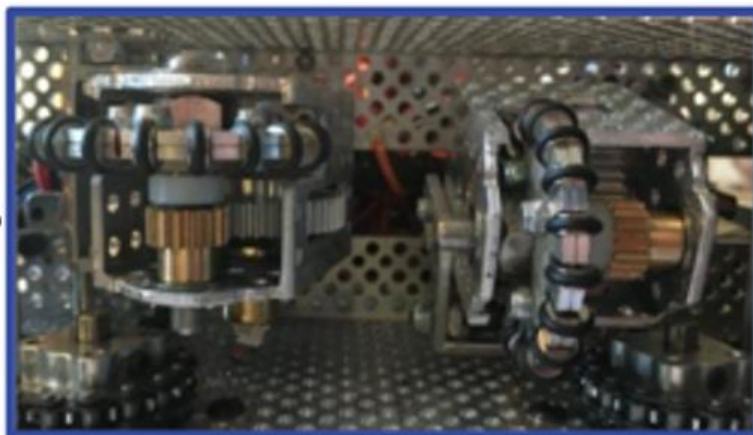
(a) Brainstorming on white board



(b) CAD Design



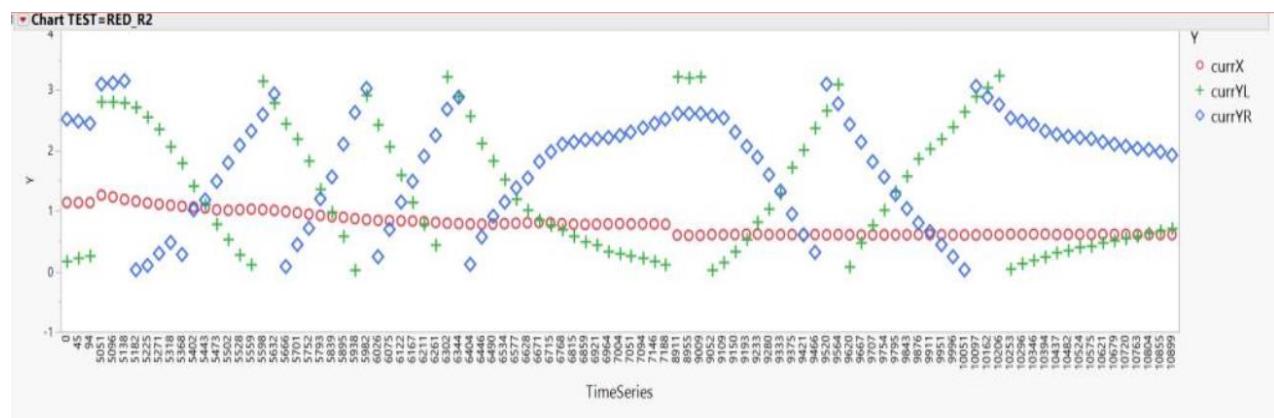
(c) Prototyping



(d) Fabrication

Figure 7. Odometry (a) Brainstorming (b) CAD Design (c) Prototyping and (d) Fabrication phases

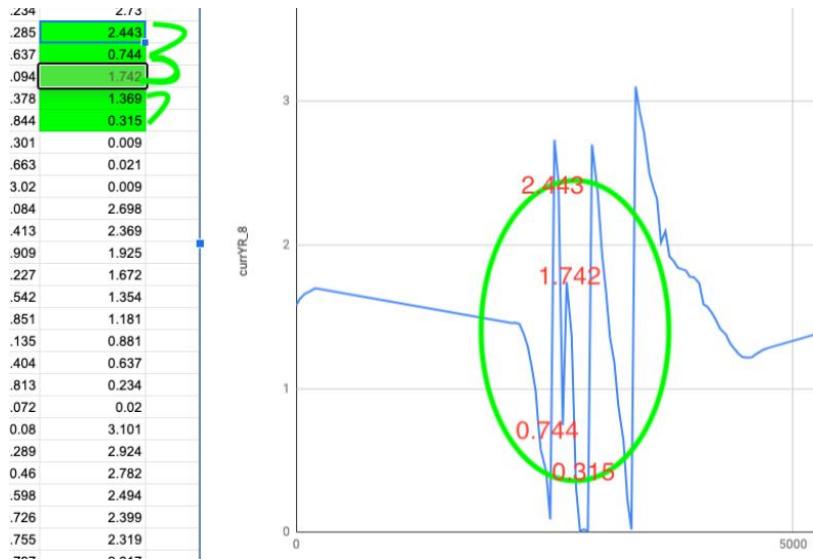
The team did thorough testing and collected data on voltage jumps to expose errors in the autonomous phase and made debugging very efficient. The data gave the team reasons to switch from absolute MA3 to E4T encoders. To visualize errors, the team logged the reading and then parsed certain values of interest to show normal trends and discrepancies (Figure 8a) where CurrX indicated the robot side move and CurrYL/ CurrYR indicated the reading on the left and the right hands side along the robot heading direction. The CurrYL and CurrYR in Figure 8a showed jumps in the reading which indicates the discrepancy issue for the robot movement. Figure 8b shows a smooth auto run with no sudden jumps for the robot side move (X) and the robot heading moving direction (YL and YR) readings, as compared to the real encoder with voltage jumps shown in Figure 8c due to the real field friction variation, etc. The example showcased the use of the Revamped Design Cycle promotes quick and iterative development process for the software quality improvement.



(a) Odometry testing data trends and discrepancies



(b) Odometry testing smooth autonomous run despite voltage jumps



(c) Odometry testing encoder voltage jumps

Figure 8. FTC robot odometry testing results

3.3 TensorFlow Machine Learning for Autonomous Phases

The team has the autonomous software program which can pick up and drop off cones to the high junctions and then park at the right zone. The diagram below shows the path that the robot takes, which consistently accomplishes high scores during the autonomous period.

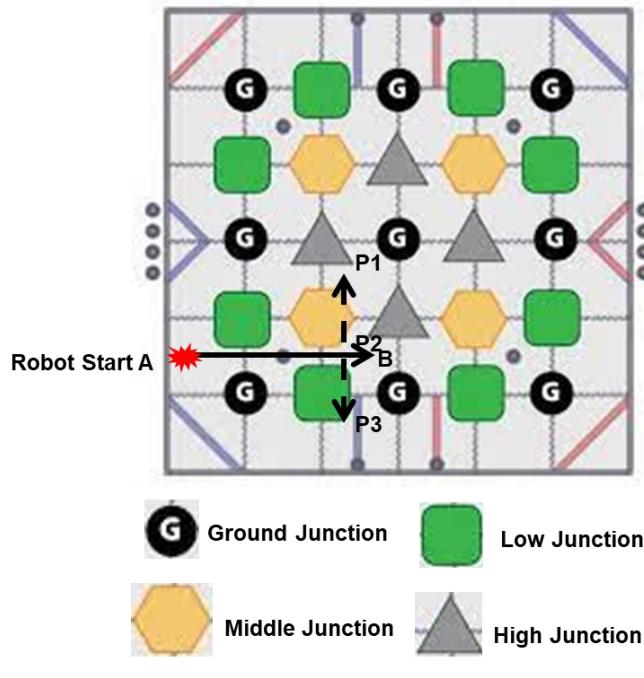


Figure 9. Illustration of autonomous robot movements

Table 2: FTC Challenge Autonomous Robot Action Sequence Example

Step	Description
1	Robot starts from 1 of 4 corner position in the field
2	Camera captures the picture on the signal cone and the robot program analyzes it
3	Determines the final stop zone using TensorFlow
4	Moves to the position near the high junction (pole)
5	Detects and picks up the cone from the pile next to the wall, and then drop off to the high junction
6	Repeat Step 5 to pick more cones with the adjusted height and drop off the cone to the high junction
7	Adjust and then move to left or right to the right parking zone determined by Step 3

One important part of the autonomous objectives is to identify the zone that the robot should park in at the end of the autonomous period accurately. Last season, the team used OpenCV to detect the team element which is a key aspect of the autonomous period. While it is simple to use, the team found that it was less reliable under different lighting conditions. This year, the team adopted the TensorFlow AI model provided by FTC and trained it on customized signal cone patterns. The TensorFlow Machine Learning process [7] utilizes image recognition (Vuforia). After the autonomous phase starts, the robot camera activates Vuforia, allowing the TensorFlow image recognition to initialize and scan the team sleeve picture on the signal cone. The Machine Learning system has been trained to recognize the pattern on the signal cone by being shown numerous pictures of this object from multiple angles, allowing for a complete perspective to be produced. Once objects are visually scanned, it compares previously learned pictures to the real-life situation. At run time, the USB web-camera on the robot captures a real-time video stream which is fed into the TensorFlow inference model to detect the randomized signal. To increase reliability, the video screen is cropped to minimize the background noise. This setup can reliably detect all three patterns, even in low lighting conditions and with a noisy background. Additionally, every iteration of the scanned objects is added to a cloud recognition database to enhance future accuracy of scanning. In some scenarios, some pictures are related to the object with different conditions, so the TensorFlow can recognize the pattern with only a part of the picture. Based on previous testing, the team was able to narrow down the expected pattern and determine the right zone to land the robot reliably.



Figure 10. Illustration of signal cone sleeve pattern detection using TensorFlow machine learning

A high scoring autonomous robot needs robust sensors and a good architecture for easy debugging and readability. To complete every autonomous task, The team structures the code following a trajectory and movement sequence that allows the team to orchestrate complex behaviors. The software runs multiple tasks concurrently allowing the team to plan more time efficient paths and dynamically generate movements

based on the robot status. For instance, if the distance and color sensors detect a junction or a stack of cones in the proximity, the software can control the turret and slides on the fly to grab a cone and score it on a junction. This prevents the autonomous failure given the trajectories are based on the sensor inputs rather than pre-programmed instructions. With the advanced troubleshooting practice and methodology, the robot can detect the right pattern accurately (See Figure 11)

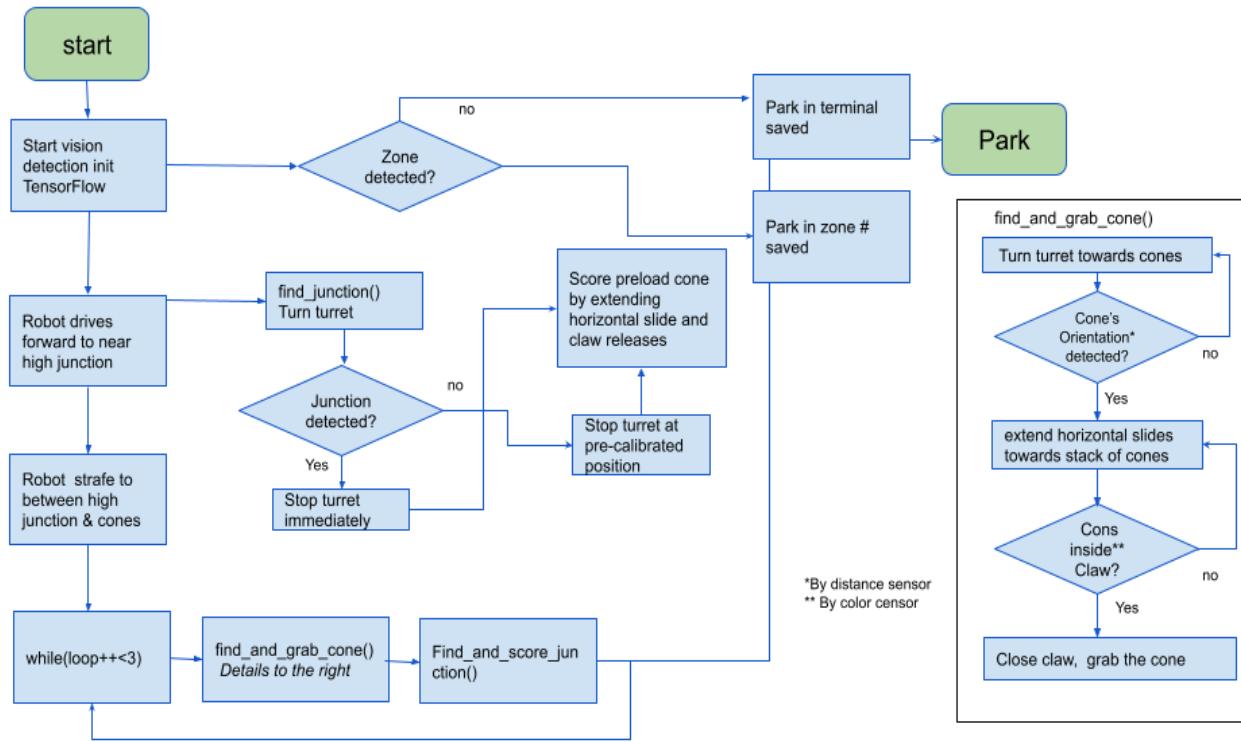


Figure 11. Autonomous program flowchart

3.4 Github and CI/CD Software Deployment Methodology to Accelerate Software Quality Improvement Process

The team has been using GitHub as a web-based version control and collaboration platform for the software development. Given the agile software development cycle in the team, GitHub allows team members to collaborate on a project more effectively by providing tools for managing possibly conflicting changes from multiple team members. The team changes, adapts and improves software from its public repositories. The team grouped the code into “Autonomous”, “TeleOp”, “test”, and “utils” code repositories correspondingly. In addition, the team also maintained selected component codes that were tested from earlier seasons and can be reused for this season (e.g., roadrunner, etc.)

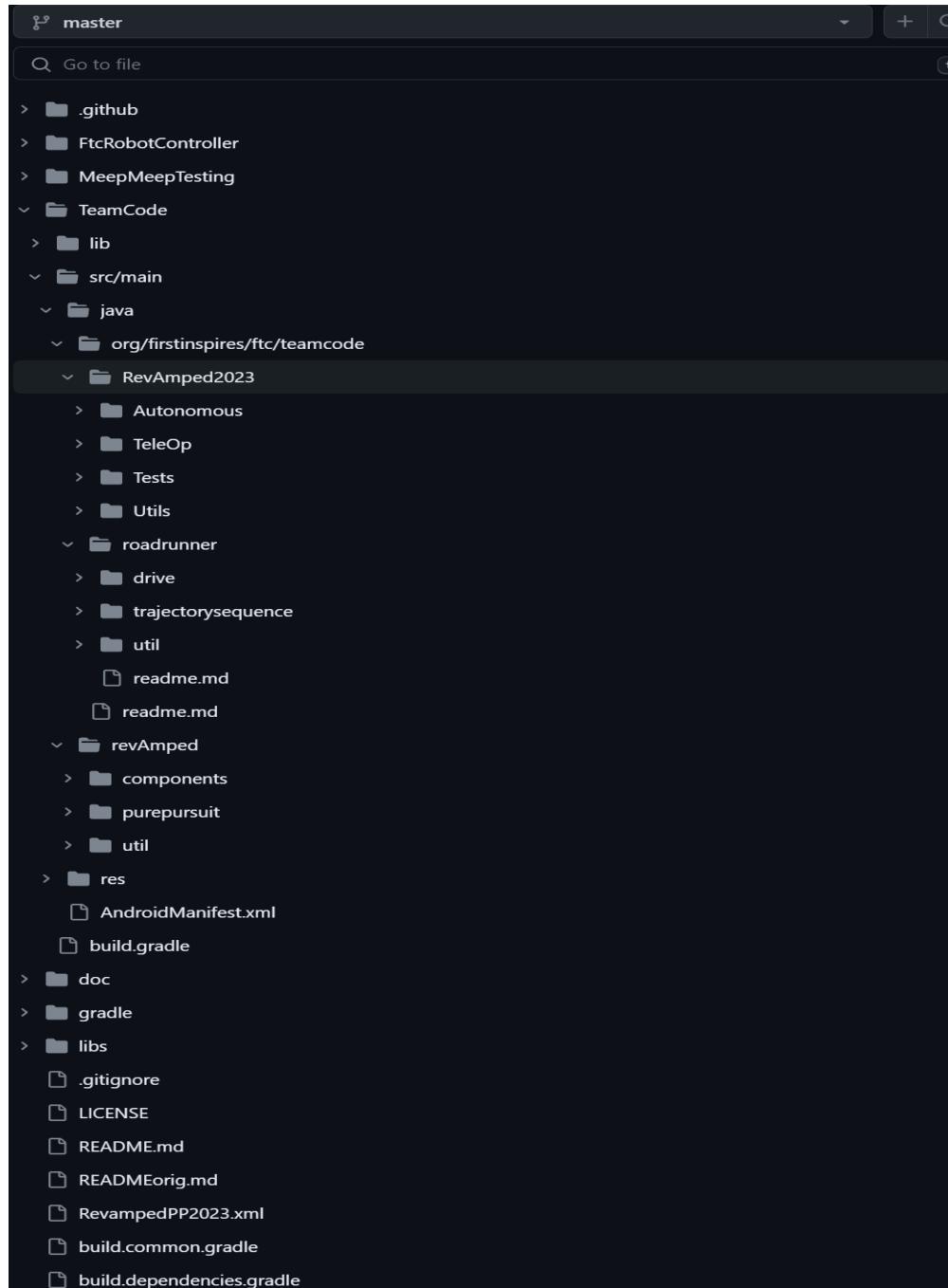


Figure 12. Revamp Robotics team GitHub code storage and management model screen shot

The team employed the Github technology below to set up different branches for different developers to ensure developers make modifications without affecting the original master code. After reviewing modifications, the original owner would “Add” the modifications into the repository on a very frequent basis. The team uses “Commit” to commit an individual change where commits are retained and interleaved onto the main project appropriately. The team member can accept the modifications and merge them with the master repository, enabling a flexible and fast revision process that spans the entire FTC season. Furthermore, the team uses “Push” to send code from a local copy to the online repository.

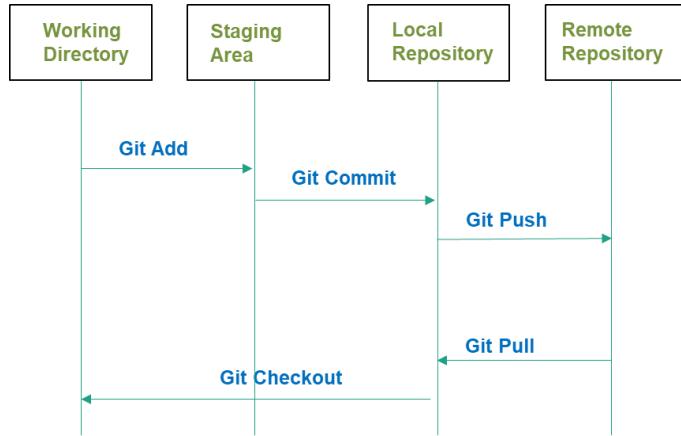


Figure 13. Flowchart for Github code storage and management

Through this journey to revamp the software quality, the team members were able to follow each other, receive updates for specific open source projects (e.g., autonomous project, etc.), and communicate learnings effectively. Github enables the team to work effectively when different team members work on the code at different times with the flawless transfer of the developed code.

The team also utilized GitHub software storage management to support integrated development environments and continuous integrated development (CI/CD) with more security and auditability [8]. By automating CI/CD throughout the entire software development lifecycle in FTC season, the team was able to develop higher quality code at a faster pace. CI/CD brought different team members together to deploy software to the robot promptly as soon as the software code change was completed and ready. As the speed of the software application to the Robot becomes a key to how FTC teams differentiate, the rate at which the code can be released has become a competitive differentiator. Revamped Robotics team was able to accomplish ~15-33% new feature implementation acceleration after employing Github software storage management, and CI/CD development and deployment methodologies (See Table 3).

Table 3 Github and CI/CD Software Development and Deployment Acceleration

Categories	Small Size Feature (Weeks)	Medium Size Feature (Weeks)	Large Size Feature (Weeks)
Legacy Deployment	0.3	1	2
Github & CI/CD	0.2	0.8	1.7
Acceleration	33%	20%	15%

4 Summary

This paper presented Revamped Robotic team's approaches in improving software quality in Android robots that integrate robot software and physical worlds together. Case studies were shared in SWOT analysis, software deep learning and Github/CI/CD in Android robots to revamp software quality and accomplish the top robot performance in FTC. The SWOT analysis is used to effectively design, prototype, fabricate and test solutions. Furthermore, the paper presented the method to improve the software quality through deep learning TensorFlow to minimize dependencies on the external physical sensor reading and the mechanism to mount the sensor. In addition, Github storage management and CI/CD process showed ~15-33% FTC feature implementation acceleration. All of these methods can be utilized in industries to improve software quality at an accelerated pace when handling physical world uncertainties and complexities.

Acknowledgments

Authors would like to thank paper reviewers, Cameron Kilgore and Nick Bonnichsen, for their invaluable assistance and discussions on software quality. Furthermore, authors would like to thank the Ramped Robotics team coach Zhunquin Wang for his excellent guidance throughout our multiple year FTC journey. In addition, authors would like to thank Oregon Robotics Tournament & Outreach Program (ORTOP) whose planning and assistance are very crucial for the success of FTC events in Oregon.

References

- [1] <http://www.usfirst.org/> (accessed May 31, 2023).
- [2] <https://firstinspires01.blob.core.windows.net/first-energize-ftc/game-manual-part-2-traditional.pdf> (accessed May 31, 2023).
- [3] https://github.com/ftctechnh/ftc_app/wiki/The-FTC-Control-System (accessed July 23, 2023).
- [4] <https://www.aha.io/roadmapping/guide/it-templates/swot> (accessed May 31, 2023).
- [5] Escalona, M.J., Nora Koch and Gustavo Rossi. 2022. "A Quantitative SWOT-TOWS Analysis for the Adoption of Model-Based Software Engineering, Journal of Object Technology (January 2022). 21(4):4:1.
- [6] <https://mitsloan.mit.edu/ideas-made-to-matter/design-thinking-explained> (accessed May 31, 2023).
- [7] <https://www.tensorflow.org/tutorials> (accessed May 31, 2023).
- [8] <https://resources.github.com/ci-cd/> (accessed May 31, 2023).

Break Silos, Build Chapters

Ashok Kumar. S

ashok.jan31@gmail.com

Abstract

As enterprises mature in their agile journey and move towards becoming more distributed cross-functional teams, testing that was once done by a single central team has transformed into a distributed capability where testers are embedded within the teams.

While this has many benefits, some potential challenges surface. When testers disperse across multiple teams they can start to feel isolated and collaboration with peers diminishes.

Duplication of tools and frameworks may emerge, transparency decreases and ways of working and practices become highly siloed and inconsistent.

To solve the lack of connection, inspiration, and knowledge-sharing, one solution is to work with testing chapters to connect testers into a shared context - a community. They provide a platform to share and organize skills and expertise as well as serve as a safe community of like-minded people to share new ideas, test experimentations, and receive feedback from peers. The term chapters in some enterprises is also called Community of Practice (COP) and for clarity of this paper, we will continue with the terminology of Chapters.

I will share our learnings from building these testing chapters, the opportunities they can provide to collaborate across teams, and the value they create. I will also cover challenges we faced, such as:

- The mindset shift needs to happen within the teams as well as with the stakeholders connected to them.
- The false notion is that external chapters are not required or valuable since agile is about autonomous teams.
- How to plan and account for time spent on chapter activities and who should take the lead on organizing and coordinating them.
- How to bring these learnings and best practices back into the teams.

I'll share how we overcame them, used gamification to keep engagement high, and made the chapters fit into an enterprise-wide agile team structure.

Biography

Ashok Kumar is the head of Quality Assurance & Engineering for Transunion Africa. In this position, he leads Testing and Quality Assurance for the Africa region which enables Transunion to roll out innovative credit bureau solutions to our African customers at scale and with high quality. Ashok works with multiple agile teams within the company and coaches them on the quality mindset. Ashok has nearly 2 decades of experience in testing across multiple industries such as financial services, retail, & credit bureau.

Copyright Ashok Kumar 25082023

Introduction

In today's rapidly evolving software development landscape, agile methodologies have emerged as a game-changer, revolutionizing the way teams collaborate and deliver high-quality software. At the core of Agile lies the principle of continuous improvement, emphasizing iterative development, flexibility, and adaptability. Cross-functional agile teams bring value to the business in terms of faster time to market and self-contained teams. However, one crucial aspect that often demands careful attention is the inability of the team members in continuing to collaborate, share and improve professional expertise within their skill groups.

In this paper, we delve into the effectiveness of creating chapters for testing professionals within an Agile Enterprise and explore how they enhance the efficiency of testers embedded within a highly distributed and scalable Agile organization. By examining the benefits, challenges, and best practices associated with testing chapters, we aim to shed light on how this approach can elevate the quality and velocity of software development. Throughout the article, we will explore the primary reasons why testing chapters are gaining traction in agile organizations. We will discuss how these chapters empower teams to build a quality culture that is focused on collaboration and implementing best-in-class quality engineering practices across the agile silos and standardizing these practices within the teams.

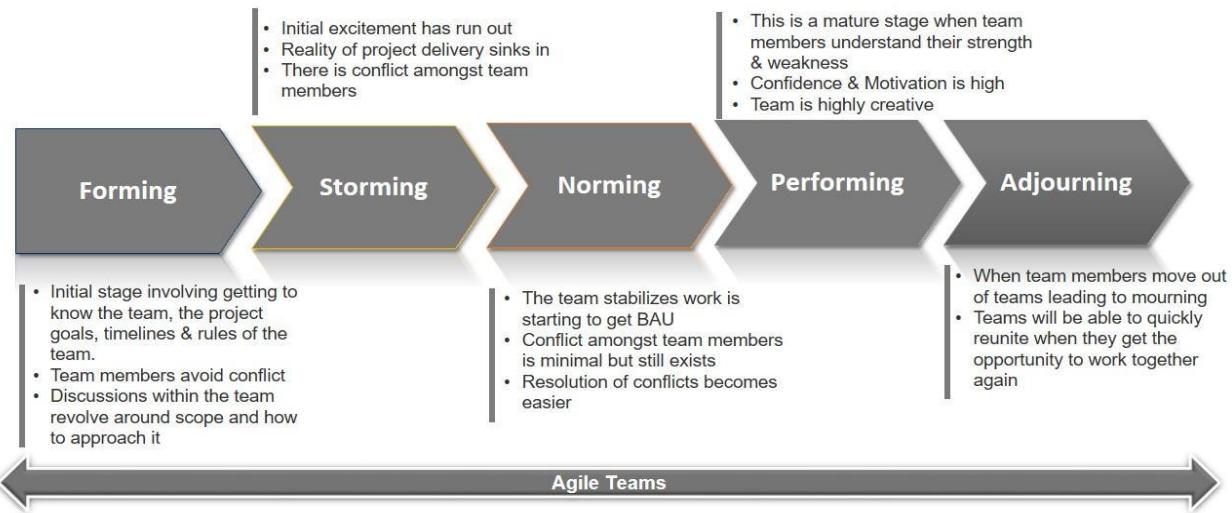
Additionally, we will address the challenges that Agile teams may encounter while implementing testing chapters, such as scaling across large teams, balancing individual responsibilities with collective goals, integrating testing chapters into existing Agile frameworks, and planning for the time required to build, scale and operate these testing chapters.

By providing insights, practical advice, and real-world examples, this paper aims to equip agile practitioners, project managers, and quality assurance professionals with the knowledge needed to leverage testing chapters effectively. Whether you are looking to establish a testing chapter or seeking to optimize an existing one, this exploration will serve as a valuable guide in harnessing the power of testing chapters to drive superior software quality and elevate the overall performance of agile teams.

1 Agile team Structure

An agile team is a cross-functional team that is self-contained with all the skills required to deliver the product's next iteration. Chapters are a team of people with similar skills or job roles or professional interests.

A group of people organized as an agile team or chapter typically goes through different stages of development and transformation. Agile evangelists often draw inspiration from Tuchman's study on "stages of group development".



Forming

The initial stage of team building is when the communication between the teams mostly revolves around scoping, introduction, timelines & challenges of meeting those timelines. Team members avoid conflict at this stage as they are new to the team and most likely will try to fit in.

Storming

This is the second and most troublesome stage of team building where teams generally tend to conflict and compete with each other as individual personalities emerge. Productivity dips as there is a lot of energy spent on conflict resolution. Team members tend to disagree on goals, and subgroups are formed under other strong personalities.

Norming

In this stage, consensus begins to evolve within the team and there is more clarity about who the leader of the team is. There is a sense of cohesion and unity among team members. Team performance increases gradually at this stage as the members learn to cooperate and work as a team. The teams' overall productivity is also on the rise considering they can focus on more productive work rather than resolving conflicts.

Performing

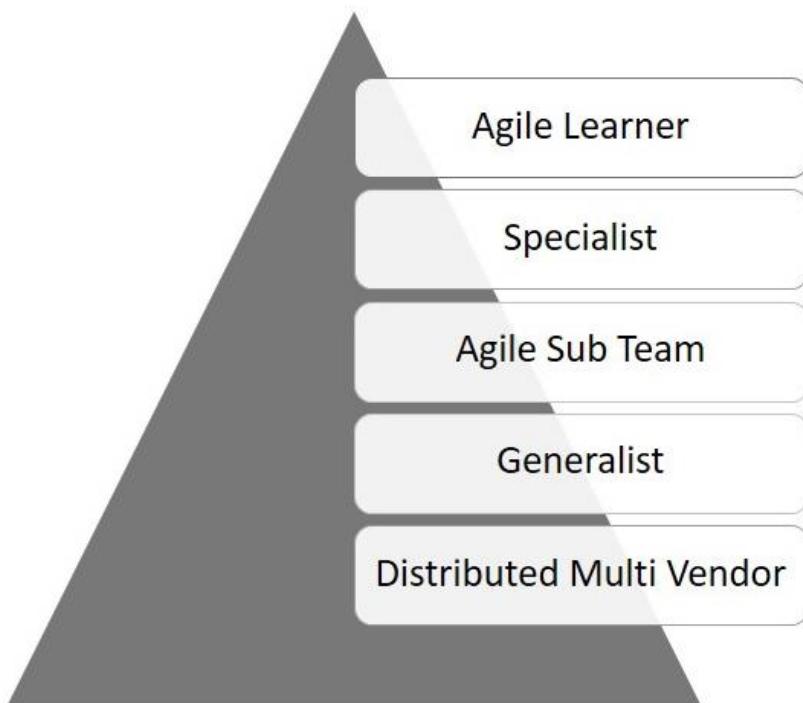
This is a mature stage where the teams reach the highest level of collaboration and cohesion amongst themselves. There is a clear and stable structure and members are aligned to the joint team goal. Conflicts still emerge however they get resolved more quickly and constructively.

Adjourning

In this stage, most of the team's goals have been accomplished. The emphasis is on closing out any final tasks that need to be completed and documenting efforts and results. As the work diminishes, individual team members move to other teams. There may be regret as the teams disband however it is expected that teams take the time to celebrate their achievements – having positive shared experiences will make it easier, if you work with some of the same people again in the future.

1.1 Transformation over time

The structure of agile teams themselves has transformed and evolved over time as a response to the business need to build high-quality software products.



1.1.1 Agile Learner

This is the initial stage of team structure agile teams typically start with. When a team transitions to Agile for the first time they need a lot of support and guidance. This structure of teams would require an agile coach to be dedicated to the teams to help them navigate the learning curve. Teams at this stage typically follow a combination of waterfall and agile practices to enable them to navigate the transition more easily.

1.1.2 Specialist Teams

This is the second stage of agile teams' transformation. They are now a team of specialists where every team member is an expert in his/her specific skills. Although teams following this structure usually deliver high-quality software they often struggle with delivering high-quality work within the framework of the sprint deadline. Also, the other challenge with this structure of teams is that team members often sit idle waiting for their specialist task.

1.1.3 Agile Sub Team

These are team structures that are made up of smaller units of teams that are focused on specific skills. Although this defeats the purpose of having cross-functional teams, some merit in this is each sub-team will have responsibility over a specific area of the application. There is a drawback in terms of too much

time spent in handover work from one sub-team to another, resulting in the overall project timeline being extended.

1.1.4 Generalist

The team consists of members that can pick up any task as they are cross-skilled and work very well together within their team. The members who are part of this structure of teams are usually very passionate and are driven by the opportunity to learn multiple skills. Also, this structure of teams is usually small in terms of size and would have team members working together in this fashion for extended periods resulting in high levels of mutual trust and cohesiveness.

1.1.5 Distributed Multi-Vendor

This is a new structure of agile teams where team members are distributed across multiple geographic locations and even come from different companies. They are aligned on the common sprint goal and work towards delivering it at the end of every iteration. This structure is complex in implementation considering that team members are not only remote but also from different parent companies. There can be an initial challenge in getting these disparate team members to be aligned on the common goal however this model is the most cost-effective way of implementing an agile multi-vendor team model.

The other challenge with such a distributed model is that the more distributed the squads or teams become the more they start to duplicate practices, frameworks & tools. An essential transformation of team structures to overcome this challenge is the formation of chapters that draw people with common professional interests into a group that can foster the standardization of practices across these distributed multi-vendor agile teams.

2 Cross-functional teams boon or bane

Cross-functional teams can be both a boon and a bane, depending on various factors and how it is implemented and managed. Let's explore both perspectives:

Boon

- **Diverse Expertise:** Cross-functional teams bring together individuals with diverse skill sets and backgrounds. This can lead to a broader range of perspectives, ideas, and problem-solving approaches, enhancing creativity and innovation within the team.
- **Self-contained teams:** Since cross-functional teams have diverse skills within the team there is very little dependency on external people which makes the decision-making process much faster.
- **Holistic problem-solving:** With members from different functions or areas, cross-functional teams can address complex problems from multiple angles. This holistic approach increases the likelihood of identifying comprehensive solutions that consider various aspects of the issue.

Bane

- **Siloed:** The more cross-functional teams are, the more they tend to become siloed. This results in duplication of best practices across teams
- **Skill Stagnation:** The drive towards nurturing more generalist talent within agile teams can result in skill stagnation whereby team members within the agile team who were once specialists now strive to move away from their specialization and become more generic professionals.
- **Lack of Standardization:** Cross-Functional teams give their team members a false sense of comfort in terms of the practices and processes they adopt. A direct outcome of the siloed manner in which cross-functional teams operate is that they lack a platform to collaborate, share and learn best practices with other team members within their skill area working across multiple agile silos.

Cross-functional teams are a key aspect of any agile structure and it becomes necessary for us to find solutions to the challenges around teams becoming siloed and the lack of standardization of best practices.

3 Enabling distributed testers through Chapters

When testers disperse across multiple cross-functional teams, they can feel isolated and collaboration with peers diminishes. Duplication of tools and frameworks may emerge, transparency decreases and ways of working and practices become highly siloed and inconsistent. A solution to improve knowledge sharing and standardized testing best practices across multiple agile teams is to form testing chapters within the agile structural framework.

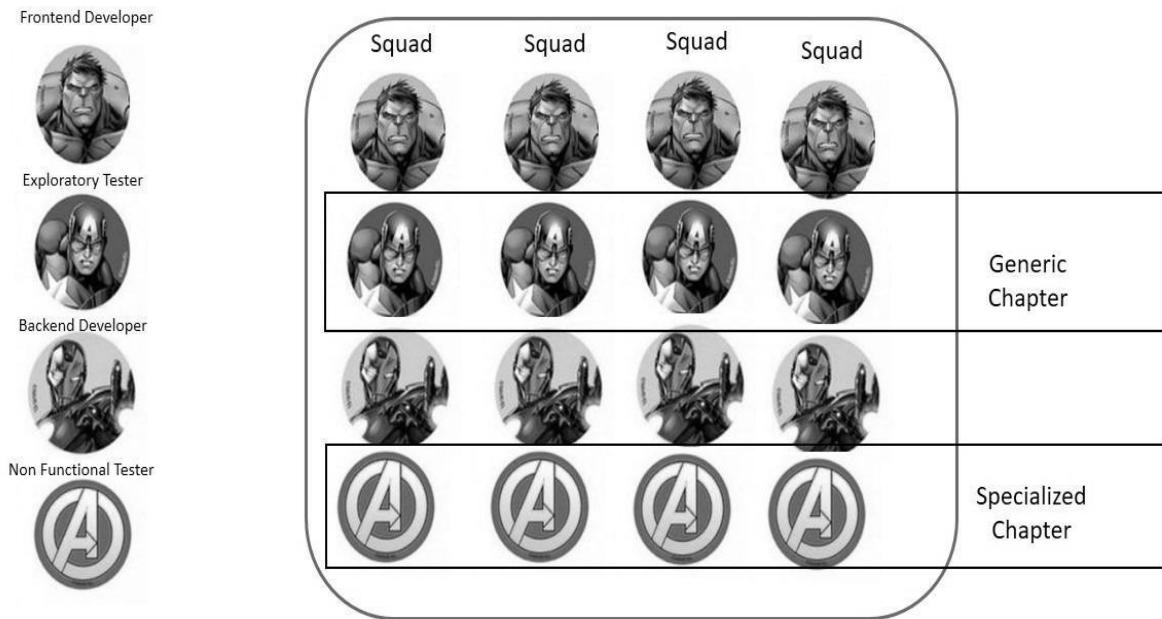
A chapter is an open group of people working for the company (including vendors) who have the same job role or are interested in the same topics supported by the chapter. The chapter can be based on expertise or based on a topic. Some chapters are officially established, and some might appear naturally as more people are interested in the solution to the same problem. Chapters are not limited to an individual project or program but are more of an amalgamation of similar skilled professionals across different projects/programs/agile teams/squads. Chapters can in certain organizations be termed as a community of practice (COP) which is again a community of similar skilled professionals practicing their skill. For the clarity of this technical paper, we will be using the term Chapter and in certain contexts can be similar to how a Community of Practice (COP) operates.

These chapters typically exist outside of the individual cross-functional teams but they are within the agile boundary of the enterprise. Testing chapters are a cohort of testers drawn from individual teams. These chapters serve as a safe sandbox where testers can practice and collaborate on their specialist skills while still continuing to wear the generalist hat within their scrum teams.

Chapters also act as incubation centers where testers can share and experiment with new ideas, and get feedback from other chapter members. Testing Chapters can also be an effective tool to standardize best practices of testing, cascading them across the different agile teams. This ensures that tools, processes, and frameworks used for testing within the teams don't get duplicated. There is also the opportunity to save in terms of cost and effort for the organization by ensuring reuse and optimization.

4 Testing Chapters our perspective

There are many categories of chapters that can be formed, in this section I will discuss on a high level what each of these are and in what situations can they be useful.



4.1 Generic chapter

Chapters can also be formed to be more generic in nature. This can be an overall testing chapter that includes team members performing various skills encompassing automation, performance, and functional testing skills. Such generic chapters can help with the cross-skilling of team members across multiple specializations within Quality Engineering. These generic chapters can be an effective lever for the optimization of skills across testing.

4.2 Specialized Chapter

These are chapters of testing that are focused on a specialized skill within the testing domain. An example of such a chapter is the one dedicated to performance testing where performance testers across the various scrum teams collaborate and share their expertise. The advantage of having such specialized chapters is that it helps in building a knowledge repository and best practices for specialized skills. Specialized chapters can also be a sandbox where specialist skills are groomed and nurtured more effectively considering that within agile teams there is an emphasis on generic skills.

4.3 Internal Chapter

Internal chapters are groups formed with team members belonging to the same company internally. These internal chapters help with more flexible ways of data sharing among the different chapter members as they all belong to the same organization. However, the drawback with this form of internal chapters is that they lack the outside perspective of solutions to problems being dealt with by the various chapter members.

4.4 External chapter

External chapters are groups formed with team members who are external and belong to different organizations. These chapters are effective in implementing unique solutions to challenges as they bring varied perspectives from people inside and outside of the organizations. External chapters also can benefit from the fact that they can draw in experts from the field not limited to the company to which the experts are attached in a professional setting.

5 Chapter Leader

A key role in chapters responsible for forming, organizing, and running chapters is the chapter leader. Some of the key responsibilities of this role are below

- To establish a lively chapter
- To invite people to be part of the chapter
- Share outcomes of the community with management, other chapters, squads, and people,
- To coordinate and moderate meetings,
- To run chapters with transparency,
- To share expertise within the community.

The essential traits of a successful chapter leader are as below

- Exceptional Leadership abilities as the chapter spans multiple agile teams and squads
- Possess deep specialist skills that are connected to the chapter
- People leadership, developing talent across the business and outside the organization
- Championing and influencing a culture of growth mindset
- Craft enablement through coaching, mentoring, facilitation, and teaching
- Technical acumen in capability delivery and thinking critically about the application of technologies to solve customer experience problems
- Effective problem-solving and decision making

6 Challenges in organizing chapters

6.1 Internal challenges

Some of the challenges that exist internally within chapters are

- Chapters are more loosely organized teams and without motivation among the chapter members they can become dysfunctional
- Chapters aren't a one size fits all model. An example is the Spotify model of agile teams might not work for all companies

6.2 External challenges

- Chapters do not necessarily have a direct impact on the sprint goal of a team. This often creates friction in the overall agile enterprise on the need for them
- Organizing chapters can become complex with collaboration and communication becoming a challenge between the various teams.

7 Solution from the trenches

In this section, I will discuss the strategies that have worked for us when building chapters and the solutions that were implemented to make our chapters more effective.

7.1 Management Air cover

The success of any chapter is directly proportional to the extent of management support that they enjoy. Since chapters are external to the agile teams that directly work on business outcomes and goals, often time's chapters and chapter goals get side-lined, because no product owner or someone in a similar role is setting aside time for it. With the right management support and cover, chapters can be effective and focused on achieving their goal. Management support is necessary for chapters to focus on the right problems to solve.

7.2 Gamification of chapters

Gamification is about leveraging the motivational aspects of games and applying them to real-world situations. It draws on the innate human desire for challenge, achievement, and rewards. By incorporating game-like elements such as points, levels, badges, leaderboards, and quests, gamification taps into our natural inclination to participate, compete, and succeed. Gamification in chapters can effectively engage chapter members, motivate them to work on tasks, and bring them to closure.

Leaderboards in gamification are a feature commonly used to promote competition and motivate players or users in a game or gamified system. They display rankings of participants based on various criteria, such as scores, achievements, levels, or progress. Leaderboards typically show the top performers or highest-ranking players, allowing others to see where they stand in comparison. This fosters a sense of challenge, encourages engagement, and drives individuals to strive for improvement. By showcasing the achievements of top players, leaderboards create a competitive environment and spur friendly competition among participants.

Chapter leaders can implement leaderboards and award members badges based on their accumulated points. The points are allocated to individual chapter members on an ongoing basis as they complete specific chapter tasks and initiatives. Points can be collected by chapter members for performing specific tasks. Some of the common tasks that chapter members can get involved in to accumulate points are specified below.

- Complete chapter-specific tasks and activities such as
 - Building an automation framework
 - Design a continuous testing pipeline
 - Formulate meaningful testing metrics to measure within teams.
- Ensure implementation of best practices and frameworks built by chapter within individual agile team/squad
- Champion chapter initiatives within and across squads
- Conduct training programs for chapter members to upskill and reskill

A sample leaderboard for organizing and running chapters.

Chapter Member	Points Accumulated	Badge
Jack	75	 
Ana	60	
Mohammed	52	
John	43	 
Sharon	38	
Jane	36	 
Watson	30	

7.3 Chapter time

This is time that is allocated for team members to work on chapter tasks. In a typical agile organization, this time is often not accounted for, and it quickly turns into a situation where activities specific to the chapter get side-lined or down-prioritized. It is good practice for chapter leaders to work with the Scrum masters of individual agile teams and agree on a specific percentage of the time being set aside for chapter activities. This time that is set aside for chapter work needs to specifically be called out in the PI planning for the teams. A practical allocation of time between sprint deliverables and chapters is 80:20 where 80% of the team's capacity is allocated towards achieving the sprint goal whereas 20% of capacity goes to chapters.

7.4 Chapter Tasks

Chapters can be involved with tasks that horizontally impact all or some of the teams within the overall agile enterprise. These tasks add value to the individual teams and can help them accelerate their velocity and capability to deliver business value for the end customer.

Some of the tasks that can be accomplished within a chapter are as below.

- Building an automation framework that can be implemented within the teams
- Functional and non-functional tool optimization and rationalization
- Designing a test architecture suitable for testing different technologies UI, API, DB, etc.
- Developing coding standards for scripting with the automation tools
- Building continuous testing pipelines
- Designing branching strategies for test code
- Conducting hackathons and technical meetups for testing professionals
- Plan and conduct test process reviews to standardize processes across teams

8 Metrics & KPI

Some of the key metrics that can be tracked to measure the success of chapters are in effect similar to the metrics that apply to an agile team as well.

Lead Time:

Lead time measures the total time from the moment a story enters the system (in the backlog) until it is completed as part of a chapter sprint. It measures the total time for a requirement to be realized and start earning value – the speed of your value chain.

Cycle Time:

Cycle time is a subset of lead time – it measures the time for a task to go from “started” or “in progress” to “done”. If cycle times are longer than a sprint, teams are not completing the work they committed to.

Chapter Happiness Index

Each chapter member is asked to vote on a scale of 1 to 5 on how happy he is in the chapter with 1 being least happy and 5 being most happy. The average of all the chapter members’ happiness index will provide the happiness index of the overall chapter. The chapter happiness index is a measure of how motivated and happy a chapter is functioning. This is an important measurement because a happy chapter can get a lot more accomplished.

9 Extend testing chapters

Our experience in building chapters for testing can also be extended to organizing chapters for other skills within agile teams. The primary tenets of organizing, monitoring, and running testing chapters can be extended to chapters of any skill. The best practices for running these testing chapters can be cascaded to other chapters. Such sharing of best practices among chapters can help chapter leaders accelerate the time required to build and organize them. A ceremony called “chapter of chapters” is similar to the “scrum of scrum” where chapter leaders across multiple chapters participate to share updates on their respective chapters.

10 Benefits

Benefits can be quantitative, such as a reduction in expenses, increase in revenue, increase in market share, or risk reduction, e.g., compliance. In addition, benefits can also be qualitative, such as improved employee morale or a more recognized corporate brand.

10.1 Qualitative Benefits

- Rapidly implement best practices through lift and shift from across the different squads
- Build reusable framework components once and deploy across multiple squads
- Increase collaboration among team members and share knowledge
- Develop agile team members into T-skilled professionals where the vertical line of the “T” indicates the specialization that they gain and nurture through chapters and the horizontal line indicates the more generic skills they develop within agile teams.

10.2 Quantitative Benefits

- Increase in velocity by implementing automation of the existing manual testing process
- Significant decrease in R&D effort within the squads as this effort is now moved to the chapters
- Optimize tool license cost by optimizing the usage of tools within the teams

Conclusion

In conclusion, chapters play a crucial role in agile organizations, contributing to the development of individuals, teams, and the overall success of projects. They provide a structured framework for career growth, knowledge sharing, and collaboration within the organization. By grouping individuals with similar skill sets or interests together, chapters promote learning, skill development, and mentorship opportunities.

Chapters foster a culture of continuous improvement by encouraging regular feedback, promoting transparency, and creating a safe space for open discussions. They enable individuals to explore new areas of expertise, expand their skill sets, and take on challenging projects. Additionally, chapters serve as a valuable mechanism for identifying and addressing skill gaps, ensuring that the organization has the necessary capabilities to deliver high-quality products or services.

Furthermore, chapters enhance communication and collaboration between teams, breaking down silos and encourage cross-functional interactions. Through regular chapter meetings, workshops, and knowledge-sharing sessions, teams can leverage the collective intelligence and experience of their members, resulting in increased innovation, productivity, and problem-solving abilities.

However, it is important to note that the success of chapters relies on effective implementation and management. Organizations must establish clear guidelines, responsibilities, and goals for each chapter, and ensure that there is adequate support and resources available to facilitate their activities.

In summary, chapters are a powerful mechanism for fostering collaboration, continuous learning, and career development within agile organizations. By creating a supportive environment that encourages knowledge sharing, skill development, and cross-functional collaboration, chapters contribute to the growth and success of both individuals and the organization as a whole in an ever-evolving and dynamic business landscape.

References

- <https://rebelsguidetopm.com/5-team-structures-for-agile-teams/>
- https://en.wikipedia.org/wiki/Tuckman%27s_stages_of_group_development
- <https://www.adaptovate.com/agile/what-are-chapters-in-an-agile-operating-model/>

A11y Advocacy: Why We Need to do More Than Just Test for Accessibility

Michael Larsen

mkltesthead@gmail.com

Abstract

Accessibility is a broad area. It can be applied to many different scenarios and can be met in many different ways. At the end of the day, though, we are dealing with people with challenges and concerns that, let's face it, most if not all of us will face if we live long enough. Accessibility is more than checking off a box that says, "We are compliant". It is advocating for people to be able to effectively participate in daily life as any of us would, with accommodations where necessary.

This paper explores the significance of accessibility advocacy in the context of inclusive design. It highlights the importance of considering diverse user abilities and the benefits of creating accessible products, services, and environments. The discussion focuses on the concept of "A11y" as an abbreviation for accessibility, the principles of inclusive design, the relevance of accessibility to various disability levels, and the guidelines set forth by the Web Content Accessibility Guidelines (WCAG). Additionally, the paper presents ten principles for effective accessibility implementation and discusses the qualities of a HUMBLE accessibility advocate. Finally, it examines useful tools and resources for evaluating accessibility and concludes by emphasizing the long-term impact and personal benefits of incorporating accessibility into design practices.

Biography

Michael has worked on a broad array of technologies and industries including virtual machine software, capacitance touch devices, video game development and distributed database and web applications. He currently works as a software testing trainer with UST xPanxion. He writes a software testing blog called TESTHEAD (<http://mkltesthead.com/>).

Michael served as a member of the Board of Directors for the Association for Software Testing from 2011-2015. He was their Treasurer and then their President. Currently, he helps teach their Black Box Software Testing classes. Michael is also the current producer and a regular commentator for The Testing Show, a podcast produced for QualiTest (available in Apple Podcasts). For 2023, Michael has been the Marketing Chair for the Pacific Northwest Software Quality Conference.

Copyright Michael Larsen August 18, 2023

1. Introduction

As we navigate the ever-changing world that is modern technology, a concept often overlooked in these latest is Accessibility. In the simplest sense of the word, Accessibility is “the quality of being easy to approach, reach, enter, speak with, use, or understand” and “the quality of being usable, reachable, obtainable, etc.” That is what dictionary.com uses to describe the term. While it is accurate, there’s more to it than that. It’s not just a buzzword. It’s literally a compass that can guide us into a more inclusive world. We’re all, to borrow a line from Prince, “Gathered here today to get through this thing called life.” Sooner or later, we are all going to face challenges that make Accessibility a personal matter. Here’s the thing; Accessibility isn’t just checking off boxes labeled “compliance.” It’s about making sure the products we create allow everyone, as much as possible, to be a part of this digital world and beyond, no matter their abilities. Accessibility goes considerably farther than the digital realm, but that is my area of experience, so I will have to focus our attention there.

The goal of this paper is not to plow through textbook definitions (though there will certainly be some of those). It’s meant to help us see the diverse user abilities that surround us and why making products, services, and spaces with everyone in mind is vital. It’s a willingness to take on a promise, to use the principles of inclusive design, follow the threads that connect accessibility to various disabilities (and levels therein), often using the Web Content Accessibility Guidelines (WCAG) to help us navigate along the way.

At the center of this paper and many of my presentations over the years is the “Ten Principles of Accessibility”. These principles were first written (at least in this way) by Jeremy Sydik, more than 15 years ago. These are more than just guidelines; they’re the cornerstone of Accessibility advocacy. Additionally, I owe a debt of gratitude to friend and colleague Albert Gareev and his introducing me to the mnemonic “HUMBLE” and what it represents. The “HUMBLE A11y” walks in someone else’s shoes, unlearns what we think we knew, and develops real empathy to help advocate for those who may not be able to advocate for themselves.

2. Accessibility vs. Inclusive Design: Navigating the Spectrum

As we venture into the realms of Accessibility and Inclusive Design, we have to understand these aren’t interchangeable. Accessibility is, to borrow a Marvel metaphor, the “super suit” that ensures everyone – regardless of their abilities – can engage with our product. It involves creating an experience that can leverage assistive technologies. These technologies consist of screen readers, voice commands, closed captioning, assistive touch devices, or any myriad other sets of tools that will allow people to overcome disabilities and interact with our products effectively.

Now, let’s meet its cousin, Inclusive Design. An apt metaphor would be to see it as sort of a “master chef”, capable of whipping up tastes and dishes that cater to every taste bud in the room. Inclusive design is about crafting products and services that are so user-friendly, they include as many people as possible. The difference is, our chef doesn’t have any special tools for this process or a need to use special tools to complete their creations. In fact, their challenge is to make dishes that are as pleasing as possible with a minimum of extra technology, or in some cases, in spite of that technology.

The key here is that Accessibility and Inclusive Design work together. Accessibility tackles the nuts and bolts, making sure that anyone with assistive tech has a smooth ride. Inclusive design helps ensure that everyone feels right at home, or at least as much as possible. They are partners in crime, working together to make sure no one is left out and that everyone can have as seamless and welcoming an environment as possible.

3. What's in a Numeronym: A11y

As I mentioned earlier, that neat little numeronym "a11y", where did that come from? Other than, of course, the specific fact that the number of letters between 'A' and 'y' in Accessibility is eleven. The simple answer is that it was used with short microblogging sites like Twitter, where there was a minimum of characters allowed. In these environments, typing "Accessibility" takes up needed space, so "a11y" became a way to represent that in fewer letters¹.

But here's the real magic: a11y works out to be more than just a string of characters. Localization uses "l10n", internationalization uses "i18n", virtualization uses "v12n". Useful in the same way "a11y" it's but a11y spells out something more meaningful. While we often pronounce "A-eleven-y", we also read it as the word "ally". Reading it as ally creates a symbol that genuinely captures the goals of Accessibility. It's more than shorthand – it's a reminder that behind every line of code or image on a screen, there's a person looking to interact with it. Often a person who has different needs than we might have and that we have a responsibility to help make those experiences for them as effective as possible.

A11y started as a way to save space but has become, in a way, a badge of honor and a call out to those who would write code or test code, or any other element of human interaction. The next time you see that numeronym out in the wild, see it as an invitation to join those who champion accessibility and advocate for it.

4. The Spectrum of Disability

The levels of disability we will deal with as we interact with products is going to vary. We can't just say, "We've checked off the boxes and all is good" because there's a lot more to this than firing up a screen reader and calling it a day.

The most challenging areas are the ones that deal with primary disabilities – these are people who navigate life with persistent challenges and are the ones we most often think about when we think of people with disabilities. These are people with low vision, low hearing, limited mobility, or profound cognitive challenges. Imagine not being able to see the screen you're reading right now or struggling to hear the melodic tone of a friend's or family member's voice. These primary disabilities are also referred to as "chronic" or "persistent," as in they will not be going away. They're part of a person's identity and everyday life, and as such they are a core part of their navigating their way through the products we develop.

In addition to the various primary disabilities, there are also what are called "situational disabilities"². Unlike chronic disabilities, situational disabilities can affect anyone at any time. Have you ever tried to take a call on your phone while attending a rock concert? Not the best use of technology at that moment. How about squinting and struggling to read your phone screen under harsh sunlight? Ever found yourself in a foreign country where the alphabet and language is nothing like your familiar language? These are all situational disabilities, issues that can have an effect on anyone and make them less useful in that environment.

So, what's the takeaway here? Accessibility isn't just about catering to the "obvious." It's about crafting experiences that embrace the full spectrum of human existence – from those who've dealt with these issues from day one to those who face unexpected challenges. Whether it's primary disabilities or those situational curveballs, our goal is the same: design experiences that work as well as possible for everyone.

One of my favorite examples and one I look to often as both a way that inclusive Design and Accessibility standards can be applied to even printed materials is the IKEA Instructions guides that come with most of their products. In most cases, their instructions are very simple, frequently have no words, have images that are portrayed in ways that tend to be easy to interpret, and ultimately do not require multiple languages to make sense of.

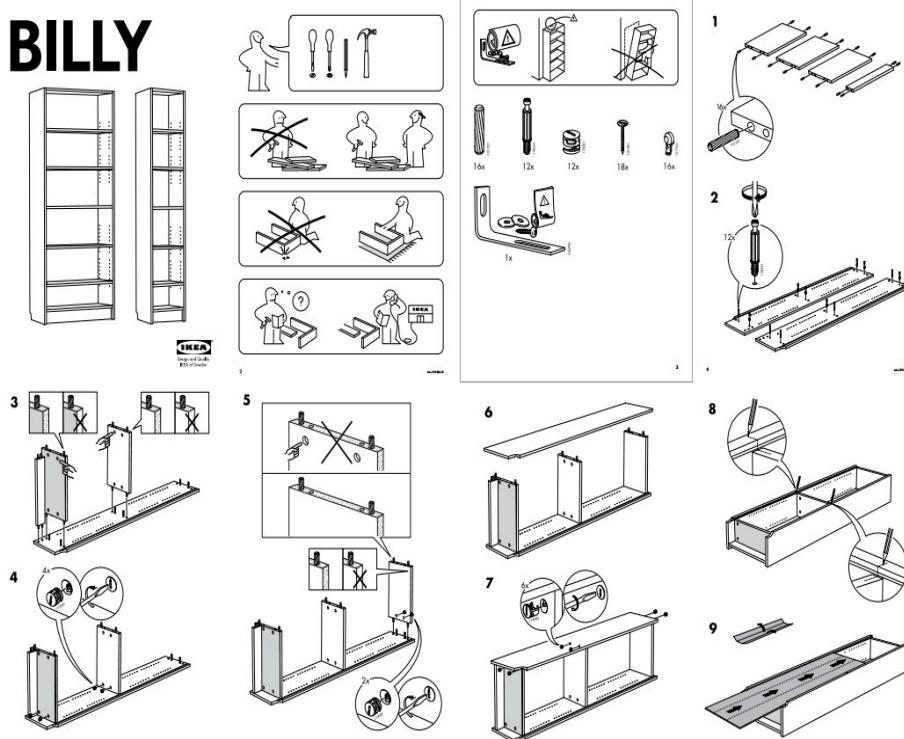


Figure 1. IKEA Billy bookcase assembly instructions

What makes this effective is that the instructions are able to be printed on just a few sheets of paper, with minimal additional details required. One set of instructions effectively works for everyone. Granted, for blind users, this is a hindrance but for everyone else, this is as inclusive as possible.

5. Ten Principles of Web/Mobile Accessibility

Jeremy Sydik's book "Design Accessible Web Sites"³ focuses on "Ten Principles of Web Accessibility". At the time he wrote the book, mobile devices were a very small part of market access compared to what it is today, but many of the principles that Sydik presented are just as applicable in the mobile space. Those ten principles are:

1. Avoid making assumptions about the physical, mental, and sensory abilities of your users whenever possible.
2. Your users' technologies are capable of sending and receiving text. That's about all you'll ever be able to assume.
3. Users' time and technology belong to them, not to us. You should never take control of either without a really good reason.
4. Provide good text alternatives for any non-text content.
5. Use widely available technologies to reach your audience.
6. Use clear language to communicate your message.

7. Make your sites usable, searchable, and navigable.
8. Design your content for semantic meaning and maintain separation between content and presentation.
9. Progressively enhance your basic content by adding extra features. Allow it to degrade gracefully for users who can't or don't wish to use them.
10. As you encounter new web technologies, apply these same principles when making them accessible.

These principles are helpful when it comes to designing applications and they are also helpful when it comes to framing how they should be tested.

5. The HUMBLE A11y

Talking the talk is one thing, but what about walking the walk? I'll never forget a presentation I participated in with Albert Gareev, my fellow Weekend Testing Americas coordinator close to a decade ago. We had a session devoted to Accessibility and one of the topics we covered was how we can actually get into the minds and bodies of people who deal with disabilities every day. Albert had the perfect acronym to help guide us. That acronym is "HUMBLE," and let me tell you, it's well named.

Again, like a11y, "HUMBLE" is not just a word, it's a roadmap. It's a way of thinking that will have the potential to transform how you see the world around you, specifically in regard to interacting with technology.

First up, "H" stands for Humanize. It's about getting cozy with the emotional side of things. No more one-size-fits-all approach – it's about understanding how these experiences hit home for real people.

Next comes "U" for Unlearn. Shed those device-specific blinders and get comfortable stepping into different habits. If you are used to using your mouse for everything, see how it feels to only use your keyboard. No mouse interaction allowed.

"M" stands for Model. Picture this: personas that put you into the situations and lives of users. It's about seeing, hearing, and feeling the issues firsthand. We're talking behaviors, pace, mental states, etc.

"B" stands for Build – build your knowledge, your testing skills, and the scripts and tools that can help you do the work necessary.

"L" is for Learn. This is the step where we put on our detective hats. What are the barriers? How do users tackle 'em? Are we effective in our abilities when we assume these people's identities and realities?

Finally we get to "E" for Experiment. Put yourself in their shoes – literally. Collaborate with the development team, share feedback, and shape those experiences. It's not just about pixels; it's about real-life moments that matter.

For those who take the time and put forth the effort to practice the principles in the HUMBLE mnemonic, I dare say you will not be able to help but be more effective, understand the situations of your customers, and build products and tools that delight them. Also, getting closer to customer needs and experiences and understanding them better in the general sense is never a bad idea.

6. The Rules of the Road: Understanding WCAG

The Web Content Accessibility Guidelines (WCAG)⁴ are not the only guideline in the world of Accessibility. Many different countries have their own standard. The United States, for example, points to Section 508 of the Americans With Disabilities Act⁵. However, the WCAG guidelines are considered to be a global agreement on Accessibility issues and by understanding the WCAG guidelines, developers and testers will have a good grasp on Accessibility and Inclusive Design issues in general, so that local regulations will often fit right into the WGAC guidelines.

It could be said that these guidelines should be considered part of the minimal standards. If we stay ahead of and implement the suggestions, we don't have as much re-factoring to do in our software or systems when new standards come out. However, if we design without considering accessibility, things like WCAG typically become legal requirements and they are terrifyingly large lifts if we want our products to be used.

WCAG is broken up into a series of principles with subsequent qualifications.

First up, pages and apps should be **Perceivable**, where content is presented in ways that every sense can interact with.

Then we have **Operable**, making sure navigation is possible and effective, no matter the tools or devices.

Understandable is the principle of clarity, that all interactions ultimately make sense to everyone.

Robust rounds out these guidelines, helping to make sure that content is stable, regardless of the tech any user might bring to the interactions.

Each of these sections have individual requirements and levels of compliance that we can interact with and confirm meets the levels of compliance desired. There are three ranges of compliance, including A, AA, and AAA. As a developer or tester goes through each subsection, they will be able to determine at which level they are compliant and more importantly, which levels they wish to be compliant. It might seem expected to have all levels be rated at AAA but there are issues here. Often, the guidelines that are rated AAA for one group of users, would be considered too extreme for other users. Often a balance has to be reached, hence the three levels of compliance for each section and subsection. With these differing levels, teams can make judgment calls as to how specifically accessible a site, pad, or app element may be.

7. Tools of the Trade: Accessibility and Inclusive Design in Action

Software developers and designers have a variety of ways available to them to evaluate and use Accessibility and Inclusive Design principles. Several tools can be implemented to help them get the most out of their interactions.

For many who are newcomers to accessibility the "Web Accessibility eValuation Tool," aka WAVE⁶, is frequently the first tool they interact with. For good reason, as it is simple to use and it also provides a high level visual view to the issues on any given page, app, or interaction. It is often easy to showcase where issues are and how we might be able to remedy those issues.

Numerous Developer Tools have been created to allow developers and testers a direct way to gauge compliance with WCAG guidelines. Examples include color contrasts, examining semantic meanings, or overall WCAG compliance.

For those curious about interacting with low vision and how to interact with systems under those conditions, Apple's VoiceOver ships for free with MacOS, while "NonVisual Desktop Access" (NVDA) is an

open-source tool available for a variety of platforms. By turning on a screen reader, the user is able to hear the page speak back to them. This in turn offers examples and insights to allow them to make good decisions based on the descriptions provided.

We can't forget our friends out there who are color-blind. Numerous variations of color blindness are present and one size does not fit all. Color contrast, with effective hue dynamics, text and images can be easily identified, untainted by their background canvas.

One of my favorite tools to work with Cognitive disabilities is the "Hemingway" app. This app analyzes the readability of text. This can be helpful for developing content to help people with dyslexia, as well as encouraging greater readability of text in general.

The World Wide Web Consortium (W3C) has a "Before and After"⁷ demonstration that shows pages prior to being complaint with WCAG and makes a side by side comparison with pages that are compliant. The source code can be viewed and compared, allowing for Accessibility and Inclusive Design issues and fixes to be examined and reviewed.

#	Title	Description	Result
+ 1.1	Text Alternatives	Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.	✓
+ 1.2	Time-based Media	Provide alternatives for time-based media.	✓
+ 1.3	Adaptable	Create content that can be presented in different ways (for example simpler layout) without losing information or structure.	✓
+ 1.4	Distinguishable	Make it easier for users to see and hear content including separating foreground from background.	✓

#	Title	Description	Result
+ 2.1	Keyboard Accessible	Make all functionality available from a keyboard.	✓

Figure 2: W3C Before/After Simulator with Report

To borrow from Heydon Pickering's book, "Inclusive Design Patterns: Coding Accessibility Into Web Design"⁸, these are some additional areas can be examined and compared:

- Ensure that images are described with alt tags and that the picture is described meaningfully. Additionally, use alt tags so that repetitive images are not all called out by using the WAI alt decision tree.
- Provide a skip link at the top of the document that will allow users to get to the main content of a page and bypass the navigation menu if desired.
- Use the "lang" attribute in tags to help programs translate or render other languages.

- Make buttons that are scalable and not tied to literal images.
- Use images that have a universal meaning (a smiley face can be rendered once, no translation required).
- Use the div tag sparingly, especially in areas where keyboard focus is important.
- Make content available in a variety of formats. If you have uploaded a video, have an option for closed captioning available. Additionally, include a full transcript of the video's content.
- If using date fields, allow for multiple ways to enter the date (text field and date picker).
- Allow Pinch-to-Zoom to let the user determine the amount of zoom and focus needed to view the page.
- Make touch areas large enough to interact with without requiring rescaling.
- Encourage the use of proportional fonts.
- Write simply and use space to aid reading.
- Review the contrast recommendations in the WCAG guidelines and encourage high contrast designs.
- Web pages may end up in other media, such as PDF files to be printed. Make sure elements that appear on the screen appear on a printed page, too.
- Remember that simple interfaces are usable interfaces. Do not make navigation or discovery more difficult than necessary.

8. Conclusion

Now it's your turn. Would you like to shine a spotlight on design decisions, to be both early and impactful in the areas of accessibility and inclusive design? It's important to realize that once a product gains momentum, changing its design is like steering a ship in a storm. That is to say, not the smoothest experience. For using on Accessibility and Inclusive Design, from the get-go, can help tremendously with giving everyone a voice and being able to effectively use it.

Accessibility and Inclusive Design are, in effect, an example of the "curb cut effect". When we designed curb cuts to help wheelchair users navigate sidewalks, we made life a lot easier for our able bodied population, too. Runners, people with strollers, shopping carts, etc. all get to reap the benefits of that decision. In short, these changes are things that can make even "normal" customers happier.

Embrace the diversity that makes each of us uniquely human. Practice empathy as you dive into design decisions. Ultimately, we should embrace the act of being an a11y. Make accessibility advocacy your position from day one. The person who one day might benefit the most from your foresight may very well be you.

9. References

- ¹ . O'Hara, Scott, August 26, 2017, "a11y and a brief numeronyms primer", <https://www.a11yproject.com/posts/a11y-and-other-numeronyms/> (retrieved Aug. 19, 2023)
- ² . Userway, "How Situational Disabilities Impact Us All", <https://userway.org/blog/how-situational-disabilities-impact-us-all/> (retrieved Aug. 19, 2023)
- ³ . Sydik, Jeremy J. 2007. "Design Accessible Web Sites: Thirty-Six Keys to Creating Content for All Audiences". Pragmatic Publishing
- ⁴ . W3C, "WCAG2 Overview", <https://www.w3.org/WAI/standards-guidelines/wcag/> (retrieved Aug. 19, 2023)
- ⁵ . Justice.gov, Civil Rights Division, "Section 508 Home Page", <https://www.justice.gov/crt/section-508-home-page-0> (retrieved Aug. 19, 2023)
- ⁶ WAVE. Web Accessibility Evaluation Tool. <http://wave.webaim.org/> (retrieved August 19, 2023)
- ⁷ W3C. Web Accessibility Initiative. Before and After Demonstration. <https://www.w3.org/WAI/demos/bad/> (accessed July 8, 2017)
- ⁸ Pickering, Heydon. 2016. "Inclusive Design Patterns: Coding Accessibility into Web Design". Smashing Magazine GmbH

Continuous Testing Integration with CI/CD Pipeline

Junhe Liu

Junhe.liu@carrier.com

Abstract

When the software development community started moving toward Agile methodology from Waterfall by the turn of the century, a critical phase of the software development model - deployment – was still stuck in Waterfall time. From the conflict of Agile and Waterfall deployment practices came the idea of DevOps. At the same time, continuous testing -- automated tests integrated within a CI/CD pipeline -- has taken software testing one step further, and it is a direct benefit of the DevOps transition.

In this article, we give a brief overview of the path from Waterfall methodology to Agile, the rise of the DevOps practice, and the benefit of continuous testing. We also show an example of how to integrate automatic testing into the CI/CD pipeline.

Biography

Junhe Liu is currently a DevOps engineer at Carrier. He has over 30 years of technology and technology management experience. At Carrier, he integrated various automatic tests into the CI/CD pipeline. Junhe Liu studied Computer Science at Portland State University and Stanford University. He's also a co-author of Enterprise Java Developer's Guide.

1. Waterfall methodology

The Waterfall Model originated in the nineteen seventies. In Waterfall, a project consists of a few phases, namely requirement analysis, system design, coding, testing, deployment, and maintenance. Each phase needs to be completed before next phase can start, the cost of going back to a previous phase is prohibitive, hence the name waterfall.

In the requirement analysis phase, a product manager would write a paper detailing market opportunity, product capabilities, dependencies, competitive analysis, etc. This paper is often called product requirement documentation

With the product requirement documentation, engineers will write a corresponding system design that meet the business requirement. The system design will include technical architecture, data model, business logic, user interface, etc.

Then there are the coding, testing, and deployment phases. The whole process could take years. When the customers finally see the product, not all features in the product requirement documentation work as anticipated. In addition, some needed features are not in the product, either because the product manager didn't have the foresight, or the requirement have changed over time.

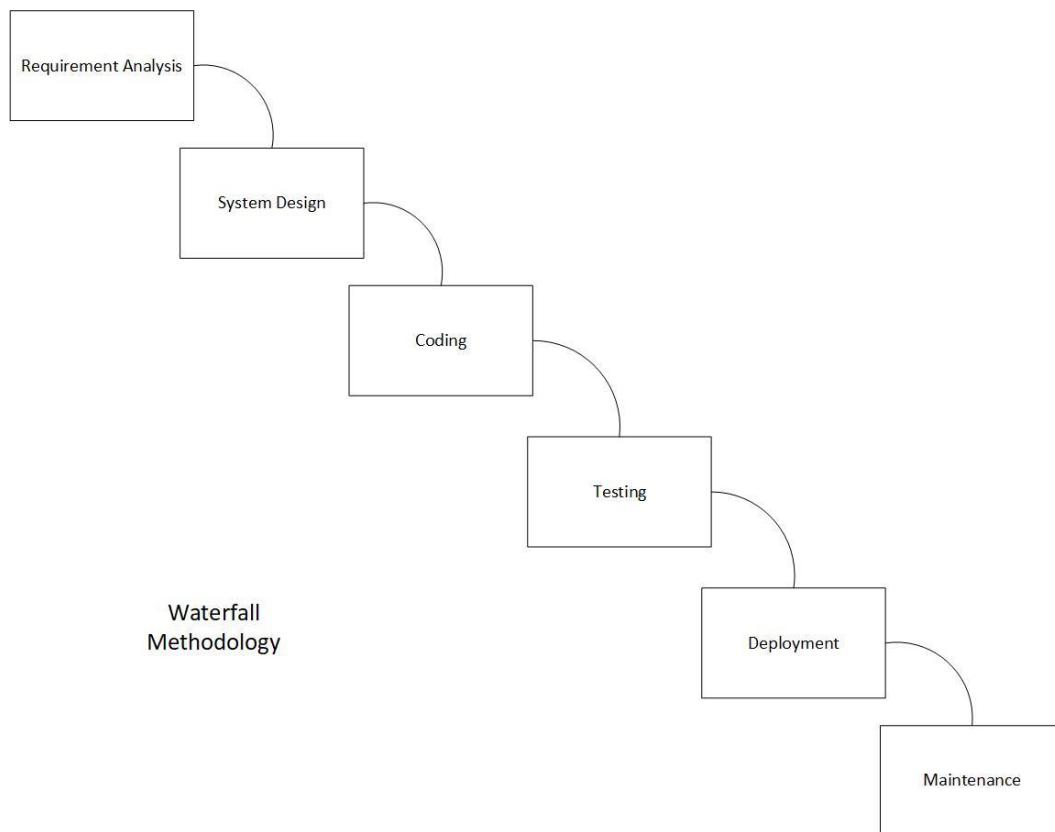


Fig 1: Waterfall methodology

Waterfall was successful before the Internet age because at the time, there were few third-party libraries, and many applications needed to be built from the ground up. Therefore, it takes a long time to develop a new product. The first commercial mobile phone, as an example, took ten years to develop [5]. The first commercial Java implementation took 5 years [6]. However, the rigidity of

Waterfall makes it unfit for the Internet age where time to market is measured in months and customer input has to be implemented as soon as possible. [1][2][3][4]

2. Agile

By the turn of the century, as the Internet became more prevalent, the problems of Waterfall became more apparent. The product would take years to finish and, by the time it was released, the nature of the requirements have changed and product likely failed to address market needs, which have changed or become unimportant. Also, there is no way for developers/product managers to continuously glean changing market needs and feed back to product development.

Agile was invented in 2001 to address the short comings of Waterfall methodology and meet the fast-changing market conditions. Unlike Waterfall, there was no heavy upfront requirement documentation in Agile. If the development team is not sure of the final requirements, it delivers the first approximation of the product and then listens to feedback. The process is iterative, like what the following picture shows:



Fig 2: Agile methodology (Source: Krusche & Company)

A development team uses a series of “sprints” to incrementally finish the product, and in each sprint (usually 2 – 3 weeks), there are phases like plan, design, develop, test, deploy and review. Customer feedback is the key to Agile, and feedback is usually addressed in later sprints.

Agile has nicely addressed the issues of Waterfall, including long product development cycle and no customer feedback during development. The finished products are more aligned with changing market needs. However, there was still one problem – deployment. Agile calls a new deployment for each sprint, therefore customers can see the newly released product. However, nothing significant has happened in the deployment process – in other words, deployment process hasn’t changed much since Waterfall methodology. In the early days of Agile, there was conflict between development teams and operation teams, who usually handled deployment. That’s where the idea of DevOps was created.

3. DevOps

In the golden days of Waterfall, operations teams deployed software releases once every several months. As Agile became popular, such a deployment frequency no longer satisfied the need of development teams. There were constant conflicts between the development team and the operations team. In 2009, during O'Reilly Velocity Conference, technologists John Allspaw and Paul Hammond from Flickr gave a talk titled: "10 + Deploys per Day: Dev and Ops cooperation at Flickr".

In their talk, they described the state where developers and operations blamed each other and how that attitude could be detrimental. They suggested instead that developers and operation should collaborate and make deployment as frequent as possible. That was a defining moment in what was later called DevOps movement.

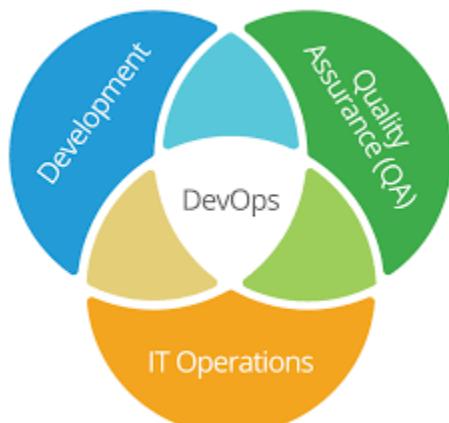


Fig 3: DevOps (Source: Smartsheet)

The idea of DevOps is to fix the inefficiencies of deployment process in Waterfall. In addition to more frequent deployment, DevOps is also set to achieve:

- Shorter lead time for a new release
- Minimized downtime
- Significant improvement in product quality
- Automation in repetitive tasks

Organizations that implement DevOps tend to have more satisfied customers and are more capable of building the “right” products because of more frequent deployment.

In the beginning, DevOps appeared to be the intersection of Dev and Ops. Since then, it has evolved to becoming more than just deployment. Now it also encompasses continuous testing – the testing methodology that allows developers to continuously receive testing feedback during the coding phase. [7][8]

4. CI/CD Pipeline

If you search the Internet for the definition of DevOps, the result could be quite abstract. It is a set of practices, tools, and cultural philosophies. In practice, it can be also viewed as a CI/CD pipeline. CI stands for continuous integration. It is the practice which developers of a team merge code together into a repository. CD stands for continuous delivery (or continuous deployment).

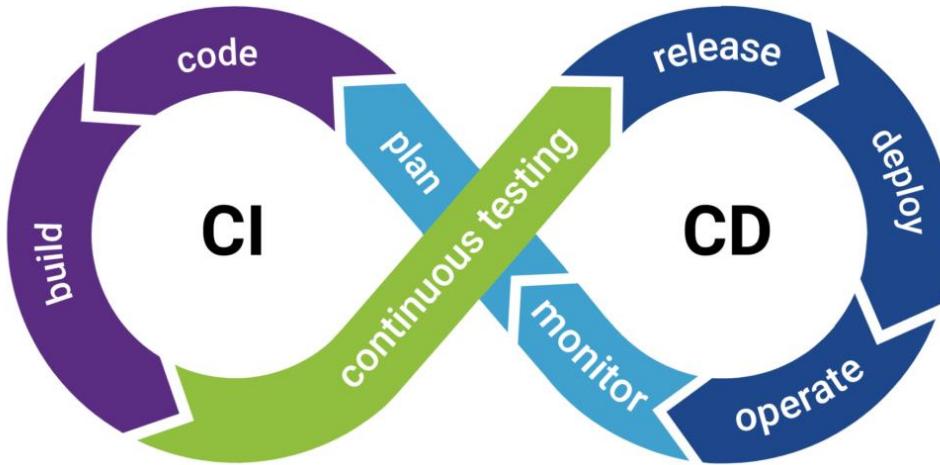


Fig 4: CI/CD Pipeline (Source: Synopsys)

Stages of CI/CD include:

- Code

The first stage of CI is coding, where developers generate source code according to requirement and design. Tools used at this stage is usually an integrated development environment (IDE). After source code is created, a developer would check the code into a code repository and version control system such as Git.

- Build

At this stage, the CI tool (sometimes called a build pipeline) would check out source code from the code repository, combined with relevant libraries, internal or external modules and dependencies, and compile all components into artifacts. Developers will be notified of the build result, whether successful, possible error code, and build logs.

The build process may also include extra tools such as scripts to generate executables or package the artifacts into runnable instances for deployment.

- Deployment

There are usually several environments for deployment, some examples include development (DEV), QA, Staging, and Production. The first three environments are for internal purpose and Production is what the end-users see. If a build is successful, the runnable instance is usually deployed into the first environment – in this case DEV.

Depending on the company and the application, an environment may either be in a data center or in the cloud.

- Continuous testing

Because of the impact and cost of software failure, continuous testing has quietly become the centerpiece of DevOps.

5. Continuous Testing

In the Waterfall model, testing only happened after coding is done. Continuous testing is Agile's approach to software testing. Testing happens throughout the CI/CD pipeline. We don't have to wait until a large chunk of coding is done and then test. Test is done during build, and there are tests after each deployment.

The benefits of continuous testing include [9][10]:

- Expose defects early:

The later a defect is found in the development process, the higher the cost of fixing it is. By moving tests early in the development stage, we can also find defects early.

- Be able to improve the test process

Unlike in the Waterfall model, where testing is done at a specific time. In Agile development, product becomes more complex over time, so are the tests implemented. By setting up tests in every stage of the development, we can gather more data on testing and improve the process over time.

- Immediate feedback

Because all tests are automated, the development team will immediately receive feedback in all stages of testing. This saves time and eventually means faster product release times.

6. Implementation Example

I have been a DevOps engineer in the Supra organization of Carrier for the past several years and have had the opportunity to implement continuous testing integrated with our DevOps pipeline. Let me describe how we did it in this paper.

6.1. CI/CD Pipeline – TeamCity/Octopus Deploy

We use TeamCity as the CI tool and Octopus Deploy as the CD tool.

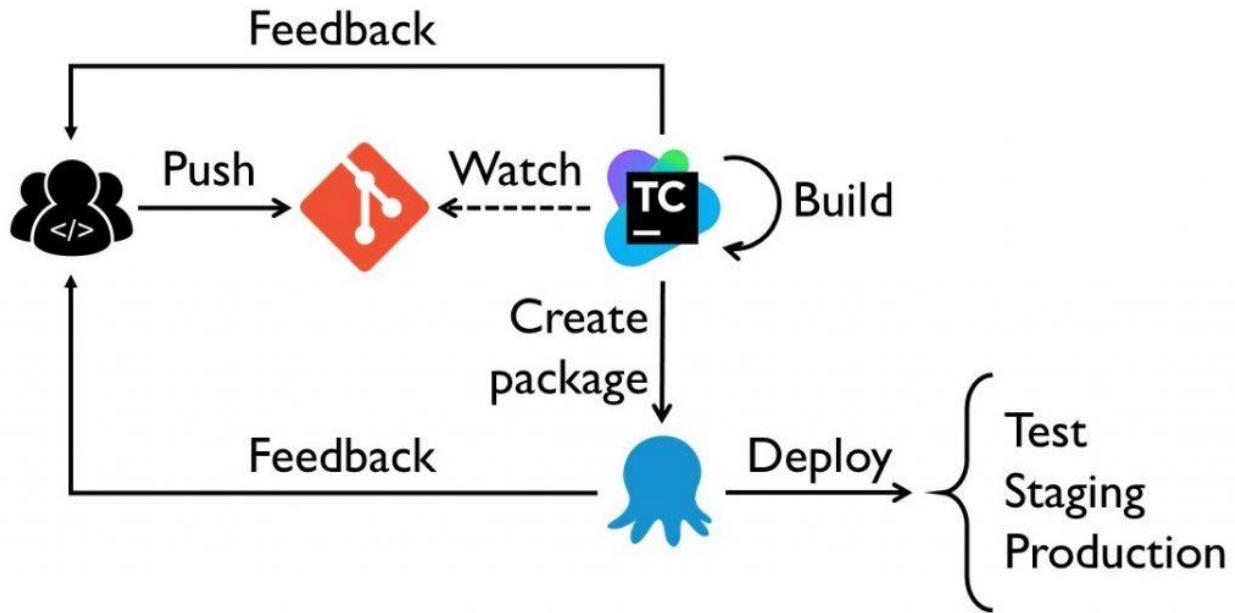


Fig 5: CI/CD Example (Source: boyan.io)

TeamCity is a popular continuous integration server that supports all major IDEs and version control systems, including Git (which we use). It has native support for languages such as Java, .NET, Python, and Ruby. Many other software vendors have written numerous plugins so that their software can work with TeamCity, including Octopus Deploy, Smartbear, Jfrog, etc.

Octopus Deploy is a continuous deployment tool for .NET applications. It works with CI tools such as TeamCity to securely deploy applications to different environments, whether they are on premise or in the cloud. Octopus Deploy enables DevOps engineers to bring deployment related automation into a single place.

6.2. Test Integration

Many types of tests can be integrated with the CI/CD pipeline, including unit tests, integration tests, API tests, UI tests, etc.

6.2.1. Unit Test

Unit test is about low-level tests performed close to the source code. Individual methods, subroutines, functions, or properties are tested. We integrated unit test with our CI tool – TeamCity. TeamCity has built-in support for multiple types of unit tests including nunit and vstest.

6.2.2. Integration Test

Integration test is about whether individual modules work as expected when connected. The test may also include whether the modules can interact with external databases or

APIs properly. Because the test requires other applications to be up and running, it is run in a particular environment. Therefore, it is integrated with the CD tool – Octopus Deploy, which runs tests against the deployment environment.

If the environment is on premise, the test is quite straight forward, including parameter passing.

If the environment is in the cloud, the implementation is a bit more complicated. First, the script that runs the integration test has certain parameters that needs to be passed in during deployment. Those parameters may include description about external resources and what tests to run. Special care must be taken for the cloud deployment. Secondly, because the key user interface is Octopus Deploy, the test report must be passed from Azure to Octopus Deploy. Mechanisms must be established for Octopus Deploy to find out when the test finishes and then download the test report from the cloud to Octopus Deploy.

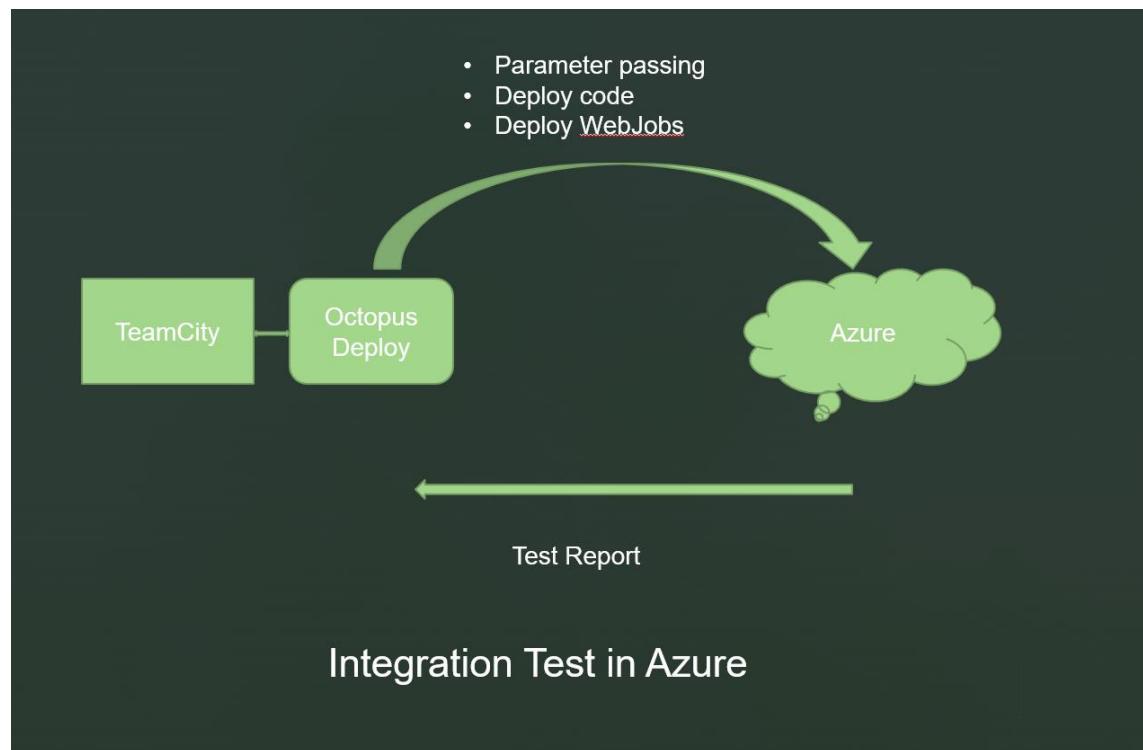


Fig 6: Integration Test integration

6.2.3.API Test

API tests validate the Application Programming Interface (API). Usually, a tool is needed to automate API tests. In our case, we use ReadyApi from Smartbear to run the test. Here are the steps to integrate ReadyApi test into our CI/CD pipeline:

1. Install SOAP UI's TeamCity Plugin on the TeamCity server.
2. Extensively modify the main plugin script so that parameters such as test environment, test type, script repository branch, etc. can be passed into the script.

3. Invoke TeamCity build configuration associated with ReadyApi test from Octopus Deploy using a REST API call.
4. Octopus Deploy keep polling TeamCity until the test finishes.
5. Octopus Deploy then invoke a REST API call to download the test result.

A key consideration in this implementation is job queuing. If multiple tests are invoked at the same time, there should be a mechanism to queue test jobs one by one. Without job queuing, ReadyApi might return unexpected results. To provide this synchronization, the TeamCity build agent has automatic job queuing capability. Therefore, our implementation can handle situations where test jobs are invoked simultaneously.

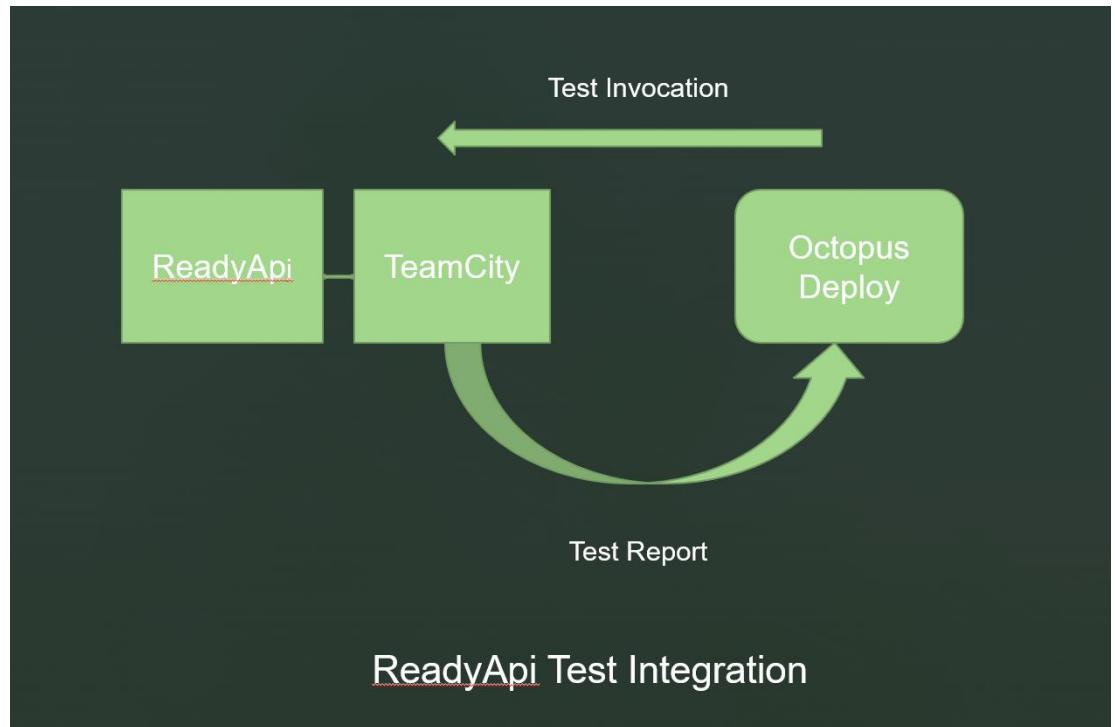


Fig 7: API test integration

6.2.4. UI Test

Automated UI tests can also be integrated post-deployment. We have done a Web application UI test integration with TeamCity and the same test can easily be integrated with Octopus Deploy. We are also in the process of integrating a Mobile Application UI test utilizing both Selenium and Apium.

7. Continuous Testing Process

With automatic tests integrated with the CI/CD pipeline, the development process has changed. After new code is checked into the repository, builds are automatically started, and then unit tests are run. After deployment to a certain environment, say DEV, the integration tests, API tests and other tests such as UI tests can be invoked automatically against the DEV environment. As we promote the application through DEV, QA, Staging, and Production environments, multiple tests are run against each environment, as the following diagram shows:

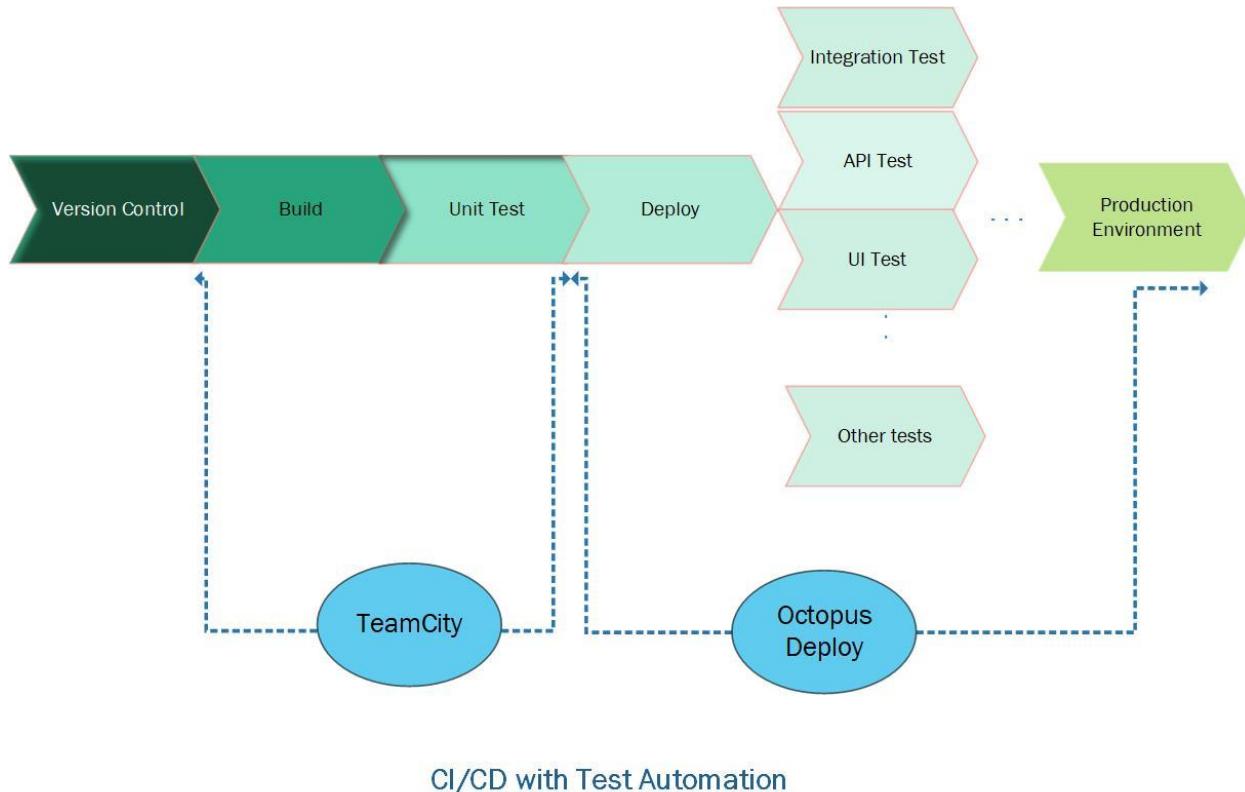


Fig 8: fully automated testing integrated with CI/CD

With test automation integrated with the CI/CD pipeline, Agile team deploys more frequently, tests more frequently, test early, receive feedback early. Not only does the team have better ability to build the “right” product, but also build a quality product.

8. Conclusion

Traditionally software testing starts after developers finish coding. With the fast-moving Agile development method, there is a need for developers to receive immediate testing feedback with new code commits. In addition, after the advent of DevOps practice, there is a need for automatic testing against important deployment environments such as staging and production. Both needs are addressed by continuous testing – automatic testing integrated with a CI/CD pipeline. In other

words: Agile is just the beginning; the road to quality software goes through DevOps and continuous testing.

References

1. To Agility and Beyond: The history—and Legacy—of Agile Development (<https://techbeacon.com/app-dev-testing/agility-beyond-history-legacy-agile-development>)
2. An Introduction to Agile: The history of Agile and project management (<https://www.cambermast.com/agile-blog/2020/05/07/an-introduction-to-agile-the-history-of-agile-and-project-management.html>)
3. The Complete History of Agile Software Development (<https://agilemania.com/history-of-agile-software-development/>)
4. A brief history of the agile methodology (<https://www.infoworld.com/article/3655646/a-brief-history-of-the-agile-methodology.html>)
5. Inventor of cell phone: We knew someday everybody would have one - CNN.com
6. Java (programming language) - Wikipedia
7. The history of DevOps – How has DevOps evolved over the years (<https://www.harrisonclarke.com/devops-sre-recruiting-blog/the-history-of-devops-how-has-devops-evolved-over-the-years>)
8. A Brief History of DevOps (<https://www.knowledgehut.com/blog/devops/history-of-devops>)
9. Continuous Testing in DevOps Best Practices and its Adoption (<https://www.xenonstack.com/insights/what-is-continuous-testing>)
10. What Is Continuous Testing In The DevOps Pipeline (<https://insights.daffodilsw.com/blog/what-is-continuous-testing-in-the-devops-pipeline>)

Reducing Ambiguity in Requirements through Multi-modal Communication

Evan Masters

emasters@critical-logic.com

Abstract

In an era where software development is accelerating at unprecedented rates, clear and concise communication of requirements has never been more critical. Yet, businesses continue to grapple with the challenge of effectively conveying requirements to their software development teams. The consequences? Misinterpretation, rework, inflated costs, and missed deadlines.

Enter the realm of multi-modal communication; a game-changing approach to requirements communication that leverages various forms of communication mediums to ensure clarity and reduce ambiguity. This talk will explore this innovative strategy, which has been successfully adopted by forward-thinking organizations worldwide.

Unlike traditional methods that rely heavily on written documentation, multi-modal communication employs a combination of visual, verbal, and written mediums. It taps into the power of diagrams, prototypes, and face-to-face discussions, fostering a deeper understanding and alignment between business stakeholders, the developers, and the quality assurance team.

In this paper, you will discover how this approach can significantly minimize the risks associated with unclear communication, such as scope creep, budget overruns, and project failure. In addition, I will explore how multi-modal communication can lead to more accurate requirement interpretation, faster development cycles, and ultimately, high-quality software products that meet users' needs.

The associated talk will provide practical tips and techniques for implementing multi-modal communication in your own organization, regardless of its size or the complexity of its projects. From leveraging visual aids to promoting active listening, we'll delve into strategies that can transform your requirements gathering and communication processes.

Biography

Evan Masters is passionate about seeing customers succeed. With over 10 years' experience in the software quality assurance industry, Evan has helped companies around the globe overcome challenges and implement innovative new technologies in support of business goals and objectives. Working with companies of all sizes, from startups to Fortune 5 companies, and thriving on his ability to listen to and understand customers' real needs, he has equipped and supported teams with the tools necessary to take their capabilities to the next level.

Copyright Evan Masters, August 2023

1 Introduction

1.1 Overview of the current landscape of software development as it pertains to the subject matter.

To meet the ever-increasing demands of delivering high quality software products and solutions at a pace matching rapidly evolving business needs, Software Quality Assurance (SQA) professionals need to leverage every resource at their disposal. An effective strategy SQA professionals can employ to ensure the quality of the product or solution is to engage with stakeholders at every stage of the SDLC, not just the Testing stage. A tool that can be used to implement this strategy is a diagram of their understanding of the intended behavior of the business system. This helps to prevent any misalignment in the understanding of the intended behavior across the various stakeholder groups.

Creating (good) models and diagrams to augment text-based definitions of the desired system offers many benefits which I will discuss in depth throughout this work. Creating models and diagrams allows SQA professionals to become engaged much sooner in the SDLC than typically realized, putting quality front and center from the start of the process. These models and diagrams give the SQA professionals a layer of abstraction that can be used to convey an understanding of the system being developed in a way that stakeholders of any technical or business level of expertise can understand.

Whether your project follows traditional linear development methodologies, such as Waterfall, or the more contemporary (and these days more common) iterative methodologies, such as those that fall into the Agile category, you can benefit from engaging in multi-modal communication via requirement/story visualizations using models.

1.2 Challenges faced by business stakeholders in effectively conveying requirements to software development teams.

Business stakeholders, such as Business Analysts (BAs), are typically responsible for describing what the business solution should do to satisfy a need. When they undertake this task, they have a goal to describe the intended behavior as clearly as possible so that the development team can understand what is being asked of them. However, these business stakeholders face several challenges in effectively conveying these requirements to the development teams. These challenges include:

- **Organization structure challenges (i.e., silos):** It is not at all uncommon for business organizations to be functionally segregated. Sometimes there are good business reasons for this. One downside to such an organizational structure is that it prevents the people on these segregated (i.e., siloed) teams from seeing the bigger picture. Therefore, when one of them attempts to describe a business need, they can't see the full impact of how the solution will affect the organization as a whole.
- **Lack of a full understanding of the need:** Another common shortcoming that business stakeholders face when attempting to describe a business need is that they don't fully understand it. This isn't meant as a slight or criticism about these stakeholders. On the contrary, I have personally found myself in this situation numerous times. Rather, it's an acceptance of the fact that business stakeholders often find themselves faced with a problem of finite resources. The resources could be time, accessibility to Subject Matter Experts (SME's), technical expertise, overwhelming amounts of context around the need, just to name a few.
- **Not experienced with conveying the business need to technical resources:** Even the best-intentioned business stakeholders may have a hard time conveying their need in ways that a development team could take and turn into something deliverable. While this does get easier with experience, additional complications to this can arise for even experienced authors. Complications such as offshore teams, infrastructure considerations and constraints, and complex system interactions. Each of these complications adds a layer of difficulty the business stakeholder will have to contend with to convey their need.

In any case, whenever a business stakeholder is unable to completely convey all aspects of a business need to the technical resources that will be developing and testing the solution, they have created ***ambiguous*** requirements. The next section will explore the implications of this result.

2 The Impact of Ambiguous Requirements

2.1 Consequences of ambiguous requirements

Cutting right to the point, the biggest consequence of a business stakeholder creating ambiguous requirements is cost; specifically, increased cost. Whenever ambiguous requirements are used as input to create something designed to serve a business need, it costs more than it would if the definition were clear and unambiguous from the start. This increased cost can come from:

- **An increased number of communication feedback loops:** This happens when the business stakeholder communicates either directly or asynchronously with the development team to clarify their intent. In essence, the more meetings or more email chains that must happen as part of the development process, the more expensive the development is.
- **Rework:** This occurs when the development team interprets what the business stakeholder gave them as input without getting full clarification. The result of this can be a pattern of the developer presenting the solution they've built and including the question, "Is this it?". The longer the development cycle, and the more of the solution built and delivered before showing the stakeholder, the greater the risk of a large amount of required rework.
- **Opportunity costs:** Opportunity costs can occur when the product being delivered contributes to revenue generation. The longer the product is 'in-house' and not in the marketplace, the more time passes without revenue coming in from the product. The other opportunity cost to consider is that the longer the development team spends on a project, the more time passes before they are available to work on the next project. The next project could be either a revenue generator or a cost-saver. Either of which would have a direct cost impact.

There are, of course, other consequences of ambiguous requirements that are not directly cost related.

- **Missed deadlines:** One clear impact of having unclear/ambiguous specs is that it will take longer for the development team to create the proposed solution. If this isn't considered when project timelines are developed (and it almost certainly is not), then there runs the very real risk that any deadlines that have been set are in jeopardy of being missed. While there certainly can be a direct impact to cost when a deadline is missed, I chose to include this consequence as an indirect cost. I did so based on the experience that I have both as a member of an SQA team that been on the receiving end of such a situation, and as a higher-level consultant observing members of my hosting organization's SQA group. What happens more often than not is that the time is made up at the typical end of the SDLC process; namely, it stretches the SQA team. This can result in extra hours worked for the SQA team, a reduction in the amount of valuable quality assurance tasks that can be undertaken, or both.
- **Technical debt:** Technical debt is the implied cost of future work that must occur in response to implementing a solution that is not the best long-term solution, but it works sufficiently in the short term. An analogy for technical debt from the physical world is as follows. Imagine that you are having a home built. You can choose to use materials that are high-quality, recommended by various industry experts, and cost more. By doing so, your home will be more resilient over time, cost less to maintain, and will likely pass any sort of required inspection. Alternatively, you could choose to go with building materials that are lower-quality, acceptable to various industry experts (though not recommended), and initially cost less. The result of which is a home that will end up being less resilient over time, cost more to maintain, and dubiously pass inspections. The latter choice is analogous to choosing the easier and less desirable software implementation option and incurring technical debt.
- **Erosion of trust and confidence:** In less concrete terms, if business stakeholders were to consistently produce ambiguous requirements to the development team and expect them to in

turn deliver perfect software that meets their every need, only to turn around and be frustrated with the development team when the delivery falls short (through no fault of the hard working dev team), an attitude shift will undoubtedly happen over time. The dev team would come to expect that no matter how hard they work and how many extra hours they put in to meet the business-defined deadlines, their work will likely be rejected, and they will have to 'open the hood', sometimes multiple times, before their deliverable is accepted. In turn, the business stakeholders will come to expect that whatever the dev teams show them first, it won't be right, and they will have to go through multiple rounds before they get software that meets their needs. This erosion of confidence by both groups in their counterparts can, in the extreme, become adversarial, and in less extreme circumstances lead to less productive work by both teams.

3 Introduction to Multi-modal Communication

3.1 Explanation of multi-modal communication as an approach to requirements communication.

What is Multi-modal communication? In short, it can be described as expanding the way something is communicated to more than just one method. For example, most software specs are authored in a text-based communication mode, and only a test-based communication mode. Thing of user stories. These are authored in some kind of management system and are primarily written down in a text field. If one were to augment this with, let's say a wireframe or mockup, a second type of communication mode has been introduced. This makes it a multi-modal communication mode.

Multimodal communication is the use of multiple modes of communication to convey a message. This can include verbal language, written language, images, sounds, and gestures. Some of these modes are easy to capture and codify to make a persistent record. Other are more ephemeral, though that does not necessarily reduce the importance or effectiveness of the mode.

Multimodal communication can be used to improve the clarity and understanding of a message by providing different ways for the receiver to interpret it. By providing multiple different communication modes that describe the solution, there is a much greater likelihood that the stakeholders will have a complete understanding of the intended behavior of the system.

In the context of requirements communication, multimodal communication can be used to:

- Ensure that all stakeholders have a shared understanding of the requirements.
- Reduce ambiguity and misinterpretations.
- Make the requirements more accessible to a wider range of stakeholders.

3.2 Exploration of how it can ensure clarity, reduce ambiguity, and create a common understanding of the intended requirements.

Multimodal communication can help to ensure clarity by providing different ways for the receiver to interpret the message. For example, a written description of a requirement can be supplemented with images, diagrams, or videos to help the receiver visualize the requirement.

Multimodal communication can also help to reduce ambiguity by providing multiple perspectives on the same message. For example, a requirement can be described in different ways, such as in natural language, formal language, or graphical form. This can help to ensure that all stakeholders have a shared understanding of the requirement. To illustrate this, consider the following example:

On Monday, you tell your boss that you will be meeting with your customer next Thursday. He interprets this to be the next Thursday on the calendar, three days from Monday. What it meant to you and the customer was the Thursday the following week. The use of informal language has resulted in an ambiguity.

Formal language can be used here to precisely define that by “next Thursday”, you mean Thursday, September 14th, 2023”. This multi-modal communication has eliminated any possible ambiguity.

Finally, multimodal communication can make the requirements more accessible to a wider range of stakeholders. For example, people with different language skills or learning styles may be able to better understand the requirements if they are presented in multiple modes. Some stakeholders may be visual learners who best understand things presented in picture or graphical format, while others may be more apt to consume information presented in a bulleted list or If/Then format. By providing both, you are increasing the approachability of your design to a broader audience.

Here are some specific examples of how multimodal communication can be used to improve requirements communication:

- A requirements document can include:
 - A written description of the requirements.
 - Diagrams or illustrations of the requirements.
 - Use cases or scenarios that illustrate how the requirements will be used.
 - Screen mockups or prototypes of the user interface.
- Requirements can be gathered through interviews, workshops, or focus groups, and these sessions can be recorded and transcribed to capture the different ways that stakeholders communicate about the requirements.
- Requirements can be modeled using UML or other modeling languages, which can help to visualize the requirements and identify potential ambiguities.
- Requirements can be tested using a variety of techniques, such as user testing or acceptance testing, to ensure that they are understood and can be implemented correctly.

By using multimodal communication, you can be sure that the requirements are clear, unambiguous, and understood by all stakeholders. This can help to avoid costly rework that leads to delays and frustrations.

4 Success Stories: Multi-modal Communication in Practice

4.1 Case Studies of how multi-modal communication has helped organizations of all sizes, in various industries, and around the globe achieve higher quality business systems.

I have a number of case studies that show how engaging in Multi-Modal Communication has resulted in major successes for the customer. These are available on the Critical Logic website [here](#). Additional whitepapers can be found [here](#).

5 Multi-modal Communication Versus Traditional Methods

5.1 Comparison of multi-modal communication with traditional methods that rely heavily on written documentation.

Traditional methods of requirements communication rely heavily on written documentation. This can be effective in some cases, but it can also lead to ambiguity and misunderstandings. The word 'biweekly' for example is inherently ambiguous. Dictionary.com defines 'biweekly' as both twice a week and every other week. There are many more examples like this in the English language, and likely in other languages as well.

Multimodal communication, on the other hand, uses multiple modes of communication to convey a message. This can help to improve clarity and understanding by providing different ways for the receiver to interpret the message. Imagine visualizing your intent of the word 'biweekly' by showing an image of a monthly calendar with every other Tuesday highlighted. This clearly shows your intent of the meaning of 'biweekly'.

Some of the benefits of using multimodal communication over traditional methods include:

- Increased clarity and understanding of the requirements.
- Reduced ambiguity and misinterpretations.
- Improved accessibility of the requirements to a wider range of stakeholders.
- Increased engagement of stakeholders in the requirements definition process.

5.2 How multi-modal communication resets the standards for communication between all participants in the SDLC.

Multimodal communication can reset the standards for communication between all participants in the SDLC by:

- Ensuring that all stakeholders have a common understanding of the requirements.
- Facilitating communication between stakeholders with different language skills or learning styles.
- Making the requirements more accessible to stakeholders with disabilities.
- Improving the efficiency and effectiveness of the requirement development and communication process.

Overall, multimodal communication is a more effective way to communicate requirements than traditional methods that rely heavily on written documentation. By using multimodal communication, you can help to ensure that the requirements are clear, unambiguous, and understood by all stakeholders. This can help to avoid costly rework and delays in the development process.

Here are some specific examples of how multimodal communication can be used in the SDLC:

- During the requirements gathering phase
 - Stakeholder interviews can be recorded and transcribed to capture the different ways that stakeholders communicate about the requirements.
 - Requirements workshops can be used to facilitate communication between stakeholders and to get feedback on the requirements.
 - Use cases or scenarios can be used to illustrate how the requirements will be used.
 - Diagrams can be created based on Use Cases and/or Scenarios that augment the information contained in the Use Case/Scenario.

During the requirements analysis phase:

- Requirements can be modeled using UML or other modeling languages to visualize the requirements and identify potential ambiguities.
- Requirements can be modeled using a variety of techniques, and then shared back with the stakeholders who originated the requirements to ensure that they are understood and can be implemented correctly.

During the requirements validation phase:

- The requirements can be presented to stakeholders in multiple formats, such as written documents, diagrams, or prototypes, to ensure that they are understood.
- Stakeholders can be asked to provide feedback on the requirements in multiple formats.
- Tests can be developed from the visualized requirements and included in the validation assets so stakeholders know how their requirements will be validated.

By using multimodal communication throughout the SDLC, you can help to ensure that the requirements are clear, unambiguous, and understood by all stakeholders. This can help to avoid costly rework and delays in the development process.

6 The Benefits of Multi-modal Communication

6.1 Explanation of how this approach minimizes risks associated with unclear communication.

Multimodal communication can help to minimize the risks associated with unclear communication by providing multiple ways for the receiver to interpret the message. This can help to avoid misunderstandings and misinterpretations.

For example, a written description of a requirement can be supplemented with images, diagrams, or videos to help the receiver visualize the requirement. This can be especially helpful for stakeholders who have different learning styles or who are not native speakers of the language in which the requirements are written.

This will also lead to more accurate requirement interpretation. Specifically, Multimodal communication can lead to more accurate requirement interpretation by providing different perspectives on the same message. This can help to ensure that all stakeholders have a common understanding of the requirement. For example, a requirement can be described in different ways, such as in natural language, formal language, and graphical form. By doing so, this can help to ensure that all stakeholders are able to understand the requirement, regardless of their background or preferences.

Resulting in faster development cycles

Multimodal communication can result in faster development cycles by reducing the need for rework. When requirements are clear and unambiguous, there is less need to go back and forth between stakeholders to clarify the meaning of the requirements. This can save time and money in the long run.

Producing higher quality business systems

Multimodal communication can produce higher quality business systems by ensuring that the requirements are accurately implemented. When requirements are clear and unambiguous, there is less

risk of errors being introduced into the system during the development process. This can lead to a higher quality system that meets the needs of the stakeholders.

In addition to these benefits, multimodal communication can also help to improve the engagement of stakeholders in the requirements definition process. When stakeholders are able to communicate their needs in a way that is comfortable for them, they are more likely to be engaged in the process and to provide valuable feedback. This can lead to a better understanding of the requirements and to a more successful project overall. By engaging in Multi-Modal communication, you are addressing both the concrete, hard requirement of developing something implementable and testable, and also the softer, more nuanced requirement of allowing all stakeholders, regardless of their specific style, to better and more effectively contribute to the development of the desired solution.

6.2 Discussion on how multi-modal communication can lead to more accurate requirement interpretation, faster development cycles, and higher quality business systems.

Images and diagrams can be a helpful way to illustrate requirements and to make them more understandable. For example, a diagram can be used to show the flow of information through a system. Another type of diagram can be used to show, for the same system, how data elements of a system relate to each other.

Video can be a helpful way to demonstrate how requirements will be used in practice. For example, a video can be used to show a user interacting with a system or to demonstrate how a process will be carried out. Videos can include everything from stakeholder interviews to a demonstration of how an app will function based on a series of mockups.

Prototypes can be used to test requirements and to get feedback from stakeholders. For example, a prototype of a user interface can be used to get feedback on the usability of the interface. Prototypes are also effective for validating accessibility considerations before development.

By using multimodal communication, you can help to ensure that the requirements are clear, unambiguous, and understood by all stakeholders. This can help to avoid costly rework and delays in the development process. It can also lead to higher quality business systems that meet the needs of the stakeholders.

7 Practical Tips and Techniques for Implementing Multi-modal Communication in Organizations

7.1 Guidance on implementing multi-modal communication derived from real-world experience.

In my personal experience, there is a short list of things that you can do to improve your ability to get higher quality requirements from stakeholders by using Multi-modal Communication techniques.

- Identify the stakeholders
 - The first step is to identify the different stakeholders who will be involved in the project. This includes the users of the system, the business owners, the developers, and the testers.
 - This seems obvious but I've seen many projects fail because key resources have failed to identify critical stakeholders.
- Understand the needs of the stakeholders
 - Once you have identified the stakeholders, you need to understand their needs and expectations. This includes their understanding of the requirements, their preferred communication methods, and their level of technical expertise.

- Choose the right communication methods
 - Once you understand the needs of the stakeholders, you can choose the right communication methods. This may include written documents, diagrams, videos, prototypes, or even face-to-face meetings.
 - Remember to keep in mind that you may need different communication modes to convey the same message to different stakeholders
- Use multiple communication methods
 - It is important to use multiple communication methods to ensure that all stakeholders can understand the requirements. This is especially important for stakeholders with different learning styles or who are not native speakers of the language in which the requirements are written.
 - Again, multiple modes may need to be implemented.
- Be flexible and reactive
 - It is important to be flexible in your approach to multimodal communication. The best communication method for one stakeholder may not be the best method for another. Be willing to adapt your approach based on the needs of the stakeholders.
- Get feedback
 - It is important to get feedback from the stakeholders throughout the process. This will help you to ensure that the communication is effective and that the requirements are understood.
- Continuously improve
 - The process of multimodal communication is an ongoing process. As you learn more about the stakeholders and their needs, you may need to adjust your approach. Be willing to continuously improve your communication methods to ensure that the requirements are communicated effectively.

8 Conclusion

8.1 Recap and final thoughts

In conclusion, there are a number of significant benefits to employing multiple modes of communication when defining the requirements that define the intended behavior of your business system. Too often, “process” gets in the way of doing what needs to be done to achieve true quality.

Stepping back to remember that all stakeholders are unique individuals with distinct styles will help to keep perspective in mind. Done intentionally, Multi-Modal Communication will greatly improve the chance that you satisfy your stakeholders’ communication preferences and thereby greatly improve the outcome of developing the software that truly meets the needs of the business end user.

You’re on the same team as your stakeholders. Use Multi-Modal Communication to give them what they need to help you help them. You’ll end up with the best product in the end.

Quality Engineering Considered as a Helix of Semi-Precious Quadrants

Vivek Sahai Mathur

mathurvs@outlook.com

Abstract

How has the scope and role of Quality Engineering changed over time, and what does it mean for you?

Software applications have moved a great deal from well controlled, procedural code on green screens supported by a single processor, single database with a limited and trained user base. The present day utilizes multi-device, multi-user, multi-threaded, event-driven workflows, with distributed data, multiple compute instances and fractured business logic.

All this complexity is compounded by increased awareness and implementation of security and privacy regulations and industry mandated best practices.

The increasing complexity has fundamentally and irreversibly changed the definition of "Fitness for use" of software, and non-functional compliance requirements.

The end result has been an evolution in the roles and responsibilities of pod members, and especially in the relationships between the quality team and the developers. This paper casts the trends in software and team dynamics in terms of the software functionality quadrant, and also links it to the classic Hugo & Nebula award winner story by Samuel Delany.

Biography

Vivek S Mathur started his IT career in 1997.

Vivek has 30+ years of varied work experience ranging from the Government to software product and service companies. His focus has always been on setting up robust processes, and seeing their impact on the quality of deliverables and productivity of the workers. Making changes that delivery teams internalise and adopt, and add value to the organization.

Vivek has set up core software development processes for product and service companies like McAfee and Intelligroup, and has set up and led large delivery teams for multiple customers.

Vivek has always tried to give back to the community, and has been associated with STeP-IN Forum (www.stepinforum.org) actively for the past 11 years. He is currently the President, STeP-IN Forum

Experienced in Information Security, Data Analytics/ Big Data projects, and Software Quality, and in merging the three into the Product Lifecycle Management to achieve highest productivity and predictable delivery while leveraging the partner ecosystem.

Copyright Vivek Sahai Mathur 9 Jun, 2023

1 Introduction

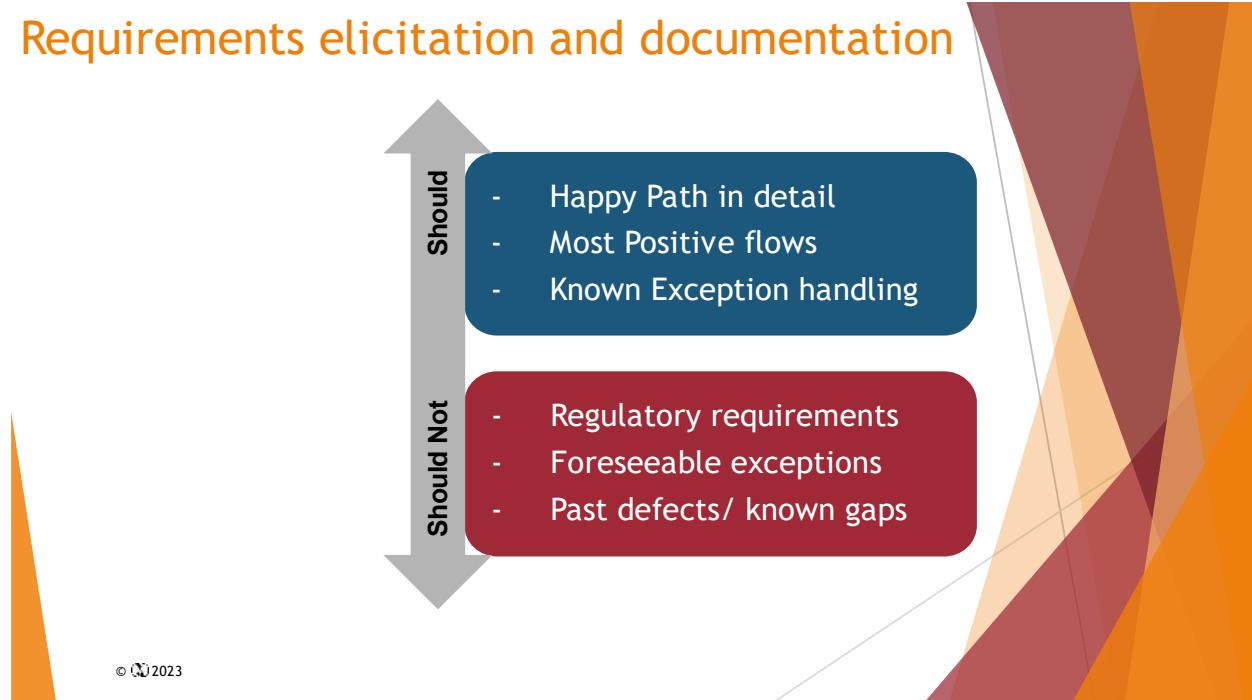
Software applications have been in constant evolution since the first basic automation of work procedures. The impact is apparent in the way we use software, up to the present day, when the application is ubiquitous and central to the business transactions. Every step in the workflow can take place in a different environment. Distributed compute and data storage alters the way applications are built, and the way the transaction integrity is retained.

This paper casts the trends in software and team dynamics in terms of the software functionality quadrant, and also links it to the classic Hugo & Nebula award winner story by Samuel Delany.

1. Evolution of software complexity
2. The expected trends and the roles qualiticians can hope to face in the future
3. How to mature the relationships within the pod and improve the output quality.

2 The evolution of software complexity

Traditional application development lifecycle followed the waterfall approach, in which each step had to be completed before the next one could start. So the requirements gathering and analysis preceded the remaining stages.



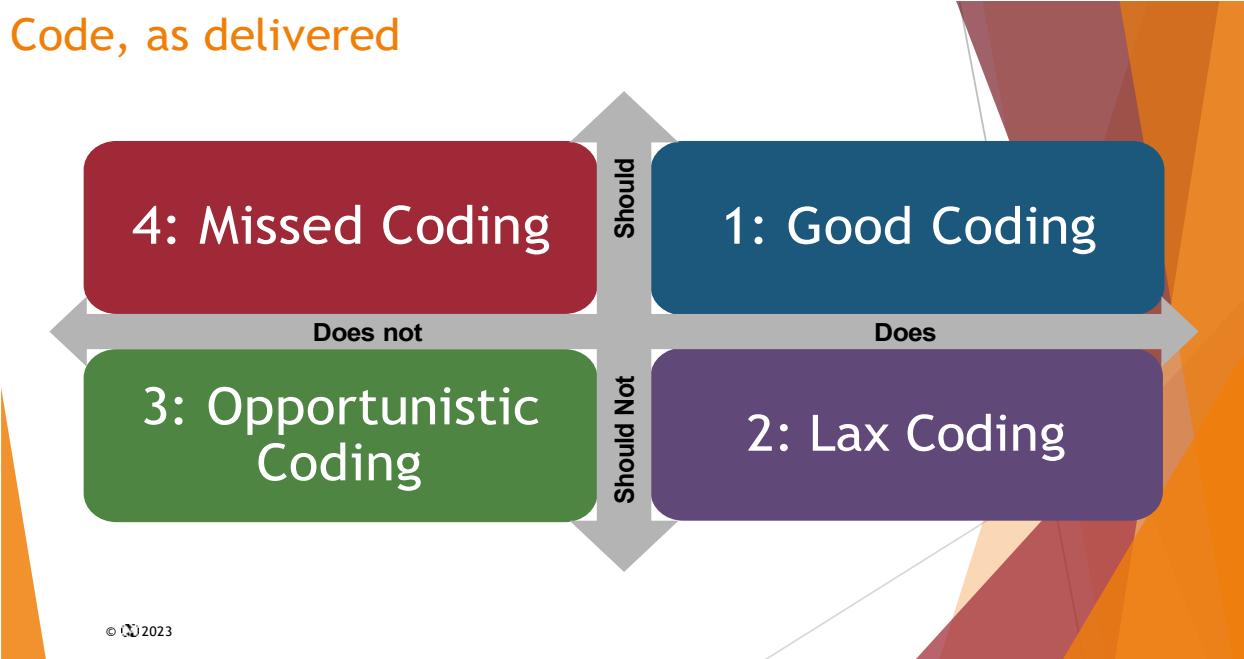
The requirements identify what the software should do and what it should not do, on a single axis. The code that is written should conform to both aspects.

Due to paucity of time and effort available, and the need to ensure that the MVP (Minimal Viable Product) can be delivered at the earliest, complete delineation of the design and code is not possible, and gaps emerge. Taking the code developed as the second axis, we get the 2 aspects of whether the code actually implements the requirements.

The process of translating requirements into design and code leads to the first set of quadrants:

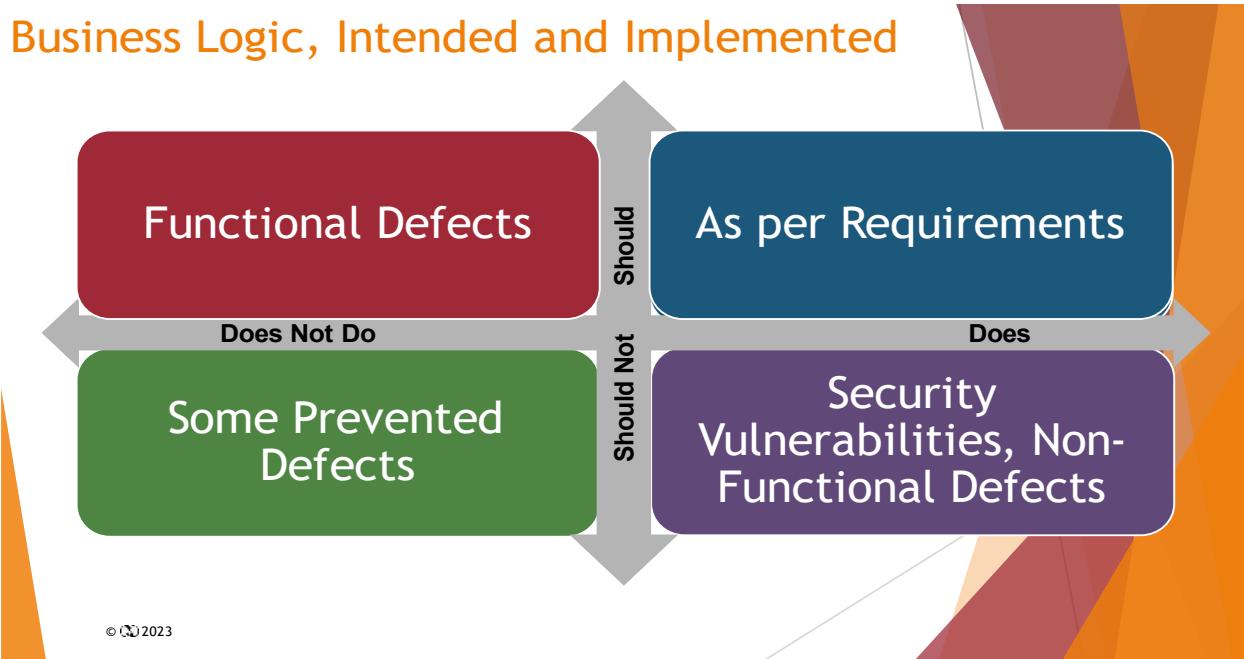
1. Code that does what it should do: This is the correct code, and covers the most common and straightforward functionality.
2. Code that does what it should not do: This is defective code, and may not be easy to identify, since the hidden functionality will not be visible during normal use of the application.
3. Code that does not do what it should not do: This is the code built to address the ‘negative requirements’.
4. Code that does not do what it should do: These are functional defects and should be identified through tests, if the coverage is adequate. Defects in corner cases may reach Production deployment and be hidden for relatively longer periods of time.

Code, as delivered



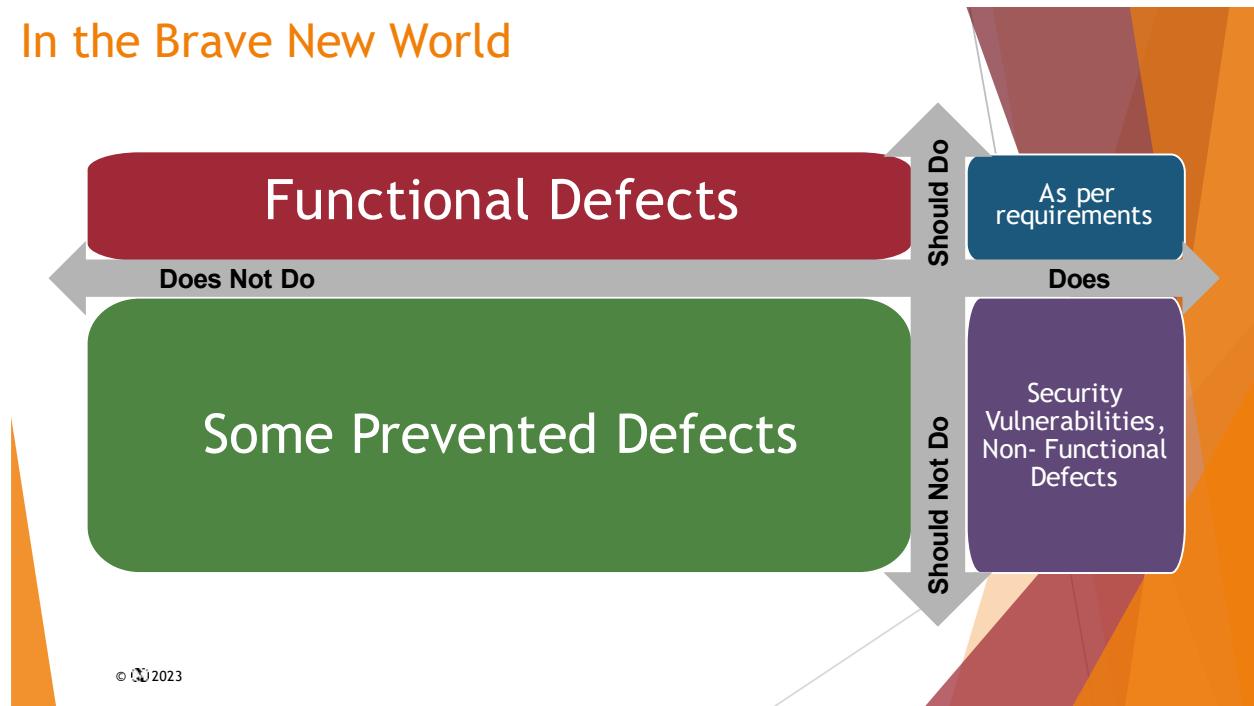
From the aspect of delivered functionality, the results are as follows:

Business Logic, Intended and Implemented



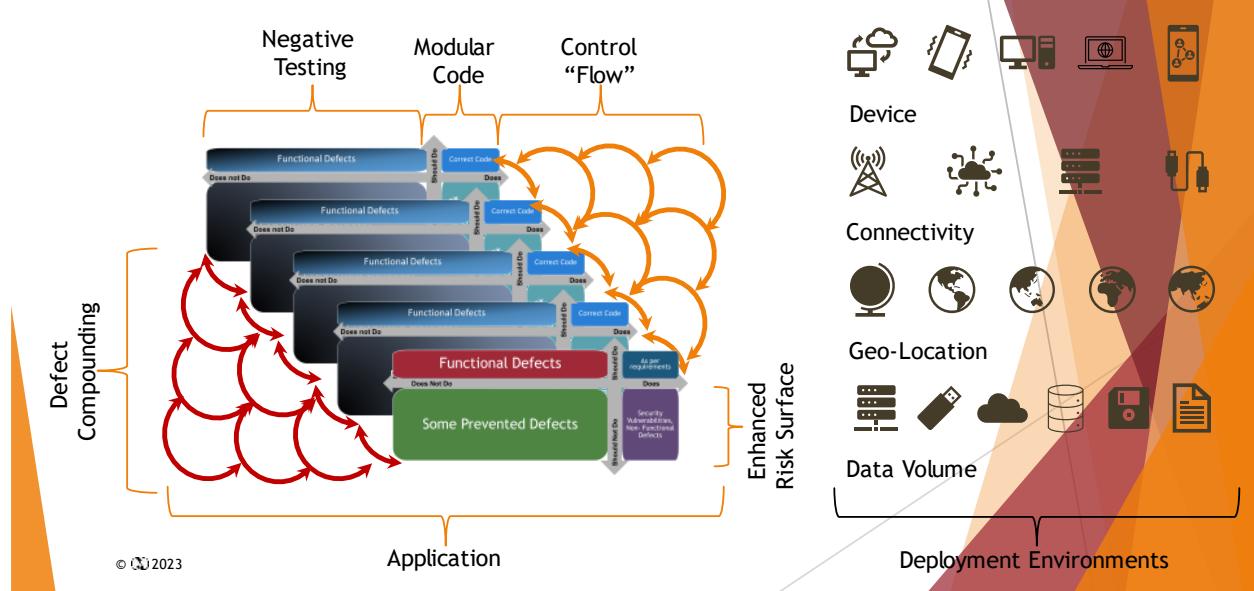
While the above depiction covers the standard monolithic application, recent trends have reduced the functionality in each code module. This makes it difficult to ensure that negative functionality defects are completely covered, and the scope for both nonfunctional and functional defects that need to be avoided increases.

In the Brave New World



Since a lot of single function modules (e.g. Microservices) need to work together, the business transaction itself needs to jump between multiple modules to reach completion:

Zoom Out ...



Taking each of the aspects pointed out in the figure in sequence:

2.1 Application

1. Modular Code: Expected to complete a single function in a robust and repeatable fashion based on the input parameters but without the transaction context. Needs to be able to handle all inputs correctly.
2. Control Flow: The transaction itself may bounce from one code module to another based on the events that are triggered. While there is a “Happy Path” that can be assumed to complete the workflow in the simplest way, there is no force to follow that path, and the deployment environments add another level variability.
3. Enhanced Risk Surface: The expected ability to handle the “Should Not” aspects increases.
4. Negative Testing: The requirement for Negative testing and assuring that incorrect logic cannot occur increases
5. Defect Compounding: As the event flow is unpredictable, the errors in the workflow can impact downstream errors and make the debugging and data correction difficult.

2.2 Deployment Environments

The transactions and workflows have to handle the changes in the following:

1. Device – desktop to mobile devices and back. Databases on-site to on-cloud.
2. Connectivity – switching between telecom network cellular towers to Wi-Fi or wired network, while retaining the transaction context and data.
3. Geo-Location – Regulatory requirements address data processing, data flow, and data storage of non-public data based on the geo location of the data subject, controller processor, or sub[-processor].
4. Data Volume – data processing can occur in multiple location and on multiple devices, necessitating the transfer of transaction data to follow the location (and network, device) where the transaction is being processed.

2.3 What future trends do we see?

The approach to software development will continue to evolve, and some of the trends that can be identified are listed below:

- Supply Chain/ ecosystem – Outsourced components and reuse of the Open Source
- Constant upgrade churn – OSes/ Browsers/ Dev Frameworks will be upgraded, and the applications need to be built to accommodate these constant changes
- “My Name is Vuca” (Volatility, uncertainty, complexity and ambiguity)-changes in the real world scenarios that the software has to address will only increase with new technology and user interfaces (VR, Speech, brain implants?)
- “Would you like some more PII?” – The increasing regulatory controls on the use of personally identifiable data, covering the rights and duties of the data subjects, controllers, processors and sub-processors, and impact of the locations of data origin, processing, and storage will require more real-time monitoring to retain compliance with regulations.
- Built to sell – The software is not built cover a complete business transaction, but as a part of one.
- Edge Computing – technology that distributes the processing and data manipulation, along with the UX management necessarily slices the business process.
- Gen AI – still need to understand the impact this will have on IP, and the backdoors it can open up in the codebase.

3 The evolution of testing expectations

3.1 The evolution so far

How did we get here?



Testing and code development have always evolved, and the relationship between the functions has also been changing through the years.

- Purpose built applications were tested by the developer as part of the coding process, and there was no separate testing phase in effect.
Relationship – non-existent
- With the approach of structured coding practices, the focus for the tester was to achieve Verification and Validation (V&V, V-model, W-model, etc.) success. Test cases were created, with the objective of being MECE (Mutually Exclusive and Collectively Exhaustive) and leaving no requirement un-covered. The testing effort was clearly defined, and had temporal and sequential integrity.
Relationship – unaware/ ignorant
- Test Automation for faster regression testing, Model based testing (MBT) approaches for faster test case development happened when the factory model of code development transformed with Agile approaches. Scrum Pods included a testing resource,
Relationship – Condescending
- There were brief periods of adoption of Pair Programming, TDD, xTDD, etc. with a much closer scrutiny of coders by testers, and vice-versa. Dedicated testing positions in the scrum pod were eliminated.
Relationship - Competitive
- Finally in the present moment, the tester role has split into in-sprint testing in the dev phase, and confirmatory testing at the end of the SDLC. The focus is on business domain experience and exploratory testing to trap corner cases.

Relationship - Cooperative

With the increased adoption of CI/ CD and more automated complex error finding, the tester needs to govern the code development process while validating the business functionality.

4 How the Helix explains the changing relationships

1. "Later HCE finds Arty the Hawk on his doorstep. Arty explains that he sought him out because their relationship was about to undergo a change. HCE is puzzled, but eventually realizes that Arty and he are about to become rivals. Arty will try to buy him out, then to kill him, because that is the way the world works. If he survives and prospers, he and Arty will eventually become friends because there will be more profit in cooperating than in competing. He tells Arty this, and Arty wholeheartedly agrees. He responds that HCE is starting to think holographically, just like Maud and Special Services. He departs, leaving HCE to contemplate his future." – Wikipedia

Alternatively:

2. "First they ignore you"
"Then they laugh at you"
"Then they fight you"
"Then you win" – attributed to MK Gandhi

References

Internet Hyperlinks:

- Hugo and Nebula Award Winners, 1968: <https://nebulas.sfwa.org/nominated-work/time-considered-helix-semi-precious-stones/>
- Plot Summary (last paragraph):
https://en.wikipedia.org/wiki/Time_Considered_as_a_Helix_of_Semi-Precious_Stones
- Helix linkage to relationship evolution:
<https://www.lyricsondemand.com/tvthemes/thepratridgefamilylyrics.html>
- Theory of Emergent Evolution: https://en.wikipedia.org/wiki/Emergent_evolution
- Quote attributed to Mahatma Gandhi: <https://www.snopes.com/fact-check/first-they-ignore-you/>
- My Name is Luka: https://en.wikipedia.org/wiki/Solitude_Standing -> Track listing

Designing and Creating Testing Datasets for Medical AI/ML End to End Testing

Nancy McCormack

Mayo Clinic, CDH AI Engineering

McCormack.Wei@Mayo.edu (working), or nhou_wei@yahoo.com (personal)

Abstract

Testing datasets are essential for ensuring that models work correctly in the real production environment. They can be used to test a wide range of functionality, including API, application, and data pipelines. End-to-end testing is particularly important, as it tests the entire model from start to finish.

Designing testing datasets can be challenging, as it requires a deep understanding of the model and the production environment. It is also important to consider the protection of PHI (Protected Health Information) when designing testing datasets.

This paper describes the following items:

- What the Mayo Clinic CDH AI Engineering team does
- What the AI Engineering Verification and Validation Engineers do
- What end-to-end testing is (based on what the Mayo Clinic CDH AI Engineering team does)
- How to design testing datasets with the protection of PHI (Protected Health Information) for end-to-end testing based on a real Mayo Clinic AI/ML project.
- How to create testing datasets for end-to-end testing
- Issues and challenges during the creation of the testing dataset

Biography

I am a Sr. Full Stack Verification and Validation Engineer for the Mayo Clinic CDH AI Engineering organization, who takes responsibility for verifying and validating models in real production environments to ensure they are safe and effective by collaborating with project team members and cross-function teams. I also lead an offshore V&V team that creates automation frameworks and builds internal tools to help improve the AI Engineering team's productivity.

Before joining Mayo Clinic, I had 8 years of functional management experience and 14 years of hands-on testing engineering experience in the Semiconductor, Networking, and IT industries. Accomplished at managing and leading diverse quality teams using both waterfall and agile methodologies, I am adept in all aspects of the SDLC but excel in test strategy, planning, process, standards, and metrics.

During my free time, I enjoy traveling, watching sports, and tasting different types of foods with my husband, Simon McCormack. I also enjoy dancing and watching real mystery shows.

1. Introduction

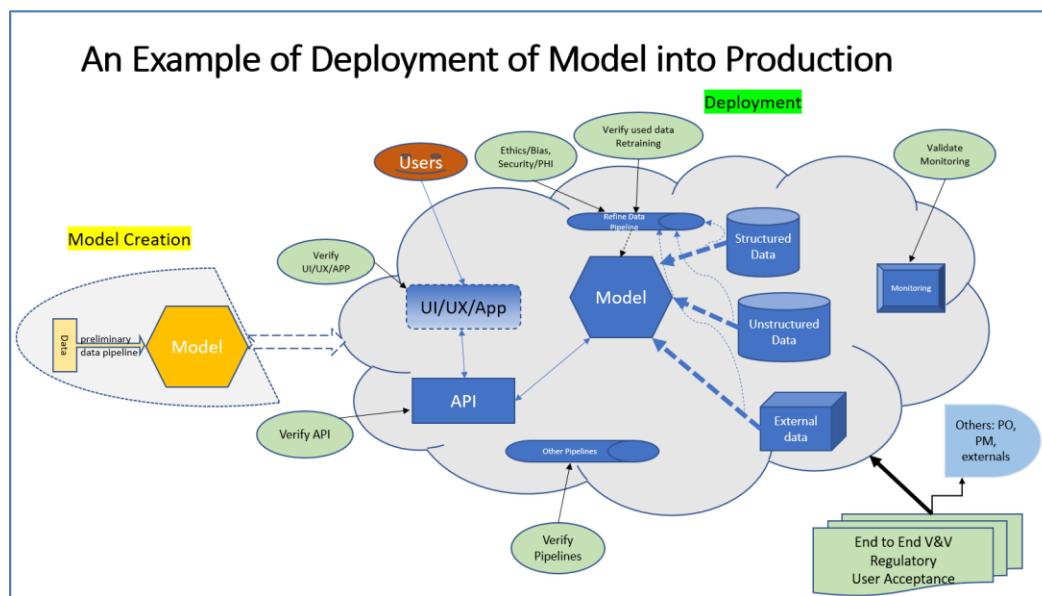
Mayo Clinic has been ranked the No. 1 hospital overall in the nation and top-ranked in fourteen specialties by U.S News and World Report for seven consecutive years.

The Center for Digital Health (CDH) creates innovative digital experiences, aligned with Mayo Clinic's mission to improve the health and well-being of people worldwide, CDH collaborates with partners to establish innovative ways to yield greater value, agility, and effectiveness.

CDH AI Engineering team takes responsibility for deploying models into real production environments. Those models can dramatically reduce the time spent by clinic users who are reviewing patients' history data and information. They also help clinic users make predictions and plan for patients more efficiently and properly. The CDH AI Engineering team is making a significant contribution to improving the quality of care at Mayo Clinic. Our work is helping to save lives and improve the lives of patients.

As a Verification and Validation Engineer on the AI Engineering team at Mayo Clinic, I work closely with MLOps engineers, data scientists, data engineers, IT engineers, and medical informatics specialists to ensure that the models that are deployed in production are safe and effective. I work across the entire lifecycle of a model, from development to deployment to monitoring, to ensure that the model meets the needs of the users and the requirements of the production environment. My work includes developing and implementing verification and validation plans, collaborating with stakeholders to understand the requirements, verifying and validating the API, UI/UX, applications, pipelines, bias, compliance, and other items that support model running in production, and focusing on end-to-end testing, monitoring, reporting, and retraining.

1.1 What exactly do AI Engineering and V&V do for deploying models into production environments?



A Basic Model Deployment

Model Creation

When data scientists build models, they take care of data validation with data engineers, model creation, training, and testing to finalize a model that is ready for deployment, and then pass it to our team, the AI Engineering team.

Model Deployment

To deploy the model into the real production environment, our MLOps Engineers build the following items:

- Data pipelines/API: get proper data from different databases including clinic data from the clinic database, clinic notes, documents, and lab results from the Clinic document database, and external data from external databases.
- Other pipelines: Integration pipeline, deployment pipeline, CI/CD pipeline that are necessary for deployment work.
- UI/UX applications: The end clinic users can use the final product on a web browser, and they are normally built by UI engineer with UX designer.

Verification and Validation

V&V Engineers start with simple API and UI testing and finish with end-to-end testing and monitoring.

Here are some examples:

API Testing	UI Testing	End to End (Full Stack) V&V
Functional	Functional	Data Flow
Data formatting and defaults (based on data cards)	Usability	Data bias
API Performance	PHI	Use cases
Negative/Error handling	Security	Multi-users
PHI	UI/UX standards	End to end performance
Scalability	Basic UI Performance	Differential
Stability	Negative/error handling/messages	Compliance
Simple Integration	Testable and automatable	Model retaining and monitoring

Monitoring, reporting, and retraining will continue in production after we are done with deployment work.

1.2 How to design verification and validation testing datasets for end-to-end testing?

Those are some important items to consider when designing testing datasets for Verification and Validation (V&V) testing, especially for end-to-end testing.

- How the model works: This includes understanding the model's input data, its output data, and how the model makes predictions.
- Data that the model needs: This includes the type of data, the format of the data, and the amount of data that the model needs to make accurate predictions.
- Data range: This includes the minimum and maximum values of the data, as well as the distribution of the data.
- Data bias: This includes understanding what data can cause model bias to help with creating data that can reduce and remove those bias.
- The relationship between all data: This includes understanding how the different pieces of data are related to each other, and how they affect the model's predictions.
- How MLOps Engineers make the model live in production: This includes understanding the production environment, the infrastructure that the model will run on, and the monitoring tools that will be used to track the model's performance.
- Data source: This includes understanding where the data comes from, how it is collected, and how it is cleaned and prepared for use in the model.
- Data flow: This includes understanding how the data flows through the system, from the data source to the model to the production environment.

By considering these items, you can design testing datasets that will help to ensure that the model is working correctly and that it is meeting the needs of the users.

Verification and Validation (V&V) testing datasets need to be designed to cover all necessary combinations of data points and different data flows. This is to ensure that the model can handle all possible scenarios and that it is not biased toward any particular set of data.

Here are a couple of specific examples of how V&V testing datasets can be designed to cover all combinations of data points and different data flows:

- The testing datasets can be designed to include all possible combinations of input data. For example, if the model is a medical diagnosis model, the testing datasets could include all possible combinations of symptoms, test results, and patient demographics.
- The testing datasets can be designed to include different data flows. For example, the testing datasets could include data that is collected from different sources, such as electronic health records, patient surveys, and laboratory tests.

For example, one of projects that I worked on is to develop a patient-focused and physician-driven diagnostic model and visualization framework for Acute Leukemia or MDS (Myelodysplastic Syndromes) patients.

Myelodysplastic Syndromes (MDS) are a group of diverse bone marrow disorders in which the bone marrow does not produce enough healthy blood cells. MDS is often referred to as a “bone marrow failure disorder”. MDS is primarily a disease of the elderly (most patients are older than age 65), but MDS can affect younger patients as well.

The model uses structured data: the patient's most recent bone marrow test data, cytogenetic test data, and genomic test data to calculate IPSS-M and IPSS-R risk scores and levels. These risk scores and levels can be used to help physicians make decisions about treatment options for patients with MDS.

Clinic users can also modify the data based on unstructured data: clinic documents and CBC (Complete Blood Count) test results. This allows for a more comprehensive assessment of the patient's condition and can help to improve the accuracy of the risk scores and levels.

The data, clinic documents, and CBC test results can be internal or external.

Note 1: The model is an open-source model and is used in the Medicare industry popularly.

Note 2: The model is used for adult patients, who are older than 17 years old.

Note 3: Risk score and levels are not impacted by gender.

IPSS-M (Molecular International Prognostic Scoring System) Levels <-> Scores:

IPSS-M Level	IPSS-M Score
VL	<= -1.5
L	>-1.5 to -0.5
ML	>-0.5 to 0
MH	>0 to 0.5
H	>0.5 to 1.5
VH	>1.5

IPSS-R (Revised International Prognostic System) Levels <-> Scores:

IPSS-R Level	IPSS-R Score
Very Low	<= 1.5
Low	>1.5 to 3
Intermediate	>3 to 4.5
High	>4.5 to 6
Very High	>6

Structure data is used for calculating IPSS-M & IPSS-R scores and levels:

Data type	Data	Data values	IPSS-M R(Required)/ O(Optional)	IPSS-R R(Required)/ O(Optional)
Bone Marrow	Bone Marrow Blasts	0-30%	R	R
	Hemoglobin	4-20 g/dL	R	R
	Platelet count	0-2000 le9/L	R	R

	Absolute Neutrophil Count	0-15 le9/L	O	O
	Age	18 to 120 years	O	O
Cytogenetics	Presence of del(5q)	Yes/No	R	N/A
	-7/del(7q)	Yes/No	R	N/A
	-17/del(17p)	Yes/No	R	N/A
	Complex Karyotype	Yes/No	R	N/A
	Cytogenetics Category	Very good, Good, Intermediate, Poor, Very Poor	R	R
	Number of TP53 mutations	0, 1, 2+	R	N/A
Genomic	Loss of heterozygosity at TP53 locus	No, Yes, N/A	R	N/A
	FLT3 Mutations includes MLL PTD	No, Yes, Not Assessed	R	N/A
	FLTs ITD or TKD	No, Yes, Not Assessed	R	N/A
	13 Genes (Individual weighs)	Non-mutated, Mutated, Not Assessed	R	N/A
	15 Genes (number of residual mutations)	Non-mutated, Mutated, Not Assessed	R	N/A

Examples of Testing Datasets

The testing datasets need to be comprehensive enough to test all different risk scores and levels, and different data flows. This means that the datasets need to include patients with all different types of data, including patients with complete data and patients with incomplete data.

Including patients with incomplete data is important because it allows the project to test how the model handles imperfect data. This is important because in the real world, patients will not always have complete data. The project needs to be able to handle imperfect data correctly in order to be accurate and reliable.

Here are some specific examples of how the testing datasets could be used to test different risk scores and levels, and different data flows:

- The datasets could be used to test how the model calculates risk scores for patients with different types of data. For example, the datasets could be used to test how the model calculates risk scores for patients with complete data, patients with incomplete data, and patients with no data.
- The datasets could be used to test how the model handles different data flows. For example, the datasets could be used to test how the model handles data that is entered in different ways, data that is entered in different formats, and data that is entered with errors.

- The datasets could be used to test risk level categorization using the risk scores, especially using the end points of each score ranges.
- The datasets could be used to test bias by using different ages and genders that impact risk scores and levels and possible races, locations, which should not impact risk scores and levels)
- The datasets could be used to test data pipelines by having data from different data sources including internal/external structured and unstructured data.

By including patients with all different types of data, the testing datasets can be used to test the model's accuracy and reliability in a variety of situations. This will help to ensure that the model is able to provide accurate and reliable risk scores and levels for patients with MDS.

Some examples of testing datasets not including entries intentionally missing some data or no data that is a part of testing datasets.

Age	IPSS-R Risk Level Score	IPSS-M Risk Level Score	BMB	Hemoglobin (g/dL)	Platelet	ANC (1e9/L)	del(5q)	del(7q)	del(17p)	Complex Karyotype	Cyto Category	TP53 Mutation Count	Max TP53 VAF	TP53 locus LOH	Missing Genes	Most Recent BoneMarrow Test	Most Recent Cytogenetic Test	Most Recent Genomic Test	CBC Result	
47	Intermediate 4	Very High 1.87	11%	12	222	5	fasle	fasle	fasle	fasle	Good	2 or more	4%	true	TP53, TPS3loh, MLL_PTD	None	Internal/External/None	Internal/External/None	Internal/External/None	
18	High 6	Very High 4.26	0%	4	0	0	fasle	True	True	fasle	Poor	2 or more	4%	true	TP53, TPS3loh, MLL_PTD	None	Internal/External/None	Internal/External/None	Internal/External/None	
24	Intermediate 4	Moderate Low -0.30%	13	1001	9.8	fasle	fasle	fasle	fasle	Good	1	23%	false	TP53	None	internal/External/None	internal/External/None	internal/External/None	internal/External/None	
51	Low 3	High 1.22	30%	12	222	5	fasle	fasle	fasle	fasle	Very Good	2 or more	80%	true	TP53, TPS3loh	None	internal/External/None	internal/External/None	internal/External/None	internal/External/None
30	Low 3	N/A -0.86	16%	12	1000	6	fasle	fasle	fasle	fasle	Very Good	1	2%	N/A	TP53loh	internal/External/None	internal/External/None	internal/External/None	internal/External/None	
72	Very High 7	Very High 7.1	15%	12	1000	6	fasle	fasle	true	fasle	Very poor	1	31%	N/A	BCORL1, CEBPA, GATA2, IDH1, NF1, PHF6, PPM1D, PTPN11, STAG2	TP53loh	internal/External/None	internal/External/None	internal/External/None	internal/External/None
120	Very Low -3.23	Very Low -3.23	3	18.2	1073	8.4	fasle	fasle	true	fasle	Very Good	0	0	false	NRAS	None	internal/External/None	internal/External/None	internal/External/None	internal/External/None
17	Intermediate 4	Moderate Low -0.30%	13	1001	9.8	fasle	fasle	fasle	fasle	Good	1	23%	false	TP53	None	internal/External/None	internal/External/None	internal/External/None	internal/External/None	
121	High 6	Very High 4.26	0%	4	0	0	fasle	True	True	fasle	Poor	2 or more	4%	true	TP53, TPS3loh, MLL_PTD	None	internal/External/None	internal/External/None	internal/External/None	internal/External/None

Tools that help with designing testing datasets

- Public Calculators**

For this project, I used public IPSS-M and IPSS-R calculators to help me design testing datasets that use different data combinations to generate different risk scores and levels.

This was a time-consuming process, as I had to manually figure out the data combinations based on risk level and score. However, the effort was worth it, as it allowed me to create a comprehensive testing dataset that will help to ensure the accuracy and reliability of the model.

- ChatGPT**

Two methods are used to generate testing datasets using ChatGPT:

- Generated Knowledge Prompting

We ask to create an artificial data set with corresponding IPSS-M scores and then IPSS-R scores by using the following example of prompts sequence with not filling any real patient data:

Do you know what an IPSS-M score is?

- Return: Factors, description of necessary data

Can you create a data set containing these factors that correspond with varying IPSS-M scores from very low to very high?

- Return: Factors are then compiled into a data set with corresponding IPSS-M score.

Can you then calculate IPSS-M scores given relevant data?

- Return: required data needed to calculate.

Finally, create data for those values so that the corresponding IPSS-M scores cover the range of possible values.

- Return: completed data table.

Findings: ChatGPT has limited knowledge of IPSS-M and R scores, and more focus on bone marrow test data.

Issue: scores sometimes are not right. For example: IPSS-M score should be -0.38, but get 1 from ChatGPT. It probably due to aggressive rounding or a simplified score calculation method.

Here is an example of a data set that ChatGPT created:

Patient	Blasts (%)	Cytogenetic Abnormalities	Hemoglobin (g/dL)	Platelet Count ($\times 10^9/L$)	Absolute Neutrophil Count ($\times 10^9/L$)	del(5q)	del(7q)	del(17p)	Complex Karyotype	Cytogenetic Category	Mutated Genes	Missing Gene Variants	IPSS-M Qualification	IPSS-R Qualification	IPSS-R Score	IPSS-M Score
1	18	2	8	120	0.6	TRUE	FALSE	FALSE	TRUE	Intermediate	TP53, FLT3	IKZF1	H	Intermediate	2	0.5
2	7	1	15	500	1.2	FALSE	TRUE	TRUE	FALSE	Very Poor	NRAS, JAK2	-	ML	Poor	3	1
3	2	0	4	800	0.2	TRUE	TRUE	FALSE	TRUE	Very Poor	CEBPA, FLT3	-	VH	Very Poor	4	0.5
4	12	0	18	300	0.8	FALSE	FALSE	TRUE	FALSE	Very Good	-	NPM1, DNMT3A	ML	Good	0	-1
5	4	1	11	1000	0.4	TRUE	FALSE	FALSE	TRUE	Good	-	ASXL1	H	Good	0	-1
6	30	2	5	1800	0.7	FALSE	TRUE	TRUE	TRUE	Very Poor	TP53, FLT3	RUNX1	VH	Very Poor	3	1.5
7	15	0	9	1500	1.5	TRUE	TRUE	FALSE	TRUE	Intermediate	-	ETV6, TET2	H	Intermediate	2	0.5

- Few-Shot Prompting

We ask ChatGPT to replicate existing patterns in pre-existing data that is fake data. We label the original data set as a validated pattern. The pattern is then analyzed by the model and replicated first by generating random data for each column (within given ranges) and second by calculating IPSS-M and IPSS-R risk score for the created data.

Could you analyze the original data and then create random data that adheres to that pattern?

Return: Data with corresponding IPSS-M scores.

Findings: We need to ask to generate randomized data to avoid the exact same dataset as the original.

Issues: Risk level data is always off by one full qualification step. For examples, Low should be Moderately Low and High should be Very High. It might because data is generated only based on 10 original datasets. Believe that the larger number of datasets in the original datasets will increase the accuracy.

Here is an example of the original dataset:

IPSS-R Score	IPSS-M Score	BMB	Hemoglobin (g/dL)	Platelet (date)	ANC (1e9/L) (date)	del(5q)	del(7q)	del(17p)	Complex Karyotype	Cyto Category	TP53 Mutation Count	Max TP53 VAF	TP53 locus LOH		Missing Genes
Intermediate 4	Very High 1.87	11%	12	222 31-Mar-2023	5 31-Mar-2023	false	false	false	Good	2 or more	4%	true			None
High 6	Very High 4.26	0%	4	0	0	false	True	True	Poor	2 or more	4%	true	TP53, TP53loh, MLL_PTD		None
Intermediate 4	Moderate Low -0.41	30%	13	1001 31-Mar-2023	9.8 31-Mar-2023	false	false	false	Good	1	23%	false	TP53, TP53loh, MLL_PTD		None
Low 3	High 1.22	30%	12	222 31-Mar-2023	5 31-Mar-2023	false	false	false	Very Good	2 or more	80%	true	TP53		None
Low 3	N/A -0.86	16%	12	1000 31-Mar-2023	6	false	false	false	Very Good	1	2%	N/A	TP53loh		TP53
Very High 7	Very High 7.1	15%	12	1000 31-Mar-2023	6	false	false	true	Very poor	1	31%	N/A	TP53, FLT3, CBL, DNMT3A, EZH2, KRAS, NPM1, NRAS, SF3B1, U2AF1, BCOR, BCORL1, CEBPA, GATA2, IDH1, NF1, PHF6, PPM1D, PTPN11, STAG2		TP53loh
Very Low -3.23	Very Low -3.23	3	18.2	1073	8.4	false	false	true	Very Good	0	0	false			None
Intermediate 4	Moderate Low -0.41	30%	13	1001 31-Mar-2023	9.8 31-Mar-2023	false	false	false	Good	1	23%	false	NRAS		None
High 6	Very High 4.26	0%	4	0	0	false	True	True	Poor	2 or more	4%	true	TP53, TP53loh, MLL_PTD		None

An example of a generated data set:

Patient	Bone Marrow Blast (%)	Hemoglobin (g/dL)	Platelet Count (nanoliters)	ANC	del(5q)	del(7q)	del(17p)	Complex Karyotype	Cyto Category	TP53 Mutation Count	Max TP53 VAF	TP53 locus LOH	Mutated Genes	Missing Genes	IPSS-M Qualification	IPSS-R Score
1	19%	14	1900	3	False	True	False	False	Good	>=2	6%	True	TP53, TP53loh, RUNX1	None	H	High Risk
2	4%	10	1500	0	False	False	False	False	Poor	0	0%	False	None	None	VL	Very Low Risk
3	27%	12	1000	6.8	False	False	False	False	Good	1	15%	False	TP53	None	L	Low Risk
4	22%	11	1800	7	False	False	False	False	Very Good	2 or more	85%	True	TP53, TP53loh, MLL_PTD	None	ML	Intermediate Risk
5	13%	12.5	1200	6.5	False	False	True	False	Very Good	1	5%N/A		FLT3, CBL, DNMT3A, EZH2, KRAS, NPM1, TP53TP53loh	ML		Intermediate Risk
6	16%	13	1000	5.5	False	False	False	False	Very Poor	1	25%N/A	RUNX1	TP53loh	H		High Risk
7	2%	16	900	9.2	False	True	True	False	Very Good	0	0	False	NRAS, TP53	None	ML	Intermediate Risk
8	28%	14	1100	8.5	False	False	False	False	Good	1	18%	False	TP53	None	L	Low Risk

Overall, Generated Knowledge Prompting: creating model generated context information improves prompt efficiency and Few-Shot Prompting: Providing expected output improves output format accuracy. The combination of those two methods can lead to better testing datasets.

ChatGPT can help to create testing datasets much quicker. However, due to ChatGPT's limited knowledge and the fact that it does not have the latest data, we still need to use the public calculator to verify and confirm the datasets. Hopefully, when ChatGPT has the latest data, the accuracy of creating datasets can be improved.

1.3 How to create testing datasets?

There are different ways to create testing datasets with advantages and disadvantages.

Creating fake data in a local database

Advantages:

- Does not require dependency on other teams.
- Can be easily modified when needed.
- Can be used to design and create common data sets for future similar projects.

Disadvantages:

- Takes a significant amount of time to understand the format and content of clinic notes, documents, medications, and lab results in order to create realistic fake data.
- The data flows in a local database may not be the same as the data flows in the real production environment, which can impact end-to-end testing.

Overall, creating fake data in the local database can be a quick way to create testing datasets to start V&V, but the disadvantages of creating fake data in a local database should be carefully considered before using this approach.

Creating test patients in Mayo databases

To protect patient health information (PHI), using real patients in testing and automation is not allowed that is important for compliance with HIPAA and other regulations, so we need test patients based on the testing dataset we design and to meet verification and validation purposes.

Test patients are fake patients with fake data but created in Mayo real testing systems/databases. It required to follow the specific process and work with multiple teams.

For special reasons and special cases, test patients can be created in production with going through much more difficult approval process.

Creating test patients while following the Mayo process in Mayo databases is important.

Advantages:

- Protect PHI.
- Use real data flow.
- Get accurate data created by experts that includes clinic documents, medications, and lab results.

Disadvantages:

- Requiring the cooperation of other teams.
- May be difficult to get priority for test patient creation from other teams.

Overall, creating test patients in Mayo Databases can be a valuable tool for testing models. However, it is important to be aware of the advantages and disadvantages of this approach before using it.

1.4 Challenges I had during test patient creation

Finding test patients

Before creating test patients, we always need to find out if there are any existing test patients in systems to avoid unnecessary work for all teams, but finding test patients with characteristics that align with the functionality under test can be challenging.

The challenges are:

- Creating queries to find data is challenging for those unfamiliar with data structures and without a medical background.
- Finding test patients takes time due to the large size of our database and the time it takes to run each query.
- Finding common test patients that have all mandatory data is time-consuming since some of them may not have all of the mandatory data.

Overall, finding test patients can be a challenging task, especially in large databases. However, there are a few things that can be done to make the process easier. These include:

- Working with a data engineer to create efficient queries.
- Indexing the database to improve the performance of the queries.
- Using tools that can help to search through the database more efficiently.

By taking these steps, it is possible to make the process of finding test patients more efficient and less time-consuming.

Creating test patients

During test patient creation, there are some challenges too:

- Incomplete understanding of the data used by the model can lead to missing data for test patients.
- Limited knowledge of multiple data sources can slow down the process of test patient creation.
- Inconsistency in lab test results from multiple CIS can make test patient creation challenging.

Overall, creating test patients can be a challenging task. However, there are a few things that can be done to make the process easier. These include:

- Use a data card to document the data that is used by the model. This will help ensure that the test patient creation tool has access to all of the necessary data.
- Learn from database experts in the organization to understand the data schema. This will help you work with the right resources to create realistic and accurate patient data.
- Develop a data mapping process to translate data from one format to another. This will help ensure that the data from the different data sources can be used by the test patient creation tool.

Note: All test patients created based on testing datasets are stored and managed by Mayo Epic testing systems and environments. To modify test patients for good reasons, we will have to submit requests to work with multiple teams to get requests completed.

2. Conclusion

Our end-to-end testing can ensure that the model is deployed into production correctly, can help to ensure that the model works correctly in the real world, as it is exposed to different data and different environments, and help to ensure that the model is secure and compliant with regulations. So, the proper testing datasets are the key to end-to-end testing.

To design the correct testing datasets, we need to define the scope of the testing, need to understand how the model works, how the model lives in production environments, what data the model uses, the relationship between data, how data flows, and how MLOps engineers deploy the model into production environments. Bias, security, regulations, retraining, and monitoring must be considered as well.

Use a variety of data sources to create the testing datasets. This will help to ensure that the model is not overfitting to any particular dataset.

Use a variety of data ranges to create the testing datasets. This will help to ensure that the model is not biased towards any particular range of values.

Use a variety of data relationships to create the testing datasets. This will help to ensure that the model is able to handle different types of relationships between the data.

Use a variety of data flows to create the testing datasets. This will help to ensure that the model is able to handle different types of data flows.

Choosing the proper way to build testing datasets is based on different types of tests. Building testing datasets correctly from the beginning of the project with appropriate resources would improve the effectiveness of the testing process and the quality of the project.

Necessary help from medical experts is needed and collaboration is critical to the successful creation of testing datasets.

3. References

- IPSS-M Risk Calculator: <https://mds-risk-model.com/>
- IPSS-R Basic Calculator: <https://www.mds-foundation.org/ipss-r-calculator/>
- ChatGPT Model: GPT-4 <https://openai.com/research/gpt-4>
- GPT-4TechnicalReport:
<chrome-extension://efaidnbmnnibpcajpcgjclefindmkaj/https://cdn.openai.com/papers/gpt-4.pdf>

4. Acknowledgements

John Skiffington Mayo Clinic CDH AI Engineering Director

Eric Will Mayo Clinic AI/ML Software Architect and project technical lead

Max Comfere Mayo Clinic AI Engineering Summer Intern (ChatGPT researcher)

Quality & Risk Management Challenges When Acquiring Enterprise Systems Revised in the Age of Cloud and AI

Jack McDowell & Ying Ki Kwong

jackmcdowell@gmail.com

Abstract

This paper investigates the significant shifts in IT project management paradigms necessitated by the adoption of artificial intelligence (AI) and cloud technologies, with a focus on large-scale system acquisitions in the public sector. Initially, we delve into the nuances of Requirement gathering and refinement in this new paradigm, emphasizing how AI and cloud technologies necessitate an iterative and dynamic approach to requirements gathering. We then scrutinize Organizational Readiness, highlighting how enterprises must develop comprehensive strategies to prepare their infrastructure and workforce for these advanced technologies. Contract Alignment with System Development Life Cycle (SDLC) is also considered, stressing the need for flexible and adaptive contractual frameworks that account for the unique attributes of AI and cloud-based systems. Subsequently, we underscore the need for Business-Aware Testing, demonstrating how thorough validation processes must ensure the functionality, security, and reliability of AI and cloud systems. The crucial role of Organizational Dynamics is examined, stressing the human factors that may influence the successful adoption of these technologies. The paper also highlights key Lessons Learned from Testing, pointing out the importance of maintaining robust testing protocols despite the novel challenges posed by AI and cloud implementations. We then delve into the ethical implications of adopting AI and cloud technologies, elucidating the need for robust ethical frameworks and governance mechanisms to ensure responsible usage. Illustrative case studies from the public sector are then presented, highlighting practical applications and potential pitfalls of implementing AI and cloud technologies in the public sphere. The paper concludes by reiterating the imperative for a multi-faceted approach to managing large IT projects in the era of AI and cloud, emphasizing the need for a judicious balance between the adoption of innovative technologies and maintaining robust standards of functionality, security, and ethics.

Biography

Jack McDowell is the Statewide QA Program Manager with the State of Oregon. The Statewide QA Program provides Quality Assurance services for Oregon's Major IT Projects, and Quality Assurance consultation to Oregon State Agencies. Before this, he was a web developer and the chief editor of a community newspaper in Arlington, Virginia. Originally from Buenos Aires, Argentina where he lived before attending college in the US. He holds a master's degree in political science from the University of Oregon and a certification in ITIL.

Ying Ki Kwong is the E-Government Program Manager with the State of Oregon. Prior to this, he was Statewide QA Program Manager and IT Investment Oversight Coordinator with the office of the State CIO and was Project Office Manager of the Medicaid Management Information System Project with the Oregon Department of Human Services. In the private sector, he held R&D, marketing, and general management roles in large companies and startups. He holds a PhD degree in applied physics from Cornell University and certifications in project management (PMP), ITIL, and IT Service Management.

1 Introduction

Today's large-scale organizations face intricate challenges when acquiring enterprise systems. Among the varied complexities, they must ensure efficient risk management and quality assurance to execute successful system implementations. Agile methodologies have emerged as a popular solution to these challenges, as discussed in our PNSQC 2018 paper (Kwong & Lew, 2018). However, the continuously evolving landscape of information technology has introduced two additional transformative elements: cloud computing and artificial intelligence (AI) (Sills, 2017; Crawford, 2014).

The aim of this paper is to delve into the multi-faceted implications of integrating Agile methods with cloud computing and AI technologies during the acquisition of enterprise software systems. This integration embodies a paradigm shift in the realm of software development, deployment, and utilization, which necessitates a deeper exploration of its effect on diverse aspects of system acquisition processes (Armbrust et al., 2010).

Beginning with the discovery and refinement of system requirements, the paper navigates through the prerequisites of organizational readiness, and the importance of aligning contracts with the Software Development Life Cycle (SDLC). Emphasizing the role of business-aware testing in assuring quality. The paper also examines the considerable influence of organizational dynamics on project outcomes (Crawford, 2014).

The paper then outlines significant lessons learned from testing processes, shedding light on how these processes may need to be tailored when AI and cloud computing technologies are in scope. Following these discussions, the paper will conclude with an in-depth analysis of the crucial takeaways for enterprises aiming to incorporate Agile, AI, and cloud computing in their system acquisition strategies (Brynjolfsson & McAfee, 2014; Marston et al., 2011).

The exploration of these aspects is intended to provide comprehensive insights for decision-makers navigating the complex pathways of modern system acquisitions, with the hope of encouraging more effective enterprise IT investments.

2 Requirements Discovery and Refinement

Acquiring enterprise systems poses challenges of many facets, starting with requirements discovery and refinement. The underlying processes are essential for any software project, providing a foundation for all aspects of a project's life cycle to follow (Kotonya & Sommerville, 1998).

Historically, methodologies like Joint Application Development (JAD) and the Rational Unified Process (RUP) have played crucial roles in requirements gathering. JAD emphasizes collaboration, using team discussions and meetings to gather insights and ensure comprehensive understanding of user needs (Wood & Silver, 1995). On the other hand, RUP uses a structured, iterative approach, heavily reliant on the Unified Modeling Language (UML) for requirements and related analysis. RUP's focus on early prototyping is particularly notable, helping teams identify and address project risks early in the development life cycle (Kruchten, 2003).

The Agile methodology, while newer, prioritizes continuous collaboration. It brings together stakeholders and developers to work in iterative cycles, delivering value incrementally (Cohn, 2004). Introducing AI, especially Machine Learning (ML), adds complexity to this process. ML's capability to process large datasets and identify patterns can enhance the efficiency of requirements gathering. However, using AI requires adjustments in the traditional methods and tools used in this phase (Jordan & Mitchell, 2015).

Cloud computing offers new opportunities and challenges for software development and deployment, emphasizing scalability, flexibility, and cost-effectiveness (Marston et al., 2011). With this technology, there's a need to consider additional requirements related to performance, scalability, security, and data management early on (Mell & Grance, 2011).

In essence, the application of AI and cloud technologies expands the canvas of requirements discovery and refinement, inviting new complexities while also offering opportunities for efficiency and innovation. This requires organizations to rethink their traditional approaches, considering not just functional requirements, but also AI-specific and cloud-specific needs, as part of their enterprise system acquisition strategy. By combining traditional methodologies with new tools and processes, organizations can create effective strategies for enterprise system acquisition.

3 Organizational Readiness

With requirements generally understood, the readiness of an organization to adopt and exploit the advantages of emerging technologies such as cloud and AI becomes a determining factor for the successful acquisition of enterprise systems (Bharadwaj et al., 2013). Such readiness extends beyond mere technical preparedness, enveloping a variety of facets such as infrastructure, human resources, data management, security, and cultural readiness (Kwong et al., 2019 and Kappelman et al., 2018).

Traditional development models, prominently the Waterfall model, are characterized by their linearity and sequential phases, ranging from requirements definition to system design, implementation, verification, and maintenance (Royce, 1970). This systematic approach, while offering predictability, a structured framework, and predictable contracting models, can sometimes limit the nimbleness required to adapt to the rapid advancements and dynamic nature of AI and cloud technologies. For organizations steeped in traditional development practices, the journey to embracing AI and cloud solutions might necessitate not just technological upgrades but also a paradigm shift in their project management and development methodologies. This transition would demand a re-evaluation of organizational practices, workflow adjustments, and potentially, a cultural transformation to accommodate the speed, dynamism, and iterative nature of contemporary digital technologies. It emphasizes the reality that organizational readiness is as much about the willingness to evolve and adapt as it is about technical preparedness (Sommerville, 2011).

In the Agile framework, readiness typically refers to the preparedness of an organization to adopt Agile principles and methods, including its capacity to welcome iterative processes, flexibility, and constant collaboration (Dingsøyr et al., 2012). However, when considering the integration of AI and cloud technologies, organizational readiness takes on an even broader scope.

AI readiness involves developing competencies in data management and analytics, given that AI technologies heavily rely on the availability and quality of data for effective functioning (Chui et al., 2018). Organizations also need to consider the ethical and regulatory implications of AI, particularly when dealing with sensitive information (McDowell & Kwong, 2022 and Russell et al., 2015). This is coupled with the need for skilled personnel who are proficient in AI technologies, and an organizational culture that embraces AI-driven decision-making.

Concurrently, readiness for cloud adoption demands infrastructural preparedness and robust security measures. The transition to cloud platforms involves re-engineering traditional IT infrastructure, ensuring connectivity, and implementing measures to address the unique security challenges presented by cloud technologies (Brender & Markov, 2013). Here again, human factors play a crucial role, requiring training and awareness campaigns to educate employees about safe and efficient cloud practices.

Moreover, the intersection of AI and cloud technologies presents opportunities to enhance organizational readiness. Cloud platforms can provide the necessary computational resources for AI applications, while AI can help automate and optimize cloud infrastructure, a concept encapsulated in the emerging field of AIOps (Gartner, 2017).

In essence, the journey towards adopting Agile in conjunction with AI and cloud technologies requires comprehensive organizational readiness, which transcends traditional boundaries to encapsulate a multifaceted set of considerations. As such, readiness should be a strategic concern for organizations aspiring to successfully acquire and implement enterprise systems.

4 Contract Alignment with SDLC

With the swift rise of technologies like AI and cloud, aligning contractual agreements with the Software Development Life Cycle (SDLC) takes center stage (Haley, Laney, & Peter, 2018). Proper alignment not only ensures the meeting of project objectives but also systematically manages the distinctive risks these advanced technologies pose (Turner & Cochrane, 1993).

Cloud contracts, notably, must delve into the intricacies of data ownership, privacy, security, and adherence to local and international standards. Central to this is the establishment of well-defined Service Level Agreements (SLAs) that clarify expectations regarding service availability, downtime allowances, data recovery mechanisms, and swift incident response procedures (Katzen, 2010). Instead of micromanaging every technological facet, these contracts pivot towards overarching system performance. Service Level Agreements (SLAs) for cloud and AI services, therefore, underscore the fulfillment of business objectives, emphasizing outcomes over inputs. This translates to a commitment to ensuring that the entire system operates seamlessly to meet business needs, rather than guaranteeing the functioning of individual components (Katzen, 2010). It's a holistic approach, where accountability is tied to the system's overall delivery and effectiveness in serving business goals.

Contrast this with traditional, technology-centric contracts, especially prevalent in the Waterfall methodology. Here, every software component, subsystem, and license stand under scrutiny. Contracts often delineate the health, updates, and interplay of each component, ensuring they function in harmony (Royce, 1970). While this offers granular control, it might sometimes sidestep the broader business goals, focusing more on technological particulars than organizational outcomes.

In AI's realm, the landscape merges both models. Contracts touch upon the technology—transparency of algorithms, data handling protocols, and model retraining—but also stress the business end, addressing AI's efficacy in solving real-world business challenges (Duan, Gu, & Whinston, 2020). With AI's dynamic learning capabilities, there's an imperative to continually retrain models, address biases, and clarify intellectual property rights, especially concerning AI-generated content (Duan, Gu, & Whinston, 2020). Furthermore, AI contracts should elucidate on model accountability, longevity, and the ethical dimensions tied to its deployment (Raghavan et al., 2020).

In Agile SDLC, where fluidity and change are the norms, contracts demand flexibility. Unlike traditional Waterfall methodologies, where the project's scope, timeline, and cost are predominantly fixed (Royce, 1970), Agile projects champion evolution, iterations, and adaptability. Thus, Agile-based contracts must permit scope and deliverable adjustments, anchored to each development iteration. The core Agile tenet - continuous delivery of valuable software - should ideally be enshrined within the contract, offering incentives for timely deliveries (Beck et al., 2001). This melds the business-centric and technology-centric views, allowing flexibility in deliverables while emphasizing business outcomes.

Conversely, in traditional Waterfall projects, contracts often have a well-defined scope and clear boundaries, making them less adaptive to changes post-initiation. This rigidity can be both an advantage and a limitation, especially when integrating modern technologies like AI and cloud. Thus, a clear understanding of the project approach, be it Agile or Waterfall, is crucial when drafting contracts (Sommerville, 2011).

In sum, the evolution of contracts in the AI and cloud era presents a multifaceted tapestry, intertwining technology, business needs, and project management methodologies. Striking a balance between the broader business objectives and intricate technological specifications is both an art and a science, demanding nuanced comprehension and dexterous execution.

With the adoption of new technologies such as AI and cloud, an organization needs to ensure that the contract and Software Development Life Cycle (SDLC) are properly aligned to account for the unique characteristics these technologies introduce (Haley, Laney, & Peter, 2018). This alignment is critical in ensuring that the objectives of the project are met and that potential risks associated with these technologies are properly managed (Turner & Cochrane, 1993).

Cloud contracts, for instance, need to address issues related to data ownership, privacy, security, service levels, and compliance with local and international regulations. These contracts should also specify the cloud deployment models to be used (e.g., private, public, hybrid), the level of service availability, data recovery mechanisms, and incident response procedures (Katzan, 2010).

When AI components are involved, the contract should consider the transparency and interpretability of AI models, the handling and privacy of the data used to train these models, the responsibilities for the regular retraining of the models, and the handling of potential biases in AI decisions (Raghavan et al., 2020). Furthermore, issues related to intellectual property rights and ownership of AI-generated content should be explicitly addressed (Duan, Gu, & Whinston, 2020).

In Agile SDLC, contracts should be flexible to accommodate the iterative nature of the development process. Traditional contracts often predetermine the scope, cost, and timeline, which may not be suitable for Agile projects where these factors may evolve over time (Lindgren & Münch, 2012). Contracts in Agile projects should therefore allow for adjustments in scope and deliverables, contingent upon every development iteration. Furthermore, incentives for early and continuous delivery of valuable software - a key principle of Agile - could be integrated into the contractual framework (Beck et al., 2001).

In essence, aligning contracts with the SDLC in projects involving AI and cloud technologies in an Agile environment is a complex yet vital process. It involves a delicate balance of catering to the specific demands of these technologies, the flexibility of Agile, and the necessity of safeguarding the organization's interests.

5 Business-Aware Testing

With the integration of AI and cloud technologies into enterprise systems, the demands on software testing are evolving. Business-Aware Testing stands out as a refined approach to ensure that technological solutions not only function correctly but also align with overarching business objectives (Jalali & Wohlin, 2012).

5.1 Business Context:

Traditional vs. Business-Aware Testing: Traditional testing focuses on ensuring functionality, performance, and security (Pezze & Young, 2007). Business-Aware Testing extends its reach to encompass business goals, verifying not only if a system works but also if it serves the business effectively and delivers genuine value (Kwong & Lew, 2018 and Ali et al., 2017).

Business Alignment with AI: AI operates on a probabilistic model, contrasting with the deterministic outputs of traditional systems (Mitchell, 1997). Business-Aware Testing for AI assesses if AI decisions align with business strategies. For instance, while an AI recommendation engine may provide suggestions technically, the real test lies in its ability to enhance user engagement and drive conversions (Chen, Groves, Harrison, & Lin, 2022).

Business-Aware Testing in the context of AI and cloud-based systems is a crucial aspect to address when acquiring enterprise systems. It requires a different approach compared to traditional testing methods and needs to be designed to test not only the functionality but also the specific characteristics of AI and cloud technologies (Chen, Groves, Harrison, & Lin, 2022 and McDowell & Kwong 2022).

AI systems' behavior is inherently probabilistic and not deterministic, and it changes as the model learns from new data (Amershi et al., 2019). Therefore, traditional pass/fail test cases may not be adequate for

AI systems. Testing methods should focus on understanding the model's decision-making process, data biases, potential drift in model's behavior over time, and robustness to adversarial attacks (Pezoa et al., 2020).

For instance, considering defect metrics in AI-based systems, a key aspect to track is the performance metric that is most relevant to the specific AI task. This could be precision and recall for a classification task or Root Mean Square Error (RMSE) for a regression task (Provost & Fawcett, 2013). Moreover, the test data set should be representative of the real-world scenarios that the AI system will encounter to ensure its robustness and fairness.

As AI systems become more commonplace, ethical, and fair AI outcomes are increasingly important, as the decisions made by AI models become integral pieces of software systems. Business-Aware Testing must validate that AI outputs are ethically sound, unbiased, and aligned with organizational values (Bryson & Winfield, 2017 and McDowell, 2022).

Business Alignment with Cloud: Cloud solutions offer flexibility and scalability (Marston et al., 2011). Testing for cloud systems evaluates their readiness to adapt to varying business demands. With cloud's pay-as-you-go model, it's vital to ensure the solution aligns with business financial strategies (Mell & Grance, 2011).

Testing cloud-based systems, on the other hand, should include verification of scalability, resilience, security, and compliance with regulatory standards (Cai, Liu, & Zhou, 2016). Given the shared responsibility model in cloud, testing should verify proper configuration of security settings, including access controls, encryption, and logging. Load testing can verify if the system can handle expected user loads, and fault-tolerance testing can verify system's resilience to different types of failures (Chen et al., 2022).

The User Acceptance Testing (UAT) in this context would involve assessing the system's ability to meet business needs effectively in the cloud environment, and the AI model's ability to make decisions as expected by the business. In addition to traditional UAT methods, it might include tests to validate the explainability of AI decisions, especially in high-stake applications (Gilpin et al., 2018).

In conclusion, Business-Aware Testing in the AI and cloud context requires an understanding of these technologies and an ability to design and execute tests that assess their specific characteristics and implications on business operations.

6 Organizational Dynamics

The introduction of AI and cloud-based systems in an organization can significantly impact its dynamics, necessitating changes in both technological aspects and human factors. The dynamics of implementing these technologies often go beyond technical challenges and include factors such as organizational readiness, culture, skillsets, and stakeholder buy-in (Bresnick, 2020).

AI and cloud-based technologies can introduce new operational paradigms in the organization. The decision-making process may shift with AI algorithms providing insights or making autonomous decisions. This requires an understanding of these decisions from not only a technical perspective but also from ethical and legal perspectives, given that AI decisions can sometimes be hard to interpret and may be biased (Cath, 2018). Also, with data being a critical component of AI systems, effective data governance becomes a key part of organizational dynamics (Otjacques, Hitzelberger, & Naumann, 2007).

Similarly, a transition to cloud services alters the organization's relationship with its IT infrastructure. This shift requires changes in IT management practices and can also cause resistance from IT personnel due to the perceived loss of control over the infrastructure (Carcary, Renaud, McLaughlin, & O'Brien, 2020). This underscores the importance of strong leadership, communication, and change management during such transitions.

Staffing changes may be required as well, as both AI and cloud-based technologies require specific skills. For AI, this may involve hiring data scientists and AI specialists, while for cloud, this could mean hiring cloud architects and cybersecurity professionals (Bughin, Hazan, Ramaswamy, Chui, Allas, Dahlström, Henke, & Trench, 2018). Training existing staff can also be an important part of the organizational change process.

Moreover, both AI and cloud implementation should be driven by business needs and not just for the sake of adopting new technology. This is particularly true for AI projects, where the focus should be on solving specific business problems rather than simply adopting AI technology (Garcia, 2018).

Given these dynamics, organizations need to be proactive in managing these changes. This involves engaging with stakeholders early, setting clear expectations, fostering an organizational culture that is open to change, and continuously reviewing and adapting strategies based on feedback and learning (Cameron & Green, 2019).

7 Lessons Learned from Testing

The experiences garnered from testing AI and cloud-based systems offer several lessons, significantly enriching the insights obtained from conventional system testing. The uniqueness of AI and cloud technology necessitates a rethinking of testing strategies, presenting learning opportunities (Bertolino, 2020).

One significant lesson revolves around the unpredictability of AI systems. Traditional testing methods, designed for deterministic systems, are not fully compatible with AI. The probabilistic nature of AI, specifically machine learning models, can lead to unexpected outputs even for known inputs (Zhang, Huchuan, Zhang, & Guo, 2020). This behavior necessitates an ongoing and adaptive testing strategy to accommodate these dynamic changes and ensure that the system is performing as expected.

In terms of cloud-based systems, the major lesson is the criticality of network and security testing. Given the off-premises nature of cloud-based systems, network latency and security become primary concerns (Subashini & Kavitha, 2011). Regularly conducting penetration tests, vulnerability assessments, and security audits should be an integral part of the testing strategy for cloud implementations.

Moreover, both AI and cloud-based systems underscore the importance of data quality and the need for data-centric testing. AI systems are only as good as the data they're trained on, and cloud systems rely heavily on data for their operation (Chen, Mao, & Liu, 2014). Consequently, data verification, data integration testing, and data privacy checks are crucial when testing these systems.

Another vital lesson is the value of adopting a "continuous testing" approach. Given the rapid pace of changes in AI algorithms and cloud technologies, testing cannot be a one-time activity. Implementing continuous integration/continuous deployment (CI/CD) practices and employing automated testing tools can ensure the ongoing reliability of these systems (Duvall, Matyas, & Glover, 2007).

Lastly, the user-centric nature of these technologies mandates a more business-aware testing approach. The tests should validate not only the technical requirements but also the business requirements, usability, and overall user experience.

By drawing from these lessons, organizations can refine their existing testing strategies to better align with the peculiarities of AI and cloud technologies. For instance, normal use of System Integration Testing (SIT), User Acceptance Testing (UAT), and Pilots (Kwong & Lew, 2018) can be tailored:

System Integration Testing (SIT): For complex cloud and AI systems, various components or subsystems are often developed in parallel. SIT ensures that these subsystems and components work cohesively when integrated. Given the nature of AI and cloud systems, integration issues may arise from inconsistent data flows, incompatible interfaces, or unpredictable AI model responses when interfaced with other systems.

In cloud environments, integration becomes even more critical due to potential dependencies on third-party services or platforms. SIT in such contexts not only checks the interactions between subsystems but also ensures that any external services, APIs, or databases integrate seamlessly (Leung & Chan, 2003).

User Acceptance Testing (UAT): UAT is crucial for any system, but its importance is amplified in AI and cloud domains due to the user-centric nature of these technologies. It focuses on validating the system against the user and business requirements. Given the inherent unpredictability of AI, users may often face unforeseen results. UAT provides a platform to validate whether such unpredictable outcomes are still within acceptable and understandable bounds for the end users. In cloud systems, UAT also tests the system's accessibility, scalability, and response times from various locations, ensuring a consistent user experience across the board (Perry, Votta & Wade, 2000).

Pilots: Pilot testing is essentially a limited release used to gain feedback before a full-scale launch. For AI and cloud systems, pilots are indispensable. Given the probabilistic nature of AI, pilots can help in observing the system in a more controlled real-world setting, identifying any anomalies or unexpected behaviors. Furthermore, they provide insights into real-world user interactions, potential training needs, and areas of improvement in user experience. For cloud systems, pilots can help in identifying bottlenecks, especially when accessed by users from different geographic locations, and provide insights into any latency, accessibility, and dynamic resource allocation issues. They also assist in understanding how well the cloud infrastructure scales with increased user load, and if any additional resources or optimizations are required before a full-scale deployment (Kruchten, 1995).

In summary, while AI and cloud technologies bring about their unique challenges, tailored testing approaches like SIT, UAT, and pilots provide a robust strategy to address these challenges. By integrating these approaches with the overarching lessons from AI and cloud testing, organizations can ensure the delivery of reliable, user-friendly, and high-performing systems.

8 AI and Cloud Ethical Considerations

As we incorporate artificial intelligence (AI) and cloud computing into our digital ecosystem, we must also acknowledge and address the ethical considerations that arise in the context of these advanced technologies. AI and cloud technologies undoubtedly bring transformative potential, but their deployment also raises issues around data privacy, security, and AI fairness and bias that need to be thoughtfully addressed (Cath, Wachter, Mittelstadt, Taddeo, & Floridi, 2018, McDowell & Kwong 2022).

One of the primary ethical concerns associated with cloud technology is data privacy. As the custodian of user data, organizations need to ensure that the privacy of this data is safeguarded (Martínez-Cámarra, Martín-Valdivia, Ureña-López, & Montejo-Ráez, 2014). This responsibility extends to compliance with the relevant data protection and privacy regulations in the jurisdictions where they operate. The global nature of the cloud can make compliance with these regulations complex, but it remains a critical aspect of ethical cloud usage.

Additionally, as organizations employ AI technologies, they must also address the issue of AI bias. AI systems learn from the data they are trained on, and if these data sets reflect biased human decisions, the AI could inadvertently perpetuate these biases (Mittelstadt, Allo, Taddeo, Wachter, & Floridi, 2016). This can lead to unjust outcomes and discriminatory practices, potentially causing harm to individuals or groups. It is crucial, therefore, for organizations to consider methods to mitigate such biases and ensure fairness in their AI systems.

Finally, security considerations are a significant concern in both AI and cloud technology deployment. Breaches can lead to significant losses, both financial and in terms of reputation. Thus, the adoption of stringent security protocols, continuous monitoring, and robust response plans is imperative in managing these risks (Pfitzmann & Hansen, 2010).

In summary, while AI and cloud technologies offer tremendous potential for organizational efficiency and innovation, organizations need to be cognizant of and prepared to address the ethical considerations they

bring. A responsible approach to AI and cloud usage is not just about integrating these technologies into the organization's processes and systems, but also about taking seriously their associated ethical implications.

9 Conclusion

In this paper, we have sought to explore the complex interplay of artificial intelligence and cloud computing technologies in software testing. We started with the importance of requirements discovery and refinement, considering the rapid pace of technological advancement, the expanding possibilities brought about by AI and cloud technologies, and the evolving demands of businesses and users. Our exploration of the role of organizational readiness illustrated the need for organizations to adapt their structures, processes, and culture to fully leverage these advanced technologies.

We touched upon the alignment of contracts with the Software Development Life Cycle (SDLC), underscoring the necessity of rethinking contractual frameworks to accommodate AI and cloud technology implementations. In terms of business-aware testing, we examined the significance of incorporating AI and cloud technologies to enhance business outcomes, through improving efficiency and quality, and enabling faster responses to market trends.

By discussing organizational dynamics, we pointed out how AI and cloud technologies can impact communication, power structures, and collaboration within organizations, with a potential for profound transformative effects. In analyzing lessons learned from testing, we emphasized the importance of an iterative, learning-oriented approach, and of sharing knowledge within the software testing community.

We introduced two new sections to add depth to our exploration - ethical considerations and government case studies - which underscored the significance of ethical, legal, and societal dimensions of AI and cloud technology use, especially in the context of government use.

In closing, the interweaving of AI and cloud technologies into software testing and development is more than a technological revolution. It is a complex endeavor that reaches into the realms of organizational culture, business strategy, ethical considerations, and societal impact. As we continue to push the boundaries of these technologies, we must ensure our approaches to software testing and development evolve in tandem, cognizant of the broader implications and opportunities they present.

The future of software testing in an AI and cloud era is undoubtedly filled with both challenges and promises. It behooves us to embrace the complexity, maintain an unwavering focus on the end-user, foster continuous learning, uphold ethical considerations, and share our lessons learned. By doing so, we ensure that we are not just creating software that works, but that we are shaping a future in which technology enhances societal wellbeing and propels us towards new horizons of discovery.

To quote Isaac Asimov, "The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom." Let us strive to close that gap, for in the intertwining of knowledge and wisdom lies the true potential of AI and cloud technologies.

10 References:

- Agile Alliance. (2021). Agile 101. Agile Alliance. <https://www.agilealliance.org/agile101/>
- Ali, S., Petersen, K., & Schneider, K. (2017). Business-aware testing in practice. *Journal of Systems and Software*, 134, 221-237.
- Amatriain, X. (2019). Navigating the ML testing landscape. <https://queue.acm.org/detail.cfm?id=3321612>
- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50-58.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). Manifesto for agile software development.
- Bertolino, A. (2020). Software testing research and practice. *Advanced Software Testing and Quality Assurance*.
- Bryson, J. J., & Winfield, A. F. (2017). Standardizing ethical design for artificial intelligence and autonomous systems. *Computer*, 50(5), 116-119.
- Cath, C., Wachter, S., Mittelstadt, B., Taddeo, M., & Floridi, L. (2018). Artificial Intelligence and the 'Good Society': The US, EU, and UK Approach. *Science and Public Policy*, 45(1), 27-37.
- Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile networks and applications*, 19(2), 171-209.
- Chen, Y., Groves, T., Harrison, R., & Lin, Y. (2022). Business-Aware Model Training, Monitoring, and Testing. *IEEE Transactions on Knowledge and Data Engineering*.
- Crawford, J. K. (2014). Project Management Maturity Model, Third Edition. CRC Press.
- Desouza, K. C., & Jacob, B. (2017). Big data in the public sector: Lessons for practitioners and scholars. *Administration & Society*, 49(7), 1043-1064.
- Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6), 1213-1221.
- Duan, Y., Gu, B., & Whinston, A. B. (2020). Artificial Intelligence and Network Platforms: Business Models and Public Policy. *Information Systems Research*.
- Duvall, P. M., Matyas, S. M., & Glover, A. (2007). Continuous integration: improving software quality and reducing risk. Addison-Wesley Professional.
- Gartner. (2022). Top strategic technology trends for 2023. Gartner. <https://www.gartner.com/smarterwithgartner/gartner-top-strategic-technology-trends-for-2023/>
- Gaziano, J. M., Concato, J., Brophy, M., Fiore, L., Pyarajan, S., Breeling, J., ... & O'Leary, T. J. (2016). Million Veteran Program: A mega-biobank to study genetic influences on health and disease. *Journal of Clinical Epidemiology*, 70, 214-223.
- Haley, B., Laney, D., & Peter, M. (2018). Cloud Computing: Concepts, Technology & Architecture. Prentice Hall.
- Jalali, S., & Wohlin, C. (2012). Systematic literature studies: Database searches vs. backward snowballing. *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, 29-38.

- Katzan, H. (2010). On the privacy of cloud computing. *International Journal of Management & Information Systems (IJMIS)*, 14(2).
- Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution.
- Kitchenham, B., & Pfleeger, S. L. (2002). Principles of survey research part 5: populations and samples. *ACM SIGSOFT Software Engineering Notes*, 27(5), 17-20. <https://doi.org/10.1145/568790.568794>
- Kruchten, P. (2003). *The Rational Unified Process: An Introduction*. Addison-Wesley Professional.
- Kruchten, P. (1995). The 4+ 1 view model of architecture. *IEEE software*, 12(6), 42-50.
- Kwong, Y.K., Lew, P., and McDowell, J. (2019), "Software Based Disruptive Change Initiatives Require a Culture of Quality," Portland, OR: PNSQC 2019 Proceedings, Pacific Northwest Software Quality Conference.
- Kwong, Y.K. and Lew, P. (2018), "Quality & Risk Management Challenges When Acquiring Enterprise Systems," Portland, OR: PNSQC 2018 Proceedings, Pacific Northwest Software Quality Conference.
- Leung, H. K. N., & Chan, L. K. (2003). Risks in software development projects: evidence from a survey. *International Journal of Project Management*, 21(4), 252-262.
- Lindgren, R., & Münch, J. (2012). Agile contract creation. *Software*, IEEE, 29(4), 68-73.
- Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing — The business perspective. *Decision Support Systems*, 51(1), 176-189.
- Martínez-Cámarra, E., Martín-Valdivia, M. T., Ureña-López, L. A., & Montejo-Ráez, A. R. (2014). Sentiment analysis in Twitter. *Natural Language Engineering*, 20(1), 1-28.
- McDowell, J. and Kwong, Y.K. (2022), "Artificial Intelligence is the New Astrology of Software Quality," Portland, OR: PNSQC 2022 Proceedings, Pacific Northwest Software Quality Conference.
- McDowell, J. and Kwong, Y.K (2021), "Managing Accessibility in Software Systems," Portland, OR: PNSQC 2021 Proceedings, Pacific Northwest Software Quality Conference.
- McDowell, J. and Kwong, Y.K. (2020), "Towards a Culturally Inclusive Software Quality," Portland, OR: PNSQC 2020 Proceedings, Pacific Northwest Software Quality Conference.
- Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. *Special Publication 800-145*.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.
- Mittelstadt, B. D., Allo, P., Taddeo, M., Wachter, S., & Floridi, L. (2016). The Ethics of Algorithms: Mapping the Debate. *Big Data & Society*, 3(2).
- Mohler, G., Short, M. B., Malinowski, S., Johnson, M., Tita, G. E., Bertozzi, A. L., & Brantingham, P. J. (2015). Randomized controlled field trials of predictive policing. *Journal of the American Statistical Association*, 110(512), 1399-1411.
- National Institute of Standards and Technology. (2023). *NIST Special Publication 800-160: Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems*. NIST. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v2.pdf>
- Office for National Statistics (ONS). (2019). Big Data Project: Faster indicators of UK economic activity. Office for National Statistics. Retrieved from

<https://www.ons.gov.uk/economy/nationalaccounts/uksectoraccounts/methodologies/bigdataprojectfasterindicatorsofukeconomicactivity>

Perry, D. E., Votta, L. G., & Wade, W. M. (2000). An empirical study of software design practices. *IEEE Transactions on software engineering*, 26(3), 213-229.

Pezze, M., & Young, M. (2007). *Software testing and analysis: Process, principles, and techniques*. John Wiley & Sons.

Pfitzmann, A., & Hansen, M. (2010). A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. August.

Raghavan, M., Barcas, S., Kleinberg, J., & Levy, K. (2020). Mitigating bias in algorithmic hiring: Evaluating claims and practices. *Proceedings of the 2020 conference on fairness, accountability, and transparency*.

Royce, W. W. (1970). "Managing the Development of Large Software Systems." *Proceedings of IEEE WESCON*, 1-9.

Sills, S. (2017). *Agile Project Management: A Comprehensive Guide*. CreateSpace Independent Publishing Platform.

Sommerville, I. (2011). "Software Engineering" (9th ed.). Boston: Addison-Wesley.

Sommerville, I. (2016). *Software Engineering*. 10th ed. Addison-Wesley.

Subashini, S., & Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1), 1-11.

Tore Dybå, Torgeir Dingsøyr, Nils Brede Moe. (2014). Agile project management. In: M. Halling, S. Fraser, K. Tuma, N. Christiansen, B. Aichernig, J. de Groot (eds) *Agile Methods. Lecture Notes in Business Information Processing*, vol 180. Springer. https://doi.org/10.1007/978-3-319-06862-6_3

Turner, R., & Cochrane, R. A. (1993). Goals-and-methods matrix: coping with projects with ill defined goals and/or methods of achieving them. *Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis*.

Vasilescu, B., Serebrenik, A., Filkov, V. (2015). A data set for social diversity studies of GitHub teams. In: *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15)*. IEEE Press, Piscataway, NJ, USA, 514–517.

Wood, J. R., & Silver, D. (1995). *Joint Application Development*. John Wiley & Sons.

World Economic Forum. (2021). AI and Ethics: A roadmap for navigating opportunities, laws, and norms. World Economic Forum. <https://www.weforum.org/whitepapers/ai-and-ethics-a-roadmap-for-navigating-opportunities-laws-and-norms>

Zhang, Huchuan, Zhang, L., & Guo, X. (2020). Deep learning for image-based object detection. *BMC Bioinformatics*, 19, 257.

Art in Code and Quality

Sophia McKeever

s.mckeever@pokemon.com

Abstract

Code. What do you think of when you hear or see that word? Maybe you think of programs or applications. Or perhaps you think of services, APIs or specific technologies. Code can represent many and more of these things but maybe there is something you're not thinking of when you read code. What if the development of code and the process of qualifying it is a creative process? What if the act of writing code is, in fact, an act of creating art and reviewing that code is an act of artistic appreciation?

This paper will take you on a journey beyond the scientific and the mathematical. A sojourn past the technical aspects of the work that you do and bring that work contextually into the heart of the humanities. This paper is an invitation for you to consider the very last thing on your mind when you consider code and the quality processes used to validate it. An invitation for you to think about your work as art.

Biography

Sophia McKeever (She/Her/Hers) is a self-taught Software Development Engineer in Test currently working at The Pokémon Company International with over ten years of experience within the Software Quality industry. Throughout her career she has positioned herself as a test automation framework architect at the various companies she's worked at including Apple Inc., Microsoft Azure, and DataSphere Technologies Inc. She holds a Certificate in Python Programming from the University of Washington's Continuing Education Program and has experience in a wide array of qualitative technologies including Selenium, qTest, mabl, and Cypress. She is an artist at heart with a deep love for digital illustration and building artistic code projects.

Copyright Sophia McKeever, June 4th, 2023

1 Introduction

Be you a software engineer or a quality assurance engineer you likely spend a good portion of your day reading or writing code. Chances are you are looking at pull requests, reading the code in a repository, looking at snippets on the web, or writing the code yourself. You have probably read thousands if not millions of lines of code throughout your career; given feedback on a senior engineer's code or perhaps chuckled at the mistakes of a junior engineer. Have you ever stopped, however, and taken in the true nature of what you're seeing? Have you taken a moment to appreciate the syntactical splendor on your screen and understand the magnitude of what it represents? There is the obvious answer that the code is a list of instructions to build an application or service. Although technically correct, this is not the full picture of the code you see on your screen. What you're not realizing is that when you read that code you are interpreting a pure creative work of art.

Our work as software engineers is a unique form of art that combines scientific and mathematical principles with poetic prose complete with stanzas and verses. Our product owners come to us with problems, and we return to them with solutions, but those solutions are truly unique to us individually. In fact, if you were a product owner requesting a solution to a problem from two engineers on the same team, the solutions would be very different. Their code would, in the end, produce the same result but the inner workings of their code would be unique to everyone built upon their own understanding, past experiences, and even their emotional state at the time of writing. Chances are you've seen this in action and not given it much thought. You may have been wowed by the 'beauty and elegance' of a solution or indifferent to one that seems 'run of the mill'. In those moments you are engaging in artistic appreciation for the code you are reading.

We've all had that one person on our team at some point who takes feedback and constructive criticism on their code personally. Maybe they're on your team right now, maybe you're them. Why do they fight or get upset? We often chalk that up as someone who thinks they are better than everyone else, someone who thinks they are above the rest. Truly think about that person. What drives them? Is it that they are stubborn or self-centered, or is it that they are an artist defending their work? One could argue that this individual needs to be able to accept constructive criticism and feedback, and this is arguably true; however, as an artist it can be hard to accept this criticism and not also feel the need to stand by your work. Constructive criticism for artists can be difficult to accept, even when it's a mutually agreed upon two-way street. Rather than thinking the programmer is self-centered, we need to think of them as an artist and better know them as a human in the context of their work.

Throughout this paper, we'll explore the philosophy behind the act of writing code as a creative process in both the methodology and aesthetic beauty of the code. Then we will take an introspective look at the influence our emotion plays in the work that we do as creative individuals. We'll learn how our emotional intelligence in addition to providing gentle, constructive, and collaborative feedback in our code reviews can help us better connect with the human behind the code. By learning how to engage our emotional IQ, we'll be able to foster integrity and respect while building relationships within our teams and partners, improve the state of quality in the products we work, and ultimately delight our customers through improved team cohesion.

2 On the Philosophy of Writing Code as a Creative Process

2.1 The Definition of Art According to the Dictionary

To understand how code is an art form we need to first define what art is. Merriam-Webster has six definitions of art as a noun but there are two definitions we should focus on. We'll start with the first definition.

"Art (noun) - Skill acquired by experience, study, or observation." (Merriam-Webster)

Whether you are self-taught, taught through an educational institution, or learnt by watching or reading the work of others you have acquired a skill. That skill grants you the ability to make a computer do a specific set of tasks. Line by line you create an application, service, automated test and so on, drawing on that knowledge and your previous experiences to write that application. It takes knowledge of writing in your preferred programming language, knowledge of the ecosystem you're building for, and knowledge of the type of program you're building. You've gained that skill through a mix of either study or observation and through experience. You have acquired the art of writing code, however there is more nuance to this philosophy, so we need to now look at the next important definition of art.

"Art (noun) - The conscious use of skill and creative imagination especially in the production of aesthetic objects" (Merriam-Webster)

The first definition allows us to define our work as art because it is an acquired skill while this second definition expands that concept by taking the concept of that skill and using it with creative imagination to produce aesthetic objects. Let's break down the process of developing. Whether you are assigned a task or just wanting to build a new application on your own, you are starting with a blank canvas. Since the solution doesn't exist, you begin to really think about how to build it, drawing on your skill and experience, engaging your imagination on how to build it. Of course, you're using the principles of mathematics and logic to think through how the code of the application will work, however, in addition your imagination is powering the dream to bring that application to life and to bring it into the real world.

2.2 Meter, Rhythm, Poetry - The Aesthetic Object of Our Code

The aesthetic part of the application comes in several forms. There is the obvious piece, if the application has some sort of UI or output you could consider that aesthetic, but we can go deeper. Truly look at the code you produce. Read it thoroughly, but more importantly read past the words in the code sheet. When you look at the code, not for the words or their meaning, but something more you begin to see it. The meter, rhythm, rhyme. Stanzas, verse, prose. The use of whitespace to separate parts of the code to make it readable? No, beautiful. This code snippet will help illustrate this concept.

```

1
2     def first_unique_character(string_in):
3         dict_char = {}
4
5         for char in string_in:
6             dict_char[char] = dict_char.get(char, 0) + 1
7
8         for key, value in dict_char.items():
9             if value == 1:
10                 return key
11
12         return None
13
14 result = first_unique_character("abcdeedebaxb")
15 print(result)
16

```

(Figure 1: A Python code snippet for returning the first unique character of a given string)

Read the code above. It should look obvious as a function that finds and returns the first unique value of a string in Python. Perhaps it's not the most efficient method of the desired result but that's not important for this concept. Read the snippet again. What do you see, the same function? Keep reading it until the names, keywords and values lose meaning. You might begin to see it, the meter inside the code. This might help:

```

1   """  

2   define name parameters enter  

3       name equals new dictionary  

4  

5       for item in string  

6           dictionary item equals value  

7  

8       for pair in dictionary  

9           if value operator condition  

10          return key and break  

11  

12      return nothing then end  

13  

14      result equals function parameter  

15      output result from function  

16 """

```

(Figure 2: The Python code snippet represented in pseudo code)

This pseudo code represents the same function, however its more than that. It shows the meter of the code as it is read. “Define name parameter enter”, “name equals new dictionary”, and so on. The meter is four beats per line. Let’s break down the snippet one more time to see the true nature of it.

```

1   """  

2   one two three four  

3       one two three four  

4  

5       one two three four  

6           one two three four  

7  

8       one two three four  

9           one two three four  

10          one two three four  

11  

12       one two three four  

13  

14       one two three four  

15           one two three four  

16 """

```

(Figure 3: The meter of the code)

And there it is. When you realize it uses mathematics as its meter and logic as its prose, you see the aesthetic object of the code. Poetry. Simple and beautiful and yet profoundly complex. A piece of creation that came from the heart and mind, a marriage of talent and skill. As a developer, you have been writing poetry to the computer and probably never realized it.

2.3 Art in Almost Everything We Do

In his book *Understanding Comics: The Invisible Art*, American cartoonist and comic theorist, Scott McCloud asserts that “in almost everything we do there is at least an element of art” [2] because he defines art as “any human activity which doesn’t grow out of either of our species’ two basic instincts: survival and reproduction.” [2]. This philosophy is a driving force to understanding the art within our craft as software engineers. The fact that we need to follow a creative process to create our code is proof that we, by nature, are truly artists under the guise of Computer Science. Related to the creative process,

McCloud breaks down the creative, artistic process into a six-step process, one that we follow as developers without even being conscious of it.

1. “Idea/Purpose: The impulses, the ideas, the emotions, the philosophies, the purposes of the work. The work’s context.” [2]
2. “Form: The form it will take... Will it be a book? A chalk drawing? A chair? A song? A sculpture? A comic book?” [2]
3. “Idiom: The ‘school’ of art, the vocabulary of styles or gestures or subject matter, the genre that the work belongs to... Maybe a genre of its own.” [2]
4. “Structure: Putting it all together... What to include, what to leave out... How to arrange, how to compose the work.” [2]
5. “Craft: Constructing the work, applying skills, principal knowledge, invention, problems solving, getting the ‘job’ done.” [2]
6. “Surface: Production values, aspects most apparent on the first superficial exposure to the work.” [2]

2.4 The Creative Process as It Applies to Development

Consider, for a moment, how you write your code and design your projects. You likely have a process of build that you adhere to, be it consciously or unconsciously, that ensures you get the job done. There's a possibility you approach your work in a similar methodology outlined in these six steps. It may look something like this:

1. Idea/Purpose: Upon being assigned a task you begin to react to impulses and ideas on how to do the work. The task you receive or app you imagine may defines the context of your work and then you apply your knowledge to generate ideas for the solution.
2. Form: You decide the form the task or app will take. Maybe it's a command line app, maybe a Lambda in AWS, maybe a service or a script, or perhaps a mobile app.
3. Idiom: In this case, idiom defines a lot about the process of writing the code. What programming language to use, the coding paradigms you align to, how are you going to unit test it?
4. Structure: Putting the coding project together, creating the directory structure, defining the namespaces, perhaps adding your tests if you subscribe to Test Driven Development (TDD).
5. Craft: Writing the code. Applying your knowledge and skills to solve the problem and bring the code to life.
6. Surface: Building and running the code. Handing off to your test team for validation, building test automation against it. Delivering the solution to your customers or stakeholders.

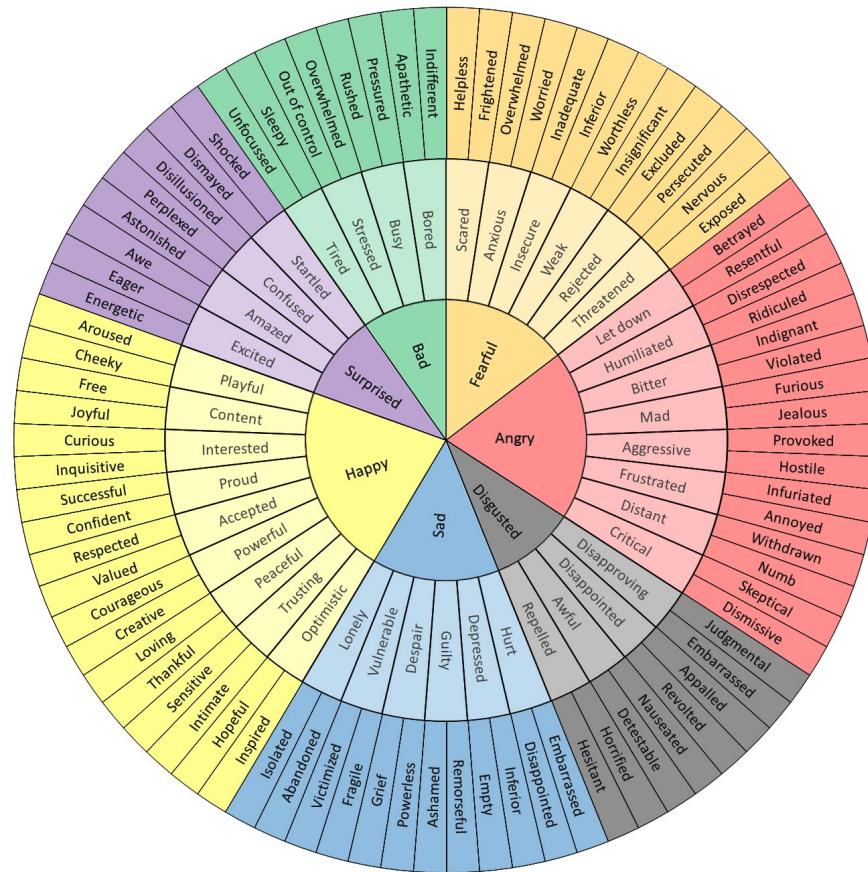
Perhaps you don't engage in all six steps, perhaps you've been engaging them in every task, and you haven't even noticed, perhaps you have additional steps. Regardless of your process, you are engaging in an act of creation with each line you write and each task you complete. Creativity, passion, skill, art flowing from finger to key to code file. These principles apply whether you're a software engineer, software development engineer in test, software test engineer, or any other member of a development team. There are aspects of art in nearly everything we do, writing code, software development is no exception.

3 Emotions and Their Effect on the Creative Process

Now that we've established the philosophy that we, as engineers, are artists, let us consider how our emotional state affects the quality of the work we do. The concept might seem obvious, “feel bad, do bad work”; however, how our emotions affect how we code is more nuanced than that. Our emotions can affect more than just the way our code functions- it can affect how it looks, how it reads, the amount of white space we have, our commenting, the way we name things, the way we test our code and more.

3.1 Defining Our Emotions and Feelings

Think about your base line for writing code. You're in the groove, you feel confident, perhaps a little happy, you can think clearly, the task is simple to achieve without being complex. You likely have a very distinct style to your code that shines through on the code sheet in this state. Now consider a time you were writing code for a project, and you were frustrated, or sad, anxious, or upset; likewise consider a time when you were extremely excited, elated, happy, or maybe even creative. Have you ever noticed a difference between the times you're operating in at your baseline and times when you are operating under a given emotion? You may have noticed that your emotional state affected the way you wrote your code. To see this in action, let's first define the veritable continuum of emotions a human can express.



(Figure 4: *The Emotion Wheel* by Geoffrey Roberts, based on *The Feelings Wheel* by Dr. Gloria Wilcox)

In October of 1982, Dr. Gloria Wilcox published the original *Feelings Wheel* in volume 12, issue 4 of the *Transactional Analysis Journal*. In 2015, Imgur user Geoffrey Roberts derived the *Emotions Wheel* from her work [3]. This version expanded the feelings listed on the original wheel and represents a wide, albeit not comprehensive, range of human emotions and feelings. The intended use of this tool is to help people identify what they are feeling to better understand their current emotional state and raise their self-awareness. We will use this tool to identify feelings associated with specific code samples to see how they differ from a baseline.

3.2 How Emotions Can Affect the Code You Build

Important to Note: Some of the code snippets below have been modified or recreated by the original artists to ensure the removal any proprietary or confidential information. All snippets come from the same

writer who requested that their identity not be revealed for privacy purposes, for the purposes of this paper they will simply be known as *The Poet*.

3.2.1 A Sterile Baseline – Code Written for Educative Purposes

```
@patch("example_class.BaseClass._format_error")
@patch("example_class.BaseClass._service_uri")
def test_make_request_error(self, patch_uri, patch_error):
    # Validate we catch when a service returns an unexpected response code.
    # Note the mocking of _service_uri
    patch_uri.return_value = "some uri"
    session = MagicMock() # Mock session used to ensure requests.session is not called.
    expected = MagicMock() # Mock used as the result
    expected.status_code = -3
    session.request.return_value = expected

    # Validate Assertion error gets thrown, note no other assertions are made within the 'with' statement.
    my_class = BaseClass("prod", "", session)
    with self.assertRaises(AssertionError):
        my_class._make_request("stuff", "in", "a", "box", 200)

    # Validate the expected function calls are made.
    session.request.assert_called_with("stuff", "some uri/in", json="a", timeout="box", verify=True)
    patch_uri.assert_called()
    patch_error.assert_called_with(expected, 200)
```

(Figure 5: The “Base Line” Code Snippet written in Python)

The Poet asserted that this is the type of code that they are known to write on a day-to-day basis, especially since it was used as an example for writing unit tests for their development team. As we can see in this code snippet, the code is tidy, neatly organized, and well named. It has no inherent flaws, no major issues. The code might not be efficient; however, it gets the task done of unit testing a function within the code of the application. We’ll use this as a base line as we look at examples from the same developer.

3.2.2 Code Influenced by Feeling Unfocused

```
10  @patch("some_library.namespace.some_class.ElementTree")
11  def test_init(self, patch_tree):
12      parse_mock = MagicMock()
13      parse_mock.return_value.getroot.return_value = "stuff in a box"
14      patch_tree.parse.return_value = parse_mock()
15
16      class_to_test = SomeClass("some path")
17      assert class_to_test.name == "stuff in a box"
18      assert patch_tree.parse.assert_called_with("some path")
```

(Figure 6: The “Unfocused” Feeling Code Snippet written in Python)

The Poet explained that they wrote this when they were feeling “unfocused”. They mentioned they were able to resolve the issue but only after a half hour of debugging and triaging. If you were reading this code in either a pull request or a peer programming session and they said they were unable to get the test to pass, what would you see? The immediate issue is the mistake on line 18 as part of the assertion on the “assert called with” call. In this case, its attempting to assert that the return value of the function is not none, however that function has no return value and thus asserts as false and causes the test to fail. Additionally, a minor mistake on line 14 exists where The Poet initializes the mocked document parser even though it is already initialized on line 12 as a magic mock. This isn’t that critical of a problem however it may result in some odd behavior when investigating assignments and calls within the mock

itself later if the test was more complex. If we're unfocused, we tend to forget these little nuances. We tend to overlook minor facts of our jobs and it takes longer for us to figure out the issues.

3.2.3 Code Influenced by Feeling Infuriated

```

11  @patch("datetime.timedelta")
12  @patch("time.sleep")
13  def test_wait_some_iterations(self, mock_sleep, mock_delta):
14      condition = MagicMock()
15      mock_delta.return_value = 1
16      se = [False, False, True, True]
17      condition.side_effect = se
18      scalls = [call(0.5) for _ in range(len(se) - 2)]
19      cccalls = [call() for _ in se]
20      StaticHelpers.wait_for(condition)
21      mock_delta.assert_called_with(seconds=60)
22      condition.assert_has_calls(scalls)
23      mock_sleep.assert_has_calls(cccalls)

```

(Figure 7: The “Infuriated” Feeling Code Snippet written in Python)

In this snippet, The Poet explained that they were struggling to mock the functionality of the Time Delta function in Python. In Python, Time Delta is treated as immutable which means the function itself cannot be mocked. The Poet wanted an easy way to control the function so they can test and make sure the proper deltas were being applied. With each try attempt that failed, they got more and more angry at the situation. They got to the point where they felt “infuriated” and were making simple mistakes, naming things “whatever” to complete the task. As we can see, many variable names have poor naming, it’s hard to understand what the different variable names represent. Additionally, there is no white space separating out logical areas in the code. Finally, the code is so incoherent that even after revisiting it in this state several days later, The Poet themselves couldn’t really make sense of it. According to them, it took them quite a while to finally give up and just make the code easier to mock by extracting the time calculation into its own function. They claimed that when they finally let go of the track they were following and refactored the code they felt disappointed that they couldn’t make their original design work.

3.2.4 Code Influenced by Feeling Inspired

```

11  def test_get_ids_from_document(self):
12      parser = MockedDocumentParser("Remember the rain")
13      list_ids = [
14          "Near and far beloved, each drop a blessing from heavens above",
15          "And how as time flowed on those waters became one",
16          "Streams, rivers and lakes reaching for the horizon and far beyond",
17          "They carry onward however changed with each brief reflection",
18          "by setting sun, by storm's wake. Til welcomed home to gentle sea"
19      ]
20
21      list_properties = [MagicMock() for _ in list_ids]
22      for index in range(len(list_ids)):
23          list_properties[index].get.return_value = list_ids[index]
24
25      parser.document.find.return_value = list_properties
26      assert parser.get_ids_from_document() == list_ids
27
28      parser.document.find.assert_called_with("property")
29      for property_listing in list_properties:
30          property_listing.get.assert_called_with("propertyId", str)

```

(Figure 8: The “Inspired” Feeling Code Snippet written in Python)

Lastly, we'll look at a time when The Poet was coding while feeling inspired. They mentioned that during this time they had solved a problem in a novel way and that it made them feel inspired and hopeful. As they unit tested the code, they added some lyrics to the song they were listening to as part of the return value of a given mock. They mentioned that it was probably overkill for what they were doing but it didn't take any additional time to build the test. Ultimately, they were happy to build the test in this manner and they ended up finishing this specific suite in this manner. Their colleagues even commented on the tests noting that they found it a fun way to write tests that also accomplished the task.

3.2.5 Summarizing The Impact of Emotions on Our Code

Emotion	Effects	Recommendations
Unfocused	<ul style="list-style-type: none"> • Syntactical Errors • Lapses in Critical Thinking • Decline in Debugging Skills • Lack of enthusiasm 	<ul style="list-style-type: none"> • Resolve whatever is taking your focus. • Engage in simple mental exercises to free up your thought process. • Self-Care/Handle basic needs (Eat, Drink, use the bathroom, etc.)
Infuriated	<ul style="list-style-type: none"> • Lapses in critical thinking • Difficult to understand or inappropriate variable names. • Creation of difficult to read code. • Compounding frustration when trying to "force it to work" 	<ul style="list-style-type: none"> • Step away - let yourself calm down. • Engage in grounding exercises to restore calm, rational thinking. • Learn to let go of solutions that won't work or that need refactoring. • Reach out to friends or colleagues to vent your frustration.
Inspired	<ul style="list-style-type: none"> • Playfulness in code or comments • Fluidity in solutions • Easier to read and parse code 	<ul style="list-style-type: none"> • Continue as you are. Shine on.

(Figure 9: A table summarizing the emotions The Poet experienced, their effects on the code, and recommendations for resolving them)

The table above outlines the feelings The Poet encountered while developing and the effects that they had on their code. As we can see, our emotions can influence the way we build our code. It truly can mean the difference between a clean, well-written function and a one that has bugs, poor naming conventions, and inconsistencies. This doesn't mean we shouldn't do our jobs under the influence of given emotions, but it can inform us and our colleagues of how we're feeling in that moment. This information can help you not only better understand yourself but also understand your colleagues and inform how your colleagues understand you as well.

4 Artistic Appreciation, Emotional Intelligence, and the Quality Mindset

We've now established that emotions affect the work we do, the art we create code we write. Not only can they affect the whitespace and naming conventions, or rather the aesthetic object of the code, directly but emotions can affect how we approach engaging in the creative process and building our code. When you look back at code that you built under the influence of your emotions, it has the power to invoke those emotions again. It can also invoke additional feelings- embarrassment, pride, joy, insecurity, and so on because we are creative, and we place meaning in the work we do. The same can be said about when we look at code our colleagues have built under their emotions.

When the code you read invokes feelings, it connects you better to the person who wrote the code. Additionally, you can begin to pick out patterns in their code, ways that communicate their feelings in the Excerpt from PNSQC Proceedings

PNSQC.ORG

Page 9

context of their work. You can begin to identify when the person was feeling sad, frustrated, or any other host of emotions simply by reading the code and connecting with the human behind it. This is an incredible opportunity to help build a relationship with that individual and reach out to check in on them. The more you connect with the people on your team, the more it'll foster integrity and respect within the team and partners that you work with.

As a quality engineer, we excel at finding flaws in code and calling out the need to refactor code; although this is critical to our job, we can often alienate those we work with. By taking that opportunity to truly understand the people on our team we can help them truly shine and be their best selves. Not only does it help us build relationships with our partners, but it can also bring new opportunities for mentorship, knowledge sharing, and general improvement in the quality of the work the team does. We have opportunities every day to engage our artistic appreciation and emotional intelligence to help foster respect and comradery in our teams. By doing so, we can motivate our partners teams to engage in good quality practices and delight our customers in the process.

5 Conclusion

Code. What do you think of when you read that word now? Perhaps now there is a new synonym in your mind when you read it: Art. Whether as a quality engineer or a software engineer, our artistic ability grants us an ability no other artist has. The ability to bring life to the inanimate. The ability to conjure something from nothing as we paint across the canvas of a code sheet. Our art sets alight the fire of creation within lifeless circuits and pixels. An art that is so profound that not only does it influence the way that human and machine interact together but it inspires the very fabric of our universe to come alive and dance in harmony. Our poetry inspires electrons to flow from atom to atom as bits and bytes flowing along traces within the computer and help it understand, calculate, store and recall data. A beauty that transcends its mathematical and scientific roots, piercing right into the heart of the humanities. As computer programmers that is the power we wield.

As we are artists, it is our duty to see the beauty and humanity in our partners' work. We should take an opportunity to realize that even though we might think we have "the right answers" when we're developing our code or giving feedback on other's code, that others developing code are artists as well. The way they write their code may be influenced by the emotions they feel at the time, and we should be conscious of those emotions and how they define them as a person. By changing the perspective on how we think about code, realizing the artistic nature of code, we can reframe how we give feedback and nurture the relationships between us and our peers and partners. In nurturing those relationships and approaching our feedback from the lens of artistic appreciation, we can begin to foster a new level of integrity and respect in those we work with which will ultimately help improve the acceptance of our qualitative feedback.

References

1. Merriam-Webster.com Dictionary. n.d. *art*. Accessed June 1st, 2023. <https://www.merriam-webster.com/dictionary/art>
2. McCloud, Scott. 1993. *Understanding Comics: The Invisible Art*. New York: Kitchen Sink Press.
3. Roberts, Geoffrey. 2015. *Emotions Wheel*, Imgur, entry posted March 15, 2015, <https://imgur.com/a/CkxQC> (accessed June 1st, 2023).

Common pitfalls in modern QA

Leandro Melendez

leandro@perfbytes.com

Abstract

Are your software quality efforts constantly struggling to keep up?

It is getting more challenging, especially in these modern times, where our quality processes constantly need to catch up with each sprint. We often require QA tasks to rush to be completed before the sprint ends.

These and many more problems come from various sources, such as harmful practices, archaic best practices, misunderstandings, and issues affecting software development teams.

Psychology plays a big part in these problems. Several cognitive biases affect teams and industries. Some are the man with a hammer syndrome, priming, hindsight bias, the Dunning-Kruger effect, semblance, congruences, the halo effect, and many more.

Those may generate problems such as keeping the teams in infinite cycles updating automations, becoming the bottleneck of the entire team by waiting long periods for their executions to finish, and many more problems.

We will present several examples of organizations suffering from these practices and highlight real-life situations, identifying solutions and recommendations for modern best practices.

We will have endless funny analogies, thought exercises, and riddles that will leave the audience pondering and wondering of how many biases they fall into.

Biography

Leandro is a performance testing advocate with Grafana k6 helping everyone to ramp up their performance practices.

He has over 20 years of experience in IT and over 10 in the performance testing practice, where he served multiple S&P500 customers all over the USA, Mexico, Canada, Brazil, India, Austria, etc.

Author of the popular performance testing blog Señor Performo (www.srperf.com), where he curates a diverse set of learning material for performance testers and engineers.

He is an international public speaker participating in multiple conferences, events, and webinars, with keynotes, workshops, and multiple talks on his belt.

And last, the author of "The Hitchhikers Guide To Load Testing Projects", a fun walkthrough that will guide you through the phases or levels of an IT load testing project. <https://amzn.to/37wqpyx>

1 Introduction

Do you think QA efforts have been challenging to keep up with instead of getting more accessible and agiler?

You may think that it is the fault of those Agile practices that make your automations constantly break from so many changes. On top of that, you may be the least popular in halting the sprint's release as, again, you need help to keep up with so many things. Especially since the team is giving you testable things on the 7th or 8th day of the sprint.

And you know you are not alone because many more people you know doing QA are struggling with similar issues.

Well, let me tell you that among many other reasons, a predominant one is that you are all human beings. The human condition comes with multiple psychological phenomena that most probably all are falling for.

They are called cognition biases, syndromes, fallacies, and even more picturesque names. And guess what? They affect our everyday lives. Every activity you do is affected by them. Yes, QA is affected by them. And here, you will learn about them. We will provide some examples and even some tips to cope with them.

2 Cognition Biases

2.1 What are they?

You may think your brain, mind, and thoughts are all intelligent, rational, and excellent.

Well, I am sorry to bring this up to you. Most are not. Human beings are fascinating creatures that act in peculiar ways that are everything but rational. Even as now you might think our actions and thoughts are rational, you will see they are not always like that.

All this happens because our brains have evolved, given our past circumstances, in a way that started not to help us be smart but survive.

After that, the brain evolved larger than other animals, demanding more energy. But keeping a balance on energy consumption, in its evolution, it developed several shortcuts to maximize the efficiency in its energy use.

The elements of survival and efficiency make our brain take great shortcuts for the mentioned reasons. But it only sometimes gives rational, intelligent, accurate, or even correct outcomes.

Those are known as cognitive biases.

2.2 It was all survival.

As mentioned, we developed cognitive biases for two reasons that assured our survival.

Our ancestors had to develop multiple reactions to the environment. Otherwise, they would have been eaten.

One example is our ability to see things where they are none. Think of a Rorschach test, the ones with dots randomly splattered on a page. Most will find a shape or meaning in those. Same as finding shapes in clouds or seeing faces in front of many car models. Yes, like the Pixar movies.

Another example is our ability for quick assessments and snap judgments. We may have met someone and already have an idea about them in about two seconds. Same for identifying a weird noise we just

heard coming from behind us. Or even interpreting the cracking in our wood-framed houses as something supernatural.

We have many more situations like those. All wrong behaviors and estimations of our brains. But those were vital in the past. As a few examples, if a striped tiger got nearby in the orange forests with vertical golden plants, it was vital for us to quickly identify the pattern. We had to see the tiger immediately if we did not want to be eaten. Same for the snap judgment that triggered our fight or flight response (most often flight) at the minimum weird noise we heard. It could be the tiger.

It was all survival.

As our brains evolved, those behaviors had to be optimized as our brains' evolution made them grow, and their energy consumption increased.

Human brains had to optimize those paths and shortcuts. Otherwise, our already enormous energy consumption would be even greater. Imagine having to eat even more time during the day to keep your hungry brain fed. We would have been easier prey. Not to mention we would need more lunch breaks.

2.3 Some of the biases.

Several psychologists and scientists have recently identified these biases through research, experiments, and notes. The actual list is long, and the names are varied and even colorful.

Here I will list some of the ones I like the most and their flamboyant names.

2.3.1 The “Man with a Hammer” Syndrome¹

If all you have is a hammer, everything looks like a nail.

Where I come from, the phrase goes like this—Who learned to hammer well sees nails everywhere.

Also known as the "Law of the Instrument" or as Maslow's Hammer (yeah, the Maslow that you probably thought of) is a cognitive bias involving an over-reliance on a familiar tool. Abraham Maslow wrote in 1966, "If the only tool you have is a hammer, treating everything as if it were a nail is tempting."

This is a psychological effect that I have suffered several times over my life, and why I am starting with this one. It is effortless to fall for it.

It does not apply only to tools, but processes, knowledge, perspectives, and many more "tools." The hammer may be an excellent tool for some things, but a dangerous one if the task at hand is to brush your teeth. Just think about it. How many things you may be trying to do with the wrong tool?

2.3.2 Halo Effect²

It is also known as the Halo Error.

Have you ever met someone who was great at something? Let's think of Michael Jordan, the basketball player. He was fantastic at playing basketball.

But departing from that bit of information, some may think (even he) that he would be great at baseball.

But it doesn't only have positive effects. We may know the phrase, "Who keeps company with the wolf will learn to howl." If someone has some bad traits or is surrounded by wrong impressions, we may infer that the judged thing or person is terrible with others.

This is a common misconception about people, places, circumstances, and many more things. When we have some information about the subject, we generalize that into other areas of the subject, whether it is a person, place, or others.

2.3.3 Sunk cost fallacy.

Almost related to the Loss aversion fallacy, this one delves into how we handle loss in our minds.

Have you ever continued to be invested in something when you knew you should have quitted long ago just because of what you already invested?

This is true for relationships, where people may know their partner is not a good choice but stay there because they already invested much in the relationship. Or how about that handwork you patched even as it already looked horrible, but you kept toying around it?

I experienced that with some programming code so messed up that starting from scratch was the best choice, but I stuck with it. You have no idea how many headaches I had because of that.

Humans tend to think that past costs, investments, or events influence our future actions. Having coin-flipped tails 9 times doesn't make it more probable to get heads on the 10th try. It is always 50-50.

2.3.4 Dunning-Kruger effect

Do you think riding a bicycle is easy? Do you consider yourself a good cyclist?

You may be surprised (if you have yet to become a connoisseur) by how deep that cycling world can go. It is profound. You need several skills to be a pro, which many underestimate from their limited perspective. And this effect often happens in multiple areas: knowledge, understanding, skills, experience, etc.

This effect was named after a study by David Dunning and Justin Kruger. This effect describes how people with some understanding of a given topic tend to overestimate their understanding or skill in each topic. But as the knowledge deepens or grows, people often realize how little they know or how unskilled they are.

The craziest part is that the true experts know how big the topic is, making them underestimate their expertise. Even at points where they are pushed towards another psychological effect, the imposter syndrome. Everything is connected.

2.3.5 Groupthink

Have you been in a meeting at work where the boss says something evidently wrong, but you don't correct the boss as no one else is raising concerns? Even worse, everyone seems to be nodding. To make things worse, you start nodding as well.

In the example above, you were aware of the error. But in many environments and examples, you may need to be made aware and take everyone else's knowledge for granted.

We evolved into social creatures who would have died if we were expelled from our group; hence, we care for that pertinence instinctively, even when we may agree with evidently wrong statements if we want to be part of a community.

3 QA and biases

A common thought is that in their efforts to objectively test and prevent defects, QA practices are one of the most bias-proof areas in IT. And that is a biased opinion that is far from the truth.

QA processes, tasks, tools, methods, and many other sides of QA are designed, planned, executed, and reported by humans. We have criticized poor humans for being full of biases and problems. They inherit all those to the QA processes, tasks, reports, and everything surrounding the discipline.

Don't believe me? Here are some real-life examples of cognitive biases hurting our QA processes. Just a few because this could be a long one if I went crazy with it.

3.1 Man with a QA hammer

We may have encountered this problem in several organizations. Especially around automation tools, or in other words, hammers.

In this case, a close relative discovered how to automate with Selenium. Selenium is fine, but this relative was super excited about its possibilities. He started using it everywhere. He went everywhere, happily automating processes, not being a tester or an IT person.

Then he had a genuine QA need to validate if migrated data from other databases arrived right in the target system.

Given he had learned Selenium so well, he started automating validations through the front end. Quickly discovered that validating one record at a time through the front end was slow, required a lot of computer power, and quickly broke with any difference in the flow. We quickly estimated that validating some million records would take days of nonstop running.

Why obsess about using the Selenium hammer if he could use the SQL screwdriver? Just query for the migrated data on the source and target environments and add a comparison on both outputs. A process executed in a few minutes, if not seconds. Something seriously related to the automation pyramid.

3.2 Loss aversion of automations

We have seen this one everywhere; you probably have suffered from it. How easily can you or your team stop supporting or even delete old automations?

We all have become part of teams with hundreds and even thousands of automations, and we cannot eliminate any of them. We may not even know what they were for, who created them, or why we have them, but we keep supporting them every time they break. We diligently fix them.

This may have a sentimental facet as we worked hard to create those automations, even if that was 5 years ago. Or our management may feel that they have paid so much for them that it would be unjustifiable to file them as old or, even worse, to delete them!

As Elsa the ice princess told us in that catchy song, we must "Let it go!"

João Proença, a master in cognitive biases in QA, recommended me a heuristic for when you are falling for this one. Imagine all your automated tests were suddenly deleted to know which ones you should stop supporting or could even delete. Think of which ones you would re-create right away. The rest, you can stop thinking about them.

3.3 The tool with a halo effect

One tool to rule them all. Or at least this is the dream of many managers when requested to procure tooling for their org. No matter how expensive, complex, or heavy it may be.

This is a common pitfall when we talk about tools. Many tools are fantastic for some things. But vendors tend to add features and functionalities to those tools until they are almost a huge test-everything suite.

The conception of the thing that the tool is good at (and it may be excellent at it) may give the impression that it is an excellent tool for everything, given the other functionalities it has. But often, that is different.

This may be confused with the man with a hammer fallacy. Let me differ on it as we use only one function of the tool for many things. The tool has other modules, functions, or features in other areas than the one

it is good at. Again, it is usual that because that tool is very good at one thing, we might think it is fantastic in the other functions it has.

3.4 Confirmation in my machine bias

This one is a true classic. How many times have you heard, "It runs OK in my machine," after you report a bug? I mean, that is almost a cliché statement nowadays.

In their interest in finishing and delivering software, developers may close themselves to other perspectives. Or in other words, they close themselves to considering other environments. This lack of consideration of other data points might make them blind to other environments' configurations or requirements. In this situation, the QA specialist may be biased as well.

Often developers need to consider elements that may be missing when they are testing a report of a defect right away without considering the possible reasons why they might be seeing that defect. They are not interested in fixing or diagnosing bugs, just detecting and reporting them.

Both sides are so interested in their truths that they are blind to the rest of the information. We know how much cat-mouse chasing this brings, and usually, solutions take time to come. The trustworthy source of the problem is usually detected after a lot of chasing.

3.5 Many more biased defects!

As you can see from the examples, QA endeavors are plagued by biases. These were just a few examples, but there are many more. Wrong assumptions, hunches, anchoring when proven wrong, and much more happens on teams delivering software that passes through a QA process.

The list is long, and most of the time, the teams are unaware that they are falling for those biases.

Now you may wonder how we can avoid falling so much for these effects. Well, let's go over some tips.

4 How to avoid falling for biases?

Well, this is awkward. You may be unable to avoid falling for cognitive biases in your QA efforts. As mentioned, those biases are part of the human condition, and anyone 100% free of them may not be human. I mean, even AI falls for biases!

So, is everything lost? Not so much.

Below are some tips to avoid falling so much for cognitive biases. And some others that will help you recover or mitigate their impacts in your QA efforts.

4.1 Recognition

We are all human beings. Well, most of us. And as stated earlier, we are bound to these biases. That is just our nature. We have it and cannot remove those mental patterns from our brains.

And believe it or not, if we do not recognize that we are constantly falling, we fall for a cognitive bias that will get us into even more trouble.

Don't worry. This may feel weird to admit that one is flawed. But it is OK, trust me. No one is perfect. You are not expected to be. You constantly fall for cognitive biases and will continue to do it. If you do not recognize that, you may be in denial or unable to see it when you fall for it.

4.2 Know the menu

The second step is to be aware of the many tricks our mind falls for. In this case, ignorance is not bliss. You do not need to be an expert on them, but at the very least, you must be aware of them.

Like the menu of your favorite restaurant, you may not know what is in the ingredient list of each dish, but at the very least, you know what each one is.

It would be best if you aimed at knowing them even with all their somewhat scientific and even flamboyant names. Learning them is a fun experience, and I guarantee you will be entertained and slightly scandalized.

4.3 Shameless teams

A big blocker for recognizing biases or fixing their outcomes is a shame. That will cause the team to go into Groupthink and herd mentalities. Ironically one bias takes you to many others.

At this point, everyone on the team should feel safe presenting a different perspective and even pointing out when someone may fall for a bias. No shame. Remember, we all have biases. Everyone in the team is human and can fall from them. Teams must be safe families that allow pointing mistakes without making them personal. The focus must be on finding a solution rather than a faulty human in the team.

This is a two-way avenue. The person pointing out the problem must feel safe doing it. But at the same time, the person receiving it must feel safe to admit it and work on it. No victims nor executors, just safe environments full of team members who understand we all err and that it is more productive to find a solution.

4.4 The outlander

Sometimes we need to see the forest for the trees. We may be so close to the problem, so deep in our biases, and even the whole team may be unaware of them. What can you do if no one feels they can tell the emperor that there are no new clothes and he's naked?

Bring a fresh perspective. Someone outside of the team, the division, and even the organization. By this, I do not necessarily mean a consultant. You only need someone from the outside or even with a different perspective. This could be your grandma, a foreigner, and, as mentioned, someone from another team, division, or background. You never know what people with fresh perspectives will be able to see.

Now, some consultants and organizations specialize in providing that feedback. They provide feedback that indicates which biases you are falling for. Always consider bringing external experts who may be used to catching errors in processes and conceptions.

4.5 Keep looking

You will keep falling for biases even if you accept that you are vulnerable, know the possible vulnerabilities, and team up to identify them, even if you bring external contributors. This is the sad truth.

But this doesn't mean you should give up and stop trying to avoid or fix the outcomes of these biases. Quite the opposite, you must stay vigilant and remember that you may fall to cognitive biases every new day. Most probably, you will fall for them again and again.

Keep accepting that you are human and vulnerable to cognitive biases.

Keep identifying new ones as scientists continuously discover new ones.

Keep making it safe for the whole team or org to point out and be given indications of biases.

Keep bringing external perspectives to indicate if you may be falling for biases.

And remember to keep talking to your organization and team to establish conversations that open everyone to these recommendations.

5 Closing

We are all humans that will keep failing for biases. Denying it is a biased idea. Ignoring them is another type of bias. Making it unsafe or forcing Groupthink is biased. And closing to external perspectives is yet another bias. And sticking to cognitive biases will only bring more biases and unwanted outcomes.

Stay vigilant for our human behaviors and love each other!

Adios and gracias!

Footnotes

1. Man with a hammer – AKA Law of instrument https://en.wikipedia.org/wiki/Law_of_the_instrument
2. Halo effect - https://en.wikipedia.org/wiki/Halo_effect
3. Sunk Cost fallacy – Economies definition https://en.wikipedia.org/wiki/Sunk_cost
4. Dunning-Kruger - https://en.wikipedia.org/wiki/Dunning%20%93Kruger_effect
5. Groupthink - <https://en.wikipedia.org/wiki/Groupthink>

References

Books:

McRaney, David. 2011. You are not so smart. Why You Have Too Many Friends on Facebook, Why Your Memory Is Mostly Fiction, and 46 Other Ways You're Deluding Yourself. Gotham press

McRaney, David. 2013. You Are Now Less Dumb: How to Conquer Mob Mentality, How to Buy Happiness, and All the Other Ways to Outsmart Yourself. Gotham press

Kahneman, Daniel. 2011. Thinking fast and slow. Farrar, Straus and Giroux

Interview

Interview with João Proença: <https://www.youtube.com/live/V7gpLUkFZPs>

Real time failure point detection and operational health monitoring of Cloud Infra

Vittalkumar Mirajkar
VitalkumarrMirajkar@gmail.com

Srivatsan P
srivatsan.p@skyhighsecurity.com

Abstract

Cloud Infra down time directly results in lost Engineering productivity time. Infra resilience is every engineering team's responsibility. In a complex cloud software solution, there are interconnected dependencies. This exposes us to multi point failure.

Current cloud infra monitoring process has challenges, the tools monitor individual parameters, and a real time co-relation is missing. In a complex interconnected services infra, a component failure has cascading effect. In case of an infra outage, reaching actual issue causing component is major portion of the troubleshooting time.

Proactive health monitoring is limited to checking feature running status post deployment. There is a gap how the system behaves during the actual deployment.

Our proposed solution to building robust software health monitoring system draws parallel from the health care process. During a critical operation, the entire patient monitoring is kept active to keep the overall situation under control.

On a similar analogy, in this paper we present how we addressed infra failure issues and real time monitoring during active deployment and regular time. We correlate Dependency tree, AWS Realtime logs, db queries, Jenkins and Harness logs, network connectivity, CPU consumption and many other attributes. We take inputs from multiple monitoring systems and correlate the data for better infra health status.

Biography

Vittalkumar Mirajkar is a Sr Manager at Skyhigh Security, with 18+ years of testing experience ranging from device driver testing, application testing and server testing. He specializes in testing security products and building CI/CD systems. His area of interest is performance testing, soak testing, data analysis and exploratory testing.

Srivatsan Parthasarathy is an SDE at Skyhigh Security, with 10+ years of experience in Automation Framework and tool Development. He specializes in building end-to-end frameworks and tools for automation testing for desktop, web, and mobile applications. He has expertise working with Cloud Providers and a deep understanding of Cloud FinOps.

1. Introduction

In cloud world, infrastructure outages are costly. They not only cause service disruption, outage cause reputation loss, and monetary loss as well. Not to account for the endless time spent by engineering teams to troubleshoot and customer sentiment management.

Cloud outages can be classified into broadly, but not limited into following categories.

External outage categories:

- Cascading failure of dependencies
- Outage caused by service providers.
- Cloud infrastructure failure due to resource starvation
- Network infrastructure issues

Internal outages categories

- Product issues
- Deployment related failures
- Resource starvation of nodes due to resource consumption due to prolonged running time.

In following chapters, we understand in some detail the cause for each of these failure categories.

Not all factors can be controlled, however as a SaaS provider some of these failure points can be predicted and mitigated.

2. Cloud Outage Classification

Below is a quick look at some of the reason for both External Outages and Internal outages,

2.1 External Outages reasons:

Factors contributing to external Outages, though not in control can be broadly classified into (but limited to) following categories.

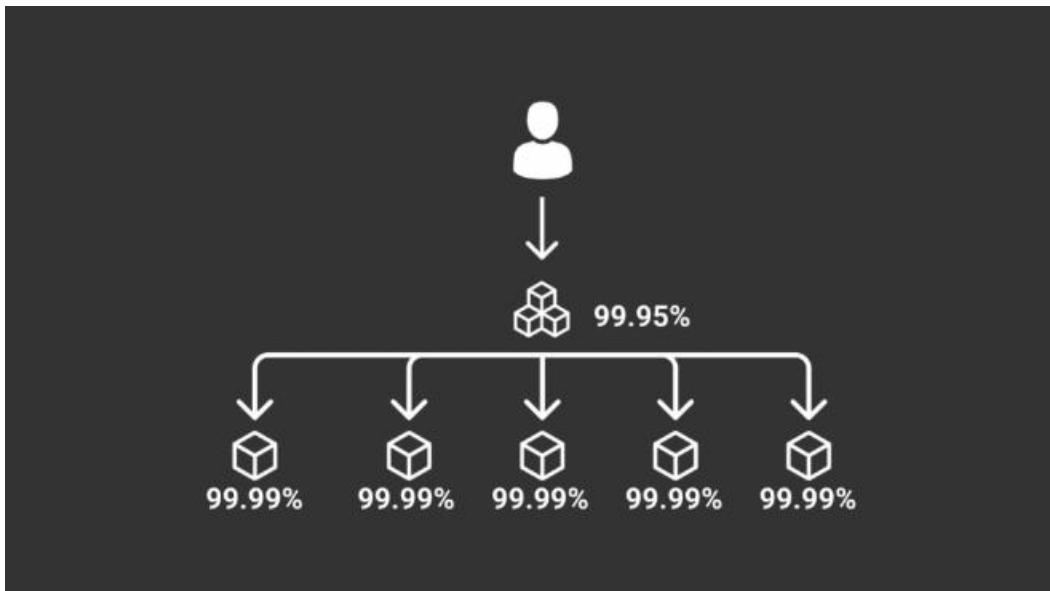
2.1.1 Cascading failure of dependencies

A SaaS solution is built on other apps. On an average digital business relies on 137 different service providers to power their software (Martens 2023). This introduces a large set of external failure points. Even when the SLA is 99.99% availability for each of the dependencies, in case of cascading failures, we need to move beyond SLA and determine a composite SLA.

$$\text{Composite SLA} = \text{No Of Dependencies} * 0.9999 \text{ (assuming 99.99% availability)}$$

For example, an app with 5 dependencies, all with four nines of availability, can only offer a 99.95% SLA itself.

Composite SLA = $5 * 0.9999 = 0.9995$, this is 99.95 % availability.



Service Availability: Even when you have Service Level Availability for all upstream dependencies at the highest level of 99.99%, it does not mean Service uptime (Neumeier 2018).

(System) Uptime != (Service) Availability

Availability Level	Uptime	Downtime per Year	Downtime per Day
1 Nine	90%	36.5 days	2.4 hours
2 Nines	99%	3.65 days	14 minutes
3 Nines	99.90%	8.76 hours	86 seconds
4 Nines	99.99%	52.6 minutes	8.6 seconds
5 Nines	100.00%	5.25 minutes	0.86 seconds

The higher the service availability opted or redundant fail-safe plan, would directly result in cost increases. Added, we need to have contingency plan of having multiple service providers and real time roll over playbook.

2.1.2 Outage caused by service providers:

Almost all cloud services providers have had outages (Korolov 2022) (Hicks 2022) (Taylor 2022). These are caused by various reasons, some popular one causes are.

- Network upgrades causing issues.
- Routine maintenance causing service disruption.
- CSP services failing and causing cascading effects on end users.

And many many more. While we do not have control in such a case till services resume at a CSP end. One potential de-risking factor to host services on multiple CSP. This though have a cost factor associated.

2.1.3 Resource Starvation:

Insufficient resource sizing could lead to processing choke points, compute starvation or network choke points or just not enough IOP's being factored in. Any of these could lead to service disruption.

While right sizing is the simplest solution, get the right sizing is not simple.

2.1.4 Network Outages:

In last couple of years Network outages are seen among even the biggest cloud solutions provider. A wrong Network routers configuration causing network shutdown, a surge in network activity (Taylor 2022), causing the entire service to collapse. Network outages led to cascading failures for downstream service to be left in unreachable state.

There is limited action where Downstream SaaS product can do in such a case. Visibility into upstream dependencies help us know if the failure is "Is It Me or Them?" (**Martens 2023**). Identifying the root cause is saves a lot of time lost in debugging. However, once the fault line is established as "Them", there is very little to do except for waiting for the upstream services to resume.

Though these outages constitute a major section of outages, this paper does not refer to addressing this issue. The focus of this paper is addressing areas which is under the control of SaaS application development team a.k.a internal outages.

2.2 Internal Outages reasons:

Internal outages can be anything where the root cause originates from within the company. Broad categories (but not limited to) are:

- Product issues
- Deployment related failures
- Resource starvation due to resource consumption of prolonged running time.

2.2.1 Product Issue:

Product feature failure directly resulting in service failure. While service monitoring will flag a service failure, but it is after the outage or service disruption. This disruption broadly can be controlled with better test infrastructure mimicking production load, production conditions.

Our Paper focus area is for below mentioned TWO Internal Failure Points.

2.2.2 Deployment related failures:

This is a unique case where maintenance window or a deployment window turns into an outage. This is due to not all the factors being considered during the product deployment. There could be unforeseen conditions which arise at run time, causing service outages.

Visibility into infrastructure, upstream dependency mitigates to some degree, but not fully covering this scope.

2.2.3 Resource starvation

Under resource planning, be it compute, network bandwidth, or storage creates bottlenecks resulting in service failures. This can be resolved by rightsizing. But as mentioned earlier is not that easy. Being dynamic in meeting resource demand is the best-case scenario.

Again, an comprehensive visibility into infrastructure and services helps us to scale up or down the resources. However, this does not trigger until a certain threshold is reached.

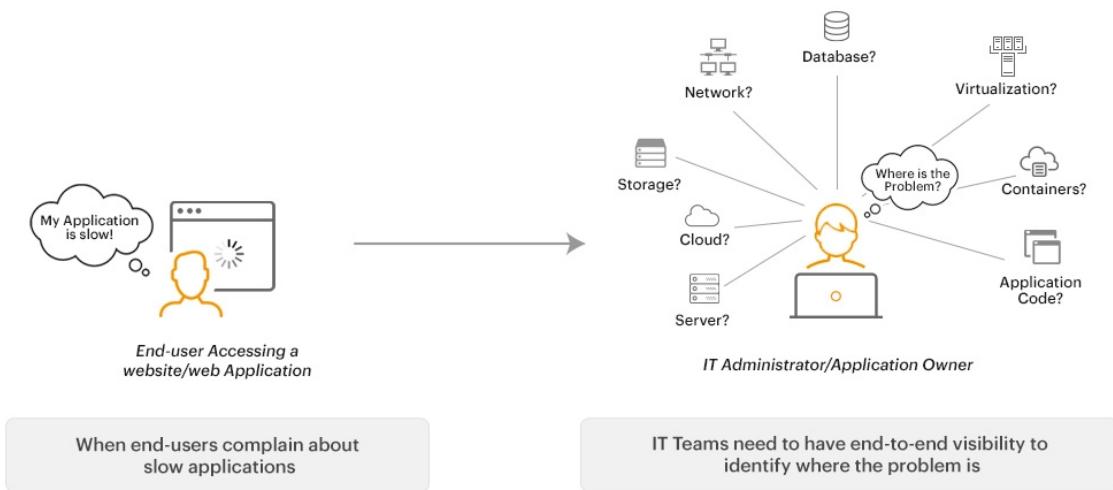
Not much has been proposed to mitigate Internal Outages reasons. Visibility into services and infrastructure is wildly consider as the go to in addressing this.

In the next chapter, we will look at what is the current visibility approaches and its shortcoming which we propose to solve.

3. Current Visibility approach and challenges

Effortless monitoring and Seamless troubleshooting, visibility from infrastructure to application both on external dependencies and internal dependencies is the key to having outages under control. This is easier said than done, as current tools used have limitation.

What we expect as SaaS solution provider when we talk about Visibility (Ellis 2022)



Despite the best planning, current monitoring and visibility challenges are,

- There are multiple tools used in monitoring.
- Still 1/3rd issues detected manually.
- 1/4th SaaS solution provider don't have full visibility (Basteri 2022)

There are just too many tools to be monitored and any real time correlation needs to be done by the engineer, introducing human elements into decision-making. The proposed solution to building a robust infra monitoring is a **2-track** problem.

- Address the correlation of existing multiple tools for better real time decision making.
- Proactively determine node failure by introducing real time monitoring, avoid maintenance windows flipping into outage windows.

4. Visibility across the spectrum and proactive node failure detection

There are multiple tools to do multiple monitoring, and all give out data into different tangents. These data points need to be correlated to make information readily accessible in decision making. This will greatly help in avoiding failures during maintenance windows where unforeseen issue arises turning a maintenance activity into managing outage.

Drawing our analogy from patient monitoring system, where entire health of patient is monitoring before operating on the patient, irrespective of which area is being operated. This is done to ensure we don't have cascading failures when the focus is on fixing / correcting a certain issue. If this is not done, a simple exercise can turn into crisis management.

Musing on similar approach we propose solution for the following TWO use cases.

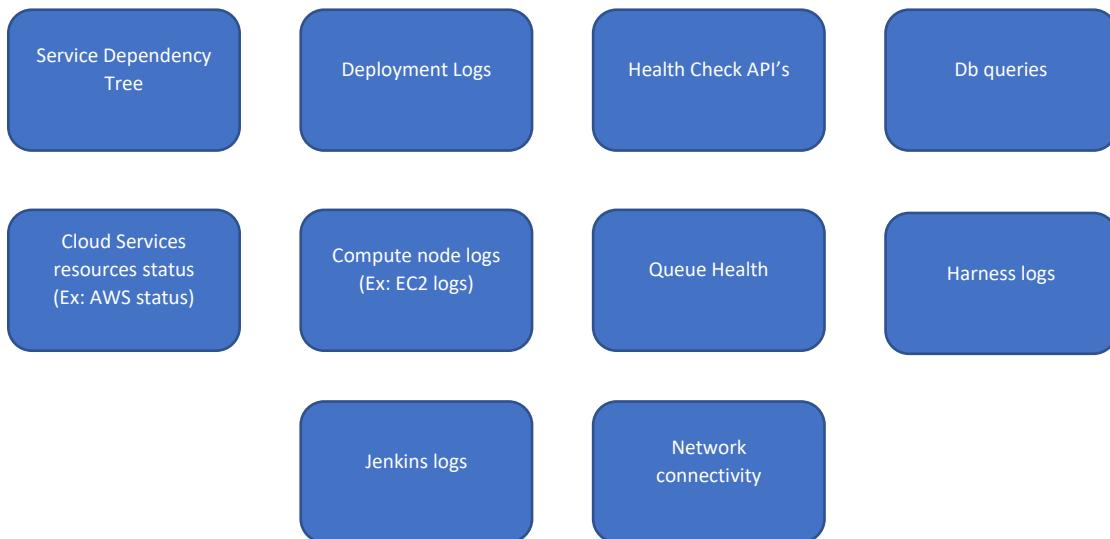
1. During upgrade / maintenance window how to ensure there are no unforeseen failures leading to outages
2. Proactively detect failure of node based on its node health (resource consumption)

4.1 During product deployment / upgrade window:

Monitor overall health of the infra to know if the systems which are to be patched are operating under near ideal condition. This ideal state is decided based.

- Current CPU consumed.
- Monitoring memory consumed vs available memory.
- Current event in processing queue
- Are the upstream dependencies fully functional and nodes in healthy condition for the component under question to undergo maintenance.

There are multiple monitoring utilities which are scattered and capture individual parameters only. Realtime co-relation is missing.



Irrespective of any activity (patching or upgrading), following checklist could provide a guideline.

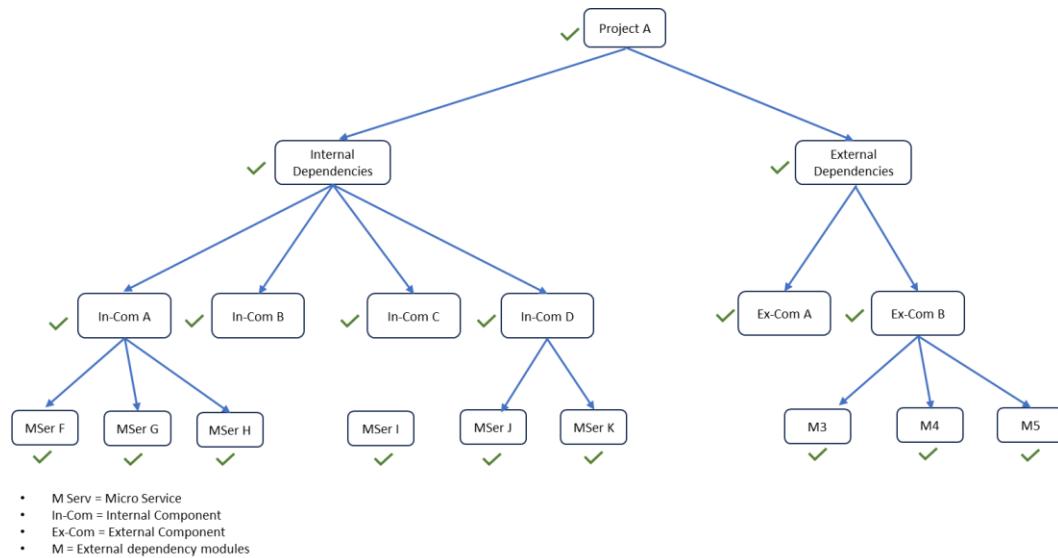
Ensure current state of the infra is healthy, this can be ensured by

- Current state of services running condition.
- CPU consumption < 60 %
- Memory consumption < 60 % of overall capacity
- Network availability < 50 %

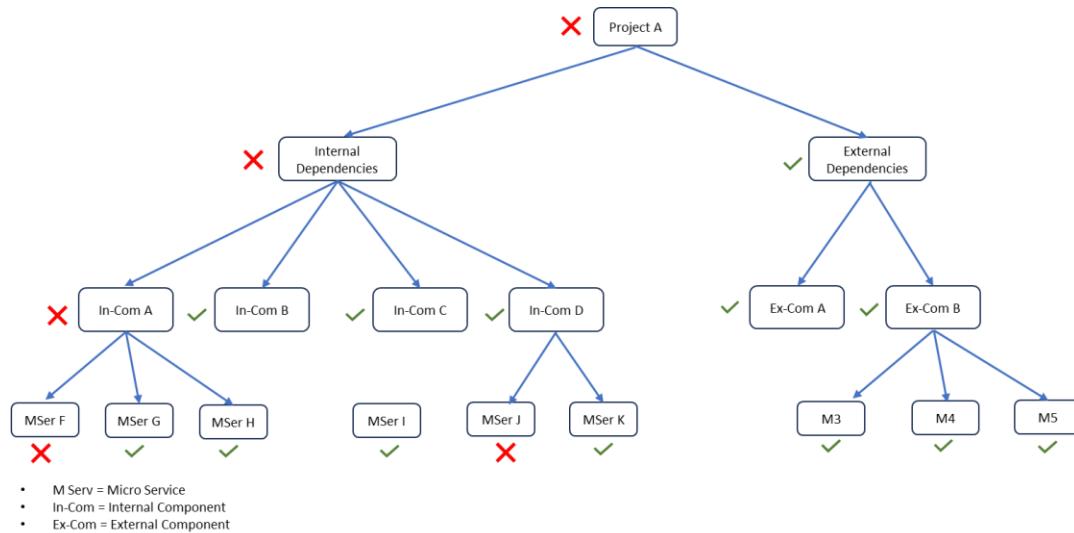
Once a stable state is established, infrastructure is ready for upgrade or patching. Beyond infrastructure monitoring, we need to monitor product upstream / downstream dependencies and their real time running states.

Below is the design of our design for Cloud Resource Monitoring and Service Dependency tree monitoring.

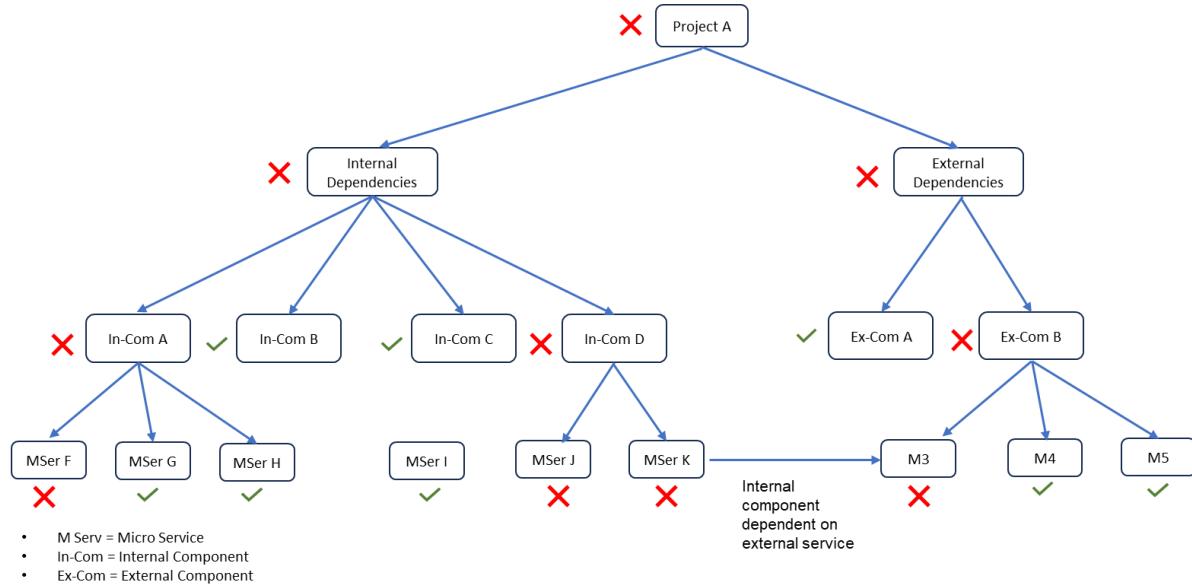
When all internal and external dependencies are Operational



When some internal and external dependencies are Non-Operational (Specifically when an internal dependency is down)



This logic can be extended to external dependencies as well. This tracking mechanism is true for both Infrastructure dependencies as well as Software Service dependencies, provided this is a clear method to establish its operation status.



Once Infra stability state is achieved, handling service dependency order is the next critical item in the flow. The sequence of draining the queue and downstream services shutdown is critical. Similarly, once the upgrade is completed, top-down sequence of restarting the flow is critical to getting back a working infra back online.

One thumb rule to be followed:

- Before upgrade: Shutdown Downstream dependencies 1st
- Upgrade
- After upgrade: Restart Downstream dependencies.

4.2 Proactive failure detection

A service failure could be because of following two issues.

- Software failure
- Infrastructure failure on which the software is hosted.

4.2.1 Software failure:

A service failure during deployment is easier to detect, whereas when a functional system fails, it takes time to isolate the problem. A real time service running status dashboard with clear upstream and downstream dependencies outlined helps quickly isolate the problem origin.

A node could be in running condition, however nonfunctional. Service running state alone might not be enough to establish infrastructure being functional. A periodic smoke test running across all critical paths covered, can aid a long way to determine if the nodes are fully functional.

4.2.2 Infrastructure failure:

During the normal working, because of any of the following (but not limited to) reason we may experience service disruption or node failure.

- Processing threads are hung some time from prolonged duration of usage.
- There is un-anticipated network spike / processing spike, choking the queues.
- Due to gradual resource consumption, the node is heading towards a failure, but not yet failed. Current monitoring system wait till the node under question reaches RED zone before triggering an alarm. Can this be preempted and highlighted when the node is in the YELLOW zone of resource consumption. This approach will enable us preempt node failures.

5. Implementation details and results

We implemented our model on one of our QA cloud infrastructures having more than 400+ nodes and running 140+ microservices.

We monitored,

- Infrastructure node level resource consumption
- 140+ micro service functional status

This greatly helped in real time health monitoring of the infrastructure. If one of the nodes under observation, health starts deteriorating, leading to failure, we can detect this condition real time and mitigate the same. Also, during the regular operational status, periodic sweeps of the environment with a smoke test ensured all the servers are not just running but also in functional state.

By this dual approach, we were able achieve.

- Environment stability and service availability, functional up time increased from 90% to 95%.
- We were able to eliminate resource starvation-based outage by >95%

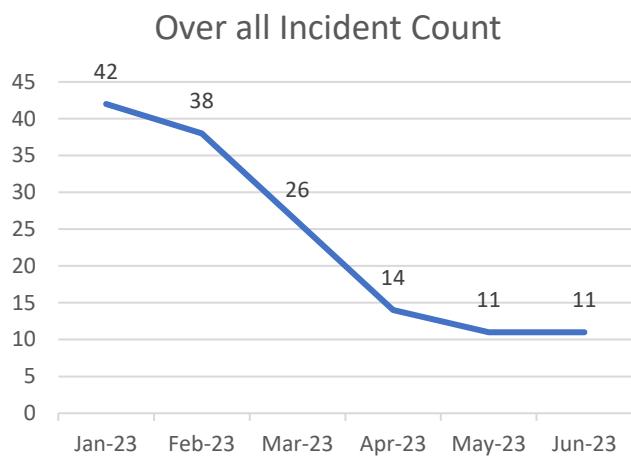
Following data parameters was measured spanning across 2 quarters.

- Total Outage Incident Count
- Post deployment failure
- Internal Service dependency related failures
- External Dependency failures
- Resource starvation related failures
- Time taken for debugging service failures (in hrs)

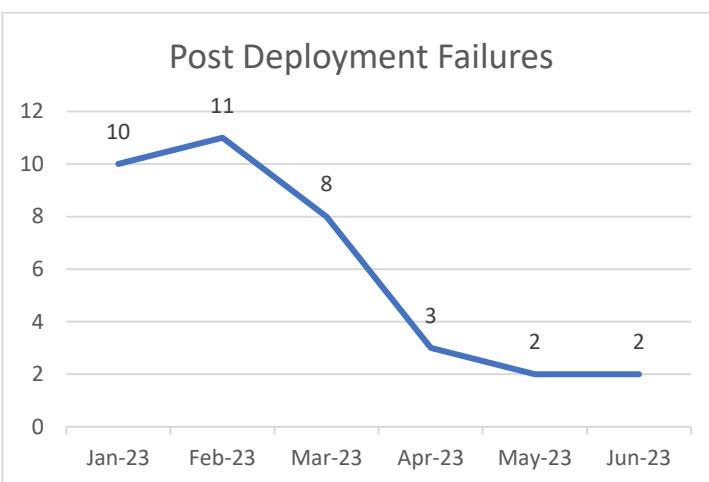
Q1 (Jan – Feb – March) this data represent condition before applying our monitoring framework.

Q2 (April – May – June), new monitoring system was implemented. We noticed considerable reductions in the time taken to identify and isolate the failure. The time taken to troubleshoot the failures also reduced.

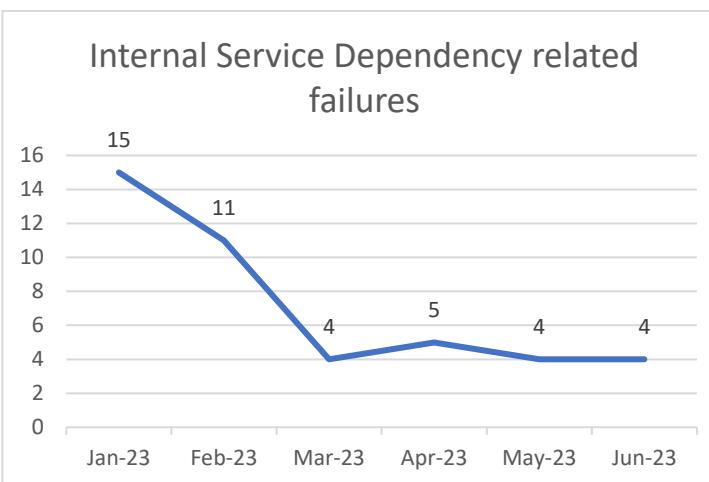
Below is data set captured for Q1 vs Q2 data pointers for the above said areas. There is an improvement observed.



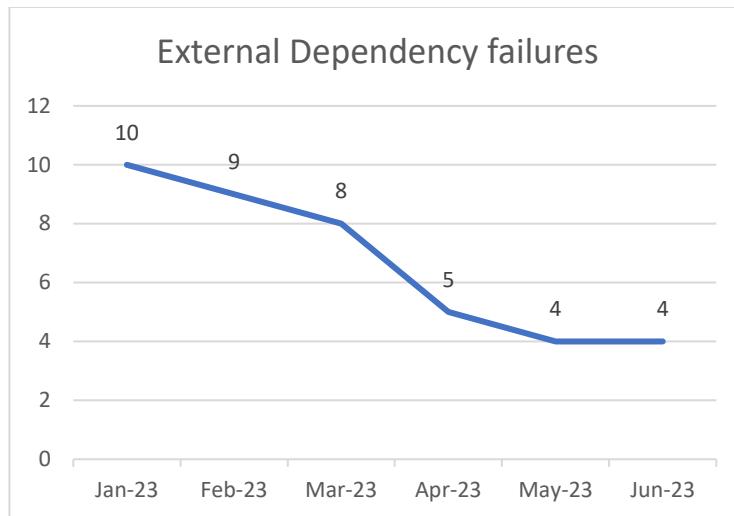
- In 2 Quarters we have observed overall drastic reduction in Incident count.
- Some of this reduction is direct related to the proactive monitoring system implemented.



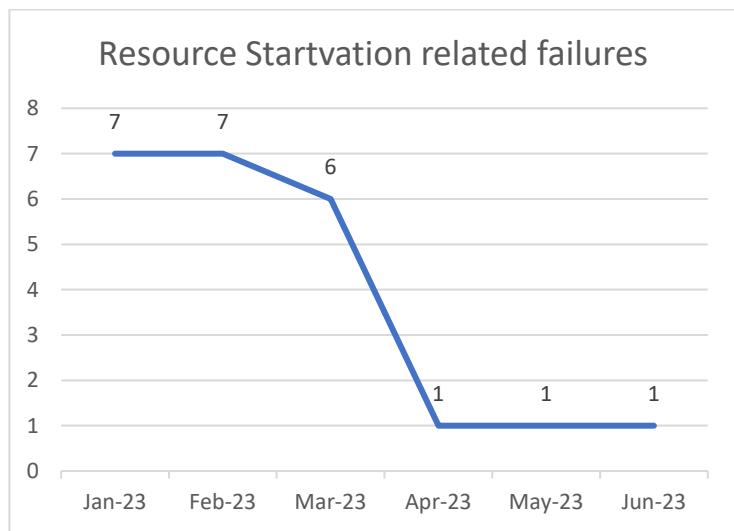
- Post deployment failure which are detected real time as there is real time 360degree monitoring of services.
- Post deployment failures are now handled in deployment window only.



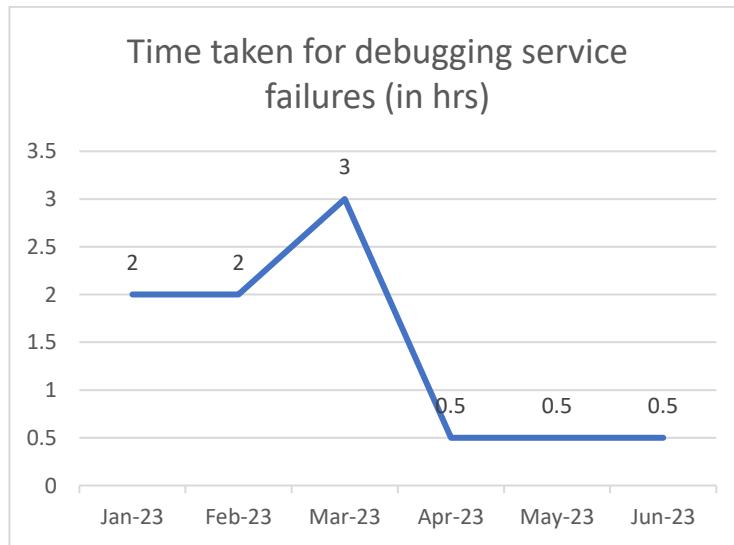
- Internal service-related failures are now instantly detected to the actual root cause.
- Service status check and interdependencies mapping is the key in helping in achieve ~80% reduction in incidents.



- There is little we can do when an External Failure is detected.
- Having visibility into 3rd party Infra Dependency / Software Dependency will help us address the question, "**Is it US or THEM**" as the root cause.
- Our monitoring system now gives instant insight into knowing the origin of the issue.



- Using the proactive monitoring, we have observed resource starvation related node failures are almost eliminated.
- We have been able Detect and Mitigate potential failures well within time.



- One major advantage we have seen is in time to reach root cause of issue.
- The monitoring system directly exposes the trigger service which has caused the failure.

6 Learnings and Take aways:

Some of our key take aways in our journey of getting our cloud infrastructure stable.

1. There is no one tool which can help you monitoring everything.
2. Correlation between data generated from different tools need to be automated.
3. Service status visibility is a must and most importantly service dependencies should be mapped.
4. Proactive hardware infra monitoring can help identify nodes which are leading towards failures, thereby giving us time to mitigate potential failures.
5. You will not be able to avoid all failures. When service failures happening how quickly we are isolating the root cause is equally important.
6. Periodic smoke tests on production environments helps us proactively check functional working status of all the services.

References

- Basteri, Alicia. 2022. *2022 Observability Forecast*. September. Accessed July 20, 2023.
<https://newrelic.com/observability-forecast/2022/about-this-report>.
- Ellis, Brent. 2022. *SaaS Outages: When Lightning Strikes, Thunder Rolls*. April 12. Accessed July 25, 2023.
<https://www.forrester.com/blogs/saas-outages-when-lightning-strikes-thunder-rolls/>.
- Hicks, Mike. 2022. *Seven Outages That Shook Up 2021*. January 6. Accessed June 12, 2023.
<https://www.thousandeyes.com/blog/seven-outages-shook-up-2021>.
- Korolov, Maria Korolov and Alex. 2022. *Top 10 outages of 2021*. January 31. Accessed July 5, 2023.
<https://www.networkworld.com/article/3648352/top-10-outages-of-2021.html>.
- Martens, Jeff. 2023. *The Overlooked Culprit Behind 70% of SaaS Outages*. Feb 23. Accessed June 10, 2023. <https://metrist.io/blog/the-overlooked-culprit-behind-70-of-saas-outages/>.
- Neumeier, Sascha. 2018. *The Uptime Myth: Why Uptime Does Not Mean Availability*. October 8. Accessed June 25, 2023. <https://blog.paessler.com/why-uptime-does-not-mean-availability>.
- Taylor, Twain. 2022. *7 Biggest Cloud Outages of the Past Year*. February 11. Accessed July 15, 2023.
<https://techgenix.com/7-biggest-cloud-outages-services-2021/>.

AnaCov - A novel method for enhancing coverage analysis

Mustafa Naeem, Ahmed Tahoob, Omar Ragi, Reem El-Adawi
mustafa.naeem@siemens.com ahmed.tahoob@siemens.com
omar.ragi@siemens.com reem.eladawi@siemens.com

Abstract

One of the many methods used for bug detection in software testing is the tracking and analysis of source code coverage, which helps reveal the percentage of original code covered by the software QA engineer's regression suite. This percentage can help identify any gaps that may exist in testing.

We introduce AnaCov, a tool that facilitates this process of code coverage testing. Using coverage data obtained from GCOV and stored in Git repositories, AnaCov maps testcases to functions and source files in an SQL database. This mapping enables quality assurance personnel to track coverage across time using versioned reports, and more importantly, to quickly test coverage of newly added code. This functionality is invaluable to both developers and QA engineers as it helps ensure better coverage over time, while minimizing time and disk space usage. It also allows users to merge several runs of coverage, creating a combined coverage report that is easy to analyze.

The AnaCov tool is designed for ease of use, with a user-friendly single command line interface that provides access to all functionalities, with no suite or product restrictions. Invoking the AnaCov tool allows users to map testcases to source files or functions within the original code, which enhances efficiency when analyzing coverage related to a selected subset of files.

The ability to easily query which testcases cover certain source files has a great benefit when it comes to quickly investigating coverage of code newly added to the source files. This knowledge enables better selection of pre-check-in testcases by targeting cases with the largest coverage of functions and source files under test, which reduces the time and disk space overhead necessary for adding and testing new code. Together, these functions result in higher quality code shipped to customers.

Biography

Mustafa Naeem is a SW Team Lead QA Engineer for Siemens Digital Industries Software. He has a B.Sc. degree in Electronics and Communications from Alexandria University, Alexandria, Egypt.

Ahmed Tahoob is a Senior QA Engineer for Siemens Digital Industries Software. He has a B.Sc. degree in Electronics and Communications from Menofia University, Menofia, Egypt.

Omar Mohammed Ragi is a SW QA manager for Siemens Digital Industries Software. He has a B.Sc. degree in Computer Science from Arab Academy for Science, Technology & Maritime Transport, Cairo, Egypt.

Reem El-Adawi is a Director of Quality for Siemens Digital Industries Software. She holds a B.Sc., M.Sc., and Ph.D. from Ain Shams University, Electronics and Communication department, Egypt.

1 Introduction

In today's fast-paced and technology-driven world, software developers and quality assurance (QA) engineers are under immense pressure to accelerate the pace of their software development life cycles. Meeting tight deadlines while ensuring good code quality becomes paramount in this competitive landscape. Code quality is the ultimate goal for every developer and QA engineer, as it directly impacts the reliability and performance of the software.

In the realm of computer science, code coverage emerges as a critical metric for evaluating the effectiveness of testing efforts. It quantifies the degree to which the source code of a program is executed when a specific test suite is run. Code coverage can be expressed as a percentage or a ratio, indicating the proportion of covered lines, branches, statements, or functions within the codebase.

High code coverage indicates that a significant portion of the source code has been executed during testing, suggesting a lower likelihood of undiscovered software bugs. On the contrary, low code coverage implies incomplete or missing tests, leaving room for hidden bugs or gaps in the application's functionality.

A general guideline is to aim for at least 80% code coverage, though the specific target may vary depending on the project's complexity and requirements. Achieving high code coverage yields numerous benefits, including increased confidence and trust in the codebase, reduced risk of introducing bugs or defects, saved time and effort in writing, testing, and debugging, and avoidance of rework or technical debt.

Automatically measuring code coverage at critical points in the development workflow allows developers and QA engineers to proactively identify areas with insufficient testing and take corrective actions. By integrating code coverage analysis into the development process, teams can continually monitor and improve their testing strategies, ensuring better code quality and a more robust end product.

With this goal in mind, the AnaCov tool is introduced to provide QA engineers with capabilities to ease the code coverage analysis.

2 AnaCov Philosophy

Coverage run result directories can quickly consume substantial disk space, leading to a need for frequent deletions to maintain sufficient storage capacity. When frequently adding new testcases to a regression suite in an attempt to increase code coverage, QA engineers face the challenge of preserving old coverage run results. These results must be preserved to merge them with the newly created testcases and generate a comprehensive coverage report for analysis. However, the large disk space occupied by coverage runs makes it impractical to retain them for an extended period, hindering the completion of the incremental coverage measurement process.

Additionally, the absence of a straightforward method to determine which testcase(s) cover specific source files or functions poses a challenge. Consequently, modifications to a source file require the entire regression suite to be run. With constant additions of new features and bug fixes, the QA team creates automated testcases to validate the changes. However, this process raises critical questions about the sufficiency of new testcases in covering all possible scenarios and identifying unreachable or dead code. Hence, tracking the coverage of newly added source code, functions, and lines is essential to address these concerns effectively.

The AnaCov tool built in the Go programming language [1] to tackle these challenges by providing a solution to compress and save coverage data without affecting the ability to generate coverage reports, map testcases to source files and functions, and measure incremental coverage while using publicly available applications (GCOV and LCOV) data [2,3]. In the following subsections, we define the main principles of AnaCov and show how they are incorporated into its design.

2.1. Resources optimization

The AnaCov tool prioritizes resource optimization as its most important principle. Conducting coverage runs can be time-consuming, often taking 20 times longer than normal runs. When dealing with hundreds of thousands of testcases, performing regular full coverage runs becomes challenging, as resources are finite and shared across all teams. Moreover, the large number of testcases can lead to disk space issues, making it difficult to retain coverage runs for extended analysis periods.

In scenarios with a high rate of code changes where only a few tests might be impacted, limited hardware resources may prevent running the entire regression suite for each code change. Instead, it is preferable to minimize the number of tests to be run by selecting those that cover the modified code only. This approach benefits not only QA engineers but also developers, as running fewer tests before committing code changes significantly improves turnaround time.

By optimizing resource usage and selectively running relevant tests, AnaCov empowers development and testing teams to efficiently manage their testing efforts, leading to faster testing cycles, reduced resource consumption, and improved productivity.

2.2. Usability and user experience

AnaCov's second principle revolves around usability, aiming to enhance the code coverage analysis experience for new or inexperienced QA engineers. The tool is designed to provide a mix of necessary functionalities in a single command with minimal options, enabling quick and efficient execution. By merging certain functionalities, AnaCov ensures speedy coverage analysis and swift access to results.

2.3. Centralization

Centralization is a vital concept for successful and maintainable testing automation. By centralizing key components of testing scripts and tools, development time and costs are minimized, avoiding redundant work. Maintenance becomes simpler as modifications or fixes can be applied in one location, benefiting all testcases that use the centralized tool. AnaCov embraces centralization to enhance maintainability. With multiple teams using the same coverage analysis tool, generating combined coverage reports for products sharing functionalities becomes effortless. The tool's shared database allows querying testcases executing specific functionalities across different products, streamlining testing expansion. Centralization in AnaCov optimizes efficiency, lowers costs, and fosters collaboration among teams, contributing to successful testing automation.

2.4. Modularity

AnaCov is designed with a modular approach, where each part of the tool is developed as separate code source files. This modularity ensures that if a defect occurs in one module, it does not impact other modules or pieces of code. Additionally, the modular structure simplifies the process of identifying and resolving errors, enhancing the overall robustness and maintainability of the tool.

3 AnaCov Components

AnaCov, a comprehensive code coverage tool, is built upon three primary modules that collectively facilitate efficient data management and analysis. These modules are the Data Storage module, the Search/Query module, and the Historical Coverage module (Fig. 1).

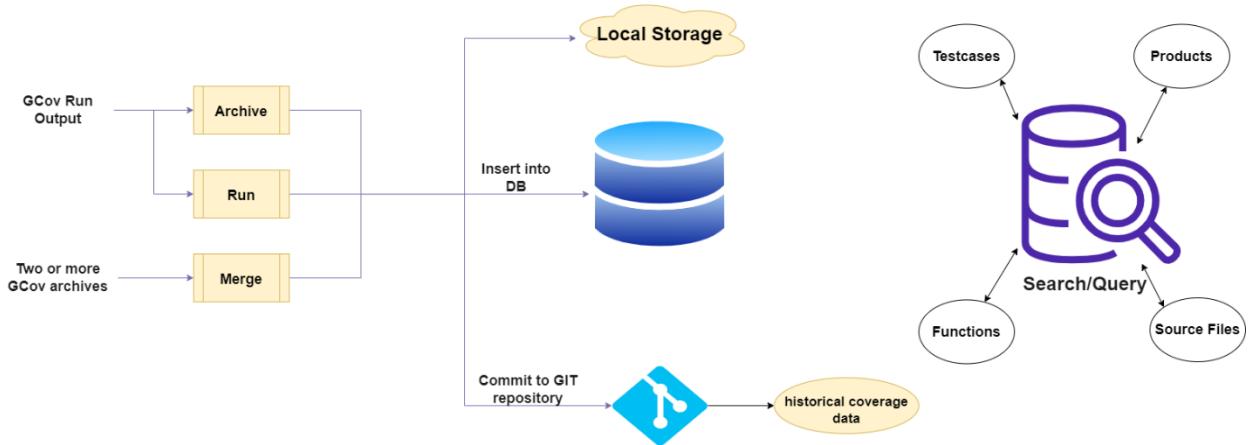


Fig. 1. AnaCov components

3.1. Modules

Regarding input/output, AnaCov takes as input the GCOV run data, which contains information about the code coverage achieved during testing. The output of AnaCov includes a detailed coverage report, showcasing the percentage of code covered by tests. Additionally, AnaCov provides a selected list of testcases, functions, or source files, enabling developers and QA engineers to focus on specific areas of the codebase that require further testing or improvement.

3.1.1. Data Storage

The Data Storage module serves as the backbone of AnaCov's functionality. It comprises two essential components: Local Storage and a Database. Local storage acts as a centralized local space in a shared location to house all the coverage data. This centralized approach ensures easy accessibility and management of data across different projects and team members. This space doesn't have any special control, it is just used to have all the data managed in one place rather than different locations for the same team members.

The database component plays a critical role in AnaCov's effectiveness. It stores the coverage output data generated during code analysis. By leveraging database capabilities, users can efficiently retrieve and utilize coverage data instantly or at a later stage, enabling them to make informed decisions and track code coverage progress. This database is supposed to keep the latest data, so for the first time a new testcase, a new source file, or a new function is added, new records are added corresponding to that in the database, otherwise the current records are updated to match the latest updates. The QA engineer has also the capability to store the data of different releases by adding a corresponding tag to the product name.

3.1.2. Search/Query

The Search/Query module acts as the user-facing interface, bridging the gap between the database and the end-users. Through this module, developers and QA engineers gain access to powerful tools that allow them to interact with and extract relevant information from the stored data. This module empowers users to perform targeted searches, obtain specific code coverage metrics, and gain insights into the quality of their codebase.

3.1.3. Historical Coverage

The Historical Coverage module is another essential component of AnaCov. It utilizes the Git version control system to maintain a comprehensive history of coverage status across different development

milestones or releases. This feature aids in tracking code coverage trends over time, facilitating a deeper understanding of codebase evolution and providing valuable insights into testing and development progress.

The key concept is that essential data for creating a coverage report is kept within an individual text file for each testcase (coverage.info). Thus, preserving this file along with its historical modifications ensures the capability to access a coverage report snapshot at any given moment (e.g., a particular release or date).

3.2. Processes

AnaCov includes several essential processes that contribute to its effective code coverage analysis.

3.2.1. Archive

The Archive process is designed to optimize disk space usage. As coverage runs can consume significant disk space, this option automatically extracts only the necessary files from the directory and generates an archive. The required files within the archive are those that store the coverage information and are essential for generating the coverage report. By archiving and compressing the necessary data, AnaCov ensures efficient storage while retaining the vital information needed for analysis.

3.2.2. Run

The Run process is the core of AnaCov functionality. It takes either the previously generated archive or the coverage data directly to generate the comprehensive coverage report. The process involves analyzing the codebase and the corresponding test suite to determine the extent of code coverage achieved. The generated coverage report provides valuable insights into the effectiveness of the testing efforts and highlights areas of the code that may require additional testing. Additionally, the Run process can feed the coverage data into the database and/or commit it to the Git remote repository, facilitating further analysis and collaboration among team members.

3.2.3. Merge

The Merge process enables the combination of multiple archives. This feature is particularly useful when dealing with large-scale projects that produce separate coverage reports for different components or modules. By merging these individual archives, AnaCov can provide a unified and comprehensive coverage analysis for the entire project. Merge ensures a holistic view of the code coverage, helping QA engineers, and developers to understand the overall test coverage across the entire codebase.

4 Archiving

When dealing with tens of thousands of testcases per product and adding more testcases on daily basis, testcase data needs tens of terabytes of disk storage. Teams should always be looking for ways to minimize the disk space that files occupy on a hard drive and reduces the time needed to transfer them. This reduction of space and time can result in significant cost savings. Compressed files require significantly less storage capacity than uncompressed files, meaning a significant decrease in storage expenses. Compressed files also require less time for transfer while consuming less network bandwidth. Which can reduce costs and increase productivity.

AnaCov provides two modes to deal with full coverage runs and archived runs.

4.1. Full Runs

In this mode, AnaCov handles full coverage runs. Since a coverage report can be generated using only coverage files, only coverage files are extracted from the coverage runs results directories and

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

PNSQC.ORG

Page 5

compressed. All other testcase-related files are disposable. The compressed archives are by default added to a central directory where all product archives are kept.

In the *Archive* command, only the full unarchived coverage run argument is mandatory for this mode. There are two optional arguments: updating the DB with the coverage data, and/or updating the Git repository.

```
Anacov archive -m <full coverage run> --update_db --update_git
```

Using the *Archive* command for a Siemens product reduced the disk space of a full coverage run from 670 GB to only 20 GB. The *Archive* command deleted all the testcase-related files and kept only the coverage files, then compressed them.

4.2. Archived Runs

When interacting with a generated archive, the initial inclination may be to let the user unzip the archives manually and interact with the data this way. While this approach isn't necessarily wrong, the AnaCov application provides a more controlled environment for interacting with the archive.

The *Run* command provides an interface for interacting with the archives. It should be used to generate coverage reports from existing archives (zip files) previously generated by the *Archive* command.

This command saves the user the hassle of unzipping the archive, generating a coverage report, and cleaning up by performing all these tasks automatically.

The archived coverage run argument is mandatory for the *Run* command. There are two optional arguments: updating the DB with the coverage data, and/or updating the Git repository.

```
Anacov run -m <archived run> --update_db --update_git
```

4.3. Merge

The AnaCov *Merge* command allows user to merge multiple archives together. When users have multiple zipped files from multiple products or different suites in the same product, they may need to merge them together as a method of cleanup, or for experimentation purposes.

```
Anacov merge output_archive [input_archive1 input_archive2 .....]
```

5 Mapping

The mapping of testcases to source files and functions is a crucial and fundamental feature in the AnaCov solution. This functionality holds significant importance due to its ability to provide valuable insights into the relationships between testcases and code components, facilitating comprehensive code coverage analysis.

By establishing these mappings, QA engineers can easily determine which testcases cover specific source files or functions and vice versa. This information aids in selecting representative testcases for the functions being tested. Ensuring the proper functioning of existing functionalities is essential when adding new features to the software. Running regression tests before implementing any changes is a standard practice to validate code integrity. However, regression suites can be extensive, leading to time and resource consumption during testing. By selecting a subset of regression testcases that may be affected by the new code, QA engineers can save time and hardware resources while ensuring adequate code coverage.

For example, if a developer is modifying an existing file or function, running only the testcases that cover that file or function can be sufficient to verify that the new code does not disrupt any previously functioning functionality.

AnaCov's mapping functionality enables the generation of lists of testcases that represent specific functions or source files based on various parameters like product name, a list of source files, etc. To achieve this mapping, data must be stored and easily accessible over time. Therefore, AnaCov uses a PostgreSQL database with dedicated tables to handle testcase names, products, source files, functions, and other related information.

By employing a robust database and implementing the mapping functionality, AnaCov empowers software development teams to conduct efficient code coverage analysis, make informed testing decisions, and ensure the reliability and stability of their software products.

Fig. 2 is the diagram of the entity relationship diagram (ERD) database providing an overview of the tables, and the relations between them.

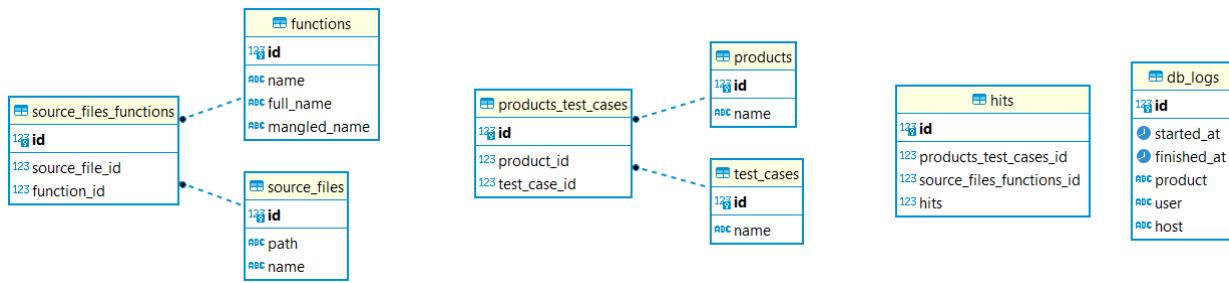


Fig. 2. AnaCov database diagram

5.1. Insertion

Data insertion into the database occurs during archiving or running processes in AnaCov. The commands provide `update_db` option to update the database with new or updated data as needed.

5.2. Querying

AnaCov offers multiple query options to identify coverage gaps and obtain relevant information. QA engineers can use queries to find testcases that cover specific files, functions, or the entire testcases for a particular product. Additionally, queries can be made to identify source files generating the covered code for specific testcases and products. The following code examples demonstrate various query options.

Testcases matching query:

```
Anacov search testcases { -p [...products], -s [...source_files], -f [...functions] } [flags]
```

Source files matching query:

```
Anacov search sourcefiles { -t [...test_cases], -p [...products], -f [...functions] } [flags]
```

Functions matching query:

```
Anacov search functions { -t [...test_cases], -s [...source_files], -p [...products] } [flags]
```

Products matching query:

```
Anacov search products { -t [...test_cases], -s [...source_files], -f [...functions] } [flags]
```

The optional *flags* provide additional functionality to the commands, such as generating output in a comma-separated value (CSV) file, limiting result numbers, or using regular expressions for searching instead of exact names for testcases, source files, or functions.

6 Incremental Coverage

As part of a new feature's validation process, QA engineers create testcases. A critical question is how to make sure created testcases efficiently cover the newly added or modified codebase of the feature under test. AnaCov's Incremental Coverage feature helps by checking the coverage of the newly added or modified code using the created testcases apart from measuring the whole coverage, which is more efficient, faster, and more optimized.

Also, whenever QA engineers need to measure coverage of a feature developed in a certain period, they must contact the developer who implemented that feature asking for a specific list of functions and source files that cover the feature only to be able to measure its coverage alone. By using the AnaCov Incremental Coverage feature, that request process is not necessary anymore.

The main idea behind generating a coverage report for the newly added or modified code only, is manipulating the instrumented lines in the cumulative coverage file. Instrumented lines are the lines that represent code lines (comments, includes, and empty lines are not instrumented lines). If the code line number is mentioned in the coverage file, it's an instrumented line that represents an executable line of code.

When QA engineers provide AnaCov with two dates of certain commits and as a future enhancement AnaCov will support an input of two source code branches, AnaCov checks which lines of code were modified or added between the two dates, and then keeps those lines only in the cumulative coverage file. Only those lines retained in the cumulative coverage file are instrumented lines and old code or unchanged code is considered to be source code comments, and its coverage is not measured.

After that, when a HTML coverage report is generated, only recently modified, or added code will be measured for coverage.

6.1. Use Model

The expected use model for AnaCov's Incremental Coverage feature involves QA engineers measuring feature coverage before starting validation and testcase creation, and then measuring it again after completing all testcases in the test plan.

Using a single command and a new coverage run for the created testcases, it becomes effortless to measure incremental coverage whenever a new testcase is created to validate a missing scenario.

Moreover, the incremental coverage mode can be employed in automated regular coverage runs, generating continuous HTML coverage reports. These reports can be used by developers and QA engineers to monitor and enhance the coverage of newly modified or added code.

The process is simplified for QA engineers, who only need new coverage run results, two dates representing the coverage report's time frame, and a list of source files to generate the report for the newly modified or added code. This seamless process facilitates efficient coverage analysis and contributes to the overall success of feature testing and code validation.

QA engineers should ensure the new date is set at most to the date the coverage run is given to the command itself or before.

Anacov datedcov -m <coverage run> -N <New date> -O <Old date> -f <source files>

6.2. Output files and HTML report

Three main files are generated when using the *DatedCov* command:

- *Dated_Coverage.info* → Coverage.info file used by LCOV to generate the HTML report
- *HTML_report* → HTML report to visualize the functions and lines coverage
- *Source_files_versions* → source files versions that were checked between the two dates

The generated HTML report has three types of lines (Fig. 3):

- 1- Lines highlighted in blue are the lines that were executed by the testcases.
- 2- Lines highlighted in red are the lines that were not executed by the testcases.
- 3- Unhighlighted lines are comments, includes, empty lines, and lines of code that were not changed during the two dates provided by the QA engineer.

```

3420 :     if(!validation_status)
:     {
:         extern Pex_message_handler Global_Message_handler;
0 :         Global_Message_handler.print_summary();
0 :         safe_exit( 1 );

```

Fig. 3. AnaCov HTML report

7 Coverage History Tracking

The Historical Coverage feature in AnaCov utilizes the powerful Git version control system, to maintain a comprehensive record of code coverage status across different milestones or releases. When AnaCov is run over a regression suite or a part of it, the coverage data is committed to the remote Git repository, preserving a historical log of the coverage metrics.

By leveraging Git's version control capabilities, AnaCov's Historical Coverage feature empowers software development teams to gain valuable insights into code coverage trends over time. It facilitates a deeper understanding of the evolution of test coverage and aids in making informed decisions to improve code quality and testing strategies. QA engineers can use this historical data to identify patterns, assess the effectiveness of testing efforts, and ensure continuous improvement in software development processes.

7.1. Features

To retrieve and make use of this historical coverage data, AnaCov provides three essential commands: *Get*, *Peek*, and *Log*.

7.1.1. Get

The *Get* command allows users to download coverage data for a specific regression suite or a list of testcases based on various criteria. Users can specify a date or a commit hash or request the latest data (HEAD) to obtain the desired coverage metrics. For example, QA engineers can check the coverage for a specific product release or track the coverage progress from one release to another.

Examples:

A specific date: *Anacov get –product <product name> –date <date>*

A specific commit: *Anacov get testcase –product <product name> –commit <commit hash>*

The latest: *Anacov get testcase –testcases <list of testcases> –product <product name>*

7.1.2.Peek

The *Peek* command provides a list of testcase names available in a particular commit, date, or the latest data. This command is particularly useful when new testcases are added over time to cover new product features or enhance coverage for existing functionalities. By tracking these changes, QA engineers can gauge the impact on overall code coverage.

Examples:

A specific date: *Anacov peek –product <product name> –date <date>*

A specific commit: *Anacov peek –product <product name> –commit <commit hash>*

The latest: *Anacov peek –product <product name>*

7.1.3.Log

The *Log* command prints the commit logs and their associated hashes from the local repository. These commit hashes can then be used with the *Get* and *Peek* commands to access the relevant coverage data for specific commits or time ranges.

Examples:

All: *Anacov git log*

Date range: *Anacov git log –since <date> –until <date>*

From specific commit: *Anacov git log –from <commit hash>*

8 Use Models

This section will present illustrative examples of utilizing the AnaCov tool across various scenarios and demonstrating the combined use of its features to fulfill specific use cases.

When QA engineer starts the validation of a new feature, a critical aspect is assessing the extent to which newly added code will be covered by forthcoming testcases. AnaCov's Incremental Coverage feature can address this by gauging the coverage of feature-related code before and after testcase creation. This two-step coverage measurement ensures thorough coverage of the newly developed feature.

Leveraging the Git feature in AnaCov eliminates the need to retain old coverage files, enabling effortless generation of HTML coverage reports for historical coverage runs. This simplifies the presentation of coverage statistics for different source code branches.

For customized builds or daily code changes, optimizing regression testing becomes imperative. AnaCov's database facilitates the generation of a tailored list of testcases covering modified code only. This allows for targeted test runs instead of full regressions with every code alteration.

In large product environments, distinct QA engineers often test different product features. Utilizing AnaCov's *Archive*, *Merge*, and *Run* commands, coverage runs for different features in the same product can be organized and merged to reduce disk space. This consolidated archive can then be employed to update Git repo, map testcases to code elements, or generate comprehensive HTML reports for the product under test.

Enhancing partially covered source files and functions often involves modifying testcases executing code from these components. By utilizing AnaCov's database to map testcases to these files/functions, QA engineers can strategically modify testcases to boost coverage.

These are just a few illustrative scenarios of AnaCov tool application. The tool's versatile features can be combined in numerous ways to cater to diverse use models and enhance testing practices across different development contexts.

9 Conclusion

By measuring code coverage at critical points in the development workflow, developers and QA engineers can proactively identify and correct any deficiencies. Introducing automated code coverage analysis into the development process enables teams to continually monitor and improve testing strategies, ensuring better code quality and a more robust end product.

The innovative AnaCov tool provides developers and QA engineers with multiple automated capabilities to simplify, speed up, and standardize code coverage analysis. Through its use in production environments, we demonstrate the AnaCov successfully simplifies the activities related to code coverage analysis, making the tool invaluable for software testing purposes. Users with any level of expertise can quickly and easily interface with coverage data, perform the necessary analysis, and develop a future plan to provide better code coverage in their test suites, without sacrificing huge amounts of time or disk space.

10 Acknowledgements

The authors would like to thank Siemens Calibre QA team for their help with AnaCov testing, including their insightful and helpful ideas and feedback that contributed to making the tool more mature. We would also like to thank Shelly Stalnaker for editorial assistance in the preparation of this manuscript.

References

- [1] Go programming language: <https://go.dev/doc/>
- [2] GCOV—a Test Coverage Program: <https://gcc.gnu.org/onlinedocs/gcc/GCOV.html>
- [3] LCOV: <https://LCOV.readthedocs.io/en/latest/>

Chatbots As Meeting Stand-In: Enhancing Remote Work Productivity

Author(s)

Kavitha.naveen@tektronix.com Naveen.p@northeastern.edu

Abstract

The COVID-19 pandemic has led to a significant increase in remote work, resulting in a higher number of online meetings and causing stress on team members. While remote work provides flexibility, it has also led to an increased dependence on online meetings. A recent study found that 70% of meetings keep employees from doing productive work (Benjamin Laker, n.d.). Excessive meetings have been shown to negatively impact productivity and cause frustration among employees (DEEB, n.d.).

One potential solution to this problem is to develop a chatbot that can substitute team members who are unable to attend meetings, providing answers to common questions and updates on project status.

This paper recognizes that while the use of technology to replicate human presence in meetings is innovative, its applicability is contingent upon the specific nature of interactions. The research underscores the potential advantages of utilizing meeting stand-in chatbots for routine information dissemination and status updates, presenting a case for reducing stress and enhancing work-life balance among employees. By freeing valuable time from repetitive meetings, employees can redirect their efforts toward tasks that demand their unique expertise, potentially fostering heightened job satisfaction and overall productivity.

By investigating both the "build" and "buy" approaches to implement such chatbots, the paper provides insights into the decision-making process for organizations seeking to leverage this technology. Furthermore, the analysis highlights the implications of introducing chatbot representation in meetings, including factors like context understanding, communication style, and technological limitations.

The paper also addresses key considerations for the successful implementation of a chatbot, such as regular updates with accurate information, and trained to respond to a wide range of questions. The design flow of implementing a chatbot is also discussed.

Biography

Kavitha is a senior lead with over 20 plus years of experience in the field of software quality assurance and project management. She is a highly motivated professional who is passionate about quality and processes, and has worked extensively in improving the product quality of various products in Tektronix Inc. She has a strong background in developing and implementing quality assurance processes, procedures and methodologies,

Pawan Kumar is a MS student at Northeastern University Massachusetts, Boston, United States

Copyright Kavitha Naveen:09/01/2023

1 Introduction

In a traditional office setting, face-to-face conversations often take place by simply walking over to a colleague's cubicle. However, with the work-from-home arrangement, even minor discussions necessitated scheduling online meetings. This increased frequency of meetings has several drawbacks, including a potential decline in productivity and negative impacts on individuals' well-being.

It is important to acknowledge that not all meetings require the physical presence of every participant, and exploring innovative alternatives can be beneficial.

By utilizing Artificial Intelligence (AI) for meeting representation, we can address some of the challenges posed by excessive meetings. AI-powered solutions can automate certain tasks, provide real-time assistance, and streamline communication during meetings. This reduces the need to attend unnecessary meetings and allows participants to focus on essential discussions, thus enhancing productivity and alleviating the burden on individuals.

The subsequent section of the paper explores the significance of meetings and provides insights into leveraging an AI-powered chatbot as a representation in meetings.

2 "Meeting Overload: Impact on Productivity"

2.1 Why do we require Meetings?

1. Understand the requirements during project requirement phase.
2. Planning meetings to discuss on the project plan and timelines.
3. Discuss on design details during design phase.
4. Understand the testing aspects during testing phase.
5. Understand the progress of the project.
6. Discuss on issues and defects.
7. Discuss on learnings to take it forward.
8. Project closure meetings

The above are high level meetings that are inevitable. Hence, we can see that, there are good many advantages in scheduling and attending meetings. It is up to the team members to understand and plan for the meetings that they need to attend or alternatively find out innovative ways to attend the meetings.

2.2 Meeting Calendar

Consider the typical calendar day depicted in Figure 1 below. This day is packed with meeting requests, commencing in the morning, and extending into the late evening hours. Between these scheduled meetings, there exists a valuable gap of ~ 2.0 hrs. (excluding the designated lunch break) that can be optimally utilized for project-related tasks. Hence optimization in attending meetings plays a key role.

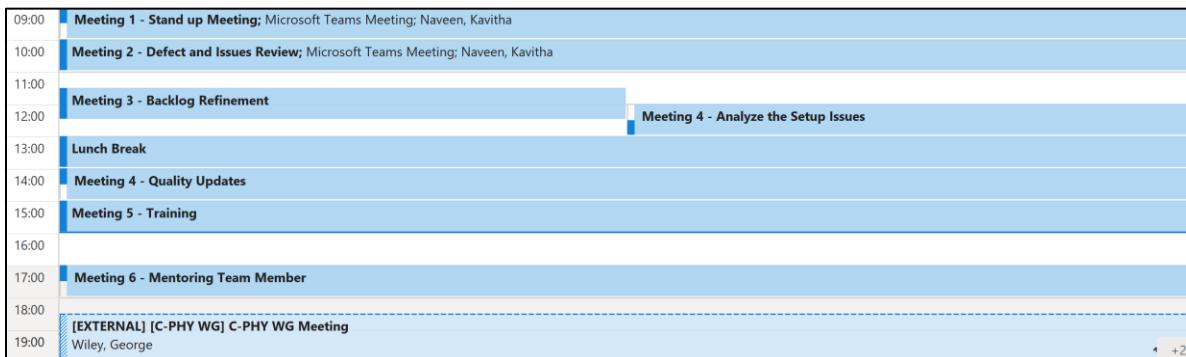


Figure 1: Meeting Calendar

2.3 Root Causes of Ineffective Meetings

The top reasons why meetings are ineffective includes the following:

2.3.1 Time lost in meetings.

Usually for every meeting there are few team members who join late and some team members who extend their conversations and hence meetings get extended. For example, as per the Table 1 you can see that if we consider any three meetings in a day, due to the start time and end time, the time lost is around 400 minutes considering typically a team size of 5 to 7 team members.

Meeting Details	Actual Meeting Time	Start Time	End Time	Team Size	Extended Time (min.)
Meeting 1	9:30 AM to 10:00 AM	9:35 AM	10:10 AM	6	90
Meeting 2	2:30 PM to 3:30 PM	2:35 PM	3:45 PM	5	100
Meeting 3	5:00 PM to 5:30 PM	5:00 PM	6:00 PM	7	210
Time lost due to Meeting Extension					
Per day due to 3 Meetings	400 min ^{Note 1}				
Per week assuming 5 working days/week	2,000 min ^{Note 2}				
Per month assuming 20 working days/month	40,000 min ^{Note 3}				
Per Year assuming 12 months	8,000 hrs ^{Note 4}				
Total cost to the company assuming 40\$/hr.	3,20,000 \$ ^{Note 5}				

Note 1: (90 + 100 + 210) min. Note 2: (400min x 5days) min. Note 3: (2,000 * 20) min. Note 4: (40,000 * 12) /60 hrs. Note 5: (8000 hrs * 40 \$)

Table 1: Time lost in meetings.

2.3.2 Efficiency of Meetings

A study by the University of North Carolina (source [Stop the Meeting Madness \(hbr.org\)](#)) surveyed 182 senior managers in different industries about their meetings and the results showed that:

- 71% of them found their meetings to be unproductive and inefficient.
- 65% said that meetings keep them from completing their own work.

Another limitation of meetings is irritations caused to an employee due to various reasons as given in Figure 2.

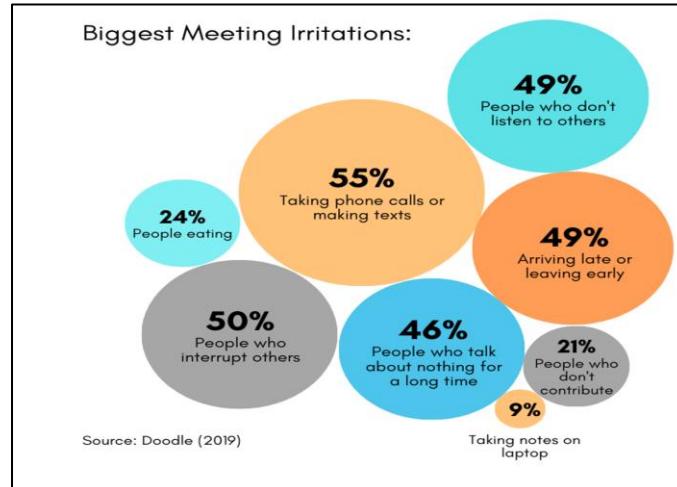


Figure 2: Meeting Irritations

2.3.3 Consequences of Ineffective Meetings

- 44%: 'Not enough time to do the rest of my work.'
- 43%: 'Unclear actions leading to confusion.'
- 38%: 'Bad organization results in a loss of focus on projects.'
- 31%: 'Irrelevant attendees slows down progress.'
- 26%: 'Inefficient processes weaken client/supplier relationships.'

Source : [Min. \(Wasted\) of Meeting: 50 Shocking Meeting Statistics | BOOQED Blog](#)

3 Chatbot as Meeting Stand-In

In Section 2, we discussed the benefits of effective meetings, as well as the reasons why some meetings can be ineffective, resulting in costs to the company. Let us examine the problem and the solution that the meeting stand-in chatbot provides, as specified in Figure 3

3.1 What is the problem and how can Chatbots help us as Meeting Stand-In?

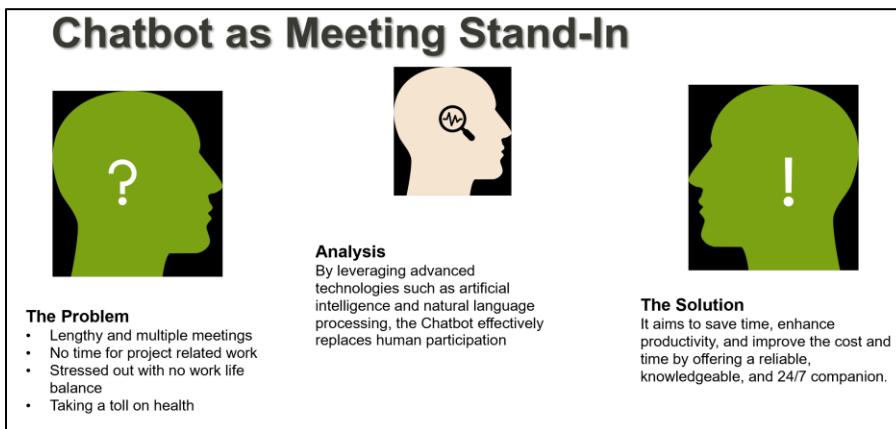


Figure 3: Problem and Solution

Let us take a deep dive on the advantages of having chatbots as meeting stand-in in the below sections.

3.2 Leverage the benefits of Chatbot as Meeting Stand-in

3.2.1 Absence Management Stand-In:

If a participant is unable to attend a meeting due to scheduling conflicts or other reasons, the meeting stand-in chatbot can step in and represent the person. It can provide pre-determined responses to common questions or address specific topics on behalf of the absent participant. This ensures that important information is still shared and discussed, even in their absence. If the organizer wants further updates, notifications can be sent to the concerned team member so that he responds accordingly. This helps to save time for both the organizer and the participant.

3.2.2 Knowledge Base Stand-In

Meeting stand-in chatbot can be integrated with an organization's knowledge base or documentation system. When a question is asked during a meeting, the meeting stand-in chatbot can search the

knowledge base for relevant information and provide accurate answers. This eliminates the need for participants to interrupt the meeting to search for information themselves.

3.2.3 Expert Assistance – Stand-In

In meetings where subject matter experts are required but unavailable, meeting stand-in chatbot act as an expert. It can possess a deep understanding of the topic at hand and provide detailed and accurate responses to questions or concerns raised during the meeting.

Additionally meeting stand-in chatbot can provide valuable assistance to enhance the overall meeting experience. It can offer features such as automated meeting scheduling and real-time information retrieval. These chatbots can integrate seamlessly with popular messaging platforms, allowing participants to interact with them effortlessly.

4 Meeting Stand-In Chatbot Design Flow

The meeting stand-in chatbot's design flow as specified in Figure 4 is accurately crafted to seamlessly integrate within the meeting ecosystem while providing a user-friendly experience. The journey begins with the meeting stand-in chatbot's integration with calendar applications to ensure automatic attendance and participation. Once in the meeting, the meeting stand-in chatbot analyzes spoken content and context, enabling it to offer insightful contributions and responses. Users can also interact with the meeting stand-in chatbot in real time, either through voice or text inputs, allowing for dynamic engagement. To maintain authenticity, the meeting stand-in chatbot employs natural language processing to emulate human-like conversations. This comprehensive design flow not only empowers the meeting stand-in chatbot to fulfill its stand-in role effectively but also elevates meeting efficiency and collaboration through its intuitive and adaptable interface.

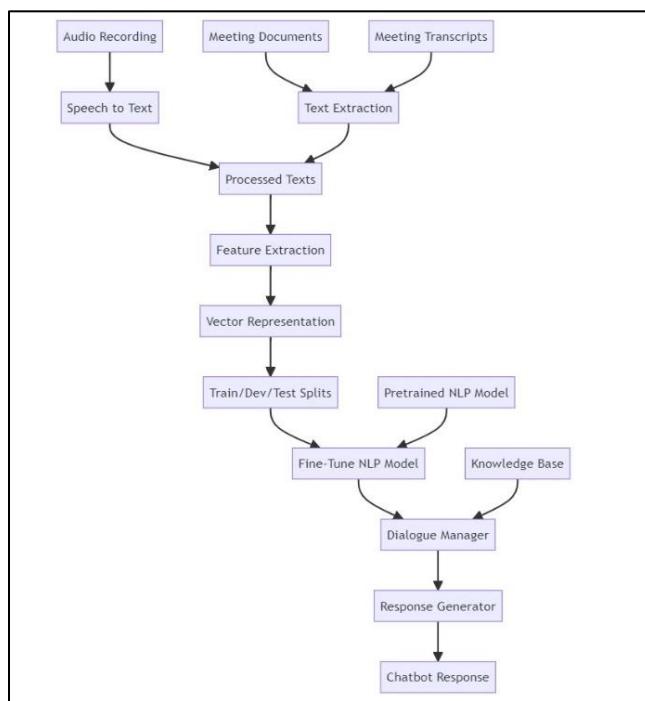


Figure 4: Meeting stand-in chatbot design flow

The meeting stand-in chatbot utilizes a standard natural language processing (NLP) architecture consisting of the following components:

Data Collection: Meeting transcripts, documents, presentations, and other relevant data needs to be collected to train the chatbot. Audio recordings may be transcribed to text using speech recognition.

Preprocessing: The meeting training data is cleaned and normalized. This includes speech to text conversion, text extraction, spelling correction, removing punctuation, converting to lowercase, etc.

Feature extraction: This includes the process of converting data into a set of numerical features or representations that can be used as input for machine learning models or algorithms.

Model Training: It refers to the process of fine-tuning a pretrained NLP model such as the DistilBERT model developed by Hugging Face or LLaMA (Large Language Model Meta AI) to recognize patterns, make predictions, or perform specific tasks based on the model's specific data set.

Knowledge Base: A database of key information relevant to meetings such as participants, agenda, past minutes, and documents. This provides facts/data for the chatbot.

Dialogue Manager: This component handles the chatbot's conversational capabilities and flow. It interprets user questions, retrieves information from the knowledge base, and determines responses.

Response Generator: Takes the output from the dialogue manager and converts it into natural language responses.

Human Expert Involvement: The chatbot is designed to ask for clarification when needed or route complex questions to a human expert. This ensures that users receive accurate and reliable information even in intricate scenarios. Meeting data needs to be anonymized wherever possible to address privacy concerns.

Testing and Reliability: Extensive testing needs to be conducted to ensure the chatbot's performance is dependable and secure.

Example dialogue:

Participant: "Can you give me an update on the project's timeline?"

Chatbot: "Sure! As of the latest update, the project timeline has been extended by two weeks due to unexpected delays in the development phase. We are now targeting completion by the end of next month."

Participant: "Can you share the presentation from the last meeting?"

Chatbot: "Of course! I've just sent you the link to download the presentation slides from the last meeting."

4.1 Meeting Stand-in Chatbot - Daily Stand-up Sequence Flow Diagram

The sequence flow diagram in Figure 5 depicts the steps involved in a daily standup meeting using a meeting stand-in chatbot. The user receives a meeting invite for the daily standup and clicks on the custom outlook add-in. The meeting stand-in chatbot asks for permission to join the meeting on behalf of the user. If the user allows the meeting stand-in chatbot to join, in preparedness for the meeting, it asks the user standard questions for the daily standup. The user can answer the questions and add additional information to enable the chatbot to participate in the meeting and represent the user. The chatbot then joins the meeting programmatically and answers the questions on behalf of the user. If the meeting stand-in chatbot is unable to answer a question, it sends an email to the user with the question.

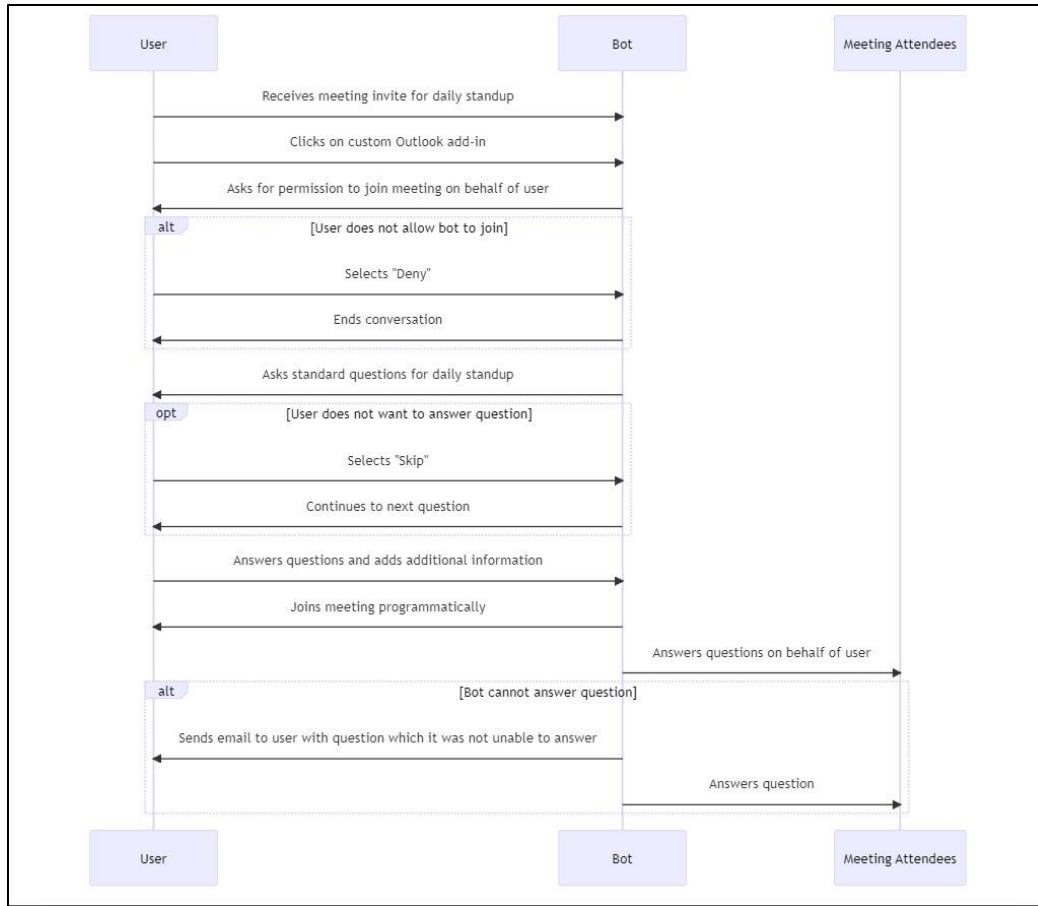


Figure 5: Meeting stand-in chatbot daily stand-up meeting

The sequence flow diagram includes the following key points:

- The meeting stand-in chatbot automates the process of collecting updates from team members, tracking progress, and sharing information with the team.
- The meeting stand-in chatbot provides a central repository for all standup updates, which improves accountability and transparency.
- The meeting stand-in chatbot can answer questions on behalf of the user, which frees up the user's time to focus on their work.

5 Meeting Stand-In Chatbot - Return on Investment (ROI)

Calculating the ROI for implementing a chatbot strategy involves a comprehensive evaluation of both building and buying options. For the 'build' approach, the costs associated with development, design, testing, and ongoing maintenance needs to be considered. These costs are balanced against the potential benefits such as increased efficiency, improved customer service, and reduced operational expenses. Similarly, for the 'buy' approach, the initial investment in purchasing a pre-built chatbot solution needs to be weighed against the anticipated gains. Comparing the two strategies involves assessing the projected ROI for each. By quantifying both the costs and benefits for building and buying a chatbot as given in Table 2, businesses can make an informed decision based on the potential ROI and align it with their strategic goals and resources. The ROI can be calculated using the formula $ROI = (Current\ Value\ of\ Investment - Cost\ of\ Investment) / Cost\ of\ Investment$.

Factor	Build Chatbot	Buy Chatbot
Price	Can be expensive - Refer Note 1	More affordable - Refer Table 3
Time to develop	Can take longer	Faster
Control	More control	Less control
Adaptability	Offers higher adaptability	Provides limited adaptability
Protection	Can provide more protection	May provide less protection
Scalability	Possibly more scalable	May have limited scalability
Input Parameters	Context size needs to be decided before the development begins and switching would involve design change	Switching between models with different context sizes is relatively straightforward
Usable Service Life	Investment in fine-tuning opensource base models, which can demand thousands of hours of computation on specialized GPUs	Utilizing 3rd party APIs, like the OpenAI API, hinges on API availability and the provider's commitment to updates.

Table 2: Meeting stand-in chatbot build and buy options.

Note 1: Cloud GPU providers like Lambda Labs offer options such as the A100 (40 GB PCIe) at \$1.1 per hour and the 1x H100 at \$1.99 per hour.

5.1 Buy Chatbot

In analyzing the total cost for purchasing a chatbot, it's essential to consider the procurement of 3rd party API (Application Programmatic Interface) access and its integration into the company's platform for inference. The cost is contingent upon performance and context size. Context size refers to the input scale usable for the model. Notably, pricing structures often revolve around tokens in input and output data. You can think of tokens as pieces of words, where 1,000 tokens are about 750 words. Below in Table 3 is a comparative pricing table for some of the most popular API available in the market:

Model Name	Context Size (tokens)	Input	Output
GPT 4	8K	\$0.03 / 1K tokens	\$0.06 / 1K tokens
GPT 4	32K	\$0.06 / 1K tokens	\$0.12 / 1K tokens
GPT-3.5 Turbo	4K	\$0.0015 / 1K tokens	\$0.002 / 1K tokens
GPT-3.5 Turbo	16K	\$0.003 / 1K tokens	\$0.004 / 1K tokens

Table 3: Meetings stand-in chatbot – Buy API Cost

5.2 Challenges Affecting ROI for Meeting Stand-In Chatbots

Error Risk - In-house chatbots may introduce transcription errors due to language complexities, undermining accuracy, and ROI.

Limited Collaboration - Self-built chatbots can't actively engage, hampering collaborative meeting recording and accuracy.

Contextual Limits - In-house chatbots might miss discussion context, leading to incomplete summaries and reduced value.

Here are some additional factors to consider when making the build vs. buy chat decision:

- The size and complexity of the business.
- The budget available for the chatbot project.
- The technical expertise of the in-house team.
- The urgency of the need for a chatbot.
- The long-term goals for the chatbot.

6 Conclusion

In conclusion, this paper demonstrates the feasibility of using chatbots to stand in for team members at meetings, potentially providing numerous benefits for remote workers and organizations. While the idea of using technology to represent individuals in meetings is innovative and could potentially streamline communication, it's crucial to recognize its applicability.

Nevertheless, in specific scenarios where the focus is on information dissemination or status updates, a meeting stand-in chatbot can prove beneficial. By allowing the chatbot to handle routine status-related meetings, employees could experience reduced stress and a better work-life balance. This approach can free employees from repetitive meetings, enabling them to allocate more time to tasks that require their unique skills and expertise. This, in turn, could contribute to higher job satisfaction and increased productivity.

The meeting stand-in chatbot is a revolutionary solution poised to transform the way businesses conduct meetings. This innovative technology harnesses the power of artificial intelligence to seamlessly simulate a human presence, enabling the chatbot to engage in meetings, presentations, and discussions on behalf of team members. By mimicking natural language interactions, the chatbot facilitates real-time communication, offers insights, and responds to inquiries just as an actual participant would. Its potential to reduce scheduling conflicts, minimize participant overload, and optimize time allocation underscores its significance in the modern corporate landscape. As organizations continue to embrace digital transformation, this chatbot stands as a testament to the limitless possibilities AI brings to redefining how we engage in professional discourse.

References

- Agrawal, Nidhi. "Here's How You Measure the ROI from Chatbots." *Mantra Labs*, 23 Sept. 2019, www.mantralabsglobal.com/measure-chatbot-roi/. Accessed 18 Aug. 2023.
- Albayrak, Naz, et al. "An Overview of Artificial Intelligence Based Chatbots and an Example Chatbot Application." *2018 26th Signal Processing and Communications Applications Conference (SIU)*, May 2018, ieeexplore.ieee.org/document/8404430, <https://doi.org/10.1109/siu.2018.8404430>.
- Deeb, George. "Too Many Meetings Suffocate Productivity and Morale." *Entrepreneur*, 8 Dec. 2017, www.entrepreneur.com/startng-a-business/too-many-meetings-suffocate-productivity-and-morale/304610. Accessed 18 Aug. 2023.
- Greyling, Cobus. "General Chatbot Architecture, Design & Development Overview." *Medium*, 22 June 2020, cobsgreyling.medium.com/general-chatbot-architecture-design-development-overview-58e145398608.
- Hristidis, Vagelis. "Chatbot Technologies and Challenges." *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*, Sept. 2018, <https://doi.org/10.1109/ai4i.2018.8665692>.
- Laker, Ben, et al. "Dear Manager, You're Holding Too Many Meetings." *Harvard Business Review*, 9 Mar. 2022, hbr.org/2022/03/dear-manager-youre-holding-too-many-meetings.
- Meshram, Siddhant, et al. "Conversational AI: Chatbots." *2021 International Conference on Intelligent Technologies (CONIT)*, 25 June 2021, <https://doi.org/10.1109/conit51480.2021.9498508>.
- Perlow, Leslie A, et al. "Stop the Meeting Madness." *Harvard Business Review*, 26 June 2017, hbr.org/2017/07/stop-the-meeting-madness.
- Wijaya, Rico Arisandy, et al. "Knowledge Based Chatbot with Context Recognition." *IEEE Xplore*, 1 Sept. 2019, ieeexplore.ieee.org/abstract/document/8901559. Accessed 18 Aug. 2023.

Increase Organizational Learning and Business Agility with Strategic Goal Setting and Innovation in your Testing Program

Kristine O'Connor
Kristine.OConnor@agilerising.com

Abstract

Organizations function like living organisms, dynamic and interconnected, with various components that must work together to thrive and grow. Each function plays a role in its overall longevity, requiring adaptability and responsiveness to change; organizations must exhibit the capacity to learn from experiences and processes and innovate. Much like the living organisms of a central nervous system, which evolves with evolution and time, the customers who use your products and the market in which your products deliver value are ever-changing and dynamic. As functional experts, adopting agile practices that support learning and growing business is a matter of when rather than why.

This paper targets testing practitioners or those in a senior testing role looking to influence, drive, or create a learning culture within your testing program. We'll start by discussing why a learning culture is so influential and how the five key disciplines of a learning organization can help align and strive toward business agility and shared outcomes. Next, we explore the opportunities of becoming partners with your leadership, finding your role as both guide and servant leader, and introducing them to the journey of continuous improvement and innovation in your testing program. Then, we do a deep dive into the five key disciplines of a learning organization and identify the critical ways in which these can be used to increase organizational knowledge and inform organizational intelligence.

Finally, I'll share three next steps you can take to implement these critical competencies in your testing program. By empowering your team members to master the functional craft of testing and establishing a differentiating quality program, you can intentionally create a learning culture, aligning quality with business outcomes to dominate the competition and delight your customers.

Biography

Kristine O'Connor spent several years in the US Navy before entering the software industry in 1998. Starting her journey at the help desk for a hotel software company, she quickly moved into software testing, finding a home for her process-driven thought process, a customer-centric mindset, and knack for tools. For over 18 years, Kristine has filled various software quality roles, from manual software testing to a Sr. QA Project Manager and Agile and DevOps practitioner in a Fortune 4 organization. In 2018, she dipped her toes in the coaching water and transitioned into the role of a certified Agile Coach, ultimately achieving her Scaled Agile Practice Consultant (SPC). Kristine has supported organizations Aetna, CVS Health, Bank of America, Northrop Grumman, NASA, and more.

Kristine is a Sr. Agile Transformation Consultant with Agile Rising, where she works as an Agile Coach, Agility Health facilitator, and SPC, helping people, teams, programs, and organizations adopt Agile and Scaled Agile practices and processes. She is also currently enrolled at Arizona State University, working on her Master of Organizational Leadership. Her passion is enabling a learning organization in a mindful and intentional way.

1. Introduction

Any organization seeking to differentiate itself from its competition to achieve long-term sustainability must engage in the constant pursuit of higher performance. Each business and functional area of your organization needs to continuously explore and be curious, seeking to understand and improve how they plan and do their work, approach value, and embrace strategic goal setting. Only then can the independent components of your organization begin to grow together and thrive, moving the larger organization forward as a single organism, running towards long-term survival. When it comes to your testing program, intentionally creating opportunities to refine our practices and identifying ways to foster a culture of continuous learning allows you to build on your organizational knowledge and inform organizational intelligence.

Let's start by creating a shared understanding of a learning culture by defining it as one that engages in "generative learning, learning that enhances the ability to create" (Oosterwal 2010, 47). A learning culture is one where its processes and behaviors mindfully target the five key disciplines of a learning organization, which is supported by the dimensions of organizational intelligence and organizational knowledge. By exploring the competencies and disciplines needed to create a learning culture, we identify the benefits of maintaining a constant state of urgency and curiosity, implementing feedback loops to learn and grow, and allowing for purposeful and intentional risk-taking. Our goal is to be intentional in the foundations we put in place. Hence, we are more successful in achieving the specific business outcomes we believe the improvement items we work on for our testing program will achieve. This propels your testing program and your functional testing team to proactive and strategic components of the larger organizational system, intentionally driving your organization toward greater levels of business agility.

What is business agility? We can define this as "the ability to compete and thrive in the digital age by quickly responding to market changes and emerging opportunities with innovative, digitally enabled business solutions" (Knaster, 2023). To still answer what business agility is, we must ask ourselves what it means for our organization to compete. What are the specific things you can do to make your test program stand out as a trusted partner, not just by validating changes to your software products but by understanding the needs and outcomes that your customers desire? How can your testing program become a proactive change agent within your organization to reduce time to market, delight customers, and enable the autonomous mastery of your functional testers? Business agility requires a learning culture; by focusing on the practices and disciplines that we'll talk more about, you can transform your testing team into agile learners, and "agile learners move quickly, adjust skillfully, and drive innovation.... organizations that get this right are 46% more likely to be first to market, 37% to be more productive, and 92% to be more likely to innovate" (Businesses by edX, 2023).

If business agility requires a learning culture, let's explore the different disciplines and dimensions which a learning organization and learning culture support.

1.1 Creating a Learning Culture

Margaret Wheatley, an organizational theorist, identified the ability to generate new knowledge and for it to be freely shared as two organizational capabilities required for long-term survival (Wheatley 2006, 92). If we say this a little differently, the ability to learn, reflect, create shared understandings, and grow our knowledge can be used to inform intelligence when we share this information and knowledge freely. A learning culture is one where there is an intentional focus placed on exploring opportunities to influence the five key disciplines of a learning organization, which include systems thinking, personal mastery, mental models, building a shared vision, and collective team learning. Why is a learning organization important? A learning organization supports the two key disciplines that Margaret Wheatley said are required for long-term sustainability: organizational knowledge and organizational intelligence, so understanding how to transform your testing program into a learning organization, supports the greater organizational learning culture.

Let's look at the first dimension of a learning organization: organizational knowledge. Let's create a shared definition of this as the accumulated knowledge of "principles and maxims of practice, images of mission and identity, facts about the task environment, techniques of operation, stories of past experience which serve as exemplars for future inquiry" (Schön, 1983, 242). This is the collective wisdom, insights, and expertise of an organization, accumulated over time, and includes lessons learned from past testing endeavors, best practices adopted, and the embedded knowledge within our team on why and how we approach testing the way we do, and tactically how it's done. We use this knowledge as our guide forward, helping us make better decisions, avoiding the repeating of mistakes, and finding efficiencies and improvements to the flow of value through our software testing processes.

The second dimension of a learning organization is organizational intelligence, which is the capacity to gather, analyze, and interpret data and information related to the software testing activities of your program. It challenges you to use data-driven insights to identify patterns, detect potential risks, and strategically optimize your testing practices for enhanced outcomes and impact, which allow you to better deliver on business outcomes. Organizational intelligence is "the mental capacity and ability to perform a task or important activities" (Bahrami et al. 2016, 191) and is made up of seven key components: strategic vision, shared fate, desire to change heart, alignment, knowledge application, and performance pressure. By harnessing the power of organizational intelligence by addressing these components, you can proactively identify emerging trends, make strategic decisions, and align your testing strategy and efforts with broader organizational objectives. This supports an environment for the analysis and understanding of practices, behaviors, and beliefs so you and your functional testers can align and address any differences or deviations from the shared understanding so that, collectively, you can learn and grow. We'll talk more about ways to do this throughout the paper.

As you build your organizational knowledge and grow your understanding of the various practices and behaviors of your testing program, you can leverage these insights to envision possible new ways of working and develop a vision for what that might look like, along with identifying the business outcomes, or shared fate, that you seek to impact. Sharing this vision and outcomes creates an urgency for change amongst your functional testers and is a necessary input for your functional testers to see how they fit into the more significant value delivery picture. The desire to change, heart, alignment and knowledge application, and performance pressure are internal motivators activated as you provide your functional testing practitioners with the autonomy to continue to master their functional expertise.

Before you begin on a journey of improvement, let us talk about the critical support and engagement needed from your leadership and ways to strategically align for impact.

1.2 Leadership Engagement and Commitment

Leadership engagement and commitment are vital for creating an environment that provides the space for creativity, collaboration, and empowerment. Empowered team members strive for innovation, propose strategic goals aligned with the organizational vision, and contribute significantly to the success and growth of the organization. Leading the change for transformation requires a comprehensive approach involving data, analyzing behaviors, and exploring practices, but everything must begin with an alignment with leadership. Change begins with a clear vision of what can be, instilling a sense of urgency around working differently and focusing on sustainable, incremental improvements.

This sense of urgency and communicating the vision for change is the responsibility of leaders, becoming a catalyst for establishing a culture that values creativity and experimentation. A culture that fosters innovation is essential for the success of any testing program, and looking for ways to actively promote and reinforce this allows for the generation of new ideas to solve problems and the open sharing of knowledge and feedback. Supporting the growth of your testing team members with

skill development, training, and embracing agile practices for continuous learning and improvement to grow, learn, experiment, and try. Daniel Pink shared in a TED talk that people are motivated to high performance because they feel like they are a part of something meaningful, and this requires autonomy, or the urge to direct their own lives, and mastery, which is the desire to get better and better at something that matters, and purpose, the yearning to do what we do in service of something larger than ourselves. People do their best work when given the autonomy to do so and master how they do their work (Pink, 2009).

In my experience working in the software quality industry, there have been only a handful of times where my direct leader had a background in software quality or testing; I made the mistake of assuming my leaders had a working understanding and knowledge of the processes which their team members followed, tools and practices which we employed, and challenges we encountered; this assumption caused us to all have our own mental models on how our testing practices and processes were functioning for us. Today, given a similar situation, I see it as my responsibility to inform and uplift the knowledge of my leadership, helping them understand where the pain points and opportunities are to. As that functional expert, I missed taking my stakeholders along the journey to understand how the testing program could work in the future and how the outcomes and outputs produced now align with business goals and outcomes. When the testing team identifies improvement opportunities, marries those to business outcomes, and defines a strategy that aligns the ideas to the vision and business outcomes, you intentionally propel your testing program forward toward those goals. Business agility requires this of us, and your leadership has a responsibility to provide the capacity to learn and grow as a team of functional SMEs. Sharing this knowledge with your leadership team is your responsibility; taking them along this improvement journey with your testing team members is critical in them seeing the value, communicating that effectively, and providing necessary support to help remove impediments and guidance on the next steps.

With a knowledge of business agility and learning culture, recognition of leadership support, and engagement, let's now do a deeper dive into each of the key disciplines of a learning organization.

2. Systems Thinking

Systems thinking is a holistic approach to understanding how different components and processes within a system interact and influence one another; remember, an organization functions much like an organism: dynamic and interconnected, consisting of various components that must work together to thrive and grow. If an organization is dynamic and interconnected, systems thinking enables an understanding of the larger software development ecosystem in which we operate, as well as how we operate within the more specialized and siloed functional area of software testing. Systems thinking is crucial for comprehensive understanding, enabling you to understand how testing activities interact with development or operation activities, and this learned intelligence is further informed by knowledge from user feedback and deployment activities. This comprehensive understanding helps us identify potential bottlenecks, dependencies, and areas of improvement that may be overlooked when we take a more narrow, siloed focus and approach to our thinking. In software testing, root cause analysis is a long-lived practice; issues and defects often have underlying causes that may originate from various parts of the development process or precede the work before it funnels into the product backlog. Systems thinking allows you to trace problems back to their source, helping you address root causes rather than just addressing symptoms; applying lessons learned systematically drastically reduces the recurrence of similar issues is a critical application of systems thinking.

Much like organisms and organizations, software systems are inherently complex, and as testing professionals, you recognize that understanding this complexity is essential for effective testing. When we practice systems thinking, we manage this software complexity by breaking down our products into interconnected parts and developing test strategies to tackle intricate problems. It also helps you identify feedback loops within the testing processes, allowing your testing program to pivot and adjust your testing strategies and approach due to customer feedback or business needs. Remember, we are seeking to implement a holistic approach to understanding and viewing the

system in which your testing program operates, how it relates to the larger software development ecosystem, and the levers to pull to deliver on business outcomes.

Systems thinking is what provides us with a framework for practicing root cause analysis, allowing us to better manage complexity, see options, optimize our processes, foster adaptability, and nurture a culture of continuous learning. It enables teams to anticipate and respond to changes in the market, whether it's new ways in which your customers use your products, new technologies, tools to support testing activities, or identifying new market trends. This adaptability is essential for staying competitive and relevant, but for testing programs getting started on their transformation journey, the pathway forward can be murky. If our goal with this discipline is to see options, optimize processes, and foster adaptability, an important tool to put in your continuous learning toolbelt is a journey map.

A journey map helps us visualize the different scenarios your functional testers engage in as they execute scenarios, such as developing automated test cases or testing mobile applications. A journey map should include (Gibbons, 2018):

Actor	This is the point of view of the persona or user which the journey map is about; the person or role who experiences the journey.
Scenario + Expectations	The scenario is the situation the map addresses, associated to an actors goal/need/specific expectation. Scenarios can be real or anticipated. Journey maps are best for scenarios that involve a sequence of events, or describe a process.
Journey Phases	The different high-level phases of the journey, and help organize and support the actions, thoughts, and emotions throughout the map. For example, with a testing a new Feature, the stages <i>might</i> be: Align (on the ask for the new Feature) Create (testing assets) Execute (test assets) Inform (update test assets, team) Reflect (feedback loops)
Actions, Mindsets, and Emotions	These are the behaviors and thoughts of the Actor throughout the journey phases on the map. Actions are the actual behaviors and steps taken by users written in narrative form. Mindsets are the thoughts, questions, motivations, and information need at each journey phase; information is gained by research. Emotions are plotted as single line across the journey phases, literally signaling the emotional "ups" and "downs" of the experience. Think of this line as a contextual layer of emotion that tells us where the user is delighted versus frustrated.

Creating a shared understanding of all the steps that your testing team members encounter while approaching and doing their work can help surface pain points and highlight opportunities for improvement. Using journey maps is one way to be aware and mindful of the entire system in which the process occurs. A learning organization thrives on continuous learning and improvement, and systems thinking fosters a culture of curiosity and inquiry, encouraging us to explore how different components interact and how understanding these interactions leads to better outcomes, and become proactive in seeking opportunities for improving and innovation. Through continuous learning, adaptability, and optimization of processes, you will begin shifting the mindset of your testing program.

3. Mental Models

The second discipline of a learning organization is mental models, and these are the assumptions, generalizations, and deeply held beliefs that influence how we understand the world around us (2010, p.53). It is the cognitive framework and belief that we use to make sense of the world and solve problems; they are essential for enhancing decision-making in the sense that they shape how people and teams perceive situations and make decisions. Our mental models have protective barriers, requiring us to share how and why we think the way we do (2010, p.54). In a learning organization where informed decision-making and adaptability are crucial, diverse mental models encourage a broader range of perspectives and lead to more well-rounded, effective solutions. Existing mental models can either stifle or foster innovation and the encouragement of diverse and flexible mental models can stimulate creative problem-solving and innovative thinking. People are more likely to generate novel solutions when they consider a wide range of viewpoints; it deepens our understanding and creates empathy for what we do not know.

A learning organization thrives on continuous learning and improvement, and our mental models influence how we approach learning. Openness to new ideas, willingness to challenge existing beliefs, and a growth mindset are all influenced by one's mental models, and a culture that values diverse mental models promotes a stronger culture of learning. Business agility is closely tied to adaptability; we've already talked about learning from feedback, intentional learning, and data-driven decisions. If we learn we are not making the right decisions, we need to be adaptable, pivot, and shift our strategy. This applies to both work on our products and services and improvements we implement in our testing program. As functional testers, it is important to challenge our mental models about the effectiveness of our testing practices and behaviors and allow ourselves to inform our knowledge on what we don't know and grow our intelligence.

You may find resistance to change as people and teams become entrenched in their existing mental models or resistance when presented with new ideas or approaches, things which otherwise would increase the ability of your testing program to adapt and learn. You may also find that there is a cognitive bias that may influence decision-making; recognizing and mitigating these biases' is essential for sound decision-making in a learning organization. It's important to keep in mind that different mental models within your testing program can lead to misunderstandings and communication breakdowns. It's crucial to establish effective communication processes that facilitate the sharing and reconciliation of diverse mental models.

One recommendation is for your functional testers to participate in a facilitated agile team self-assessment, which challenges the existing understanding of how each person on your testing team experiences the culture, processes, and challenges in your testing program. The real value of the team self-assessment is not just in answering the various questions being asked (these will vary based on the self-assessment selected) but in the discussions, the team has after each person has completed the self-assessment. When analyzing the results, there will be practices and processes that some team members do not find value in or are incredibly challenging. You'll also find things that your testing team is doing well, and you'll want to make sure you keep doing those things. After discussing the results of the assessment, the team then brainstorms on possible solutions for the identified challenges and opportunities for improvement, aligning these ideas with business outcomes. These ideas all form the strategy for achieving your vision for change, specifically aligned with driving business outcomes! This self-assessment supports each of the key disciplines of a learning organization and is an excellent way to generate urgency for personal mastery, the next discipline.

4. Personal Mastery

That takes us to our next key discipline of personal mastery, which is when we continuously learn and improve our skills, focus on our own growth, and align our personal goals with the objectives of our testing program and overarching organization. We all know that the quality landscape is constantly evolving with new tools and methodologies, and your testing program needs to stay up to date with the latest industry trends and continuously improve its skills to stay informed. Staying competent and adapting to changing requirements swiftly requires an adaptive mindset, more open to learning from experiences, both successes and failures. This adaptability is crucial for responding to shifting project priorities, emerging issues, or sudden shifts in market demand and a requirement of personal mastery.

Improving our skills, focusing on our personal growth, and finding alignment requires internal reflection and developing “deeper insights into personal beliefs, purpose, and vision. Personal mastery is achieved through a set of repeatable strategies and tools that equip individuals to perform at their best.... It’s also about the way in which an organization supports such opportunities and helps team members link their efforts to the broader mission of the organization” (Businesses by edX, 2023). The goal of this paper is to help you, the team members of your testing program, and leadership see the benefit of leveraging your testing program as a strategic lever to increase business outcomes. We just talked about the benefits of a facilitated team self-assessment. Now, imagine pairing the strategic goals identified in the team self-assessment with Agile ceremonies that make sense for your functional experts – those are tools and repeatable strategies worth working with your leadership to create the capacity and urgency to implement.

Being a tester with an agile and adaptable mindset requires quick and effective problem-solving skills, and personal mastery equips us with those, including critical thinking, creativity, and the ability to think “outside the box.” These skills are invaluable for identifying and addressing issues that can hinder business agility. Personal mastery also includes leadership development and interpersonal skills. Effective leadership and collaboration within your testing program are essential for fostering a culture of agility and learning, so pairing agile events for your functional experts would help build and support the building of these skills. Strong leadership ensures that decisions are made swiftly, and collaboration enhances teamwork, promoting agility in responding to feedback for strategic improvements. It also encourages a commitment to continuous improvement as team members who are dedicated to their own growth are more likely to actively seek opportunities for process improvement; this mindset aligns with iterative and incremental continuous improvement delivery.

Finding time for personal mastery development while balancing the need for improving skills together with commitments to delivery can be challenging for any team. Some team members may be resistant to focusing on their own personal mastery, but it is a valuable discipline to embrace. Investing in personal mastery equips us with the necessary skills, mindset, and alignment to enhance business agility. It promotes continuous improvement, adaptive thinking, and the development of problem-solving skills. Collaborating with your leadership to provide the capacity for focusing on personal mastery is a critical component of success.

5. Building a Shared Vision

Earlier, I’d mentioned that each person and role must adapt and respond to the ever-changing environment of our organization; your functional testers must exhibit the capacity to learn from experiences through applying practices and processes and innovate. Working together with the leaders of your testing program, you are tasked with developing a compelling vision of what future way of working together as a group of functional subject matter experts, sharing the role each team member has in the journey and what success looks like for both the team and organization. Building a shared vision is a key discipline of a learning organization. While the goal is to align the vision and outcomes with the outcomes and vision of your larger organization, do not allow a lack of larger

organizational goals or published business outcomes to deter you from getting started on creating yours.

Creating a vision means each person is aware of the role they play to get there and the guardrails that are in place to keep us on track; this is necessary for building a shared understanding of how we work as individuals and within a larger functional group of a testing program. Organizational intelligence has components focused on strategic visions and shared fates; while inputs come from many different sources, these are ultimately informed by the knowledge that we've newly developed and is the responsibility of the functional leadership to present a pathway and vision forward. The strategic vision of your testing program makes visible how we intend to build greater levels of quality into our practices and processes, reduce the amount of time it takes us to move work through testing, and the outcomes which we will each benefit from once success is achieved. It makes transparent our reasons for improving specific areas of our quality program, aligning everyone on the tactical activities required to achieve business agility. A well-crafted vision has the power to be inspirational and motivating, encouraging experimentation and innovation, encouraging thoughtful risk-taking, and creating that longer-term view required for systems thinking (Oosterwal,2010, pp. 55-56).

The clarity a vision provides should be an inspiring call to action for your team, challenging us to develop new ideas, think about what future ways of working might look like, and critically think about the business outcomes these ideas support. Empowering your team members to become empowered functional owners means hearing all voices of your testing organization. This diversity of input develops into a shared vision that excites everyone to a common mission, giving leadership a view forward in identifying ways to provide support and assist in mitigating any potential risks.

As you are putting together your strategy for improvement with your functional testers, identifying business goals that the work helps to advance is a way to align leadership on why prioritizing this work is important, and the outcomes which they can expect to see as a result of being completed.

Here are some possible examples:

Problem Statement	Business Goal	Objective	Outcomes
<p>Testing operates as a silo, so when code changes are ready to be tested, we usually are just learning about the work.</p> <p>These changes sit waiting to be tested while we learn about the new feature/change and create our test scripts.</p> <p>We also have other priorities to focus on.</p>	<p>Reduce Time to Market</p> <p>Delight Customers</p>	<p>Reduce the amount of time it takes for work to flow through QA.</p>	<ol style="list-style-type: none"> Parallel team activities occur on work the development team is working on, so that testing assets are created prior to the work being ready for test. Engage leadership for assistance in creating shared priorities for product delivery work.
<p>New changes or enhancements have unintended impacts on the product behavior in production for our customers.</p> <p>This behavior is having a negative impact on our</p>	<p>Delight Customers</p> <p>Build-In Quality</p>	<p>Better understand change and intended outcomes; test in an integrated test environment</p>	<ol style="list-style-type: none"> Engage leadership / Agile Coach / Product Owner for adding scope in terms of which persona or customer does this change impact, clear acceptance criteria and behavior driven acceptance tests.

clients, and in some cases rendered critical features inoperable.			2. Work with leadership to define a strategy for an integrated production-like test environment.
---	--	--	--

6. Collective Team Learning

A vision must constantly be updated based on new information learned; our last discipline is collective team learning, where the primary goal is to continuously improve and adapt through shared insights, feedback, and collaboration. One of the foundations of learning is to have a shared meaning, accomplished through dialogue or “the free-flowing movement of ideas,” and requires “the capacity of team members to suspend assumptions and judgment to genuinely “think together””(Oosterwal, 2010, p. 57). Collective team learning challenges us to apply systems thinking, explore existing mental models, and suspend our own personal beliefs and assumptions to listen with an open curious mind.

This discipline is critical for enabling competencies of building organizational knowledge, promoting knowledge sharing, collaborative problem solving, adaptive learning, data analysis, innovation, Agile decision-making, cross-functional collaboration, continuous improvement, risk management, and long-term strategic thinking. Collectively, these contribute to an organization’s ability to achieve sustainability and maintain differentiation in the market. As your testing program begins sharing tacit and explicit knowledge, you’ll begin to observe a broader sharing of information, which will begin enhancing the collective knowledge base of your testing practitioners. As your functional experts learn together and exchange insights, valuable information is spread more effectively throughout the organization. Additionally, teams that are engaged in collective learning are better equipped to collaboratively address complex problems; the diverse perspectives and expertise brought to the table enable teams to approach challenges from multiple angles, leading to solutions that are more effective and comprehensive.

Organizational intelligence relies on the ability to be Agile, to adapt, and to learn from changing circumstances. Teams who learn together are more aware of shifts in the environment, market trends related to their functional area of expertise, and emerging opportunities, such as new automation tools or DevOps. Using this knowledge to become more adaptable and make informed adjustments to its strategies and processes, you are more likely to achieve your goal of continuous testing, reducing time to market, increasing built-in quality, and higher customer delight. Incrementally delivering valuable innovative improvements to your testing program and collectively learning and reflecting on those improvements allows you to measure whether you achieved the hypothesized business impact and guide your next decisions.

Measuring and analyzing the impacts of change will guide you in determining your next steps forward, but making sure you are measuring the right thing helps you know you are on target to achieve the outcomes your team hypothesized it would. Reviewing these data points over time allows you to identify trends and outliers, as already discussed, and it also allows you to question and uncover what may not be so readily visible in your data points. This alignment with business goals and commitment to continuous monitoring allows you to uncover opportunities to develop a deeper understanding of trends and patterns, allowing your testing team to make informed decisions, identify outliers, and uncover insights. This data analysis and collective team learning allows for open discussion, consideration of unknowns, and the ability to challenge assumptions. Make sure there is clear alignment and understanding of the value that the improvement is going to bring and how it should be measured as you are defining and planning these innovation items. The intent is to measure, learn from the measurements, and make informed next step decisions based on data.

Collective team learning fosters a culture of innovation; by sharing ideas, experimenting with new approaches, and learning from successes and failures, you cultivate an environment where innovation is embraced and nurtured. This deepened trust enables your testing team to make agile and well-informed decisions. As knowledge is accumulated through shared experiences, you’ll

observe a testing program better equipped to make choices that align with the organization's goals and objectives. This trust enhances functional team member collaboration and enhances organizational intelligence as it breaks down silos and promotes the exchange of expertise, all leading to long-term strategic thinking in your testing program and increasing business agility.

7. Conclusion

In the journey towards long-term sustainability, a learning culture is critical for differentiation and success; testing programs are not excluded from this. Being able to identify opportunities to improve, measure both the progress and impact of these improvements and share the measured business impact of these improvements is critical for a learning culture. Feedback from end users, stakeholders, and customers becomes the driving force for improvement, and by adopting iterative cycles of feedback, testing programs can evolve and refine strategies to support customer needs and market demands.

Being a change agent is not for the faint of heart; employing an experienced Agile Coach can help you on your journey. However, if you need to get started on your own, here are three tactical recommendations you can begin doing right now to create a culture of learning and curiosity in your testing program and propel your organization forward to greater levels of business agility:

1. Partner with leadership and get agreement to provide capacity for improving and learning and to help with roadblocks or risk mitigation. Share the compelling urgency for improving your testing program, and together, create a vision that allows each functional expert in your testing program to understand their role in strategic improvements and achieving business agility. This vision should be inspiring, focusing on future ways in which the team could be working, defining, and implementing future improvements together.
2. Challenge existing mental models and create shared understandings of how you work together and use this information to inform journey maps and identify improvement opportunities. Leveraging an Agile Coach or certified facilitator in facilitating a team self-assessment can help your team stay focused and engaged in the conversation and be confident in achieving the desired outcomes. This collective team learning allows your team to identify and prioritize strategic improvements based on this information, linking these to outcomes and ways to measure success.
3. Support your functional testers as they define, plan, and execute these strategic improvements, but make sure the team is planning and delivering on these improvements incrementally, sharing with the larger testing program team and allowing for feedback and input. This information is necessary as it is used to make any updates needed to the vision or roadmap of improvement as needed. Remember, using new knowledge to inform future decisions enhances our intelligence as an organization.

If you're just getting started on your strategic improvement journey, you may find you have a lot of technical debt or things you need to explore to understand more;

Getting started takes time, so use your biggest asset to help garner support – your functional testers! Develop an army of volunteers to help as you get started on identifying the why's behind the urgency for change, engage with leadership, and just get started. Implementing change is not for the weak; be brave and reach out for help.

References

Bahrami, Kiani, M. M., Montazeralfaraj, R., Zadeh, H. F., & Zadeh, M. M. *The Mediating Role of Organizational Learning in the Relationship of Organizational Intelligence and Organizational Agility*. Osong Public Health and Research Perspectives, 7(3), 190–196.
<https://doi.org/10.1016/j.phrp.2016.04.007>, 2016.

Gibbons, Sarah. "Journey Mapping 101." Nielsen Norman Group, December 9, 2018.
<https://www.nngroup.com/articles/journey-mapping-101/>.

"How a Learning Culture Primes Your Organization for Agility." Enterprise Training and eLearning Built for Businesses by edX. Accessed September 1, 2023. <https://business.edx.org/blog/how-a-learning-culture-primes-your-organization-for-agility>.

Knaster, Richard. "Business Agility." Scaled Agile Framework, July 13, 2023.
<https://scaledagileframework.com/business-agility>.

Oosterwal, Dantar P. *The Lean Machine: How Harley-Davidson drove top-line growth and profitability with revolutionary lean product development*. New York, NY: American Management Association, 2010.

Pink, Dan. "The Puzzle of Motivation | Dan Pink." YouTube, August 25, 2009.
<https://www.youtube.com/watch?v=rrkrvAUbU9Y>.

Schön Donald A. (1983). 8. In *The reflective practitioner: How professionals think in action*. Basic Books. P. 242

Sinek, Simon. "Leadership Has Nothing to Do with Rank." YouTube, August 4, 2020.
<https://www.youtube.com/watch?v=LDQj-sU9Opw>.

Wheatley, MJ. "The Creative Energy of the Universe - Information." Essay. In Leadership and the New Science: Discovering Order in a Chaotic World, 3rd Ed, 93–112. San Francisco, Calif: Berrett-Koehler, 2006.

Transformation path for Tests and CI/CD when moving applications from data center to public cloud

Amol Patil

apatil@mimecast.com

Abstract

As part of the modernization effort, more and more companies are making the journey from a co-located data center to a more convenient, on-demand, self-serviced and rapidly provisioned cloud environment for better ownership and control. This change of ownership has multiple consequences depending on the strategy used for the journey like; Lift and Shift, or a full rewrite of the code or a more evolutional path. Any approach would introduce a substantial number of differences from the way how the products and services function today. Engineering needs to make the applications auto-scalable, highly available, self-healed, and easy to upgrade and deploy. The solutions would include making a choice between IaaS/PaaS/SaaS services, making current applications & services cloud native and leveraging native cloud solutions for Monitoring and Observability. The most important change introduced by cloud-native applications is a fully automated code/test/deploy/monitor process. It is probably the most expensive and difficult element of the change and transformation. There will always be the nagging question 'Is the transformation to this new world better than what we had before?'. To achieve this transformation with high level of confidence, the automated testing needs to evolve to take advantage of the on-demand and self-serviced method of working.

Biography

Amol Patil is the Director of Quality Engineering & Services at Mimecast. He is responsible for ensuring world-wide quality standards for Mimecast products that use A.I. and Machine Learning models for threat detection and analysis of emails and attachments. Amol has experience in many markets; Cyber Security, Healthcare, IoT, PLM, Mobile and cloud platforms and has been delivering new products and platform integration's using any cloud-based DevOps infrastructure leveraging Docker, Terraform and Kubernetes. Amol has a M.S. degree in Industrial Engineering from the University of Cincinnati

1. INTRODUCTION

This paper will highlight some of the challenges associated with automated testing that arise from the shift from an on-prem application migrating to running on the cloud. To address these challenges a transformation path for Tests, CI and CD needs to be figured out. There are two aspects to the transformation.

- Test Framework Changes to handle
 - Secrets and certificates
 - Logging
 - Cost
- Tests Changes to handle the different types of test levels

This topic will describe recommendations for transforming automated testing using the local stack and public cloud. In this paper, we consider the following stack as the platform:

Amazon Web Services (AWS), Jenkins, Docker, Terraform.

Additionally, the assumption is that your application/service has gone through architecture review and is already cloud native with applications built with services packaged in containers, deployed as microservices, and managed on an elastic infrastructure through agile DevOps processes and continuous delivery workflows. The focus of this paper is entirely on the changes to consider during testing, when such an application has migrated from on-prem infrastructure to a public cloud platform.

2. TEST FRAMEWORK CHANGES

2.1 Secrets and certificates

Every service or application built today requires secrets to enable the right level of access. These could be certificates, tokens, keys, connection details, root accounts and passwords. This has to be dealt with through test automation framework changes (and not tests) to minimize or better, to eliminate the risk of a secret leak or secrets being exposed.

Points to consider are the Secrets Lifecycle stages that apply to any environment; on-premise or public cloud

- Creation
- Rotation
- Revocation
- Expiration

On-prem infrastructure requires physical infrastructure and storage housed in a DC. So internal network can be used to review and analyze. But, for cloud, the capture and store of secrets happens in cloud but transmitted over the internet. So, it is better to have a central management rather than spreading secrets into individual project account. This keeps the secrets related operations optimized and consistent across all regions.

Additionally, Secrets flow is different in a Public Cloud based application testing. The mechanism commonly used on-prem is to connect to retrieve tokens stored on disk. With the move to cloud secrets can be stored natively in AWS reducing the attack vector to just AWS ingress/egress points. One way to achieve this would be to abstract the secret provisioning function into its own module. This way for any test account or secrets used by the Test Engineer/QE can be created and accessed using the module.

On-Premise example of test code using secrets

Example: Details of File: *localproperties.json* – used for storing name/value pairs

```
{
  "credentials": {
    "signature_0": "eyJhbGciOiJFZERTQSlsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..v_Tni1_9IPQsS",
    "signature_1": "eyJhbGciOiJFUzI1NkslLCJiNjQiOmZhbHNILCJcml0lpblmI2NCJdfQ.."
  }
}
```

The above data gets used in a test code like this example below

```
...
AuthManager.getInstance().setLocalCredentials(new Credentials(localCredentials.get("signature_0").getBytes()));
...
```

Cloud example of test code using secrets

The following changes will need to be done

1. Remove secrets from localProperties
2. Create the secret in AWS Secrets Manager for an AWS native way of storing secure static K/V pairs
3. Create a Helper class with methods like
 - a. getSecretValue
 - b. getResourcePolicy
 - c. createSecret
 - d. deleteSecret
 - e. describeSecret
 - f. rotateSecret
4. Use the Helper methods in your test to retrieve/read secrets

The above example of the test code will change to be like the example below

```
...
AuthHelper authHelper = new AuthHelper(logger);
String signature = authHelper.getSecretValue(TEST_SIGNATURE,"1234567890").get("signature").toString();
AuthManager.getInstance().setLocalCredentials(new Credentials(signature.getBytes()));
...
```

In this example AuthHelper is the helper class leveraging AWS Secrets Manager API's with a `getSecretValue` method declared as

```
public static void getSecretValue(String secretName, String region)
```

Additional consideration for cases where integration tests need to access secrets directly from Jenkins/CircleCI, it will be necessary to provision tokens from within the build process. In an on-prem set up this would normally be expected to be available in a known or pre-determined location.

2.2 Logging

One of the key parts of a software development process is the ability to debug a failure. Failures happen all the time due any number of reasons. The test result logs can be invaluable for debugging. Effective value can be gained by analyzing the logs and making sense of any unusual pattern of events or finding root cause of an event. So, its important for tests to implement a proper logging mechanism.

2.2.1 Log Message Structure

Most programmatic automation tests will have the format comprising different building blocks.

```
@TestClass
<definition block>
<with condition block>
<execute block>
<expect block>
build
run
```

Mostly logging can be seen only in the expect block like this.

```
assertThat(new Object(), notNullValue());
LOG.info("Object successfully created")
```

In the world of micro services and clouds, there are a lot of integration points any one of which may fail at a certain time. It is beneficial to capture useful and actionable logging info at each block.

2.2.2 Log Message Levels

For test environments, it is better to use excessive verbose logging like info and debug to help troubleshooting failures. However, it is not recommended to use similar levels for production environments because these can generate excessive data.

With the switch from physical data center where engineers can ssh into resources freely for complex test failure investigations, when operating in AWS with heightened security protocols, it is beneficial to get as much information as possible through logging. So, consider logging summary attributes for each event like

- Resources affected
- Object
- Local device identifier
- Name of resource
- Resource tag

- Cluster / hostname
- Ipv6 address

In the world of executing programmatic tests for cloud-based application or services, the logging strategy should include deciding which events and actions to log.

Example: Abstract the logging enabler in a AWS_LogHelper with example methods like

- create_bucket
- retrieve_s3_logs
- list_VPCs
- r53_querylog

The new logging information using the Logging structure and Logging level would look like the example below

```

@TestClass
<definition block>
<with condition block>
    LOG.info("Turning on Route 53 Query Logging on for VPCs in region " + aws_region + ".")
    LOG.info("List of VPCs found within account " + account_number + ", region " + aws_region + " WITH Route 53 Query Logs:")
    print(r53_working_list)
<execute block>
    LOG.info("Creating bucket in %s" % account_number)
    LOG.info("List of VPCs found within account " + account_number + ", region " + aws_region + ".")
    print(VPCList)
<expect block>
    LOG.info("List of Buckets found within account " + account_number + ", region " + aws_region + ".")
    print(S3List)
    assertThat(new Object(), notNullValue());
    LOG.info("Object successfully created")
build
LOG.info("Route 53 Query Logging Created. Resource ID:" + r53_query_log_id)
run

```

Additionally, AWS resources and services being used might need network time synchronization to ensure accurate time stamps on transactions.

2.3 Cost Optimization

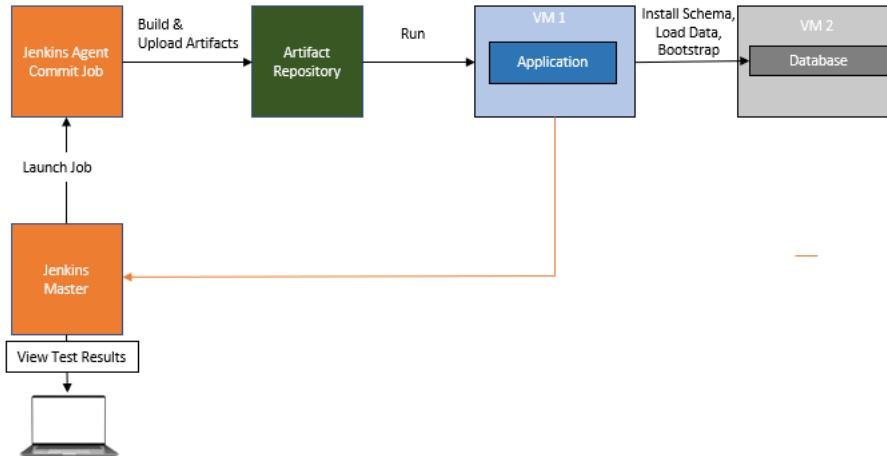
When it comes to cost considerations, the on-prem infrastructure is a large upfront expenditure with lower recurring investment (mainly around upkeep and upgrades). Whereas with the Cloud, Engineering teams have to consider daily/weekly/annual utilization and operational costs. Another theme is around storage as storage for on-premise is free and in the Cloud it has a cost.

Automated testing needs to evolve to navigate these types of nuances to avoid testing from becoming cost-prohibitive.

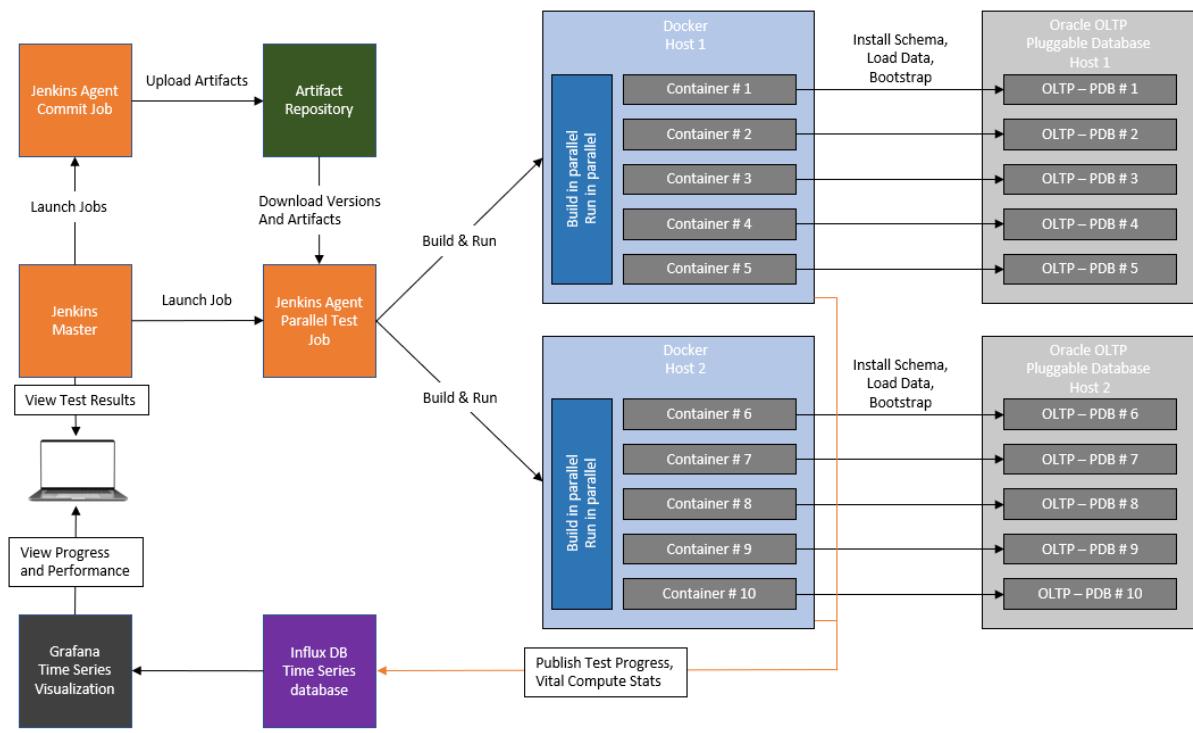
When switching to cloud, parallel testing is highly recommended. Any tests that run sequentially using automation like Jenkins will need to be converted to run in a parallel test execution framework. Some of the framework changes will include

1. Containerizing the product – the local stack of compiled classes can be deployed as an application in a microcontainer.
2. Base image for container that can be used as a common starting point for each test
3. Entry point scripts created for setting configuration and initiating security considerations like certs and tokens needed to run tests
4. Scripts that can initialize all containers and distribute tests to each container
5. Test results aggregation from each container to show a dashboard breakdown by package or sub-applications as needed by your project.

Example of a on-premise pipeline job executing tests sequentially



Example of a cloud enabled set for parallel test execution



Tooling/Scripting changes to provision and de-provision the docker and database hosts as needed will help with

- Sizing resources for your needs
- Not pay for idle resources

One more thing is E2E tests need to start building E2E image and save the image in an artifact repository.

3. TEST CHANGES

A very common example of test strategy would look like this that covers different levels of testing.

	Integration test	Functional Tests	Performance Tests	Load Tests	Resiliency Tests
Objective	confirm that the system works with other systems	confirm correctness	confirm latency and throughput requirements, scalability	confirm correct behavior under load	confirm HA and system resilience
Success criteria	Other systems can call this service	<ul style="list-style-type: none"> - Input validation - Output validation 	Measure and confirm required latency, throughput, scalability	Run system for 3 days under heavy load	<ul style="list-style-type: none"> - prove that system works when 1/3 service is down or 1/3 volume is down - prove that system can recover from failure for all volumes in 1 hour

When migrating applications/services from on-prem to Cloud a few changes in each area might be necessary to streamline testing.

3.1 AWS Integration Tests

In traditional testing familiarity with the infrastructure gives some sense of confidence on how the underlying system will behave. In a cloud-native application, there tends to be uncertainty on how it will behave, and these things cannot be assessed well when writing the application code.

It's better to create a suite of tests that are simple and can hit the API's and endpoints in AWS making sure the responses are correct. This would be a new category of tests can be used to assess the health of the endpoints before kicking off the End-to-End testing.

Here are some examples:

```

@Test
@DisplayName("Valid Endpoint")
void validEndpointReturnsForbidden() {
    given().baseUri(url + "/invalid")
        .header("Authorization", "Bearer " + authToken)
        .header("X-Tenant-ID", tenantID)
        .contentType(MediaType.APPLICATION_JSON)
        .body(getRequestBody("generateSize.sql"))
        .when()
        .post()
        .then()
        .assertThat()
        .statusCode(401);
}

```

```

@Test
@DisplayName("Invalid SQL")
void postQueryInvalidSQL() {
    given().baseUri(url)
        .header("Authorization", "Bearer " + authToken)
        .header("X-Tenant-ID", tenantID)
        .contentType(MediaType.APPLICATION_JSON)
        .body(getRequestBody("invalidSQL.sql"))
        .when()
        .post()
        .then()
        .assertThat()
        .statusCode(400)
        .body("message", containsString("There was an issue with the supplied query."));
}

```

3.2 Functional and Business flow validation using cloud endpoints

Create a collection of small AWS utilities to help with common AWS operations. Definitions can be found within the aws-sdk. Such utilities can be used through CLI or better yet as class objects in programmatic tests. Manage them as libraries in aws-commons. Some Examples

- CopyFromS3
- CopyToS3
- S3ObjectList

3.3 Performance Testing

Performance degradation arising from complexity inherent in the applications can be difficult to isolate from performance issues in the cloud due to factors such as virtualization overheads.

Performance test design should focus on areas where performance is critical. Focus on measuring key performance metrics for application caching vs in-memory data store, read-through/write-through on different latencies. These are some of the few areas where on-prem physical systems behave differently as compared to Cloud.

Another factor to consider for on-prem vs cloud performance testing is timing of the test execution. Response times in cloud could vary depending on the time of the day and time of the week. To obtain statistical confidence multiple runs need to be conducted and mean values should be used to assess the variability and then make decisions.

Learn which performance configuration options are available to you and how they could impact the workload. Optimizing the performance of your workload depends on understanding how these options

interact with your architecture and the impact they will have on both measured performance and the performance perceived by users.

3.4 Load Testing

When setting up a test to determine the load that your application can withstand, it is important to collect and evaluate metrics where workload is deployed in multiple AWS regions around the world. Some regions depending on the customer workflow might provide lower latency and a better experience.

Another advantage that cloud systems like AWS provide which is not available easily in a data center is different types of instances, storage and configurations. This can help with doing comparative load testing using virtual and automatable resources.

Example:

In AWS you can select three different EC2 instance types- small, medium, and large. Test can be designed to study their performance when they are subjected to varying intensities of load. If you see performance variability, the feedback should be for Engineering to ensure mitigations are in place in the application/service code to address potential risks.

When you use a cloud platform that easily scales to perform large-scale performance tests, you can incur high costs by deploying many servers for a long-running test and using a large amount of bandwidth in the process. To avoid this, plan ahead, and consider how much you want to spend. Model your target system accordingly to create representable results.

The feedback from load testing and costs associated with that can provide insights into discovering traditional compute activities that can benefit from moving to a serverless architecture.

3.5 Resiliency

Service failure and recovery in Cloud based applications needs handling and comes at an expensive cost and customer dissatisfaction. In a cloud native architecture where microservices are decoupled there could still be possibility of service degradation due to unexpected failures. Since Cloud based virtualization systems are considered more resilient than on-prem physical infrastructure, this is an area where more focus can be on disaster recovery and business continuity testing rather than system performance.

Think about scenarios like:

- Service A is not able to communicate with Service B
- Database is not accessible.
- The cache is not accessible.

Example Test strategy:

Run a light resiliency test against a designated service.

Outcome Expected: The service is highly available.

1. Run 2 EKS tasks, kill one.
2. Run 2 EKS tasks, kill both.
3. Stop API service intermittently, restart 2 minutes later.

Verify no records got lost.

Service failures and recovery should be built in the application codebase so that the application auto-heals and auto-communicates in the following way:

- Communication problems with the AWS managed services used by the solution is considered a critical issue and prevents the service from being operational. Set up an alerting system that will notify the team to take action.
- Internal failures in the AWS managed services used are handled by ability of the service to recover. The alerting system will notify the team to monitor the issue and take action if necessary.

3.6 Penetration Testing

Other consideration in a test strategy sometime includes Application security and penetration testing. Penetration testing helps you find cloud security gaps that create exposure and risk. It is a necessary component of security if your application is migrating to AWS, developing applications in AWS, or pen testing annually for compliance.

In the case of AWS, as a cloud service provider there is a customer service policy for penetration testing. This policy lists permitted services and prohibited services. Cloud providers like AWS welcome organizations to carry our security assessments and penetration testing against their AWS infrastructure without approval for certain services only.

Example: Any simulated Distributed Denial of Service (DDoS) testing must be performed by a pre-approved AWS partner. AWS could send out an abuse report for any unsanctioned activities related to your security testing.

4. CONCLUSION

When an application is developed, tested and run in a data center, there is a huge investment done in automated tests over the lifespan of the product or service. These tests can number in thousands. When such applications move from a DC to a public cloud platform, all this investment does not have to be thrown away. With a few changes to the existing framework and tests, a new testing paradigm for cloud-based solutions can be achieved within manageable incremental costs. Automated testing can evolve in a scalable manner to take advantage of the on-demand and self-serviced method of cloud computing without losing the leverage of an existing test suite.

References

[JSON Web Signatures for Data Integrity Proofs \(w3.org\)](#)

[Move hardcoded secrets to AWS Secrets Manager - AWS Secrets Manager \(amazon.com\)](#)

[https://docs.aws.amazon.com/pdfs/prescriptive-guidance/latest/load-testing/load-testing.pdf](#)

[https://aws.amazon.com/security/penetration-testing/](#)

Building a Culture of Accessibility: From Awareness to Action

Dhara Pattani, Younus Poonawala
dharapattani13@gmail.com, younus@rethinkingweb.com

Abstract

The digital landscape has become an integral part of our daily lives, with individuals relying on a wide range of digital platforms, including websites, mobile applications, and software solutions. However, many people with disabilities continue to face barriers in accessing and using these digital products, limiting their participation and engagement.

Integrating digital accessibility into development and testing organizations yields numerous benefits. It broadens the potential user base, allowing organizations to reach a wider audience and cater to the diverse needs of users with disabilities. By prioritizing accessibility, software engineering teams demonstrate their commitment to inclusivity. Working in concert with Product and Design teams, they help improve the overall user experience while cultivating positive brand reputation and customer loyalty.

Building a framework for accessibility within the organization encourages collaboration among team members, including designers, developers, testers, and user advocates, encouraging an environment where accessibility is a shared responsibility.

In this paper, we have listed practical techniques, tools, and steps to implement a Culture of Accessibility in your workspace.

Biography

Younus Poonawala, Director of Codevelop.us, is a visionary leader with over 15 years of experience in the software industry. Specializing in cloud computing, digital transformation, and ecommerce, he has led projects to improve website accessibility for multinational corporations. With a Bachelor's degree in Information Technology and a Masters in Computer Applications, Younus brings a strong technical foundation to his innovative work.

Dhara Pattani is a highly skilled QA analyst with extensive experience in software testing, encompassing the entire software development and testing life cycle, as well as proficiency in business requirements analysis. Certified as a Scrum Master, ISTQB Agile, and ISTQB Foundation Tester, Dhara brings a strong passion for accessibility testing. With a Bachelor's degree in Business, she combines technical expertise with a keen understanding of business needs.

1. Introduction: Developing a Culture of Accessibility

In today's rapidly advancing world, where technology and innovation have become integral parts of our daily lives, the domain of Accessibility has gained significant importance. Accessibility refers to the affordance provided by products and services so they can be easily accessed, understood, and used by individuals with a

broad range of impairments or disabilities. It encompasses physical, digital, and social aspects, aiming to eliminate barriers and promote inclusivity.

Developing a Culture of Accessibility goes beyond compliance with legal requirements; it entails building an environment where accessibility is prioritized and embedded in every aspect of an organization's operations. By cultivating such a culture, businesses, institutions, and communities can empower individuals with disabilities, enhance user experiences for all, and contribute to a more inclusive society.

This paper explores the significance of developing a Culture of Accessibility and discusses key elements and strategies to promote accessibility in various domains. It highlights the benefits of an inclusive approach and showcases how organizations can take proactive steps to embrace accessibility as a fundamental value.

We'll also emphasize on a few examples from prominent organizations like Microsoft, Apple, IBM, Procter & Gamble, and Salesforce. These examples demonstrate the successful integration of accessibility into product design, workplace practices, and community engagement. Thus, setting a standard for fostering innovation hand in hand with diversity and inclusivity

2. Why Accessibility Matters and Importance

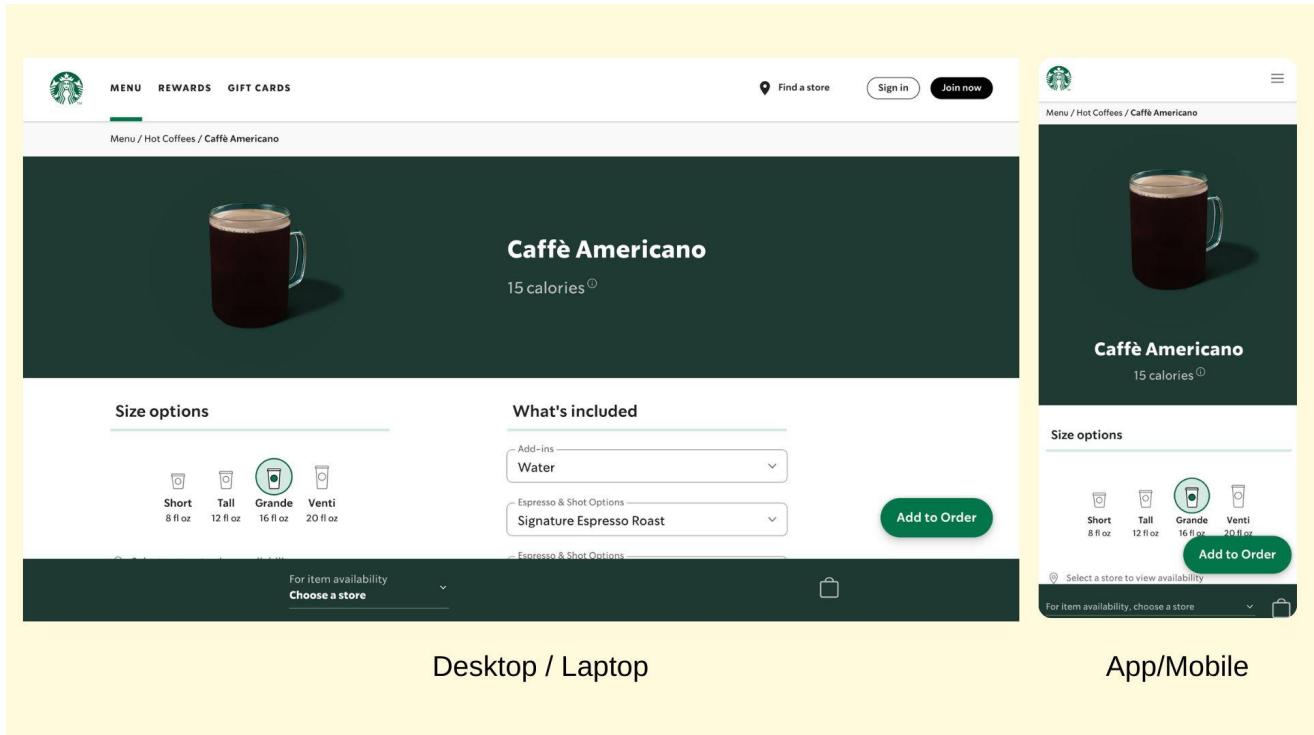
Acknowledging accessibility as a spectrum underscores the understanding that the needs of individuals with disabilities are diverse and distinct. It is our duty to offer adaptable and comprehensive solutions that cater to these varying abilities and requirements, ensuring flexibility and inclusiveness in our approach.

Key Points:

1. **Empowering Individuals:** Accessibility empowers people with disabilities by granting them equal access to information and technology, facilitating their full participation in the digital world. It breaks down barriers and provides opportunities for personal and professional growth.
2. **Enhancing User Experience:** Accessible designs not only benefit individuals with disabilities but also enhance the experience for all users. Clear navigation, structured content, and customizable interfaces contribute to improved user satisfaction, facilitating seamless interactions with digital content.
3. **Meeting Legal Requirements:** Compliance with accessibility laws showcases social responsibility and underscores the commitment to ensuring equal access for individuals with disabilities. By adhering to these regulations, organizations demonstrate their commitment to inclusivity and promote a more inclusive society.
4. **Expanding Market Reach:** Accessible products and services tap into a significant market segment comprised of diverse users. By prioritizing accessibility, organizations can attract a broader user base, expanding their market reach and driving business growth.
5. **Fostering Innovation:** Accessibility stimulates creative problem-solving and fuels innovation within the technology industry. By addressing the unique challenges faced by individuals with disabilities, it encourages the development of innovative solutions that benefit society as a whole.

Prioritizing accessibility is vital for creating a more inclusive digital world, where everyone has the opportunity to thrive. It empowers individuals, enhances user experiences, expands market reach, and fosters innovation within the technology sector. Let us embrace accessibility as a core principle and work towards shaping a future where technology is accessible to all, ensuring a more equitable and inclusive society.

Examples:



The Starbucks app is an example of a product that has been designed with inclusive principles. The app includes features such as text-to-speech and keyboard navigation, which make it accessible to people with disabilities. This allows people with disabilities to use the app to order food and drinks, just like everyone else. By designing for inclusivity, Starbucks is creating a more equitable and accessible world for everyone.

The New York City subway system is an example of a public transportation system that has been designed with inclusive principles. The system includes features such as audio announcements and tactile indicators, which make it accessible to people with disabilities. This allows people with disabilities to use the subway system independently, just like everyone else. By designing for inclusivity, the New York City subway system is making it possible for people with disabilities to participate fully in society.

3. Common Misconceptions of Accessibility

There are many misconceptions about accessibility, both among businesses and the general public. Some of the most common misconceptions include:

- Accessibility is only for people with disabilities:** This is not true. Accessibility benefits everyone, regardless of their abilities. For example, keyboard navigation can be helpful for people who have difficulty using a mouse due to arthritis. Fonts resizing, dark modes, and high contrast colors can be helpful for people who have common visual impairments. Plus voice and touch controls provide different interaction modalities that all users can choose from.
- Accessibility is expensive:** There are many ways to make digital products and services more accessible without spending a lot of money. For example, businesses can use free tools to add text alternatives to images and videos, and they can make small changes to their websites and applications to make them easier to navigate with a keyboard. More importantly, if accessibility is

integrated into the workflow, it can improve the overall development process and reduce rework. Moreover, when organizations ignore accessibility they risk potential legal action, which can exceed the costs of evolving the organization and do serious reputational damage.

3. **Accessibility is difficult to do:** This is not true. There are many resources available to help businesses make their products and services more accessible. For example, the World Wide Web Consortium (W3C) has a website with accessibility guidelines, and there are many consulting firms that specialize in accessibility. Many developers have learned the skillset to code for accessibility, and more are learning every day. These developers can influence culture and lead accessibility efforts.
4. **We don't need to conform to any Accessibility Standards:** One prevalent misconception regarding accessibility is the notion that conforming to accessibility standards is unnecessary. However, Title III of the Americans with Disabilities Act has been amended to include many websites and apps. It is vital to work toward these guidelines (such as the newly updated WCAG 2.2) to guarantee equal access for individuals with disabilities and to mitigate legal risks. Failure to conform to WCAG guidelines can lead to exclusion of a substantial user base and missed business prospects.

4. Understanding Accessibility and Building an Accessible Mindset

Approximately 1 in 5 adults in the US, accounting for over 60 million people, experience disabilities that pose challenges in accessing digital products**. This includes individuals who are blind, visually impaired, deaf, hard of hearing, or have physical or cognitive disabilities. It's worth noting that temporary disabilities can also impact accessibility. For instance, a person recovering from a temporary injury may require assistive technology.



A website that does not have alt text for images is inaccessible to screen reader users. Screen reader users cannot see images, so they rely on alt text to understand what they are about. Without alt text, screen reader users will not be able to understand the content of the image.

By prioritizing accessibility, we can break down these barriers and cultivate an inclusive digital world that offers equal opportunities for all individuals, regardless of their current abilities.

An illustration of the positive outcomes of prioritizing accessibility can be observed through the case of **Shopify**. Through enhancements in website accessibility, Shopify achieved a notable 10% increase in web traffic from individuals with disabilities. This surge translated into more customers, higher sales figures, and an enhanced reputation. This case serves as evidence that accessibility contributes not only to social responsibility but also to business prosperity.

Another example of a company that has successfully implemented accessibility is **Apple**. Apple has been a leader in accessibility for many years. The company has a long history of adding features to its products that make them more accessible to people with disabilities.

- **Built-In Accessibility Features:** Apple integrates a comprehensive range of accessibility features directly into its operating systems (iOS, macOS, watchOS, etc.) and applications. These features cover a wide spectrum of disabilities, including visual, auditory, cognitive, and motor impairments.
- **VoiceOver:** VoiceOver is a screen reader that allows people who are blind or visually impaired to use Apple devices. It reads aloud the text on the screen, as well as other information, such as labels, buttons, and menus.
- **Zoom and Magnification:** Apple devices offer features like Zoom and Magnifier, which allow users to enlarge text, images, and other content on the screen. This is particularly useful for people with low vision.
- **Dynamic Text and Display Accommodations:** Users can adjust text size, boldness, contrast, and other display settings to make content more readable based on their preferences.
- **Live Captions:** Live Captions automatically generates captions for audio and video content, making it easier for people who are deaf or hard of hearing to understand what is being said.
- **Hearing Loop:** Hearing Loop is a feature that allows people who are hard of hearing to use Apple devices in noisy environments. It transmits sound directly to a hearing aid or cochlear implant.
- **AssistiveTouch:** AssistiveTouch allows users with motor impairments to control their Apple devices with gestures.
- **Switch Control:** Switch Control allows users with motor impairments to control their Apple devices with a switch.
- **Siri:** Siri is a voice assistant that can be used to control Apple devices by voice.

Apple is constantly working to improve its accessibility features. In 2023, Apple released iOS 16, which includes a number of new accessibility features, such as Live Text and Door Detection. Live Text allows users to extract text from images, and Door Detection helps users with visual impairments navigate to doors.

Apple's commitment to accessibility is an example of how technology can be used to make the world a more inclusive place. By making its products and services more accessible, Apple is helping to ensure that everyone can enjoy the benefits of technology.

5. Inclusive Design Principles

Accessibility is the design of products, services, and environments to be usable by everyone, regardless of their abilities. The core principles of accessibility are[±]:

- **Perceivable:** Information and user interface components must be presented to users in ways they can perceive. For example, text alternatives should be provided for all non-text content, such as images and videos. This allows people who are blind or visually impaired to understand the content.
 - An image of a product on a website could have a text alternative that describes what the product looks like, its features, and how it is used.
 - A video could have a text transcript that describes the content of the video.
- **Operable:** User interface components and navigation must be operable. For example, keyboard navigation should be supported so that people who cannot use a mouse can still interact with a website or application. This allows people with motor impairments to use a website or application.
 - A website could have keyboard shortcuts for navigating to different pages, opening menus, and selecting options.
 - A website could also have a "skip to main content" link that allows users to jump to the main content of the page without having to scroll through the page.
- **Understandable:** Information and the operation of user interfaces must be understandable. For example, clear and concise language should be used, and complex or technical terms should be explained. This allows people with cognitive impairments to understand the content of a website or application.
 - A website could use plain language instead of jargon, and it could provide definitions for any technical terms that are used.
 - A website could also use a consistent design throughout the site, so that users can easily find their way around.
- **Robust:** Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies. For example, web pages should be coded correctly so that they can be read by screen readers. This allows people with disabilities to use assistive technologies to access a website or application.
 - Web pages should use semantic markup so that screen readers can properly identify different elements on the page, such as headings, paragraphs, and lists.
 - Web pages should also be coded in a way that makes them accessible to search engines, so that people with disabilities can find them through search results.

By following these principles, businesses can create websites and applications that are more accessible.

6. Integrating Accessibility into Processes

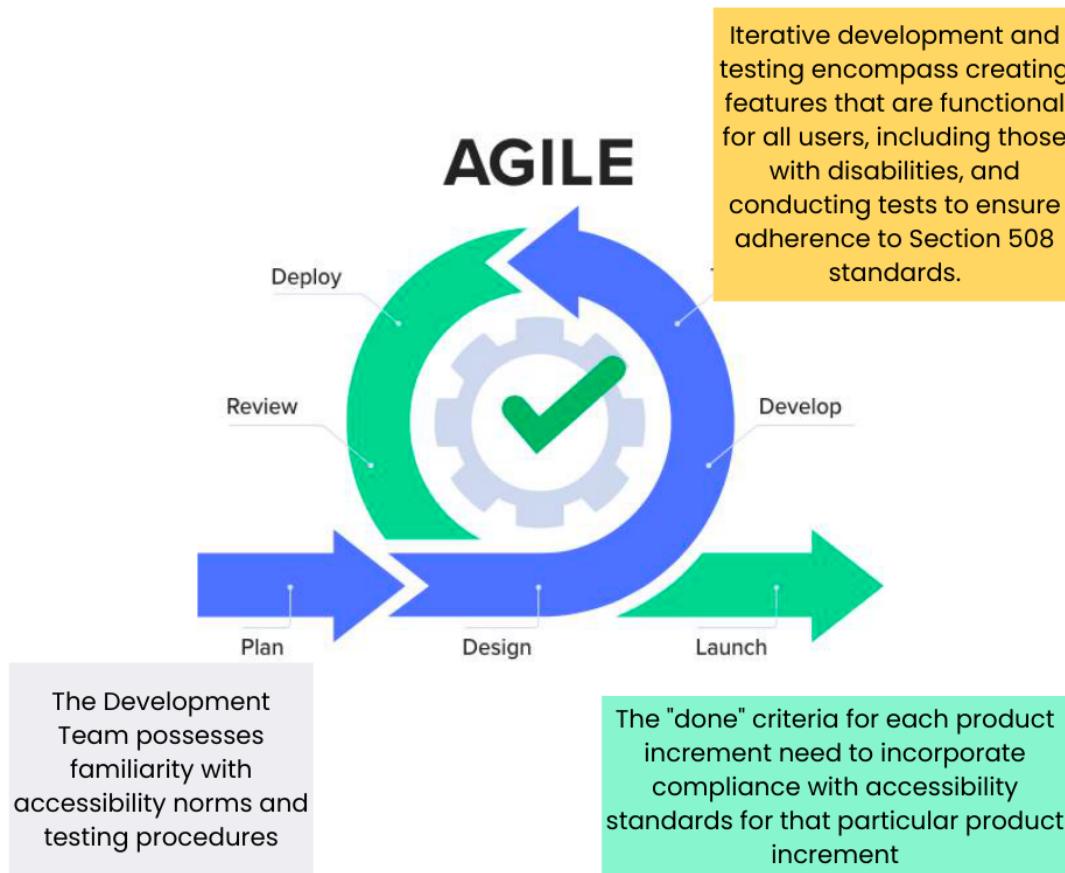
Integrating accessibility as an integral part of the development process is vital for creating inclusive digital experiences. By incorporating accessibility from the outset, we can proactively tackle accessibility issues and deliver products that cater to diverse user needs. Here are key steps to integrate accessibility effectively:

- **Start with accessibility in mind:** When designing a website or application, start by thinking about how people with disabilities will use it. Consider what features they will need to be able to access the content and interact with the interface.
- **Use accessible tools and technologies:** There are a number of tools and technologies that can help you create accessible websites and applications. These tools can help you identify accessibility issues, generate text alternatives for images and videos, and code your website or application in a way that is accessible to screen readers.
- **Get feedback from people with disabilities:** Once you have created your website or application, get feedback from people with disabilities to make sure it is accessible to them. You can do this by

conducting usability testing with people with disabilities or by inviting them to provide feedback on your website or application.

- **Continuously improve:** Accessibility is an ongoing process. Once you have made your website or application accessible, you need to continuously improve it to make sure it remains accessible. This means keeping up with the latest accessibility standards and best practices, and regularly testing your website or application for accessibility issues.
- **Make accessibility a priority:** Accessibility should be a top priority for any business that wants to create websites and applications that are accessible to everyone. This means allocating the necessary resources and time to ensure that accessibility is considered at every stage of the development process.
- **Provide training:** Provide training to all employees on accessibility and how to create accessible websites and applications. This training should cover the basics of accessibility, such as the WCAG 2.2 accessibility standards, as well as specific tips and best practices for creating accessible content and code.
- **Advocate for accessibility:** Advocate for accessibility within your industry and community. This means speaking out about the importance of accessibility and working to raise awareness of accessibility issues.

6.1 Accessibility in the Agile World



Agile development involves building products in constrained time increments called sprints. Accessibility must be ingrained within Agile processes, where all team members accept it as essential. Training is crucial for developers to write accessible code. Accessibility requirements should be included in product backlogs, design, and architecture decisions, and adhered to in "definition of done" for each increment. Testing should follow standard processes, including manual inspections, even with automated tools. Test-Driven Development (TDD) can help ensure accessibility compliance. Section 508 expertise should be consulted when needed. Agile's benefits include immediate feedback and adaptability, while potential drawbacks include ambiguous timelines and skill dependencies. Success in Agile accessibility requires consistent integration throughout the entire development lifecycle:

- Include accessibility in the "definition of done".
- Integrate accessibility standards into key artifacts.
- Implement standardized accessibility testing.
- Follow Test-Driven Development (TDD) practices.
- Provide Section 508 expertise to the team and ensure accessibility knowledge for all members.

Making accessibility a priority and **cultivating a culture of accessibility within the organization is crucial**. Providing training to employees and advocating for accessibility in the industry and community further enhance accessibility integration efforts.

By integrating accessibility into the development process, we ensure inclusive and accessible products, early identification of accessibility issues, and a more user-friendly digital experience.

7. Training and Education

There are various training and education options available to address accessibility and inclusion. Consider the following examples:

1. **Foundational Accessibility Training:** This training covers the fundamental concepts of accessibility, including an overview of accessibility standards such as WCAG 2.1, and practical guidance on creating accessible digital products.
2. **Specialized Accessibility Training:** This type of training focuses on specific aspects of accessibility, such as designing accessible user interfaces, developing accessible content, or implementing accessibility in coding practices.
3. **Inclusion and Diversity Training:** This training emphasizes the importance of fostering inclusive environments and explores strategies for promoting diversity, equity, and inclusion in workplaces and communities.

The choice of training depends on the organization's objectives, industry, and size. However, it is recommended that all organizations provide some level of training on accessibility and inclusion.

Benefits of providing training and education on accessibility and inclusion include:

- **Enhanced Accessibility:** Training empowers organizations to improve the accessibility of their digital products, services, and physical spaces, enabling people with disabilities to fully participate and benefit.

- **Mitigated Legal Risk:** By educating employees on accessibility requirements and best practices, organizations can reduce the risk of legal liabilities associated with non-compliance, ensuring compliance with accessibility laws and regulations.
- **Boosted Employee Morale:** Training fosters a culture of inclusivity, demonstrating an organization's commitment to diversity and accessibility. This, in turn, enhances employee morale, engagement, and job satisfaction.
- **Strengthened Reputation:** Organizations that prioritize accessibility and inclusion through training build a positive brand reputation, attracting customers, clients, and stakeholders who value inclusive practices.

To ensure effective training and education on accessibility and inclusion:

- Tailor the training to the audience's needs, focusing on their roles and responsibilities within the organization.
- Utilize diverse teaching methods, such as interactive workshops, practical exercises, and real-life case studies, to engage participants and enhance learning outcomes.
- Provide opportunities for participants to practice and apply their knowledge through hands-on exercises and projects.
- Follow up with participants to gauge their understanding and address any further questions or concerns, promoting continuous learning and improvement.

By implementing these strategies, organizations can effectively integrate accessibility and inclusion principles into their processes, resulting in more inclusive digital experiences and environments.

8. Tools and Technologies

In the pursuit of inclusivity, accessibility testing plays a vital role in identifying and addressing barriers that individuals with disabilities may encounter when engaging with digital products and services. To facilitate accessibility testing and ensure comprehensive evaluations, a wide range of tools and technologies have emerged. This section explores key tools and technologies used for accessibility testing in a professional context.

Automated Testing Tools

Automated testing tools are instrumental in scanning websites, applications, and digital content for accessibility issues. These tools employ algorithms and predefined rules to identify common accessibility violations, such as missing alt text for images, improper heading structures, or insufficient color contrast. Notable examples of automated testing tools include aXe, WAVE, and pa11y. These tools provide developers and testers with expedient feedback, enabling efficient remediation of accessibility barriers.

Manual Testing Tools

While automated tools are valuable, manual testing remains indispensable for detecting intricate accessibility challenges that necessitate human judgment and contextual comprehension. Manual testing involves expert evaluators systematically reviewing and interacting with digital content using assistive technologies like screen readers, keyboard-only navigation, and magnifiers. This approach allows for an in-depth exploration of the user experience, uncovering nuanced accessibility obstacles that automated tools may overlook.

Assistive Technologies

Assistive technologies are tools that can be used to help individuals with disabilities interact with digital content. Examples:

- **Screen Readers:** Software applications that convert text into speech or braille, enabling seamless access to digital content for people with visual impairments.
- **Screen Magnifiers:** Enhancing readability for individuals with visual impairments by enlarging on-screen content. Every pixel becomes a gateway to knowledge and connection.
- **Braille Displays:** Tactile devices that allow people who are blind or have low vision to read and navigate digital content through touch.
- **Alternative Keyboards:** Customized input options for individuals with motor impairments, empowering them to express themselves and explore the digital realm.
- **Switch Access:** Activating computers through external switches, enabling effortless navigation for those with limited mobility or dexterity.
- **Voice Recognition Software:** Control computers and input text using speech, empowering individuals with mobility impairments or typing difficulties.
- **Assistive Listening Devices:** Amplifying sound and improving clarity for individuals with hearing impairments, fostering engagement with audio content.
- **Closed Captioning and Subtitling:** Textual representations of audio content in videos, ensuring accessibility for individuals who are deaf or hard of hearing.
- **Alternative Formats:** Large print, audio books, and e-text, catering to individuals with print disabilities or visual impairments. Accessing knowledge in unique ways.

Expert Evaluators

Expert evaluators are individuals who have specialized knowledge and experience in accessibility testing. They can help to identify and address accessibility issues that may not be obvious to others.

By using a variety of tools and technologies, and by involving expert evaluators, organizations can create more accessible digital products and services that can be used by everyone.

Tools

1. <https://accessibe.com>
2. WAI Easy Checks <https://www.w3.org/WAI/test-evaluate/preliminary/>
3. aXe Chrome/Firefox/Selenium tool or WAVE Extension for Chrome or Firefox
4. <https://github.com/GoogleChrome/accessibility-developer-tools>
5. <https://access-works.com/>

iOS Accessibility Tools/features

1. Vision Deaf or hard of hearing
2. VoiceOver (screen reader) = FaceTime
3. Siri, Closed Captions
4. Speak Selection Mono Audio
5. Dictation Visible and Vibrating Alerts
6. Zoom Made for iPhone Hearing Aids
7. Font Adjustments Physical or motor challenges
8. Invert Colors = AssistiveTouch
9. Braille display support = Siri
10. Wireless Bluetooth = Switch Control
11. Keyboard support

Android Accessibility features

1. TalkBack (screen reader)
2. Mono audio
3. Captions = Explore by touch
4. Magnification gestures = Change font size
5. BrailleBack support = Support only available in
6. Touch & hold delay recent version of OS

9. Collaboration and User Feedback

By working with people with disabilities and other stakeholders, organizations can ensure that their products are usable by everyone.

There are many ways to collaborate with people with disabilities. One way is to form an **advisory board or committee** made up of people with disabilities. This group can provide valuable feedback on the design and development of products. Another way to collaborate is to conduct user testing with people with disabilities. This allows organizations to see how people with disabilities interact with their products and identify any accessibility issues.

User feedback is also essential for creating accessible and inclusive products. People with disabilities can provide valuable feedback on the usability of products, as well as suggestions for improvements. Organizations should make it easy for people with disabilities to provide feedback, such as by including a feedback form on their website or by creating a user forum.

By collaborating with people with disabilities and gathering user feedback, organizations can create accessible and inclusive digital products and services that everyone can use.

10. Creating a Roadmap

Creating a roadmap for accessibility is an essential step for organizations that want to create accessible and inclusive digital products and services. A roadmap can help organizations to:

- **Identify accessibility goals:** What are the organization's goals for accessibility? Do they want to make all of their products and services accessible? Or do they want to focus on a specific type of product or service?
- **Prioritize accessibility efforts:** Not all accessibility issues are created equal. Some issues are more important than others. A roadmap can help organizations to prioritize their accessibility efforts and focus on the most important issues first.
- **Track progress:** A roadmap can help organizations to track their progress towards their accessibility goals. This can help to ensure that the organization is on track and making progress.
- **Communicate with stakeholders:** A roadmap can be used to communicate with stakeholders about the organization's accessibility goals and progress. This can help to ensure that everyone is on the same page and working towards the same goals.

Here are some steps that organizations can take to create a roadmap for accessibility:

1. **Identify stakeholders:** Who are the key stakeholders who will be involved in creating and implementing the roadmap? This could include people from different departments within the organization, as well as people with disabilities.

2. **Conduct an accessibility audit:** This will help to identify the organization's current accessibility status. This could involve conducting user testing with people with disabilities, reviewing the organization's website and other digital products and services for accessibility issues, and reviewing the organization's accessibility policies and procedures.
3. **Set accessibility goals:** What are the organization's goals for accessibility? These goals should be specific, measurable, achievable, relevant, and time-bound.
4. **Prioritize accessibility efforts:** Not all accessibility issues are created equal. Some issues are more important than others. The organization should prioritize its accessibility efforts and focus on the most important issues first.
5. **Develop a plan to achieve the goals:** This plan should include specific steps that the organization will take to achieve its accessibility goals.
6. **Implement the plan:** This involves putting the plan into action and making the necessary changes to the organization's products, services, and policies.
7. **Track progress:** The organization should track its progress towards its accessibility goals. This can be done by conducting regular audits, reviewing feedback from users with disabilities, and monitoring the organization's accessibility policies and procedures.
8. **Communicate with stakeholders:** The organization should communicate with stakeholders about its progress towards its accessibility goals. This can be done through regular meetings, reports, and other communication channels.

By following these steps, organizations can create a roadmap for accessibility that will help them to create accessible and inclusive digital products and services.

11. Conclusion

In conclusion, accessibility is not just a legal requirement but a moral imperative. It is essential for creating a digital world that is inclusive and equitable, where individuals with disabilities can fully participate and access information, products, and services. By prioritizing accessibility in design and development, organizations can break down barriers and empower individuals of all abilities. It is crucial for businesses, governments, and society as a whole to embrace accessibility as a fundamental principle, ensuring that no one is left behind in the digital age. Let us continue to work together to build a more accessible and inclusive future for everyone.

References

- https://www.slideshare.net/frontenders-valtech/rebecca-topps-mobile-accessibility?from_search=0
- https://www.slideshare.net/aidantierney/mobile-accessibility-best-practices-and-trends?from_search=4
- https://www.slideshare.net/ecentricarts/web-accessibility-101-the-why-who-what-and-how-of-a11y?from_search=10
<https://www.disabled-world.com/disability/>
- https://www.bluehost.com/blog/accessibility-for-websites/?psafe_param=1&irpid=101&clickid=P61C101S570N0B5578A2D4499E0000V114&qclid=EA1aIQobChMliv_wz9Kt_wlVyCGzAB0mgAwBEAAAYAaAEgLdefD_BwE&gclsrc=aw.ds
- <https://www.w3.org/WAI/fundamentals/accessibility-principles/>
- https://medium.com/@lrfan_ali/_integrating-accessibility-across-the-agile-development-lifecycle-by-ifran-ali-from-princeton-nj-1f2b9b2994ad

Navigating Complex GraphQL Testing: Automating Integration Test Generation

Nate Smith, Ph.D.

nate@natesmith.io

Abstract

Care.com's federated GraphQL endpoint features over 450 operations, acting as a gateway to a multitude of backend data sources. Developing an additional layer of integration testing for our graph following internal staging deployments presented a significant challenge due to the vast number of possible operation variations.

Drawing on some work by Stefan Karlsson, Adnan Čaušević, and Daniel Sundmark (Karlsson et al 2020), we built a script that automatically generates GraphQL API requests by traversing the graph under test, constructing a gql query, and then populating input variables using a property testing library. The result is an easy-to-run, self-updating test script that tests every operation with a variety of fields and input values without any need for manually writing GraphQL queries to test.

I describe the results of this effort, and then discuss what I think this says about what is possible with building integration tests for a complex system like this. While we cannot automate everything, sometimes it turns out that we might be able to automate more than we first think is possible.

Biography

Nate Smith is an accomplished software professional with over 15 years of diverse experience in the industry. He began his career in the early 2000s in software QA for medical devices. Shortly thereafter, he spent the next decade earning graduate degrees in philosophy, and then returning to the industry in 2011. Since then he spent several years working in quality assurance and building automated testing frameworks. He spent several years in engineering management across many different functions, including quality engineering, site reliability engineering, data engineering, and product localization. He is currently a principal software engineer at Care.com, where he has built several automated testing frameworks and now develops backend application systems.

Nate holds a PhD in philosophy from the University of California, Davis, an MSc in philosophy & history of science from the London School of Economics, and a BA in philosophy from UC Berkeley. During his academic career, he published multiple articles and presented his work at numerous national and international conferences. He specialized in the philosophy of science.

Copyright Nate Smith, 2023

1 The Problem

Care.com's ecosystem of applications is supported by a federated GraphQL API containing more than 450 distinct operations. In 2022, we had numerous automated test pipelines testing various components of our systems including end-to-end tests for consumers of the GraphQL API, a robust unit test suite for the API itself, and unit and integration tests for the dozens of backend microservices supplying data sources. But we had no GraphQL-specific integration tests.

Integration tests at the API layer, especially if they are comprehensive, would be likely to uncover some kinds of bugs that would be unlikely to be detected elsewhere: integration failures between the graph and backend services, edge cases on GraphQL operations not current exercised by the consumers of the graph, and others. We decided to invest in a project to build out an integration test layer for the GraphQL API itself.

However, building even a basic set of integration tests making a series of GraphQL requests to cover our core operations would be an enormous undertaking, even if we distributed the work amongst our delivery teams. GraphQL requests are more varied and flexible than traditional REST requests, and to get adequate coverage across our hundreds of operations iterating over the many possible field choices and input parameters, we would need to build thousands of tests. When we get to this scale, it almost demands automating as much of this as possible to make it tractable, so we did some research into the possibilities. We found recent research into this problem by Karlsson, Čaušević, and Sundmark (2020) with a prototype and proof of concept for automatically generating GraphQL property tests and decided to operationalize this and bring it to our stack. Below I will describe the challenge in testing our graph in a bit more detail by contrasting it with testing a traditional REST API, and then describe our framework, what our results were, what we learned, and how we are moving forward.

1.1 Testing a REST API

A Representational State Transfer (REST) endpoint is a specific URI (Uniform Resource Identifier) that a RESTful web service exposes and through which the service's resources interact. This interaction is accomplished using standard HTTP methods, such as GET, POST, PUT, and DELETE.

When testing a REST API, we typically create a series of valid and invalid HTTP requests and send them to the targeted endpoint. These requests may include both required and optional fields. The complexity of these payloads can vary, ranging from single scalar values, to deeply nested JSON objects. But generally speaking, a good REST endpoint provides access to a single resource per request. We request a single user (or a list of users), but we if we want additional resources we make additional requests.

Let us consider a simple example for a GET request. Suppose we have an endpoint, “/users/{id}”. This endpoint accepts GET requests, and has a single required value for the payload, *id*. Assuming for simplicity's sake that authorization is not required, if the user entity exists with that id, the server returns a response containing the user object. If it does not exist, we get back an error message indicating that the resource does not exist.

This user entity may be quite complex: it could have dozens of attributes, some of them nested entities themselves (for example, an address that has a number, street name, city, postal code, etc.). Once we know the structure of the possible response, testing such an endpoint is relatively straightforward. We might seed the server with data with various extreme values if we can, testing the boundary conditions. We may try sending various invalid inputs for the user id, but we are still just sending a single id to this endpoint.

Endpoints can of course support more complex payloads. POST and PUT requests in particular may have deeply nested JSON payloads including dozens of fields (perhaps we are submitting a web form). Such cases are more complex to test, but once we know the possible fields that will be accepted, we can

vary which ones we send, vary the data we send for each one, and verify that we get the expected responses. Furthermore, there plenty of tools in the community that can help generate and iterate on well-documented REST endpoints.

The general challenge of testing a REST API endpoint is to understand the possible input payload, possible response, and iterate on these. We can use tools to iterate on the inputs, and assert on the responses.

1.2 Distinct Challenges of Testing a GraphQL API

GraphQL operations come in two types: *queries*, requesting data, and *mutations*, modifying data. I will primarily be concerned with queries in this discussion, and in our test framework we restricted ourselves to testing queries, but in principle we could test mutations as well. They present similar challenges, but also have their own additional challenges of mutating data on test servers. For now, we are focused on queries. Let us stick with the *user* example.

First, note that GraphQL graphs are, of course, *graphs*. Nodes on the main graph are *fields*, information stored that has one or more relationships to other fields. Fields have types, either scalar (Booleans, strings, etc.), or objects, which are in turn defined on the graph. For example, a *getUser* operation might return a top-level *user* object, defined to have an *address* (among other fields), which is itself a defined object type. An *address* field might itself have a *street number* (a scalar string most likely), and so on.

The idea is that when we send a request to a GraphQL endpoint, we request only those values ('fields') that we actually need. With a REST request for a user resource, perhaps there is a lot of information that can come back on the response payload: user information, addresses, emails, orderIds, or whatever the business context requires. For a GraphQL request, however, my client might need a single field: perhaps I simply need to know the list of orders associated with a user, so I can specify that in my request. On the other hand, in some cases, the number of resources requested by a single request can be literally infinite in a sense (there's a chance for infinite graph traversal loops in some graphs, but more on that later; that will need to be managed.)

Even if we restrict ourselves to a single operation, it is easy to start seeing why each operation gets complicated to test. Which fields do we request when we write our tests? Just some of them or all possible ones? If not all, which ones? How many combinations of fields do we test? This does not even get to the input data itself, which has the all the same complexity that testing REST endpoints does in varying the payloads. When testing our graph, we found that it was quite easy to generate requests that requested *hundreds* of fields. There is a new dimension of data -- the fields that we request -- that generates an explosion of possibilities. We need a way to create these, so we do not have to manually write them for all the operations. The challenge is overwhelming if we attempt to tackle it manually.

Even worse, there is no theoretical limit to how far the graph traversal goes. Some graphs have cycles (paths on the graph that form closed loops), and in such cases, the number of fields requested is literally infinite.

Given these challenges, we were going to have to either settle for a small set of manually written test scripts, or else find an automated way to generate them. In the next sections, we will delve into how we pursued the automated solution. It turns out that what seems to be a particular challenge ends up being a strength; we can exploit the graph structure of the API to generate test cases.

2 Prior Efforts

When researching possible ways to test our graph, we took note of a recent contribution from a paper titled "Automatic Property-testing of GraphQL APIs" published in 2020 by Karlsson, Čaušević, and Sundmark. They demonstrated that one could generate inputs for a GraphQL schema using a property-testing library, and generate the gql strings themselves (i.e., the payload that lists the fields requested) by

performing a graph traversal algorithm to build the strings. They were able to completely generate the test cases, managing both the field selection and input generation.

They built their proof of concept in Clojure, and to evaluate the efficacy of their solution, they injected faults into their graph and ran their automatic testing framework to detect these bugs. The results were promising - they identified 73% of the injected bugs. So, they are not catching everything, but they are finding a *lot*. Generating requests and using a property-testing framework to generate inputs and assert on them, resulted in finding many bugs. We decided to try this.

Care.com is primarily a TypeScript and Go shop, so we were not super excited about the idea of pulling in a new language / framework, learning it, and seeing how far we could run with it. Instead, we spent a couple weeks building out a few basic components of a TypeScript framework, that could handle a limited set of possible field operations. The initial version accepted query name, and built out a gql string to two levels deep for any query that had scalar inputs only (no objects), and then sent some canned values for the various input types. The initial results were promising (we found a few bugs immediately), and we continued handling all possible query types and more complex input objects.

3 On Property Testing

It is worth saying a little about property testing at this point; it ended up playing less of a role than we expected in our framework, but it is a core part of the idea.

At a high level, property testing is the process of varying the input to a system under test and asserting that the output conforms to a certain 'property'. This is different from another sort of test in which we validate behavior for specific input-output pairs, property tests articulate general properties of a system that should hold true for all inputs. Instead of "input A is expected to produce output B," it is "the range of possible inputs of type X should produce the range of possible outputs of type Y." To do this, we need two things: a way of generating input values, and a way of asserting on a range of output values. Rather than an expected output being, say, the *number* 5, we might expect that the output be a number in the range 1-50. We might know for a particular value that the system should only produce values in this range, so we generate a ton of inputs and make sure that no matter what we input, we still get an output in this range.

These input generators used in property testing are called 'arbitraries'. They generate a wide array of inputs, covering edge cases that a developer might not think to test. While this can include boundary conditions such as empty strings or very large numbers, the most valuable aspect of these arbitraries is their ability to generate truly random data. This randomness often leads to the discovery of unforeseen bugs and is a key reason behind the effectiveness of property testing.

When the property-testing library is generating values and asserting on the property, if a failure case is found, the library will try to isolate the range of values that produce the failed result by iterating over ranges of values and narrowing down to the failing range. This process is called "shrinking": simplifying the input range as much as possible to identify the precise failure conditions. The library will continue to generate values and test outputs until it identifies this precise range.

In the context of our GraphQL API, the goal was to build arbitraries that could generate valid queries to test against our system. These queries would have to respect the structure of our GraphQL schema while introducing enough variation to uncover any potential issues; you will see how we do that below.

4 Approach and the Framework

Care.com's GraphQL API is built with TypeScript; we decided to build the testing framework using the same stack. We wanted something familiar to our developers, and the tooling for client libraries in TypeScript is excellent given the extensive frontend GraphQL client ecosystems from organizations such as Apollo (<https://www.apollographql.com>) and Prisma (<https://www.prisma.io/graphql>).

We used FastCheck (<https://github.com/dubzzz/fast-check>) for the arbitraries; this is a property-testing library for JavaScript / TypeScript that is being actively maintained. We also used graphql-codegen for building a JSON representation of our graph from the graph schema, enabling the automatic generation of queries for testing.

All of these pieces put us in a place where we could build an automatic test-case generator and runner, using a depth-first search graph traversal to build a payload of requested fields, and fast-check to generate values for the input values. We iterated over a few weeks, and ended up with the core of our framework that generated and ran test cases for all the queries on the graph.

The core logic of the test pipeline is as follows:

- 1. Schema Retrieval:** To test the latest version of the graph and represent the graph in a form that our TypeScript code can interpret, we retrieve the latest version of the graph as a gql file. We then run graphql-codegen to transform the file into an introspection JSON; essentially a JSON representation of the schema. (If introspection is enabled on your graph, you may be able to retrieve such a JSON file directly from the graph.) This representation of the graph includes all the queries, mutations, fields, inputs, and their types.
- 2. Generate List of Queries:** We load the introspection file, and then iterate over the array of query names in the QUERIES node to extract the list of all queries in the schema.
- 3. Load Configuration including Users:** Here we load the configuration for the test run, which includes a max_depth, a field selection strategy (all vs random), a maximum arbitrary iteration number, and an array of users to iterate over.

Then, for each query to be tested:

- 4. Authenticate against the graph:** We run through each operation with each user, so first we authenticate against the graph with the user credentials.
- 5. Generate the GQL string:** Starting with the query name, run a depth-first search graph traversal to the maximum depth specified in the configuration and add each discovered field to the gql string. If we hit the maximum depth and we are at a node that has required fields, we remove fields until we get to a valid gql string. If this does not end up with a valid gql string, we throw an error.
- 6. Generate the Input Object:** Construct and input object for the variables in the query. For each input variable, the system parses its type, and then matches the type to a predefined data generator driven by fast-check. Some of these data generators produce random strings, while others generate more specific data such as zip codes or phone numbers.
- 7. Send the payload:** Send the request to the endpoint.
- 8. Iterate on input:** FastCheck steps in here; if the response is success, continue generating inputs and making requests out to a maximum specified by our configuration. If there is a failure, start ‘shrinking’ – FastCheck iterates over requests trying to find the limits of the failure range.
- 8. Assert that the response conforms to a known property.** At the most basic level, assert that the response is not an error object or error response. Fail the test if this is not the case. It would also be possible at this point to make a more complex property assertion. For example, we might wish to assert that if a given field requested is of type ‘zip code’, assert that the field returned is a five-digit integer. (We so far have not chosen to do this, as the server itself enforces its own schema validation on returned values and adding this to framework is a decent amount of work. But it would be a prudent extra verification.)
- 9. Test Completion:** The testing framework concludes the test run by compiling a report of the failures. It leverages fast-check’s property test reporting to provide detailed information on the values and ranges

that caused each query to fail. This report offers valuable insights for troubleshooting and improving the GraphQL API.

In our approach, we should note that our framework, at this stage, largely uses arbitrary or random inputs for test queries. The fast-check library, which we employ to generate these inputs, is not equipped with the inherent knowledge of the business logic associated with a particular field. Hence, unless we specifically define this logic ourselves, it generates random data inputs like strings, integers, Booleans, etc.

However, we have tried to construct more intelligent generators for certain fields to ensure they generate valid data according to our schema. For instance, for fields like phone numbers, zip codes, and user UUIDs, we have built generators that produce valid US phone numbers, valid zip codes, and UUIDs referencing actual users, respectively.

This approach highlights one of the framework's limitations: without tailored generators, the data inputs can sometimes lack the necessary contextual relevance.

On the other hand, an important feature of this framework is its adaptability to changes in the graph schema. As the schema evolves, there is no need to update the code in the tests. The test cases are generated directly from the schema, allowing them to stay up-to-date without manual intervention. This flexibility becomes particularly significant if we aim to introduce more specific business logic into certain arbitrary generators for new queries.

Lastly, it is worth mentioning that the current framework tests only queries, not mutations. We chose this approach mainly for pragmatic reasons, given the potential issues of mutating data in rapid succession on our test servers. However, running queries was our initial objective, and we consider the extension of this framework to test mutations as the next step in our progression.

5 Results

Once the framework was built and the initial bugs worked out, the resulting system had several noteworthy features. First, we set up a scheduled job to run these tests twice a day, providing consistent and regular testing of the API. This job generates hundreds of input sets for each query, covering a wide range of possible requests.

On receiving responses, the system checks if the response corresponds to a known bug or an acceptable error — these are predefined in our fixtures. If the response does not fall into either of these categories, the system asserts that it receives a known GraphQL object defined in our schema.

The implementation of this framework led to the immediate discovery of a few dozen bugs that fell into a few different categories. First, there were plenty of edge cases discovered for extreme input values. The endpoint would throw an error for certain queries and input value combinations. Second, there were some queries that just were not being used and threw errors for all the inputs they were sent. Third, there were many queries throwing errors rather than returning structured error objects defined in the GraphQL schema.

Since the framework has been stabilized, the tests have been detecting on average one bug a month. These are usually associated with new operations or changes to backend services that our system interacts with, although in a few cases they have indeed been additional failures in handling extreme input values or failures in error handling logic.

6 Discussion

From the operation and results of this testing framework, we gleaned several insights about Care.com's GraphQL setup, and indeed, GraphQL operations in general. These insights are not exclusive to Care.com's graph but are broadly applicable to any GraphQL operation.

First, GraphQL query strings can be extremely lengthy, even when the maximum depth is limited to 10 levels. In our tests, some query strings exceeded 1500 lines.

Second, consistent error handling is a crucial component for the successful implementation of a testing framework like this. It allows the system to properly respond to and record issues that come up during the testing process.

Lastly, this framework proved invaluable in identifying unused operations within our API. By testing all operations and identifying those which were never used, we were able to queue up a considerable amount of work to remove dead code, improving the efficiency and maintainability of our system.

We continue to test with the framework. After we got through the initial batch of bugs when we built the framework, we continue to see benefits: early warning on operations that are introduced to the graph but not yet functional, inconsistency between the graph's representation of a value and a backend service. (The most common version of this is a backend service that enforces a value as required, but the graph does not.) They have flagged insufficient error handling, and operations that are unused and support backend services removed (essentially pointing to tech debt we are hanging onto).

It has also been enormously helpful that since the schema is something we retrieve and update every time we run the tests, and the test cases themselves are automatically generated, we do not have any test maintenance to do as far as the framework itself. The test maintenance we do have is handling known bugs in the graph, and further improving arbitrary generation.

Beyond that, the most interesting lesson from this effort was that it ended up being possible to automate a lot more than initially thought. We had visions of ourselves writing out endless gql statements, and we think it pays to stop and think: how is this system *structured*? What about it is special such that we can programmatically solve this testing challenge. GraphQL's graph representation can be powerful and efficient for its consumers, but it also opened up a space of automated solutions.

However, there certainly have been challenges in pursuing this test strategy as well.

First, our graph is not perfect and like nearly any software system, we carry several known bugs and improvements that we have not yet prioritized. Since the test framework is automatically generating its tests based on the schema alone, we have to maintain fixtures that list known error conditions (and thus don't fail the entire test run when we see them again), and one that lists operations we simply skip entirely; these are almost always defunct operations that are no longer used that just haven't been removed from the graph yet.

Second, we have had to find a balance between random and 'realistic' inputs when generating values.

Finally, we have not yet tackled testing mutations. The additional complexity of making thousands of data mutations per test run and managing the effects of that is something that we have not yet started. The framework would work fine, we would have to make a few changes to support mutations, but we still have some thinking to do on what effect making so many data changes would have on our test systems.

7 Future Directions

As we look to the future of this testing framework, we are looking at several different improvements and additions. The first one is expanding the framework to test mutations. Mutations tend to have more complex request objects since they are modifying data, but there is nothing in the framework that would prevent us from doing so. One hurdle is managing all the changing test data, but that is something we can work through. Another is input validation: while our requests generated would be technically valid from a schema perspective, there is additional business logic behind the mutations that are not represented in the schema. We would need to add additional handling for these.

This leads us to another thing, which is that the current set of input generators, the arbitraries, are relatively ‘dumb’: they do not have any sensitivity to the business context in which they appear. For each case, we would need to write-in in this sensitivity. If we do not do this, we can end up hitting simple form validation type errors before hitting the backend logic that we want to test more robustly. This tends to be especially acute with mutations, but it can occur with any operation.

For example, suppose we were testing addresses. Our backend systems may know that a city, state, zip code combination needs to match; it might throw a validation error if we try to submit an address that has a zip code known to be in California, but we submit a state of Oregon. If we end up just spamming the endpoint with various iterations of invalid combinations like this, we certainly get a fuzz test of sorts of the mutation, we end up stopped at the first layer of validation for most input combinations. It is certainly possible to write in constraints into the framework to enforce rules like this, but those have to be dealt with on a case-by-case basis. We do have such constrained arbitraries, but there is room for many more of them and it would likely get us deeper into the logic of the system we are testing.

Acknowledgements

Thank you to the Care.com engineering organization for supporting this work and discussing these ideas with me. I would particularly like to thank Emmanuel Pamintuan Jr. for some early discussion and encouraging me to pursue this project.

References

Karlsson, S., Čaušević, A., and Sundmark, D. 2020 “Automatic Property-based Testing of GraphQL APIs <https://arxiv.org/abs/2012.07380>, (submitted December 14, 2020).

Optimize Your Testing Performance by Using Cypress.io

Ryan Song

ryan.song@iterable.com

Abstract

Automation testing is one of the essential components in the software development life cycle, yet it can be very time consuming and resource demanding to maintain. Quality Engineers constantly face the need to trade off or prioritize among test speed, test coverage, and test stability, as achieving all three simultaneously is challenging. This article will discuss the approach using Cypress.io to improve test speed, test coverage and test stability at the same time. Additionally, it includes some techniques and processes that leverage Cypress to optimize end-of-end (E2E) tests.

Biography

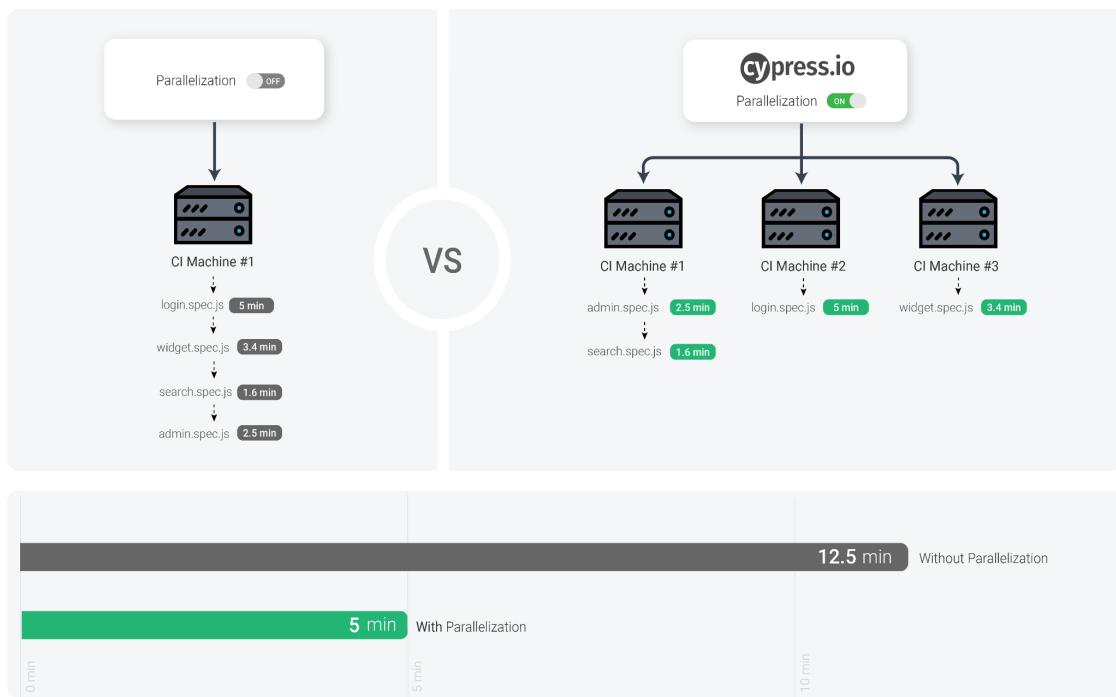
Ryan is a Staff Test Engineer at Iterable who has been working in the quality industry for 8 years. He has experience working on projects of varying sizes, such as startups, federal/defense and Fortune 50-level companies. He is passionate about Automation Testing and CI/CD process and specializes in system optimization and operation research. Ryan is located in Los Angeles and graduated from Texas A&M University with a degree in Industrial and System engineering.

Introduction

In automation testing, Quality Engineers often find it challenging to balance test speed, test coverage, and test stability with competing priorities. For example, an increase in test speed might lead to a decrease in test stability, and expanding test coverage could result in a corresponding decrease in test speed. Balancing these three aspects requires a significant amount of work experience and technical knowledge. This article will discuss the value of upholding these three pillars of automation testing and share techniques to achieve this.

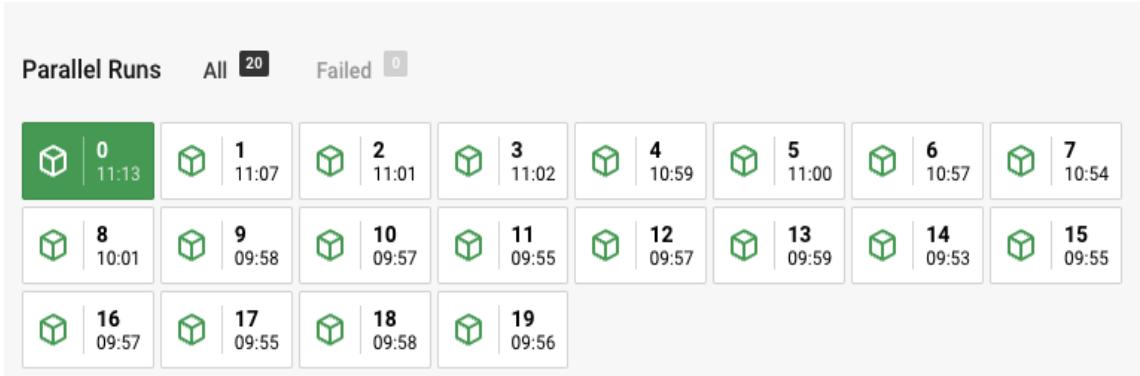
1. The importance of test speed in test automation

The speed of the test automation is one of the most important factors in agile principles, especially when companies implement continuous integration and continuous deployment/delivery (CI/CD). Running automation tests quickly enables developers to iterate faster, therefore they can promptly review test results and make necessary adjustments in a timely manner. One common issue many developers face is when test suites grow too large, resulting in significantly extended completion times for all the tests. Many companies are running all their tests during night time and only able to run one batch of tests per night. This is not ideal for teams that want to be more agile and efficient. Cypress solves this test speed problem by using the built-in feature to automatically split up test suites. This technique will empower developers to run tests in parallel and significantly reduce test run time.



Picture 1: Cypress official site for tests split

As you can see in the picture above, If all the tests are executed on a single machine, it will take about 12.5 mins to get the test results. However, when distributing tests across in three different virtual machines, the total testing times can be reduced to five mins.



Picture 2: CircleCI project page

Currently at Iterable, we run almost 300 Cypress E2E tests in 20 parallel machines within CircleCI (continuous integration tool), and each round test run can be finished in less than 15 mins. Many of our developers run many pull requests on a daily basis, frequently validating their alterations by executing E2E tests multiple times throughout the day. This strategy offers much faster feedback to the developers, a crucial aspect of our CI/CD pipeline, given the frequency of multiple deployments we conduct each day.

2. The importance of test coverage in test automation

Another important factor for software testing is test coverage. Without appropriate test coverage, many bugs will leak to production. Many companies are using percentage of code coverage as a baseline to track developer's code quality. For example, at Iterable, we use CodeClimate to check the current code test coverage. The problem with this approach is that it is very difficult to have 100% test coverage and challenging to update and maintain all the tests. There are two solutions to solve this issue. The first solution is to build E2E tests by using business scenarios to test the application. Many companies are using tools such as Segment to track customer usage and their QE team can utilize the data from those tools to design the E2E test scenarios. The table below shows the example for one team's feature usage.

The Feature % Click is calculated based on the (number of clicks/total number of clicks)* 100. The Cumulative % is calculated based on the (current Feature % Click + previous Feature % Click).

Number of Visitors	Name	Number of Clicks	Feature % Clicks	Cumulative %
3302	Action 1	95682	13.27%	13.27%
2768	Action 2	86663	12.02%	25.28%
3415	Action 3	80185	11.12%	36.40%
4753	Action 4	73524	10.19%	46.60%
1503	Action 5	46697	6.47%	53.07%
2782	Action 6	37813	5.24%	58.31%
2515	Action 7	29570	4.10%	62.41%
2606	Action 8	28099	3.90%	66.31%
2532	Action 9	25776	3.57%	69.88%
2199	Action 10	23522	3.26%	73.15%
1209	Action 11	22853	3.17%	76.31%
2199	Action 12	21720	3.01%	79.33%
1853	Action 13	11244	1.56%	80.88%
1839	Action 14	11116	1.54%	82.43%
1257	Action 15	9902	1.37%	83.80%
1116	Action 16	8741	1.21%	85.01%
1900	Action 17	8543	1.18%	86.20%
1110	Action 18	8251	1.14%	87.34%
1490	Action 19	8126	1.13%	88.47%
757	Action 20	6896	0.96%	89.42%
1415	Action 21	6579	0.91%	90.33%
1570	Action 22	5622	0.78%	91.11%
875	Action 23	5430	0.75%	91.87%
632	Action 24	4873	0.68%	92.54%
804	Action 25	4113	0.57%	93.11%
692	Action 26	3779	0.52%	93.64%
1114	Action 27	3657	0.51%	94.14%
825	Action 28	3215	0.45%	94.59%
899	Action 29	3104	0.43%	95.02%
1307	Action 30	2965	0.41%	95.43%
929	Action 31	2873	0.40%	95.83%
367	Action 32	2795	0.39%	96.22%
538	Action 33	2685	0.37%	96.59%
641	Action 34	2476	0.34%	96.93%
463	Action 35	2281	0.32%	97.25%
890	Action 36	2216	0.31%	97.56%
752	Action 37	2091	0.29%	97.85%
762	Action 38	2064	0.29%	98.13%

Picture 3: Mocked test data

As you can see in the chart above, if all actions in this chart are included in the Cypress E2E, then we can assume that **98.13%** of the business scenarios are covered. This strategy will optimize the Cypress test suite and reduce the redundant tests while maintaining high levels of test coverage.

```

cypress > e2e > testFolder > testCasesForTeamA.spec.js > describe('This is all your test cases for team A') callback
1  describe('This is all your test cases for team A', () => {
2    // Group your tests based on features
3    context('This is your test cases for feature one', () => {
4      // -- Start: Cypress Tests --
5      it('This is your test for action 1', () => {
6        // Write your test case here
7      });
8      it('This is your test for action 3', () => {
9        // Write your test case here
10     });
11     it('This is your test for action 5', () => {
12       // Write your test case here
13     });
14   });
15
16   // Group your tests based on features
17   context('This is your test cases for feature two', () => {
18     // -- Start: Cypress Tests --
19     it('This is your test for action 2', () => {
20       // Write your test case here
21     });
22     it('This is your test for action 4', () => {
23       // Write your test case here
24     });
25     it('This is your test for action 6', () => {
26       // Write your test case here
27     });
28   });
29 });
30

```

Picture 4: Example of Cypress test suite setup

The first solution is to group the test suites based on the business usage. Each team can have its own test folder and test files. The test scenarios can be broken down to feature leave and all the actions related to that feature can be grouped together based on business logic. This will provide a clear structure in the Cypress tests and easy to manage test coverages when a new feature is added to the system or existing feature is updated.

3. The importance of test stability in test automation

The third important factor for automation testing is the test stability. Many developers are facing issues of flaky tests failing on the pull request. Those flaky tests significantly impact the developer productivity and cost company time and money. There are many reasons that cause flaky tests, such as web page rendering too fast, race conditions when spins up servers, test setup depended on timezones, and etc. Cypress has many built in features that address some of those issues. There are few features or techniques that helped my team to make the test more stable. The first one is the *Test Retries* feature, which is able to retry the failed tests a few times before failing the test. The extra retry will improve the passing rates and save developer time and resources. This is a built-in feature and can be configured in the cypress.config.js file as shown below.

```
{
  "retries": {
    // Configure retry attempts for `cypress run`
    // Default is 0
  },
  "runMode": 2,
  // Configure retry attempts for `cypress open`
  // Default is 0
  "openMode": 0
}
}
```

Picture 6: Cypress retries feature

The second feature is the wait mechanism, Cypress has a built-in feature to wait for network calls to complete before continuing the next step. When running the tests, developers can intercept the network calls and let Cypress wait for a specific request to respond before executing the next test step. This feature significantly reduces the race condition that happens within test runs.

End-to-End Test	Component Test
	<pre>// Wait for the alias 'getAccount' to respond // without changing or stubbing its response cy.intercept('/accounts/*').as('getAccount') cy.visit('/accounts/123') cy.wait('@getAccount').then((interception) => { // we can now access the low level interception // that contains the request body, // response body, status, etc })</pre>

Picture 7: Cypress intercept requests and wait

The third technique is to utilize Cypress API commands to do data preparation test steps. Setting up test data in a browser and running through all the preparation steps will increase the risk of flaky tests. Moreover, test run time will increase significantly when there is a substantial amount of test data that needs to be set up. For example, the picture below shows the significant time difference between running a test through a web browser compared to using API calls to configure the test data. Therefore, directly making API calls to set up test data is a better approach. API calls are usually more stable and fast to complete.

Test steps	API	Browser
Login	3s	10s
Turn on the feature flags	5s	20s

Picture 8: Test preparation steps between API call vs browser

```
cy.request({
  method: 'POST',
  url: '/login_with_form', // baseUrl is prepend to URL
  form: true, // indicates the body should be form urlencoded
  body: {
    username: 'jane.lane',
    password: 'password123',
  },
})

// to prove we have a session
cy.getCookie('cypress-session-cookie').should('exist')
```

Picture 9: Login via API call

```
cy.request({
  method: 'PUT',
  url: `/featureFlags/${featureFlag}`,
  form: true,
  headers: {
    'X-XSRF-TOKEN': XXSRFTOKEN,
  },
  body: {
    ...param,
  },
});
```

Picture 9: Turning on/off feature flags via API call

Conclusion

This paper highlighted the importance of test speed, test coverage and test stability. There is no question that using Cypress.io with the correct strategy will help your organization to improve all three areas simultaneously. The results shown in this paper should encourage you and your team to explore and adopt Cypress.io and improve your current test framework and strategies. Utility leaving to a more robust efficient testing program. As we continue to enhance our understanding of proactive automation testing, we can drive innovation for new techniques and cultivate a culture of continuous improvement.

References

Cypress official documentation:

<https://docs.cypress.io/guides/cloud/smart-orchestration/parallelization>

<https://docs.cypress.io/api/commands/wait>

<https://docs.cypress.io/api/commands/request>

Automation - We're Doing it Wrong

Melissa Tondi

melissa.tondi@gmail.com

Abstract

There are many mis-conceptions and perceptions around automation as it pertains to software testing, so in this talk, we will discuss those and set the record straight on what automation is and, more importantly, what it is NOT.

The term automation as it pertains to software testing has been a driving force in defining the software testing industry. For many years, we've used it as a catch-all to determine whether a tester, testing team, or IT organization is successful. In this talk, we will discuss the five misconceptions that are pervasive within companies - including using a percentage or number of test cases to define success and specifying a title/role for those who automate and matrixing that role across teams versus embedding it within the project they are supporting. Melissa will discuss re-booting our current thinking of automation and show tactics to address the five misconceptions that have contributed to what many of us would consider one of the wedges that divide the testing industry. In addition, she will share practical and proven approaches you can take back with you to show immediate and valuable results.

Biography

Melissa Tondi has spent most of her career working within software testing teams. She is the Director of Quality Engineering at Guild Education and a Principal Consultant at Disrupt Testing, where she assists companies to continuously improve the pursuit of quality software—from design to delivery and everything in between. In her software test and quality engineering careers, Melissa has focused on building and organizing teams around three major tenets—efficiency, innovation, and culture – and uses the Greatest Common Denominator (GCD) approach for determining ways in which team members can assess, implement and report on day-to-day activities so the gap between need and value is as small as possible.

1 Introduction

In this talk, we introduce the four approaches that we introduced while building test automation practices that may have made sense then but may have gotten stale now. These “wrongs” serve as a reminder that innovation should be at the forefront of our industry and even best-laid plans and ideas that make it in to our QE playbook should be re-visited to ensure they are applicable, align with the industry, and are collaboratively discussed within our agile teams in order to support the highest efficiency and productivity on our quest to deliver software quicker and with high quality. Here are the top “wrongs” I’ve gathered in hopes that our community can continue to continuously improve and broaden our information sharing.

2 Wrong #1 Monolithic Suites of Tests

When we invest in on-boarding activities for new hires, we are making a statement that it's important to provide consistent information at the onset of a team member's career with the company. Far too often, we don't invest that same thought process to the rest of the team or re-visit processes put in place months (or, sometimes, years) ago to ensure they are still valid and supportive of what actually takes place.

We implemented a playbook titled Definition of Done. The playbook allows us to level-set all the services our team provides and present those to our agile teams.

2.1 Definition of Done Playbook Sample Sections

- What QA Does and How
 - Test Planning
 - Test Management
 - Test Execution both scripted and un-scripted
 - Reporting
- QA's Expectations from Development, Design, Product Management, and Scrum Master

3 Wrong #2 Automate Everything

Fundamentally, we all know that not all tests should or can be automated, but I'm still surprised when I hear teams are being measured on the number or percentage of tests automated versus weighing the overall value the automation is bringing to the team. A better way to approach this is to have an intuitive selection process to quickly determine when something should be considered for automation. Notice how I used the word "considered" instead of giving instruction to automate a test. When you are held to a meaningless metric like "everything" your creative license is essentially removed and how fun is it to be told what to do rather than doing what you know is more valuable?

3.1 The Automated of Automatable (AofA) Metric

Because we did away with the “traditional percentage of test cases that are automated” metric that tends to be an inaccurate measure of software quality, we focused deeper on a metric that showed value, not only to the QA team, but to the agile project team as well. The AofA is determined by a selection process that succinctly shows the percentage of automated valuable tests given the following sample criteria:

- Its severity to the business in terms of:
 - Revenue Loss
 - Security Vulnerability
 - Customer Loss

- Its importance to customer happiness
- Compliance

Given these, during refinement sessions, we indicate which user story and/or acceptance criteria meets any of the above criteria and consider it for automation. Once the initial sorting happens, we size the work according to the project team's norms and plan the work accordingly.

For reporting purposes (either daily or recurring throughout the sprint), we can report on the number of items that met our criteria and could be automated versus what was actually automated. We increased our percentages of AofA to the mid 90th percentile to 100% in most cases.

Using the guidelines in section 5, we worked in serial order of priorities. For example, if any test in our Priority 1 suite was not green, we swarmed to fix and did not expand any other automation until all higher prioritized automation was green.

4 Wrong #3 A Siloed and/or Non-Embedded Team of Automation Engineers

Many times when I've provided SDLC assessments for companies, I observe their team dynamics and any agile ceremonies that are followed. When I chat with QA and their management, there is a pattern of exclusion during key meetings where Development may have received more information than what was represented in the user story or its acceptance criteria. This usually results in QA not being able to size the work correctly or to assume behavior on "light" acceptance criteria. This then causes more triage by Product for bugs reported, mis-interpretation of intended behavior by QA and general tension between QA and other team members because they were not privy to adhoc or informal conversations that have taken place without them.

One option we've used to counteract that is smaller working refinement sessions. We do this by setting outcomes for refinement and have Product provide a list of prioritized stories ready to be refined at least 48 hours before the refinement session. We provide expectations for a story ready to be refined and have any one with responsibility on the story attend and contribute by providing a high-level summary of how they will approach the work. If any of those items are not complete during refinement, the story is not refined, and, therefore, cannot and should not be planned and committed to for an upcoming sprint or for Kanban, work should not be started. By holding smaller working sessions with those who have tasks, this creates a much more efficient, collaborative session where contextual information is being shared and heard, and implicit information becomes explicit and used for higher quality activities. It also ensures that each person and their practice are equally collaborative and represented for true sizing and estimates.

5 Wrong #4 Automation Deliverables are Owned, Executed and Maintained by Automation Engineers

In addition to the above, we know the Development team does their own testing, but sometimes we don't know what that is. By defining who does what and creating a playbook of each agile pillar's consistent practices at the individual contributor level, it removes ambiguity and take the guesswork out of what actually happens when that card or ticket moves to the "ready for QA" column. We advocate a collaborative discussion and refinement of work across the agile team to ensure test activities performed by QA are not only not redundant, but that are highly valuable and done at the right time during that phase.

6 Right #1 Multiple Runs for Multiple Dones

A while ago, we changed the name and mission of QA in our organization. We determined that Quality Engineering and how we defined it most matched what we were currently doing, where we eventually wanted to do, and, perhaps most importantly, where we knew our value would be emphasized the most within the organization. When given the opportunity, sometimes it's good to disrupt with a name change when a company or its leadership has misperceptions of what QA's role is versus what it should be.

This quote "Influence the Building of the Software before the Software is Built" drives our daily activities. We do this by:

- Balancing technical acumen with user advocacy and ensure we emphasize both.
- Using context-driven techniques. Given the information we have, we determine if it's enough and if not, we find more by:
 - More collaboration within Development, Product, other QE teams, Customer Support, and Customers.
 - Reaching out to the community. We both consume from and contribute to the community whenever we can.

In our QE playbook, we highlight the following traits and characteristics of the "Engineer" in Quality Engineer.

6.1 Breakdown of Engineering Traits

- Define success, outcome and measurements. Using context-driven approaches to gather the right data and information. Usually, the easiest information to gather is the that which is explicit (user stories, requirements, acceptance criteria, etc.). QE adds value when we uncover valuable information that is implicit at first.
- Design a comprehensive strategy.
- Build the solution. Write the tests, write the charters/sessions for unscripted testing.
- Execute the solution.
- Measure the results and rove them!
- Report the outcomes throughout.

By re-visiting and addressing any or all of these "do nots" you'll be operating at higher efficiency and productivity and can continue to innovate and continuously improve.

7 Right #2 Know Your Landscape

- Consider Unit/Integration tests and take action by designing a strategy (documented or not) that includes all testing being done across the team. One of the most valuable traits QE has is our ability to gather information across practices. We should use that trait to also build a comprehensive strategy that not only incorporates the tests we create and eventually execute, but considers the tests and testing that are also executed across the team
- Start with Smoke tests. After inventorying all the tests and testing happening across the team, it's a great time to also build in your test automation strategy. I recommend starting with smoke tests, but, first, let's make sure it's defined and understood across the team. I like to define Smoke tests

by following the ISTQB definition as closely as possible and also adding a bit more refinement to it, so for this paper, we will define Smoke tests as following the CRUD (Create, Read, Update Delete) actions. Those should be considered the most basic functions and should also resonate across the team.

- Next, tie it to Definition of Done/Shippable. Now that we have a more manageable starting point for test automation, we want to use our Agile methodology to apply it to “done”. In this case, I recommend that we don’t consider the ticket/work done without

8 Right #3 Using the “Tests as a Service” Approach

Organize tests in the same way your software engineers/developers are. This shifts the dynamic and responsibility of testing to the team with the QE being in a player/coach role. The more accessible we make our tests (automated or not), the easier the adoption of their states will be. Some other characteristics to consider in the TAAS (Tests as a Service) approach are:

- Concentrated in specific areas of the code
- Centralized (meaning anyone on the team can run them)
- Apply it to Regression testing
- Tests as a Service allows SWEs to validate on their own

9 Right #4 Leave No Trace

The Leave no Trace approach came about as it applies to camping. The idea is to leave a campsite or similar in at least the same condition as it was when you arrived. And, ideally, to leave it in a better state than it was when you found it. As we engage with the TAAS approach, this approach also applies! The idea here is to create, implement, execute and report on testing activities and offboard to the point where the team has everything they need to expand the tests and maintain them rather than have one individual do so. This approach is also very relevant to companies who have adopted an SDET or hybrid centralized testing approach. It also supports lateral movement across teams when specialized skill sets (like test automation) are in demand or in a deficit. Some characteristics of the LNT (Leave no Trace) approach are:

- Leave things in a state as if you weren’t, or more importantly, can’t be there
- Onboard: assess, confirm, build, and deliver
- Offboard: backlog, maintain, steady state, and leave

Follow Prime Time guidelines. I like to call the standards and consistent practice that should be applied and adhered to by anyone who is building and delivering tests as Prime Time. Things to consider when defining Prime Time for your organization are:

- Standards and coding practices are aligned with Dev’s
- Agreed to by your committee of technical QEs

Next steps and how to apply all of these wrongs and rights:

Where do some of these “wrongs” apply to you? And more importantly, where can you make them right?

- Start small – focus on a Smoke test first and gain traction. Once you have a good adoption and review rate amongst the team, then expand automation.
- Tie it to Definition of Done/Shippable. This is a great collaborative item to bridge between Product and Engineering.
- Think like a team member – even if you’re not embedded
- Your end goal should be to provide self-service tests for anyone on the team to use- that is ultimately maintained by the team so you can focus on innovation driven by change and right the wrongs!

Amping Your Success by Leading with Quality: Why "Shifting Left" Isn't Enough

Heather M. Wilcox

Heather.wilcox@hmhco.com

Abstract

A common topic for discussion in recent years amongst the quality community is the “Shift Left on Quality”. This is meant to express the idea that quality engineering should start as early in the development process as possible - within code, unit tests, automation, etc. But, is simply shifting left enough? What if organizations began to “Lead with Quality”? That doesn’t just mean putting QA engineers in the room early (although it is usually a smart move). Leading with Quality is a bigger idea than that. The concept is all about making smart decisions from THE BEGINNING of the planning process for a product. Regardless of whether you’re Agile, Waterfall, or some other SDLC paradigm, I contend that it is vital to have the right people in the room representing the right parts of your organization so that the right decisions can be made to enable the construction of a quality product. In my experience, many problems can be tied to a failure to lead with quality. For example, having non-technical people design a technical solution or perhaps choose the tools that the developers will utilize can lead to an array of problems during the implementation phase of a project. This paper will leverage my 25+ years of experience in the software industry as well as documented examples to delve into what it means to “Lead with Quality” and how the application of the concept can save time, money, and effort and will result in a higher quality product.

Biography

After leaving a potential career in Anthropology, Heather has spent over 25 years working and learning in the software industry, choosing to focus primarily on start-up and small companies. As a result, she has had a broad range of job descriptions which include, but are not limited to: Technical Support Engineer, IS Manager, Technical Writer, QA Engineer, QA Manager, and Configuration Management Engineer. More recently, Heather has moved into a more permanent relationship with a mid-sized educational company where she's been for the last 12 years. This has given Heather a wide range of experiences to draw from in her current role as a Quality and Reliability Manager. In her spare time, Heather enjoys fiber arts, equestrian sports, and training donkeys.

Copyright Heather M. Wilcox 2023

1 Introduction

When I imagined the concept of “Leading with Quality”, I thought it was kind of a “no brainer”. It felt so obvious that I wasn’t sure it was even worth discussing. However, when the idea was pitched to my peers, I got very similar responses from everyone. Consistently, the feedback was something to the effect of, “This is a great idea! It seems obvious but no company I’ve worked with has done it, so you need to talk about it!” As part of those conversations, I realized that, after seven companies (working on widely varied technologies), I hadn’t experienced it either. Many of those organizations clearly cared about quality and put resources into building better products, but none of them actually “Led with Quality” as described in this paper. That revelation, along with the encouragement from my co-workers directly inspired this work.

In my experience, the origin of new piece of code is nearly always “A technical problem” of one form or another. However, I’ve observed numerous software organizations either ignore or not realize that the problem they are solving is fundamentally a technical one. This failure happens at the most critical point in the software development process – the beginning. The direct result of this lack of insight is that the people most qualified to solve the problem - technical people - get left out of the initial conversations and that is where Quality Failures begin. (Cherry, 2019)

I’ve witnessed versions of this scenario play out many times in my career and, over the years, have heard similar stories from colleagues:

1. A need is identified: “We need a widget that stores data from a web page!”
2. Marketing has done all the research to prove that the addition of this widget would be a valuable feature for the current product line.
3. Sales swears up and down that people have been asking for this functionality for years and they’ll be able to sell it to every customer the company has ever had and, what’s more, it will attract new clients in droves.
4. It is the PERFECT idea and all that is needed is a Dev Team to build it!

Marketing realizes the value of this widget and doesn’t want development to over-think it – nor do they want to waste engineering time since developers are expensive, so they assemble a typical “Discovery Team” ([Muzyka, 2021](#)) to design the perfect piece of software – right down to the technology that will be used to build it. And then they deliver this beautifully designed concept to the development team complete with licenses to “The Perfect Tool”. Marketing believes that there’s no reason in the world why Dev can’t use their Agile methodology to have a shipping Minimum Viable Product (MVP) in 3 months.

Clearly this is a contrived example; nevertheless, as previously mentioned, I’ve witnessed variations of this in just about every company I’ve worked for, and I believe it happens often. In 25+ years of working in the technology industry – at least 20 of those in Quality, I can only think of a very few times – certainly less than 10, where the “right people” got in the room early and were able to have a positive effect on the final product. On the other hand, I can think of many examples where I or someone I knew, discovered a significant problem very late in the development process, that could have been avoided by bringing the right technical person in during the initial discussions. The failure to engage the right experts from the beginning led directly to a compromise in Quality before even a single line of code had been written.

2 The Beginning: Why We Don't Put the Right People in the Room and Why Those Reasons are Wrong

There are many reasons why companies don't put the right people in the room from the start. I know I've said some of them myself and heard versions of all of the following: "Engineers are expensive", "We don't want to disturb Bob because he's working on important stuff", "If you let an Engineer design software, you'll end up with a shiny UI with a million buttons that our customers won't be able to use", "QA doesn't need to be there until the code is finished", "Support doesn't know enough about software development to be helpful."

However, when you dig into these statements a bit, they fall apart quickly.

"Engineers are expensive." It is common belief that engineers are paid more than their Marketing equivalent. This is often used as a justification for waiting until later in the planning process to bring Senior Developers in to consult. The argument is that, if an engineer isn't working on actual software development tasks, their expensive time is wasted. The truth is that the "expensive engineer" is a fallacy (which will be addressed later in this paper). However, regardless of the cost of engineering time, bringing in the right technical person to consult on a project may save thousands of dollars (or perhaps millions, depending on the scope of the project). An engineer will understand the available technologies and their capabilities as well as the possibilities and limitations of existing infrastructure and software. They can make recommendations that will help the Marketing team set up the project in the best possible way and avoid early errors that become expensive problems later.

"We already have an architect on the team – we don't need an engineer." This argument stems from the notion of avoiding a duplication of effort - why have two individuals performing the same role in the room? Yet, this doesn't accurately reflect the situation. Solution Architects often view and understand software from a very high level. (Coursera, 2023) I have witnessed Architects that were very "hands off" and did not understand the nuances of a particular system or a piece of code within it that could complicate the proposed solution. Conversely, an engineer who has created code for the affected product or system will have knowledge of granular technical details that an Architect might overlook, but that may be vital to the success of the new project. This specific knowledge may be helpful in providing valuable insight into whether a proposed solution is reasonable, efficient, and functional. The software engineer may also be able to leverage their deeper knowledge of the existing code to suggest a more appropriate option.

"We don't want to disturb Bob because he's working on important stuff." In this case, the perceived problem is that engineers should be focused on project related work as close to 100% of the time as possible. Every moment of an engineer's day that is not spent coding or working on code is valuable time wasted, since there is always more to do than there are developer resources available to do the work. It's true that there are rarely enough engineers in a company and the best ones are in demand every minute of every day. However, as with the issue of expense, the investment of a few hours of their time at the very beginning, may save hundreds of hours of rework by multiple engineers later in the project. Isn't it better to spend valuable engineer time up front to prevent the waste of, potentially, much more time later in the project?

"You don't want to bring Engineers in too early in the process because, if you let an engineer design software, you'll end up with a shiny UI with a million buttons that our customers won't be able to use." This breaks down to an argument that Engineers are not UI designers so they shouldn't be involved in anything that might be UI design. My developer friends will probably kill me for this, but this statement is likely closer to the truth than most of us would care to admit. However, developers aren't being asked to design the UI, they're being asked to help design a software solution. A good solution and a good UI are not the same thing. Clearly, the User Interface drives the software, but the underlying technology is the core of the product. Good developers are more than capable of designing a technical solution that your UI experts can create a beautiful interface for.

"QA doesn't need to be there until the code is finished." This statement is a leftover from the days when most companies used a Waterfall lifecycle and testing, or quality assurance work was always the last step before the public saw code. The Shift Left movement has proven that sentiment to be incorrect. (Testim, 2021) QA absolutely needs to be involved from the beginning (this is shifting WAY left). Nobody will understand how to build in quality better than a Quality Engineer. They'll help identify areas where the proposed solution might be lacking – where it might not work for all customer use cases or where there might be a confusing piece of design. They will advocate for your customers in a way that Marketing cannot because QA understands both the technology and the practical usage models. (Harris, 2015)

The final common scenario: "Support doesn't know enough about software development to be helpful." This issue is caused by a lack of appreciation for the kind of knowledge that Customer Support possesses. Regardless of a support engineer's software development expertise, they are the ones who talk to users every day. They know how the software is applied in the real world and they can speak for the user in a way that even QA cannot. The support representative can confirm whether a proposed usage model makes sense from the user's perspective. They can also point out UI functionality or features that might result in a confusion and increased customer support calls, or that may not have enough value to be worth implementing.

3 Who the “Wrong People” are and Why They are a Problem

The design portion of a project is often called "The Discovery Phase", which is how it will be referenced for the purposes of this paper. (Khalimonchuk, 2022) This phase of a project is driven by a Discovery Team that normally includes a Business Analyst, UX Designer, Solution Architect, and a Delivery Manager. (Muzyka, 2019) I've also seen Product Managers (PMs), Technical Product Managers (TPMs) and Project Managers involved in the conversation. Marketing and/or the Business Analyst represent the needs of the customer. Either Technical Marketing or the Business Analyst understands the technology at a high level as well as Competitors' offerings and can describe the strategy around how the new product will elevate the company's stake in the marketplace. The Solution Architect has knowledge of the current configuration at some (again) higher level and understands the technical possibilities. The UI team knows how to design the product to be functional for the customers. The Delivery Manager understands the schedule and knows how to drive the project team to a timely conclusion. From many perspectives, this is the "dream team" for solution design. In theory, the team contains people that understand the customers and the business as well as folks that have enough general technical knowledge to cover for the actual Developers and QA that are out in the shop doing the important work of creating software. Upper management every reason to expect that the Discovery team will be successful in assembling a quality project that will be practical to implement and will meet the needs of the customer.

Once formed, the Discovery Dream Team designs a solution that they feel makes the most sense for the customer and for the company. When they've assembled all the pieces of the project, the team formally documents the plan and delivers it to the development team complete with the tools that will be used to build the new widget.

As previously mentioned, it is not uncommon for the Discovery team to select a technology or a tool for the development team to use as part of the design process. (Khalimonchuk, 2022) This selection is an early chance for the loss of Quality.

Technology decisions made by the Discovery Team can be based on any number of reasons including:

- The availability of a tool or technology the company already owns or uses.
- Information gleaned from websites. (Puzhevich, 2022)
- Financial concerns from the finance team, who might be averse to allocating funds for new technology, potentially pressuring a team into opting for free or Open Source tools.
- Pressure from a manager associated with the project. (I once had a manager demand that I re-code a fully functional build system in JAVA because he liked the language.) (Cherry, 2019)

- A salesperson from another company promises that their product will work perfectly for the project.
- Recommendation of the Solution Architect.

However, the team that will use the technology or tool may not be involved in the selection process. This can happen for several reasons. Often the Solution Architect represents the development team in tool decisions as that person usually has a higher-level understanding of the problem and the technical skills to understand what is and is not technically possible. Other possible reasons are that the software team may not yet exist, they may be busy elsewhere, or the Discovery team may not want to disturb their work. Irrespective of the rationale, the omission of the software team from the decision-making process can lead to a loss of quality, resulting in an unsuitable or suboptimal selection.

To return to the fictionalized example used in the introduction, the Discovery Team has designed a solution, purchased a tool, and handed it off to the development team. It may seem unusual that the discovery team handled the tool selection and organized the purchase, but I've witnessed this exact scenario, so I cannot believe that it hasn't happened elsewhere. It also appears to be recommended by some descriptions of the Discovery Process. (Khalimonchuk, 2022)

When the development team begins the project, the first problem they encounter is the tool that was picked. The technology was selected based on price and the assurance of the salespeople that it would absolutely work for the proposed software. Money was spent and a commitment was made. However, because developers weren't involved in the decision, there was never Proof of Concept (POC) or other deeper research done to verify the tool was the correct solution. The Architect should have done some of that work but decided that further technical discovery wasn't necessary because the software looked like a good fit, so that step was skipped. As a result, there was never a comparison of the product to POCs done with other solutions. (In an alternate version of this scenario, which I have witnessed, the developers were involved, but the Discovery Team disregarded the development team's concerns about the preferred tool.) It turns out that the purchased product doesn't quite do what is needed. But the money's been spent and there was a lot of set up required to get the tool functional. To ensure success, a third-party consulting company – "The Experts" were brought in. The decision from upper management is that enough money has been spent on tools and consultants that success should be guaranteed – there is no turning back. So, the Development team is forced to figure out how to kludge together the widget using an inappropriate tool.

The next problem is that solution has been designed as a new stand-alone tool. Had the team brought the engineering staff in on the decision, the developers would have explained that they already had something similar to the new design partially implemented as part of some work the team did in an innovation sprint 3 months ago. Or perhaps the existing software has a lot of the requirements already in place and an appropriate solution could be created in just a couple of months by leveraging the existing code. In either case, the development team is now duplicating previous efforts which is wasted time.

When the quality team looks at the solution, they are immediately frustrated. Even though it meets the needs of the market, the new design is inappropriate for the skill level of customer that will likely be using it. The Architect may have understood the technical problem, but they didn't enough knowledge about the actual user of the widget. For this product, the folks that are responsible for purchasing (customers) are much different than that the actual users of the product. The Quality Team has been aware of the problem for years since they work closely with support. As a result, the solution design, although appropriate for the people that buy the software, is completely wrong for the people who use the product. Although this seems like an unusual problem, it is not uncommon. Designated "buyers" within an organization will purchase products based on a perceived need or a set of requirements that may not accurately represent the needs of the people that will use the software. (Bird, 2019)

Unfortunately, even after all this new input and information, the plan has already been committed. Technology has been purchased, the design is in place, and a whole chain of promises have been made. (Based on a dev time guestimate from the Architect and from Engineering Managers, a delivery window is promised to Marketing. Marketing then makes promises to Sales and starts building collateral around a specific release window. Salespeople excitedly "hit the road", pitching the new widget to partners and

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

PNSQC.ORG

Page 5

promising delivery during the estimated time window.) The only option left is for the team to suck it up and deliver. And then, in the best of situations, in 6 months or a year the development team can justify why a complete redesign is necessary – only this time, it will be a better design using a better tool that will allow it to better serve the users for which it was intended. Congratulations – the team just got to build the same product twice, for twice the money. And that's the best case scenario.

The worst case is that management decides that they're already too far in on the solution – too much time and money has already been spent. Because “sunk costs” are too high (Wikipedia 2023), the dev team must continue to try to make an inappropriate, under-performant solution work for the next several years until the software is discontinued or enough time has passed to justify a re-write. In the meantime, the product is the source of never-ending support calls, and costs a fortune to maintain because the solution is delicate and difficult to work on.

I've had the unfortunate privilege of watching both situations play out and can say that they are deeply frustrating – especially for the development team that must do work that they know is throwaway for all the wrong reasons. The situation is even worse for a team that knows that they're going to have to continue to struggle to support something when they believe that there is probably a better and easier solution.

4 Who Are the Right People?

Occam's razor: **a scientific and philosophical rule that entities should not be multiplied unnecessarily** which is interpreted as requiring that the simplest of competing theories be preferred to the more complex or that explanations of unknown phenomena be sought first in terms of known quantities. (Merriam-Webster, 2023)

In other words: The simplest answer is usually the right one.

In this case, it means:

- Find your domain experts.
- Ask them for guidance.
- Listen to what they say / Do what they tell you.

Let's re-run the previously described scenario, only this time the right people get brought into the room:

A need is identified: “We need a widget that stores data from a web page!” Marketing has done all the research to prove that the addition of this widget would be a valuable feature for the current product line. Sales swears up and down that people have been asking for this thing for years and they'll be able to sell it to every customer the company has ever had and, what's more, it will bring in new clients in droves. It is the PERFECT idea, and all that is needed is a Dev Team to build it!

A Discovery Team is created to research and design the new product. They bring in a Development Lead, a QA Lead, a Support Lead, as well as a Technical Product manager, UX designer, a Product Manager, and an Architect.

The Marketing folks pitch the concept to the Discovery team. The dev lead makes some suggestions about two or three tools that they might want to use to build POCs to help make a technology choice. QA immediately points out some of the problems that have been found in similar products the company has built in the past and makes suggestions for things to avoid. Support agrees with QA and, additionally, makes recommendations for features that will really help customers use the product more efficiently. The Dev lead spends 2-4 weeks building POCs and determining the best tool for the job as well as working with the Architect and the rest of the team to ensure that the design meets customer requirements. Marketing works with Sales and Support to determine the MVP requirements. At the end of the

“discovery period” the skeleton for an MVP is delivered to the development team along with an appropriate tool set. For the sake of argument, let’s say the tools were “WidgetBuilder 4.0” and a specialty plugin that allows WidgetBuilder to work seamlessly with React, which the developers are already using for their other web apps. The team is then provided the leeway to develop a full solution in a way that makes the most sense to them while still creating the product that the customer needs.

When the MVP software is delivered, it works correctly, doesn’t contain kludges, and doesn’t require a re-write in 6-12 months. Instead, the team can take feedback from the business and incorporate it into future product versions along with new features.

In the real world, we don’t know which product took longer to develop. For purposes of comparison, let’s say that both took the exact same amount of time – 3 months. The difference is that, at the end of 3 months, one team had a product that was minimally functional but needed to be re-written almost immediately and may or may not have been optimally performant. Although the other team also had a product that was minimally functional, it didn’t need to be re-worked. Instead, the team and widget were ready to move forward with new features. There may have been a little extra money spent on the front end of the project by getting technical folks involved early, but a LOT of money was saved at the end of the project because there was no need for a costly re-write.

Even though the scenarios that have been described here were clearly manufactured to prove a point, both are based in real experiences I have had or have been directly associated with. I have felt the pain of working on a project I knew was doomed from the beginning because the right people weren’t in the room during the first conversations where key decisions were made. Seemingly simple concerns like where a particular kind of data will be stored in the database or what database to use can have widespread and devastating consequences for developers and testers later on. I can say with authority that it is (and was) incredibly demoralizing to work on a project knowing that there was an easier or better way available if anyone had asked and/or listened to the right people.

From the perspective of the company, although a demoralized team is certainly a problem, the bigger issue with this type of scenario is the monetary loss. I believe I can safely claim that Shareholders and Board Members want to see a profit. Usually, they want to that profit to be as large as possible. Having the wrong people in the room can cost a lot of money.

5 The Cost of Putting the Wrong People in The Room

For the sake of argument, let’s put some numbers around the two scenarios that I’ve described. The assumption that I’ve always made (and have heard others make many times) is that the average Senior Engineer is more expensive than the average Marketing person who often ends up making the critical decisions for a project. Interestingly, a little bit of research reveals that assumption to be untrue or, at least, not necessarily true. The table below contains some salary data courtesy of Glassdoor.com. I used the salary tool at <https://www.glassdoor.com/Salaries/index.htm> with the title listed in that cell and the location of “Portland, OR” to find the average salary for that job. (NOTE: Glass Door is continually adding to its salary database so new research may not yield exactly the same results.) Additionally, a quick consult of Google shows that, on average, there are 2080 working hours in any given year. (Zoe, 2023) That information was used to calculate an hourly wage for each job title for ease of comparison. (Yearly Salary divided by working hours.)

Title	Yearly Salary	Hourly Pay
Solution Architect	220K	105.77
Technical Product Manager	164k	78.85
Sr. Software Engineer	149k	71.63
Product Manager	130k	62.5
Sr. QA Engineer	134K	64.42
Delivery Manager	124k	59.62
Software Sales Rep	120K	57.69
Software Engineer	113K	54.33
Senior Technical Support Engineer	106K	50.96
Software QA Engineer	96K	46.15
UX Designer	96K	46.15
Business Analyst	90k	43.35

The data here is interesting. It turns out that that the old trope about the engineer being the most expensive person in the room isn't always accurate. Realistically, everyone in the room is expensive and the more time you spend with the wrong people in the room, the more money you lose. The exact amount of money wasted is a little bit academic since the hourly difference between just about anyone is minimal. With the notable exception of the Solution Architect, who is exceptionally well paid and, on the lower end of the scale, the UX Designer, Business Analyst, QA Engineer, and Support Engineer - and even they aren't inexpensive.

That brings us back to the question at hand – what is the cost when you don't get all the right people in the room?

Again, because I'm spit-balling a bit, we'll say that the "wrong" team that's designing this new project is composed of the classic representatives of a Discovery Team. That would include at least a Solution Architect, Delivery Manager, UX Designer, and Business Analyst. We will leave the Sr. Engineer, the QA and the Tech Support folks out because they're really busy and Engineers are always expensive.

Next, let's say that the project is laid out over the course of 10, 1-hour meetings. We'll also say that everyone on the team did an extra 20 hours of research on the concept. Applying basic math, we will see that getting this project put together without all the right people cost us \$7646.68

Because you should always show your work, here are the numbers for the people that were included:

Title	Hourly Pay	x30 hours
Solution Architect	105.77	3173.08
Delivery Manager	59.62	1788.46
UX Designer	46.15	1384.50
Business Analyst	43.35	1300.64
Total cost per hour Included Team	330.29	7646.68

And here's the math for the folks that were excluded:

Title	Hourly Pay	x30 hours
Sr. Software Engineer	71.63	2148.90
Sr. QA Engineer	64.42	1932.60
Senior Technical Support Engineer	50.96	1528.80
Total cost per hour Excluded Team	189.01	5610.30

By excluding the Sr. Software Engineer, the Sr. QA, and the Senior Tech Support Engineer, \$5610.30 was saved. But was it really?

The \$7646.68 team designed the solution, complete with the tools to do the job and then handed it over to the development team to implement. The development team looked at the work and said that they felt like they could have an MVP ready in 3 months or, 520 hours times the number of people on the team. A reasonable approximation of reality would be that, even though the team worked on this for 3 months, 3 weeks of that time was spent in meetings and other activities. The more realistic number of hours is probably closer to 400. Using a standard "Two Pizza" team size (Hern, 2018), a reasonable group composition would look something like this:

Title	QTY	Hourly Pay	X400 hours
Sr. Software Engineer	1	71.63	28,652.00
Sr. QA Engineer	1	64.42	25,768.00
Software Engineer	4	54.33	86,928.00
QA Engineer	1	46.15	18,460.00
Totals	7	236.53	159,808.00

Another fair assumption for this exercise is that the Product Owner, UX, and other people are outside of the team and are not dedicated full time to the project, so they won't be included as part of the cost.

In this scenario, you spent \$159,808.00 on the engineers to build and test this widget – assuming everything went according to plan. However, as laid out in section 3, it's likely that things didn't go right. There are multiple potential failure scenarios. Here are some of the more common ones that should be considered:

1. The tools didn't work as advertised and caused the team to need extra time to complete the MVP.
2. The tools weren't appropriate for the job and, although an MVP was shipped, the team ended up rebuilding the widget with a different tool set after a few months of failures in the field.
3. The solution, as designed and provided to the development team, wasn't technically appropriate and required extra work to implement.
4. All of the above scenarios.

For scenario 1, let's be kind and say that the project only slipped 2 weeks or 80 hours since the engineers were able to figure out how to manipulate the tools to do what was needed. That's an extra \$31,961.60 that was spent. If the excluded engineers had been in on the design sessions, it's likely they would have caught the tool problem by taking the time to do POCs. Your \$5610.30 savings turns into a \$26,351.30 loss.

In Scenario 2, the best-case is that the team spends another 400 hours rebuilding the widget with a more appropriate tool. It is possible that the second attempt took a little less time, since the team had a better idea about how they wanted to build things, but there's still the added cost of another tool. It is also likely that there was some ill-will generated with customers since Widget V.1 didn't work very well and wasn't very performant. (Had the product realized expectations, there would not be a need for a re-write.) However, for the sake of simplicity, we'll say that it cost the same team 400 hours. That's another \$159,808.00 minus the \$5610.30 you saved by keeping the expensive engineers out of the room. When math is carefully applied to the problem, that leaves you \$154,197.70 in additional costs.

Scenario 3 could be a relatively quick fix – perhaps costing as little as a week or two. However, depending on how inappropriate the proposed solution was, it could also lead to many weeks of struggle for the team. Again, using the least damaging version of events, we can re-use the numbers from the first scenario and say that the inappropriate design cost 2 weeks, or an extra \$31,961.60. Even with a fast fix, the company is out a lot of money!

It seems un-necessary to do the math for Scenario 4, since it's clear that the loss will not improve when all three situations are combined.

Obviously, these scenarios are simple and a bit contrived but, as stated previously, I've witnessed versions of these situations play out many times in both large and small companies). It cannot be stated with absolute certainty that things would have progressed more smoothly if the Engineers had been brought in early and fully included in the design and discovery sessions. However, I believe it is reasonable to say that having people in the room with deep knowledge of the existing infrastructure and software would not have made the situation any worse. The math clearly shows that a relatively small initial expenditure in engineering time during the design stage has the potential to save tens of thousands of dollars once the project moves into the development phase.

6 What the Right People Need to Succeed

Even when the “correct” people are included from the very beginning, they cannot just be turned loose with the expectation that miracles will be created. However, the opposite condition is also true – if a technical team is given a complete solution with no challenges, it is not reasonable to expect them to be excited and creative about the implementation process. The team will feel as if they've been reduced from engineers to “code monkeys”.

If success is the desired outcome for the technical team, provide them with a problem and allow them to pitch potential solutions to the Business representatives. Using the earlier example, this is how the Discovery Team might frame the scenario for the development team:

- “We need a widget that stores data from a web page!”
- It needs to work with our existing website.
- Stored data needs to be accessible to both customers and internal users.
- Data should be always backed up in a secondary location.
- Request response time should be less than 1 second.

Clearly, this isn’t a very complicated example, but it demonstrates the point. The desired product has been specifically described but without deep technical requirements. This is now a problem that the team can solve creatively. They may choose to build POCs with Cloud and Co-located environments so that responsiveness and cost can be compared. Once the team has accumulated some data and determined the best options, they can be presented to the Business for the final decision. However, it’s the technical team’s responsibility to define the potential solutions that they feel are most appropriate.

The result is that the Business gets the needed software, and the technical team is empowered with the freedom to design something that they are excited about creating and maintaining.

7 Implementation in the Real World

Any Software Development Lifecycle can Lead with Quality. For instance, in a Waterfall paradigm, the design process remains a specific stage that sits in front of the development portion of work. The difference is that a subset of the technical team is involved in the design stage instead of receiving the pre-defined solution from the business.

In a more Agile lifecycle, the development team may receive a simple skeleton of the design with the technology chosen and the initial work laid out along with a vision of where the product will eventually go. A potentially productive approach would involve the Product Owner coordinating between the business and the technical leads to lay out each new feature/solutioning challenge during the sprint before the one where implementation is needed.

Regardless of the methodology, it is critical to get the correct people involved from the very beginning of the process and, when the project is presented to the team, it must be done in such a way that the team is challenged and excited to build the product.

8 Conclusion

A healthy company culture abhors a large meeting – and rightfully so. The bigger the meeting, the longer and more difficult it is to make significant accomplishments. It is also more costly. But, when project planning begins by excluding the people who know the software ecosystem most intimately: Software Developers, Quality Engineers, and Support - the result is likely sabotaged before the first piece of code is written. However, if even a few of these folks are included at the initiation of the planning process and the resulting plan is constructed in a way that challenges and excites the team, the final product will likely result in saved time (and money) for the project, a happier team, and better quality software. All of which mean higher profits for the company.

9 Acknowledgements

I’d like to start by thanking my editors, Bhushan Gupta and Robert Sabourin, who are both absolute gems! They kindly and mercilessly provided great feedback that helped me transform this paper from a collection of random ideas into a coherent and (hopefully) interesting work.

I would also like to thank my husband, Justin, who still doesn't entirely understand what I do, but supports me in doing it.

And, as always, my sincere gratitude goes to my employer NWEA/HMH and my manager, who consistently support and encourage my participation with PNSQC every year.

References

- Bird, Jeremy. 2019. "How to improve users lives when users & customers are not the same people" Jeremy Bird Blog, entry posted February 13, 2019, <https://jeremybird276.medium.com/user-vs-customer-its-not-mutually-exclusive-b7f954c80b1d> (accessed July 25, 2023).
- Cherry, Denny. 2019. "Having the Correct People on a Project Is Key to Success" article posted May 16, 2019, <https://orangematter.solarwinds.com/2019/05/16/having-the-correct-people-on-a-project-is-key-to-success/> (accessed July 25, 2023).
- Coursera. 2023. "What Is a Solutions Architect (and How Do I Become One)?" article posted June 15, 2023, <https://www.coursera.org/articles/solutions-architect> (accessed July 25, 2023).
- Harris, Aaron. 2015. "QA Role in Product Advocacy" Envoc Blog, entry posted April 7, 2015, <https://envoc.com/think/qa-role-in-product-advocacy> (accessed July 25, 2023).
- Hern, Alex. 2018. "The Two-pizza Rule and the Secret of Amazon's Success." The Guardian. April 28, 2018. <https://www.theguardian.com/technology/2018/apr/24/the-two-pizza-rule-and-the-secret-of-amazons-success> (accessed July 25, 2023).
- Kaplan, Zoe. 2023 "How Many Work Hours Are in a Year?" Forage. July 11, 2023. <https://www.theforage.com/blog/basics/how-many-work-hours-year#:~:text=The%20average%20full%2Dtime%20U.S.,weeks%20in%20a%20calendar%20year.&text=O,n%20average%2C%20there%20are%202%2C080%20working%20hours%20a%20year> (accessed June 27, 2023).
- Khalimonchuk, Kateryna. 2022. "Discovery Phase in Software Development: How it Saves Time & Money" Fulcrum Rocks Blog, entry posted April 17, 2022, [https://fulcrum.rocks/blog/discovery-phase-software-development#:~:text=Discovery%2C%20\(or%20scoping%20phase\),%26%20scope%2C%20and%20identify%20risks](https://fulcrum.rocks/blog/discovery-phase-software-development#:~:text=Discovery%2C%20(or%20scoping%20phase),%26%20scope%2C%20and%20identify%20risks) (accessed July 25, 2023).
- Merriam-Webster.com Dictionary, s.v. "Occam's razor," <https://www.merriam-webster.com/dictionary/Occam%27s%20razor> (accessed June 27, 2023).
- Muzyka, Bohdana. 2021. "Project Discovery Phase in Software Development (Step-by-Step Guide)" TechMagic Blog, entry posted July 29, 2021, <https://www.techmagic.co/blog/project-discovery-phase-in-software-development/> (accessed July 25, 2023).
- Puzhevich, Victoria. 2022. "5 Steps for Choosing a Technology Stack for Your Project" Scand Corporate Blog, entry posted May 10, 2022, <https://scand.com/company/blog/choosing-a-technology-stack/> (accessed July 25, 2023).
- Testim. "What Is Shift Left Testing? A Guide to Improving Your QA" Testim Blog, entry posted June 11, 2021, <https://www.testim.io/blog/shift-left-testing-guide/> (accessed July 25, 2023).
- Wikipedia. 2023. "Sunk Cost." Wikimedia Foundation. Last modified August 13, 2023. https://en.wikipedia.org/wiki/Sunk_cost (accessed August 14, 2023)

QUALITY OWNERSHIP

Linda Wilkinson

wilkinsonlinda4@gmail.com

Abstract

While technology has changed significantly and SDLCs (Software Development Lifecycles) overall have morphed with the advent of agile methodologies, quality processes have not kept pace. The problems we had 40 years ago are problems we still deal with every day. Why are we doing this to ourselves? Our standard processes didn't work very well 40 years ago and they don't work very well now.

Quality Ownership is designed to take agile processes to a new level using shift left thinking prior to making tickets ready to work, using the expertise of the Quality Analyst to manage the quality landscape, the SDET staff to begin automation early, and the agile team itself to assume responsibility for quality as a team. Givelify has experimented with this methodology, we've learned a lot, and our results are exciting enough we would like to share them with the rest of the quality community. Our Quality Analysts are able to serve multiple teams (at a savings of hundreds of thousands of dollars), our quality levels in Production remain high, and our agile teams are better able to absorb last-minute changes to support our business.

Biography

Linda Wilkinson has been a QA practitioner for 40 years (!). She started her career in software analysis, moved into QA, and followed the traditional route of QA Analyst I, II, III, Senior, SDET, Lead, Manager, Director, VP, and took a few brief forays into DevOps, DBA, Product Management, and data analysis as it became necessary in order to accomplish quality goals. At last count, she has worked with 18 languages, helped author a book, and has had the privilege of speaking at a variety of conferences, both as a speaker and as a member of expert boards in QA and metrics. She has built QA teams (functional, automated, and performance) from scratch five times. Passionate about quality and providing business value, her teams have maintained error rates close to 1% with high customer satisfaction ratings in a highly competitive and ever-changing technical landscape.

Copyright Linda Wilkinson 072323

1 Introduction

In order to understand the ideas behind Quality Ownership, a basic understanding of the history of the field might be in order. This is a brief overview of the quality world during the past 40 years or so, told from the perspective of a practitioner.

40+ years ago, everyone was using Waterfall methodology. Many modern perceptions of what waterfall was like are, well, just wrong. Final implementation to Production might have been “Big Bang”, but large projects were broken down into phases, often a month long. You might compare it to a 4-week sprint. At that time, in the author’s world, QA staff were developers who reviewed code, made structural updates, ran tests, and fixed bugs. This was done at the end of the phase and again at the end of the entire project. That final review/test phase could bog down for months. QA was considered a bottleneck. The mission of the QA team was considered a failure if errors were detected in Production.

The QA world changed in the early 80s, when a book that had been around since 1973 started getting some traction – Glenford Meyers’ “The Art of Software Testing”. It talked, for the first time, about the psychology of finding errors; pointing out realities that still ring true today. For example, it talked about WHY it’s difficult to find errors in your own code. Consider, for example, that I write a report for the CEO. I want it to be well-received. I may read over and check my work ten times. I will find and correct some errors. If I then ask a colleague to read it just as a final check, they will find multiple things I missed – some of them painfully obvious. Why did I miss them? The reason is psychological. We don’t really want to find errors in our own stuff. In essence, developers test to prove their code works. QA Analysts test to prove it doesn’t. Just that difference in perspective drives out error. But for me the biggest take-away was unless you are writing code for other developers, your clients do not care what language your code is written in. They don’t care about your environment, your methodologies, or your expenses. YOU care about those things. They just want something that does what they want it to do and doesn’t break. Now there’s a simple concept of “quality”, although it’s incomplete (we’ll talk about that later). So the field morphed and QA as a whole started testing systems from a user perspective. While this certainly was faster, responsibility for structure, bug fixes, and everything but the actual testing devolved to the development staff. Tests were defined by QA as soon as the analysis document was available, which took time, and as code that was often untested and incomplete was passed over to QA staff for testing at the end of a phase or project, the QA process overall was still time-consuming. QA was still considered a bottleneck. Errors in Production were still considered a failure on the part of the QA team.

Enter Agile. Agile methodologies came out in 2001. What is really interesting about Agile is that it’s 22 years old and everybody is Agile. Or are they? Many of the original precepts behind success for the original agile teams have simply been lost as our field continues to evolve. Do you have a closely-knit, co-located team in the same time zone? Provide food? Have someone that protects the team and does not allow them to be interrupted? Do they pitch in and help each other out? So they’re all full-stack developers? Does the entire team do the testing? If not, you may have something, and it might work well for you, but you’re not really close to the original concepts behind Agile. And what about QA? How did their world change? Here’s the bad news. It didn’t. It’s just in 2 week chunks instead of 4 week chunks. With the advent of exploratory testing, which most companies still think means “no documentation” or preparation, our terminology changed, but the methods we use to uncover and report bugs really didn’t change. If there is a QA person on the team, they are likely responsible for testing, regardless of when or how things are delivered to them. Chances are also good QA will be responsible for test automation, and if something fails in Production, it is likely the QA team getting the call; “Why did you miss this?”. In a truly Agile shop, this would be viewed as an agile process error; everyone is responsible for testing and the quality of the project. But in reality, QA is still considered a bottleneck when the team fails to make a deadline and a failure if errors manifest in Production.

There have been and are many flavors of project methodology, but the mission and methodologies of a quality professional have not changed significantly since the early 1980s. RUP (Rational Unified Process), MBO (Management by Objective), Six Sigma, Lean, Scrum, Kanban, TDD (Test-Driven Development), BDD (Behavior-Driven Development) and all derivatives thereof – it doesn’t matter; a QA professional needs to be able to support any type of project methodology and provide value to their teams

and their company. Our job is to adapt. From process-heavy methodologies like RUP, to admin-heavy processes like MBO or Six Sigma, to the ceremonies and time-boxed Scrum methodologies and the C/CD (Continuous Integration/Continuous Deployment)-friendly looser weave of Kanban, all of these methodologies can and have been successful. Methodology of any type can and has been unsuccessful as well. Overall, success of any given team is more about the team itself. A talented, invested, dedicated, supportive and supported team is going to make things work in spite of obstacles. So why work on methodologies? Because we can and do hamstring our own teams, burn them out, and demotivate/demoralize the very people we depend upon.

2 Realities

Let's consider the nature of testing itself. While a test can be developed before anything actually exists to run it against, the testing itself cannot be performed and give you any meaningful information until has something to test. "Test early, fail often" sounds great and all testing is valuable, but the errors found as you shift left (test whatever you can as early as you can) are different than those you find later in your development cycle prior to production. My code may be tested and completely performant, yet the nanosecond I put it in place and it interacts with other code in a more production-like environment, it can fail. It can cause other code far down the line from my own to fail. Overall, it is the later phases of testing that are most indicative of performance in production. It is time, after more than 40 years, to recognize a primary truth about testing. It takes place at the end of something. It takes place once something is done. The most meaningful testing takes place at the end of a development effort when all of the pieces are ready to interact with each other in an environment as close to production reality as possible. The reason testing extends a deadline is that its very nature demands end-of-cycle execution. The purpose of testing is to find errors, all of which have to be assessed and potentially fixed by the team. This takes time and effort by human beings. It is true whether the tests are automated or manual as results must be analyzed, assessed and potentially fixed by a human.

This testing reality is true regardless of methodology. So, the first challenge in terms of change for the better is finding ways to make an activity that can't take place until something is done faster without losing any of the benefits of that activity.

Diving even deeper, what is "testing"? Testing is comparing a given result against an expected or desired result and noting anomalies so they can be repaired – either prior to Production or later. Or the decision may be made that the anomaly is acceptable as it stands. How is this decided? In other words, who decides what is good? From a practical perspective, eventually it will be your customers that decide what is good. But using our simplistic definition above, your product does what is expected and doesn't break; the expectations against which to compare the results of your testing is documented somewhere. In Waterfall days, these expectations of what needed to be produced, what it should look like, and how it should operate were in a bulky analysis document. This document took a very long time to prepare, but it did have some advantages. First, everyone on the team had a complete picture of what was desired before coding or testing started. Second, it was so cumbersome to create, changes were discouraged and normally had to go through some thorough vetting and a change process before it was accepted. This was both a blessing and a curse. The blessing was that spurious and unnecessary changes and bells and whistles were not added as often, which kept disruptions to the forward momentum of the team(s) to a minimum. It was a curse in that the teams were far less responsive to changes in the marketplace and technical issues or discoveries later in the process caused longer and more stressful updates to what had already been done, with the added burden of ensuring everyone on every team was kept in the loop about those changes.

Now most analysis work is kept in a variety of mediums. There might be one site for designs and the work is likely broken down into pieces ("tickets") in a project management tool such as Jira. Jira tickets and similar contemporary project management tools are also both a blessing and a curse. The blessing is that a ticket represents one small element of work and is fast and easy to produce. Changes are easier to absorb. While a change might have to be vetted, it might be by a single team and only to determine if it

can be handled during the current sprint or needs to be placed in a later sprint. There are a number of cursuses. Reading a single ticket does not necessarily give a very good picture of the overall effort. Changes may not be vetted as extensively for impact to the product and team or even if it is desirable from a user perspective because the "approving body" is much smaller and far more focused on their own specialized tasks. It is also unusual for a ticket to be detailed enough to use effectively as a basis for testing. Tickets are "supposed" to contain acceptance criteria. Often they do not. They may contain generic statements that are a re-iteration of the title or intent of that piece of work, but acceptance criteria go far beyond that in reality. For example, if you are creating a log-in screen, your acceptance criteria might simply state the user is able to log-in to your application. A testing professional is going to have quite a few tests associated with such a ticket. What happens if the log-in is wrong? If an incorrect log-in allows access to your application, that anomaly, or bug, will be written up and the team will have to address it. In other words, the acceptance criteria is actually every test which if failed will result in a critical bug which must be addressed by the team.

So who is responsible for determining the test set for the ticket? The product owner, manager, or BA might be responsible for creating initial acceptance criteria. But it is normally the quality professional who is responsible for taking any design documentation and the ticket and determining what needs to be tested. If the QA personnel on your teams do not begin this work until a ticket is pulled into a sprint to be worked, or they do not document what they intend to examine, your agile team is already behind. The critical analysis piece of the equation will be started at the same time as development. There is no way they will be able to use those tests for their own unit/shift-left testing, and getting automation done prior to the end of the sprint will be problematic.

Ultimately, the test analysis is the most critical part of the testing process. It doesn't matter what kind of analysis documentation you have (or don't have). It doesn't matter what kind of project methodology you use. The key to ensuring a valuable testing effort is running the right test, at the right time, in the right environment and providing the results to the team. Shocking as it may seem, it makes no difference whether that test is manual or automated. It doesn't matter who actually executes the test. The key is running the right test, at the right time, in the right environment.

This means the most important part of the entire testing process is test analysis. Determining what needs to be examined and comparing actual results to expected/desired results. Good testing is always a comparison. There is a big difference between executing code and verifying code. When there are no expected results, it is not possible to ascertain whether a result is desirable or undesirable except through the experience and expectations of the tester themselves. This will work well if your tester is highly experienced, has extensive product knowledge from your users' perspective, and is 100% available. They will make good guesses most of the time, although a chunk of time is going to be spent consulting with others to get their perspective on any perceived anomalies. But it represents a risk to your company and is ultimately not sustainable. Most companies would not want the future of their organizations dependent on guesses, yet many operate this way with varying degrees of success or lack thereof. It is also important to the team that this test analysis is done as early in the process as possible. This allows development staff to understand and fold testing expectations into their work and the entire team to agree on what "done" looks like.

Thus far, we've determined that testing takes place when something is done and that the key to successful testing is good test analysis done early. Let's talk about people next.

3 The Disconnect

There are many myths around the testing practice. Everyone who has ever tested anything, including a battery, believes they know everything about testing, that it's easy, and that the entire field is somehow a lesser activity than development. Good testing requires training, experience, and discipline. Testing is part of the development process. The specialized nature of the field should be evident; there is a reason QA professionals find so many issues after development is complete. Why aren't developers catching these errors? Where is the disconnect? Is that perception that's been around for 40+ years that "developers are terrible testers" really true?

The short answer is no. But let's talk about the disconnect. Say my company is an on-line marketplace. We add vendors to our system, which enables them to make their goods available to customers who can purchase said items on-line. A QA professional is going to know how to add a vendor, add goods, access those goods as a customer, purchase goods using a variety of payment methods, and verify the vendor and my company got paid. They may also check any analysis/reports provided to the vendor or receipts to the customer. They are going to know these things and more because they have to in order to do their jobs.

A developer, on the other hand, is likely to specialize in either front-end or back-end work and be deeply, technically involved in one small part of that chain. Generally, people learn and do what is required for them to be effective and no more; most people don't have the time (especially in the IT field) even if they have the interest. There are developers out there doing good work and contributing to their companies that have never even logged on to their own systems as an end user. Does that mean they are incapable of learning? No. But it does mean they lack the breadth of understanding of their own systems from a user perspective necessary to do good test analysis. Think about the best developers you've ever known. You know the ones – you call them at 3 AM when something goes awry, whether it's actually in their scope of responsibility or not. It is highly likely the development staff most valuable to you are those that know your system from BOTH a technical and a user perspective. A customer support person can explain what a user did and they understand instantly and ask questions customer support understands, then can pinpoint the area of code most likely at fault and engage their cohorts in finding a solution. This level of employee is going to be better at test analysis – unfortunately they are rare and unlikely to be used for such a purpose.

What this means is the disconnect is due to the learning requirements and time available to gain the expertise necessary to be an effective test analyst and not capability. It is again the test analysis part of the equation here, not the actual testing. The actual testing activity does not require any particular skill. It can, after all, be run by a machine. Anyone with time available and understanding of basic skills can run a defined test. Understanding what test needs to be run? Test analysis is a highly specialized skill.

What about other members of the team or the company? What about Product Owners? Product Owners certainly understand the system from a user perspective. What they lack is training and time. Does your Product Owner/Manager have the right skills to perform test analysis? Take a look at their tickets. What does their acceptance criteria look like? Do they provide any? Do they include how the system handles invalid or unexpected input? Do they mention downstream applications that need to be verified as part of the change? If not, even your lowest-level Quality Analyst is going to be more effective at test analysis.

What about an SDET (Software Development Engineer in Test)? This is an interesting question because there are so many assumptions and myths surrounding test automation as a whole. There's an assumption that test automation is a recent development; it's actually been around for at least 35 years. Every company, everywhere, always wants their test bank(s) automated. And why not? Something that takes manual testers weeks to execute can take less than a day, even with the human interaction portion of reviewing test results and failures. But in order to automate a test bank, you have to have a test bank. And again, who is going to design your tests? An SDET? They just want to automate tests. They are developers, not test analysts. They interact with, think like, and work like development staff. They tend to miss the same bugs development staff do. That makes sense, because their understanding of your systems from a user perspective is likely on the same level as your development staff. A common myth in the IT field is that an SDET is just like a Quality Analyst, except they can also automate tests and are therefore a better investment long-term. Unfortunately this is just not true. Even if it was, an SDET with years of experience is going to be expensive, and generally they will not enjoy and do not want to be involved with any significant tasks that do not involve automation specifically.

Where are we now? Thus far, we've determined that testing takes place when something is done, that the key to successful testing is good test analysis done early, and that Quality Analysts are best positioned to perform test analysis. The last reality to tackle is process.

4 Opportunities for Change

It is to be hoped that any company interested in innovation and with some claims to operating in an agile environment periodically review its own processes, identifies those that are holding it back, and either provides training or tosses them accordingly. That is, unfortunately, a hope in vain. People resist change. There is some comfort in familiarity, and fear of losing respect by not being expert (or being unable to be competent) in something that is new. In addition, there is a warm, fuzzy feeling when you are part of a pack or you do things the same way these other 2000 reputable, successful companies do. That is not, however, necessarily in your best interest. Instead of being a follower, you could be a leader. And leaders do what is in the best interests of their own company. Not someone else's. We need to look at our processes and be tenacious about cutting what doesn't work.

Let's look at a few agile processes for a moment. If you do not have a tight, co-located team in the same time zone, are you agile? If the answer is no and you're successful, who cares? You can call your process anything you like and laugh all the way to the bank. If the answer is no and you're struggling, with processes that do not work for you, it's time to take a second look. Are all your team members in the stand-ups or are some in a time zone that does not permit that? How do you handle that problem? Is it really handled, or are some team members unavailable or not in the loop? Are your retrospectives more than a 4-syllable word? What valuable changes have you made in the past 6 months due to retrospectives? The idea of a retrospective is great. Lots of ideas are great. But unless you appreciably benefit from an agile ceremony, let it go. What about RCAs (Root Cause Analysis)? Do you do them? Are they about improving processes and tools or providing training or capability growth, or do they simply blame people? Do your people embrace them, or do they dread them? If the answer is b., what are you going to do about it? And if you've been doing these same ineffective, damaging things for years, then just stop. Achieve your goals in some other way.

With this in mind, let's look at a typical scrum-like project sprint. A change or new feature is identified. Any new/changed screens and workflow are worked on and placed in (some) tool. The workflow and changes are broken down and estimated by a team(s). Including QA time in estimates may or may not be included. The Product Owner pulls tickets into the next sprint based on team capacity. Team capacity may be based on historical data or may not. The development staff starts working. The QA staff begin test analysis. Once an individual ticket is complete, it may be tested by either the developer or by QA in a non-prod-like environment. Bugs are written and the fix/retest cycle continues. Once all the tickets are complete or a working feature is available, the QA staff retest the entire feature in a prod-like environment. All needed tickets may not be available at the same time, resulting in some delays. In addition, several design changes might take place coming directly from the executive team. Errors in the estimation process (perhaps using story pointing) become evident, but the team is committed now and moves forward. The fix/retest cycle at this point in the process strongly impacts team delivery dates. Once the delivery date has been reached, the team schedules its next sprint, with QA staff taking time out of their testing (and developers of their fixes/resubmissions) and development begins for the next sprint, although the last one is not complete and in production. If enough time passes during the final fix/retest cycle, two sprints may be combined. Team capacity may or may not be updated accordingly. Once the sprint is in Production, tests are passed off to SDET personnel for regression automation.

Does any of this resonate with you? This process is not effective. We need to address when test analysis takes place, when a ticket is pulled in as "ready to work", how tickets are estimated, when tickets are automated, who is responsible for the physical activity of testing, and how team capacity is determined. Note there was no mention of addressing design changes coming from the executive team. If this is part of your reality, a better perspective is to ensure your final processes support these as a "business as usual" part of your landscape. Admonishing an executive team is not an effective strategy.

Before we continue with a potential solution, the above examples are IT-related, but there needs to be some discussion regarding quality as a whole. While the most simplistic definition of quality is that it does what it is supposed to do and does not break, there is actually more to the concept of "quality" than those two items. It is not a matter of who has the prettiest website either, although doing your homework and understanding what your customers want is important or no one will buy what you're selling. But when

someone thinks about quality and whether your company has it and how they “grade” you on a variety of sites is dependent on absolutely everything that touches them. You can have a killer app that has zero errors, but if your customer service rep treats someone poorly or cannot solve a problem, you’ve lost a customer. If you have any competition whatsoever, you have to pay attention to and “test” everything that touches your customer. Your website. Your social media posts. Definitely your customer service practices. Emails. Texts. Mailings. Everything that touches your customer influences their perception of the quality of your organization. This is a basic reality. Even if your customer base is locked in and has no choice but to use your apps, that could change in the future – best to treat them like the gold they are right now.

In summary, we now know that testing takes place when something is done, that the key to successful testing is good test analysis done early by the Quality Analyst, and that our process is broken and needs work. We would also like our solution to be useful to any part of our company that touches our customers. Let’s talk about a potential solution to these problems.

5 Quality Ownership

There was a conference for quality professionals back in the 1980s called “Quality is Free”. Quality is not, and never has been free. Quality is expensive, both in time and resources. It is, and always has been, difficult to justify and obtain sufficient quality professionals to handle the workload. The only way to support growing applications and company size is to expand the QA presence accordingly. What if we took a page from the book of other specialists on agile teams and made it possible for a single Quality Analyst to support more than one team? DevOps personnel, for example, might support multiple teams, as can Product Owners, UI Analysts, and others. What would that look like?

First, the single most critical contribution a Quality Analyst makes to a product team is establishing the test base. That is a task that others on the team cannot perform as successfully (see Realities above). If we agree Quality Analysts should establish the test base, when and how should that happen? If we want to solve the problems of not having tests available to be incorporated into shift-left testing efforts, having tests automated by the end of the sprint, and development beginning their work potentially weeks before QA begins theirs, it makes sense to say the Quality Analyst will establish a test base prior to a given ticket being pulled into a sprint as “ready to work”. A separate paper could be written just to deal with the “hows”, and it’s highly dependent on how a company operates now, but this would mean involving the QA Analyst during the design phase, ensuring tests are documented, stored, and attached to the ticket prior being pulled into a sprint. I use the word “sprint”, but if you work with another methodology, it would be prior to whatever block of time you divide into work chunks.

Once the development staff pick up a ticket to work, there are two possible scenarios. The original idea was to hand the ticket over to an SDET at the same time the developer pulled it in to begin work, but then an interesting conversation took place with Viktor Mateleshka of Givelify. Viktor had a team with no QA resources on it, producing some work for internal customers. To be frank, the QA team, in particular the QA Analysts, were a bit chary of Viktor; they felt he didn’t recognize their value and were afraid he wanted to get rid of QA altogether. In actuality, Viktor had a high degree of respect for the Quality Analysts and credited them with a great deal of value in designing tests and finding critical bugs. What Viktor felt was overkill were the SDET staff. That is frankly an astounding, ground-breaking, non-typical point of view. What he said was that he had a whole team of developers. With an established automation framework, any one of them could automate a test. What he did not have, and wanted, was all of the experience and understanding of the systems from a user perspective helping his teams determine what needed to be tested. In other words, he wanted test analysis. From there, he felt his team(s) could take over.

Regardless of whether this task is assumed by an SDET or a developer, tests can be developed before code is available with most contemporary automation frameworks. That is one of the foundations of TDD (test-driven development). Once elements/objects and actions/functions are defined, these are easy to build in a basic form, although more tweaking may be necessary as the code develops and the tests are actually run. As part of the development efforts, time to develop the tests would necessitate considering

them as part of the estimation process, improving the track record of the team for on-time deliveries. In addition, the availability and running of automated tests, incorporating them into CI/CD processes, would benefit the entire team in terms of time. It would encourage teams to automate, and automate early, which is very much in keeping with the direction in which agile development is moving.

The most critical and controversial part of Quality Ownership is the question of who actually performs the testing. In short, anyone on the team other than the Quality Analyst. This is a non-negotiable key point to success for the entire Quality Ownership process. Why? Because breaking that “QA will test it” mentality will not go away and the teams will not grow and move forward until those crutches are no longer available. I saw a podcast recently by Elisabeth Hendrickson that talked about a situation she had as a QA leader where her team was being blamed for a production failure and she had no people to spare to help them out any further. The development leader was furious. But reality is reality, and his team was forced to carry on with no testing support. Guess what? They became better. No amount of QA professionals on the team had improved the quality of their output, but once they were responsible and assumed responsibility for the testing, they were forced to improve.

The idea of expecting the team to test their code also forces other staff to learn how to use their own product from a user perspective, which grows highly expert, valuable personnel. The available test pool grows from one person, often trying to “save” the team by testing overtime at the last minute, to the entire team, all working together to get the job done. That is the epitome of agile methodology. Again, with testing genuinely part of the development process, estimation techniques would necessarily have to improve in order to allow for the time to execute the test/retest cycle. And those valuable, expensive QA resources can support more than one team. If their primary job is to identify the test set, once that is complete they can move over and help do the same thing for another team.

How does this relate to non-IT teams like Customer Support, Marketing, and others? Anything that touches a customer should be “tested” prior to presentation to that customer. This would be a simplified process; a Quality leader would identify what needs to be verified and ensure those things are done and acceptable prior to presentation to actual customers. It could be as simple as establishing a process whereby all blog posts are reviewed prior to posting. It could be as complex as a Marketing Campaign, with all that entails.

In the next section, we’ll talk about actual implementation and early results from experimenting with Quality Ownership on selected projects.

6 Implementing Quality Ownership

Change is hard and this one is a radical departure from what makes your teams comfortable. Those of you who have had to shepherd a team through any type of radical methodology change will understand immediately what types of challenges you’ll face when trying to implement Quality Ownership.

First the good news. Givelify is a particularly good place to experiment with new things and I would like to acknowledge their participation and support. They are a small, (slightly over 100 employees) company dedicated to “the best of the best”, which is likely why they are #1 in their particular niche, which is enabling charitable donations through an elegant set of phone apps that their customers love. They have visionary leadership and a particularly strong interest in both innovation and education. They are interested in Quality Ownership and agreed to experiment with it with one of their programs (3 teams). Those 3 teams are currently utilizing 2 Quality Analysts and 1 SDET. Since implementation, both Quality analysts are supporting more than one team, the SDET supports 3, and quality levels in Production have not declined. The quality levels at Givelify were very high to begin with (their error rate is under 1%), but what is important here is that the changes underway have not negatively impacted their client base. It is the intent of the company to continue what they’ve started and work with the new processes, adding additional teams as they gain in comfort and confidence. The company has been working with Quality Ownership for only a few months, but code delivery is no longer dependent on the availability and findings

of one individual (improved delivery), headcount has not been increased (6 were planned), and quality has remained high. Further assessment of savings and benefits/problems will become evident over time.

Atlassian, a 95+ billion dollar company, has done a great deal of interesting and brilliant work in regards to what they refer to as Quality Assistance; a link is provided below and will give you an idea as to the scope of this type of change. Where the techniques in this document differ is in the role of the quality expert in the process. Atlassian utilizes their quality staff as mentors and teachers with responsibility for making the teams better. The Quality Ownership concept includes the quality expert as part of the contributor to the team in a different but complimentary way. Part of the reason for that is purely due to human nature and the difficulty of change. A team member that produces a product, just like every other member of the team, is viewed differently than a member that proffers advice and “doesn’t get their hands dirty”.

It would be a disservice to you and to the field to not address the challenges and problems with implementing Quality Ownership. This is not a trivial change. The first problem has to do with Quality Analysts and their view of their work. Many Quality Analysts identify and define themselves through actual testing activity. While having to perform the work under a constant cloud of pressure is not especially enjoyable, testing, in and of itself, is fun. Quality Analysts are accustomed to being the one to find the errors, work overtime to save their team, and generally act as corporate heroes, even if just within their own frame of reference. That’s hard to give up. What was found during implementation of Quality Ownership was that finding some conditions under which Quality Analysts could physically test, such as cross-team efforts (especially across programs), added to their satisfaction levels. In addition, the field continues to morph into an ever-increasing world of test automation and asking your analysts if they want to remain relevant in that world will provoke some good conversations, thought, and a willingness to try it out.

On the development side, developers are going to kick back against this process. They are already tremendously busy, and they definitely do not want to add testing to the mix. Particularly if that testing includes manual testing. That’s actually a good thing, as part of the object of the methodology is to encourage test automation. After experiencing the changes for several sprints, a few major points came up that need to be considered. The Quality Analyst cannot be available to assist with the physical testing activity. If they are, the team will react by reverting to the way they always did things. There is a tendency for back-end developers to accept the methodology more easily than their front-end counterparts. This is likely because they are more accustomed to doing at least a portion of their own testing. Depending on the company and the skill sets of their quality staff, they may or may not be comfortable with “headless” testing - that is testing without a front-end feeding in data. This type of testing requires use of a tool such as Postman or SOAP/SOAPUI. It’s possible back-end developers in your company do their own testing. Front-end development staff, however, will have to become familiar with some sort of testing architecture and build their tests accordingly. This activity would be required for any type of automated or TDD/BDD methodologies, but it is human nature to resist change and busy staff members will push back even harder against learning or doing something new. There needs to be no other options. Your SDET personnel can help here both in setting up the architecture, assisting the development with automated test development, or assuming responsibility for test automation, as best suits your company situation.

It is strongly recommended that when first implementing this process that some random checking be done to ensure the tests identified were executed and verified. It may seem laughable to some (a painful reality for many QA personnel) that a developer may not understand the need to test certain areas and will therefore opt not to run given tests. When this process was begun, that very scenario arose and the random check uncovered a critical bug. You will need some method to ensure all identified tests are run and results evaluated.

Management and executive management have to hang tough. It is very hard not to cave when your teams tell you they can’t handle testing. If testing is simply not available, the teams have no choice. In these cases, reminders that there are companies out there with no QA staff that are doing just fine, that the QA staff are still available to answer questions, and pointing out the advantages to adding these skills to their toolbelt can be helpful to their careers. It has the advantage of being true. Ultimately, you cannot Excerpt from PNSQC Proceedings

move your company forward with this or any other methodology if you, as management leaders, allow individuals to undermine your efforts and thus the progress of your company. Good training, patience, and understanding are all going to be required.

Once the teams become accustomed to performing testing as a team and accepting responsibility for the quality of their code as a team, there is no reason the “no testing by Quality Analysts” rule cannot be lifted. But we need to move quality from “that person’s responsibility” to “our responsibility”.

Over time, assessment needs to be done to determine if the new methodology has had the positive impact overall originally expected. Have the team members become more expert in using the systems they build from a user perspective? Does the full team pitching in to get the testing done cut down on how long it takes to complete the fix/retest cycles? Are Quality Analysts able to handle more than one team? Have estimates improved? Is test automation done more quickly and is it being used?

Time will tell. As a personal note, I encourage all of you to experiment with this process and go change our quality world. It's about time, don't you think?

References

Myers, Glenford J (2006). The Art of Software Testing. Wiley India Pvt. Limited

Craske, A. (2021). How Atlassian Does Quality Assistance

<https://qeunit.com>

Hendrickson, Elisabeth (2023). Achieving Better Outcomes Through Structure

IT Revolution Episode 3, <https://itrevolution.com>

Defect Management System Powered by AI (Artificial Intelligence)

Peh Wei Wooi, Liu Keping, Ooi Mei Chen, Felix Eu

Wei.wooi.peh@intel.com, keping.liu@intel.com, mei.chen.ooi@intel.com,
felix.eu@intel.com

Abstract

Defect management is an important process as part of project management. It helps to provide insight into project health, quality assurance, schedule estimation, risk mitigation, continuous improvement, and resource allocations. This allows the project to raise alerts about quality and make accurate decisions based on data analysis.

The purpose of implementing the Smart Defect Management System (SDMS) is to manage a vast amount of defect data in one tool with a smarter way to provide high data integrity information for the user with regard to defect analysis, defect prevention, and defect reporting. By leveraging AI capabilities, the integration of AI into SDMS empowers the system to process and analyze extensive data, resolve complex problems, and make complex forecasting/decisions with minimum human intelligence.

The goal of SDMS is to deliver high-quality information that enables effective decision-making and enhances operational efficiency while minimizing the costs and efforts involved.

Biography

Peh Wei Wooi is a Platform Validation Lead at Intel Corporation based in Penang, Malaysia. He is certified as the ISTQB tester and holds a Degree in Information Science from UKM, Malaysia.

Liu Keping is a Technical Leader in Software Quality Assurance at Intel Corporation based in Shanghai, China. She is a certified CMMI assessor, ISO internal assessor, ASPIICE internal assessor, and CSQE, and gained 6 Sigma Orange Belt and CPMP certification in 2009. She holds a master's degree in computer science and technology from Central South University in China.

Ooi Mei Chen has been in software engineering for over 12 years and has held many roles spanning code development, design, and project management. She currently serves as a senior System Software Quality Engineer at Intel Corporation based in Penang, Malaysia. She holds a Degree in Computer Science from University Tunku Abdul Rahman, MY.

Felix Eu is a Software Quality Engineer at Intel Corporation based in Penang, Malaysia. He has held the Lean Six Sigma Green Badge since 2019, is a certified Software Quality Engineer (CSQE) from ASQ and holds a Degree in Computer Science from the University of Bolton in the UK.

Copyright Peh WeiWooi, Liu Keping, Ooi MeiChen, Felix Eu 2023

1.0 Introduction

1.1 Problem Statement

Defect management is an important process as part of project management. It helps to:

- Provide insight into project health
- Plan for schedule estimation
- Plan Resource allocation
- Allow the users to raise quality risk alerts
- Make accurate decisions based on data analysis

Traditionally, the defect lifecycle is managed in one system, and defect analysis is done in another system, possibly manually via Microsoft Excel or Microsoft PowerBI. A typical defect management system includes defect reporting, defect analysis, and defect prevention.

It is challenging to manage a large pool of defect data without a robust tool to automate the data collection/manipulation/categorization/analysis and reporting processes. According to this journal, "The Challenges of Data Quality & Data Quality Assessment in the Big Data Era", data integrity and consistency are always the biggest challenges for users to obtain up-to-date information for analysis, as the data might be collected from several databases and from different user groups.

Furthermore, the most commonly used defect tool on the market today works as a single tool for a particular purpose and it requires human intervention to configure and filter criteria to obtain the result manually. Then the data collection needs to propagate from one function to another function for analysis (For example, the defect management tool and the requirement tool are not linked together; hence the defect ticket is not able to analyze immediately and identify the validity based on the requirement specification) which creates additional overhead to hosting and maintaining the data.

1.2 Smart Defect Management System (SDMS) Concept

An integrated system, especially an AI-powered system, will help to eliminate data gaps between different tools and improve efficiency.

The Smart Defect Management System is targeted to provide a one-stop solution to users with the following benefits:

- Automatic defect classification and defect scrubbing capability
- Automatic defect prioritization and solution recommendation capability
- Real-time defect status dashboard which utilizes the Edge and Cloud services

The one-stop service hosting in the cloud enables the user to access the system without any limitation on time zone, location, hardware, etc. Information will be stored and available on a real-time basis without any delay.

Traditional Defect Management System (DMS)	Smart Defect Management System (SDMS)
Manually manage a large pool of data	Automated & more efficient
Data consistency & integrity is LOW	Data consistency & integrity is HIGH
Required multiple tools	Required ONE tool for all
Non-real time	Real-time
On-premises/Server based	Edge & Cloud services
Downtime is HIGH	Downtime is LOW
Less predictive	More predictive
Slow decision making	Quick & better decision making

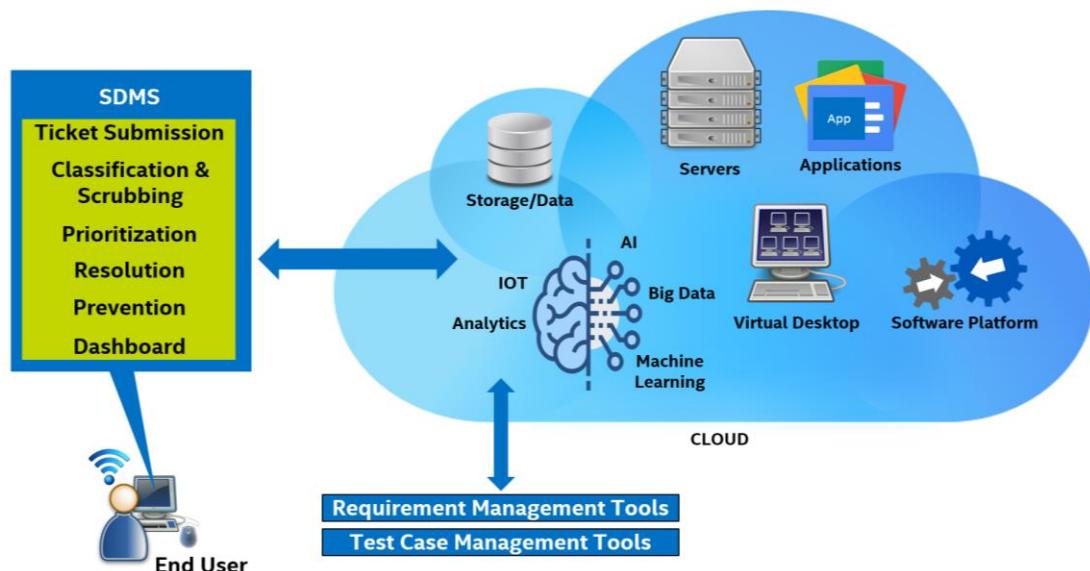
Table-1 Traditional DMS vs. SDMS

2.0 Opportunities & Solutions

By examining the problem statements, we have identified significant opportunities for advancement.

- **All-in-One Solutions** to reduce the maintenance effort & cost
- **AI-powered Defect Classification & Scrubbing** to enable efficient information retrieval & eliminate redundancy of data
- **Smart Defect Prioritization** to manage the risk & improve time efficiently with better resource allocation to meet deadlines
- **Defect Solution Recommendation** to improve the defect triage cycle time
- **Intelligent Preventive Quality Measures** to improve product quality & cost savings
- **Real-Time Dashboard** to expedite data-driven decision-making

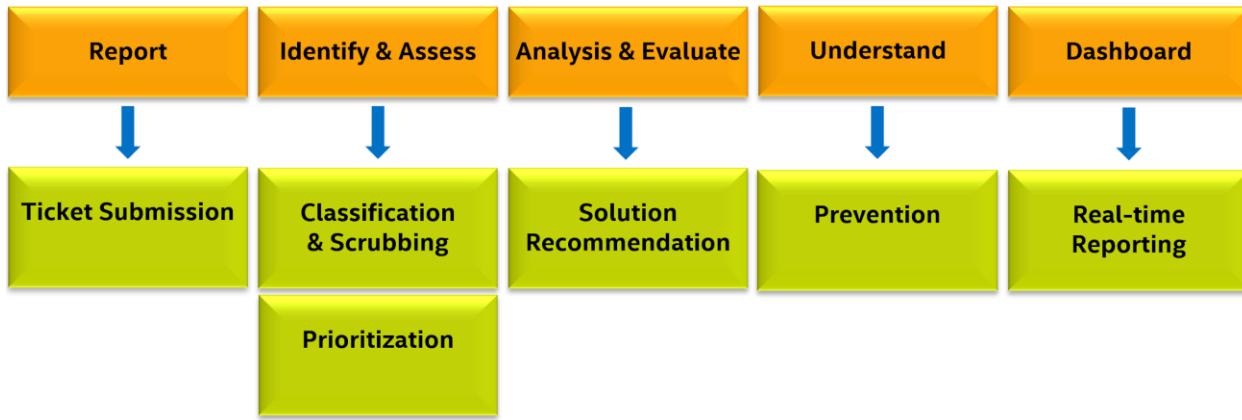
2.1 Smart Defect Management System (SDMS) Architecture



The SDMS is hosted in the cloud which enables the user to access without any limitation on time zone, location, hardware limitation, etc. All the information will be available on a real-time basis.

SDMS Architecture

AI-Powered Defect Management System



The SDMS possesses the below capabilities

- Defect reporting
- Defect classification & scrubbing
- Defect prioritization
- Defect root cause analysis recommendation
- Defect prevention recommendation
- Defect real-time dashboard

The SDMS will use a smart way to identify the validity of the defect when it is reported. Once the defect has been identified as valid, the SDMS will classify the defect based on its severity after assessing the defect fixing lead time (Including the code fixing complexity) and the release milestone.

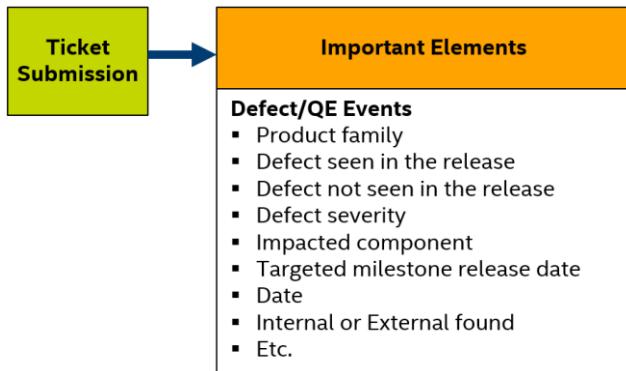
Based on the track records analysis, the SDMS will evaluate and propose a potential fix solution to the user. With the source data collected, the SDMS then automatically plots the Root Cause Analysis (RCA) and defect prediction charts via logic incorporated with AI algorithm and machine learning.

By processing the escaped defects database strategically, the SDMS will then derive an actionable prevention plan and recommend it to the user.

All the information will be in real-time and displayed in a dynamic and up-to-date manner without user intervention. This real-time dashboard provides a snapshot of critical data and metrics that allow users to monitor and track status, trends, or events as they happen.

2.2 Solutions

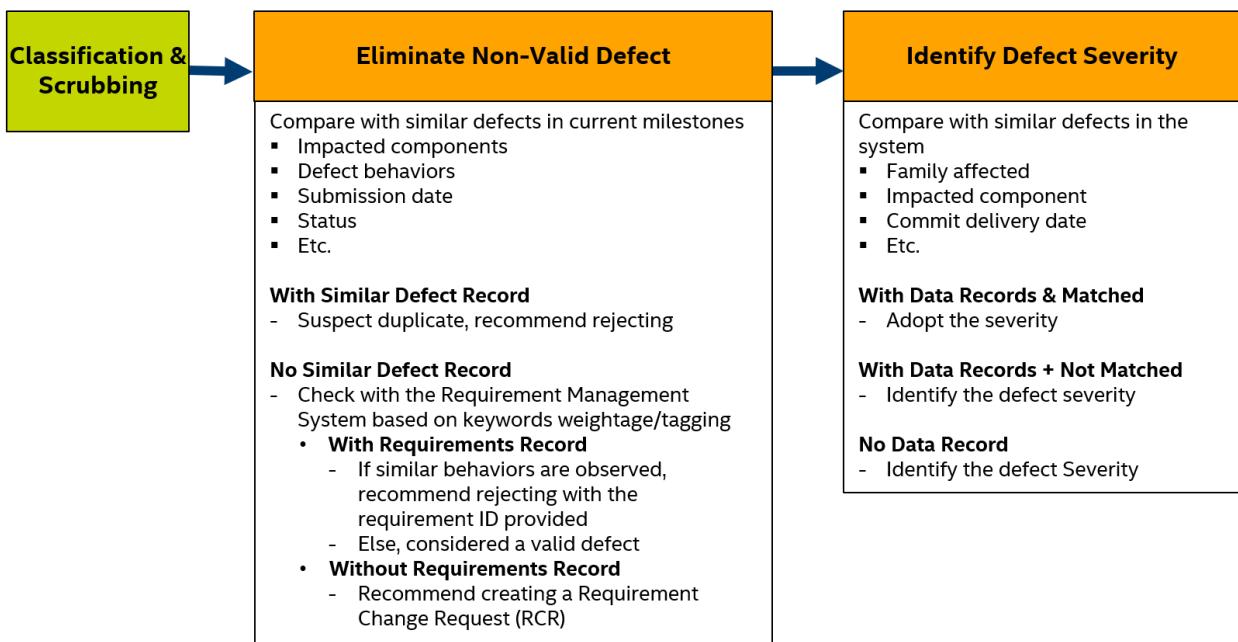
2.2.1 Defect Reporting



It is crucial to ensure all the relevant information is provided when filing out the defective ticket. This will help to:

- i) Identify the problem accurately
- ii) Efficient troubleshooting and debugging
- iii) Priority and impact assessment
- iv) Documentation and historical context

2.2.2 Defect Classification & Scrubbing



The newly submitted defect will draw a comparison with similar defects in the current milestones based on its impacted components, defect behaviors, etc. If a similar defect record is found (suspect duplication), then the SDMS will recommend the user consider rejecting this newly submitted defect.

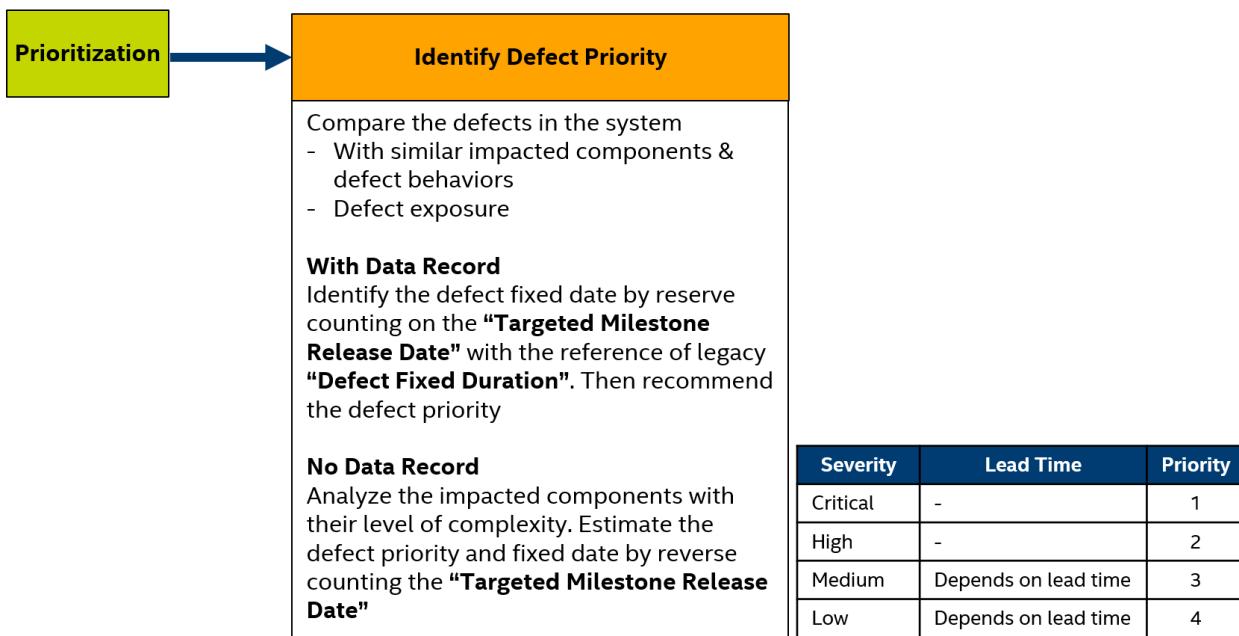
If no similar defect record is found, then the SDMS will compare the defect behaviors with the requirement specification stored in the requirement management tool. By using the defined keywords

weightage/tagging, the SDMS will evaluate if the behaviors work as designed or as a valid defect. Meanwhile, the SDMS will also recommend the user raise a requirement change request (RCR) for requirement specification improvement if there is no relevant info found. This will help to address the requirement coverage gap.

Once the newly submitted defect is identified as valid, a defect severity will be evaluated based on the impact, visibility, and standard violation. The SDMS will compare the severity level based on the defect database. The data records will be based on the similarity of the “important elements” defined in the issue ticket such as impacted product family, components, operating system, defective behaviors, etc.

- i) With Data Records & Matched – Adopt the severity level
- ii) With Data Records & Not Matched – Either adopt or assign a new severity level
- iii) No Data Record – User to assign a new severity level

2.2.3 Defect Prioritization



The SDMS will then assess the defect priority & fixed date based on:

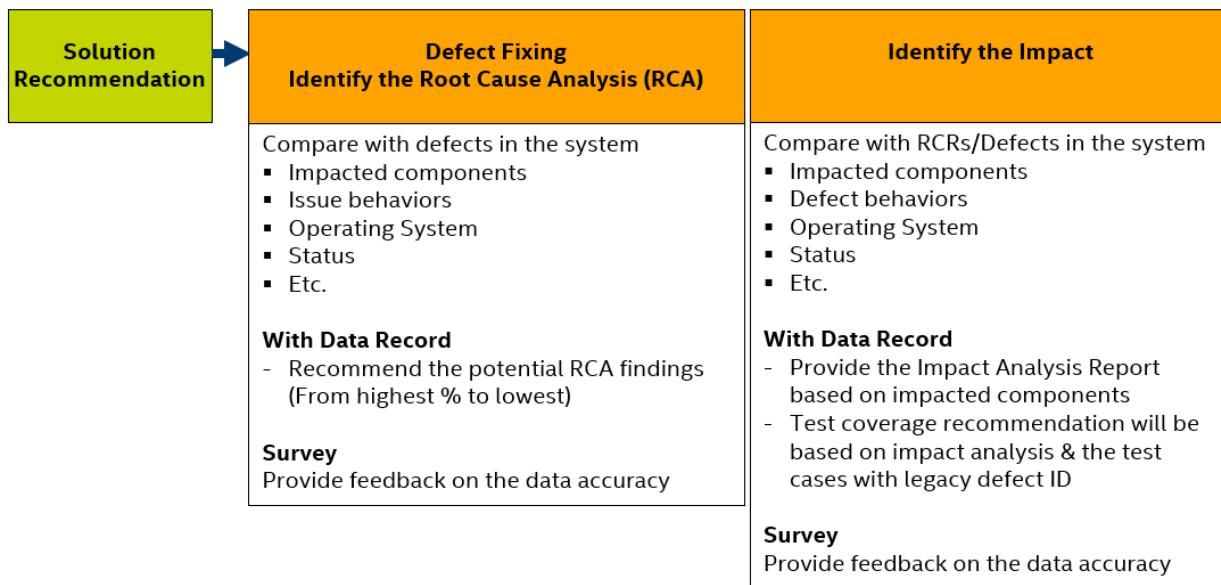
- i) Similarity of impacted components & defect behaviors
- ii) Defect severity

If a data record is found, the SDMS will identify the defect priority and defect fixed date based on the reverse counting of the “Targeted Milestone Release Date” with the reference of legacy “defect fixed duration”. The “Targeted Milestone Release Date” normally means the actual planned release date for the software.

If no data record is found, then the SDMS will analyze the impacted components with their level of complexity and estimate the defect priority and fixed date.

The SDMS must be intelligent enough to assess the defect priority level based on the defect severity. If the defect severity is high (might gate the milestone release), then the priority also needs to be high to get immediate attention to fix the defect as soon as possible.

2.2.4 Defect Root Cause Analysis & Recommendation

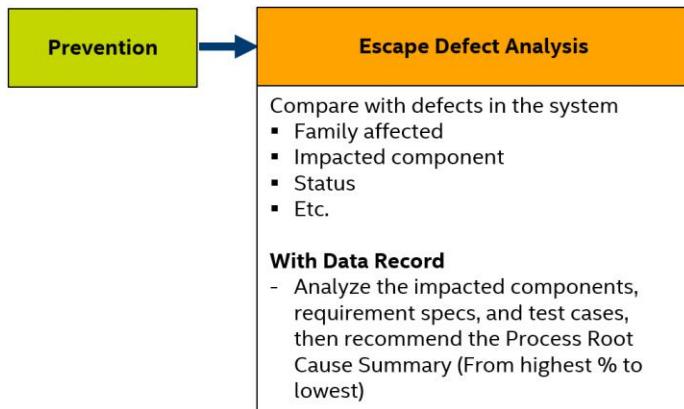


The focus of SDMS under Solution Recommendation will be on 2 areas which are identifying the defect root cause analysis (RCA) and impact analysis. The SDMS will intelligently identify the potential root cause analysis for the defect and recommend them based on similarity percentage (%). Once the RCA is adopted by the user, then the SDMS will generate the impact analysis report and recommend the test coverage based on:

- i) Test type (Unit testing, Functional testing, Stress testing, etc.)
- ii) Test case priority (1, 2, 3 & 4)
- iii) Test cases with legacy defect ID (Ensure no breakage on the legacy fixes)

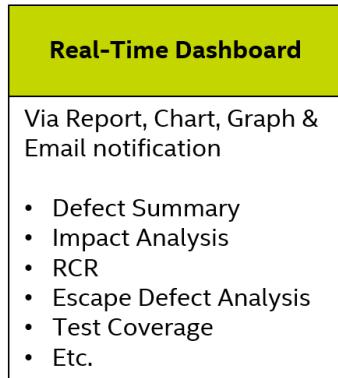
The user will conduct a review session with all the respective stakeholders (Such as the program manager, software architect, validation manager, etc.) to review the recommended RCA & impact analysis to ensure it is officially bought into before implementing it. If the information provided does not 100% fit the request, SDMS should have the ability to gather user feedback for continuous improvement.

2.2.5 Defect Prevention Recommendation



The aim of prevention is to strengthen the validation process based on the lesson learned from the escape defect. When the defect ticket is identified as an escape (reported by the customers), the SDMS will analyze the impacted component, requirement specifications, and existing test cases. After all the evaluations, the SDMS will then recommend the most related improvement process to the user.

2.2.6 Defect Real-Time Dashboard



A defect real-time dashboard provides real-time visibility and insights into the status and progress of defects. It offers a centralized view of the defects identified during testing, development, or operational stages and presents relevant metrics and information in a graphical format. The main purpose of a defect real-time dashboard is to provide the stakeholders with up-to-date information on the current state of defects.

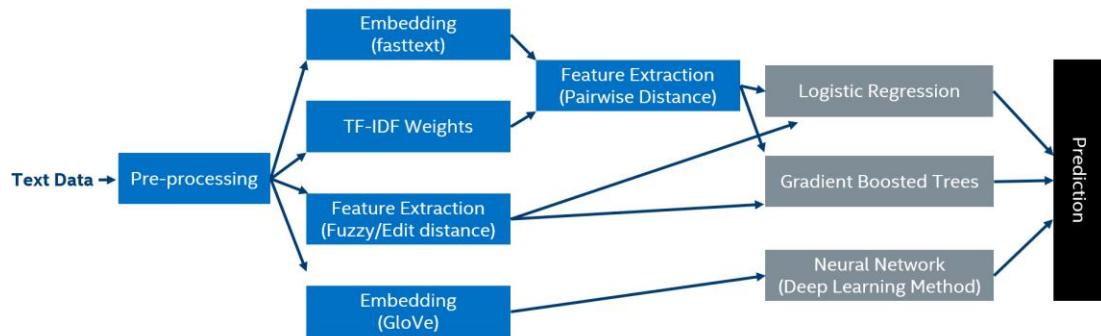
The benefits of the dashboard will be:

- i) Live updates
- ii) Metrics & trends
- iii) Visualization
- iv) Filtering & sorting
- v) Collaboration & communication
- vi) Decision-making support

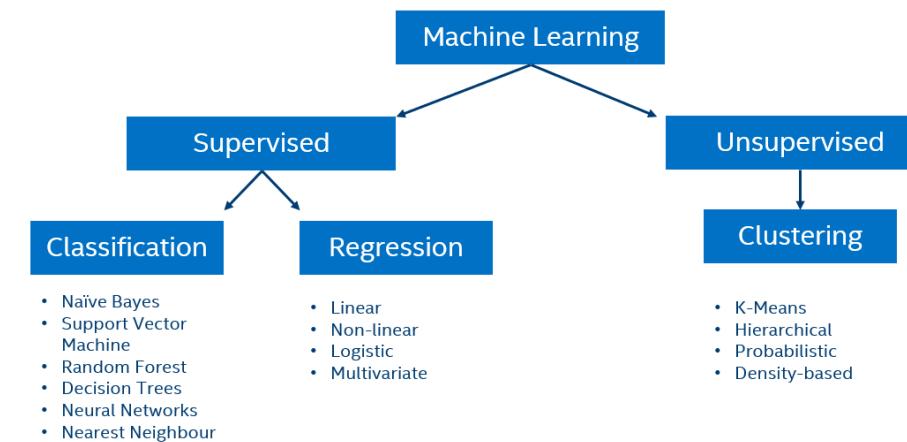
2.3 Machine Learning Methodologies

Machine learning uses computational algorithm methods to learn information directly from data. Various machine-learning approaches are used to design, develop, and implement the SDMS. These methodologies provide a structured and systematic way of solving problems and extracting insights from data using machine learning algorithms.

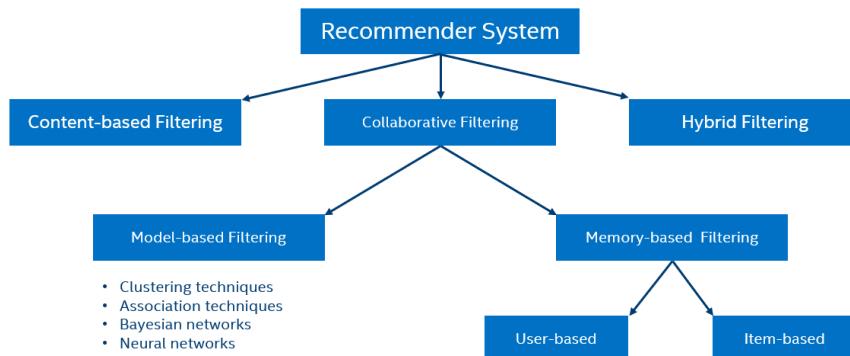
2.3.1 Identifying Similarity - Can be used for defect classification, Scrubbing & Prioritization



2.3.2 Prediction – Can be used for Resolution recommendations & Prevention



2.3.3 Recommender - Can be used for Resolution recommendations & Prevention



3.0 Estimated Benefits

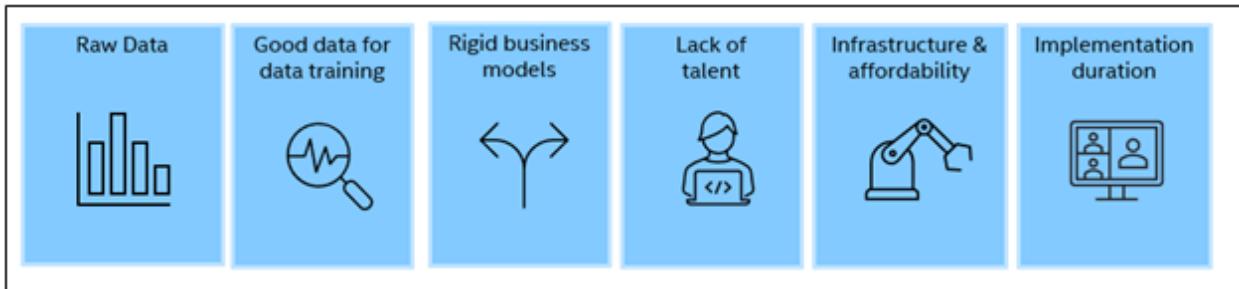


The results we use to measure the SDMS benefits can be categorized into 4 areas based on our actual study and were measured by various methods and metrics.

Our plan is to establish a baseline at the beginning of the project to serve as a reference point. Throughout the project, we will try to compare the actual resources utilized, time spent, and costs incurred against the baseline to determine the reduction achieved. This is done through regular progress tracking & reporting. Meanwhile, we also want to define the specific KPIs related to resources, time & costs for the project to measure and compare at different stages of the project to assess the reduction achieved. It is also very important to gather feedback from the relevant stakeholders to understand their perception of resource, time, and cost savings. This feedback can provide insights into the perceived benefits and efficiency improvements resulting from the project.

- In the automated and consolidated environment, we estimate at least 30% resource, time, and cost savings.
- Based on the shorter turnaround time for defect analysis & solution recommendation, we foresee at least a 40% Reduction in debugging time.
- At least 3 times of Improvement on Traceability because we have consolidated all the report & result in the One-Stop-Shop (OSS) dashboard.
- Instead of the user logging on to a different tool or application to generate the defect analysis report, escape defect analysis report, etc, and now they can collect them all in one place.
- Lastly, we also expect to have at least 3 times Faster Dashboard, and Report creation to improve data consistency & integrity because the real-time data is hosted in the cloud.

4.0 Challenges



Companies all over the world are progressively exploring “SMART” solutions to reduce business challenges and provide innovative solutions. Even though “SMART” benefits are becoming more apparent, many companies are still facing challenges in the adoption:

- Good data is needed for data training/defect analysis. To obtain the huge amounts of data required, we need time and effort to go through multiple iterations to gather the data
- For rigid business models, we need to pay extra attention to convince them to make a move and transform from a traditional defect management system to a SMART defect management system
- For lack of talent/expertise and Infrastructure & affordability, system architecture/cloud expertise is needed to set up a “SMART” defect management system and make sure the flow is smooth. Additionally, we need to consider the expenses and readiness of the infrastructure e.g., setting up a cloud server within the organization. Companies with sufficient budgets and capabilities may consider forming a team to own this “SMART” defect management system, else we also can consider joining forces with a 3rd party software house
- And lastly, for implementation duration we need to take care of the transition risk and total time & effort needed to implement the SMART defect system. Hence, it is crucial to be transparent and align the expectations with the stakeholders frequently.

5.0 Conclusion

Today, we are in a rapid pace environment that requires digesting a humongous amount of info, the traditional defect management system which included a lot of manual work without integrity is no longer able to fulfill a company’s needs. Hence, we need to find a “SMART” way to process and analyze all data more quickly than ever before to help us to make faster and more accurate decisions.

The AI-powered Smart Defect Management System is bringing a revolution in the fight to produce better quality software as it is able to:

- Provide flexibility & mobility
- Increase fast decision-making capability
- Improve quality, cycle time, effort & cost
- Avoid potential risks

There are significant advantages to be gained in owning a smart defect management system for the organization. All the parties involved such as developers, project leads, validation engineers, Software Quality Engineers, etc. can deposit their defects here and obtain instant solutions (Based on recommendation). All the defects handling processes are thus considerably simplified.

6.0 References

Li Cai & YangYong Zhu. (2015). The Challenges of Data Quality & Data Quality Assessment in the Big Data Era. <https://account.datascience.codata.org/index.php/up-j-dsj/article/view/dsj-2015-002>

Question similarity (<https://devopedia.org/question-similarity>)

Overview of Software Defect Prediction using Machine Learning Algorithms
(<https://www.semanticscholar.org/paper/Overview-of-Software-Defect-Prediction-using-Kalaivani-Beena/2162c83789f3681019a9322d5505d04e0566da44>)

Implementing Recommender Systems using Machine Learning and Knowledge Discovery Tools
(https://www.researchgate.net/publication/354259927_Implementing_Recommender_Systems_using_Machine_Learning_and_Knowledge_Discovery_Tools#pf3)