

# **Proceedings**

**40th Annual  
Pacific Northwest Software Quality Conference**

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

**October 10-12, 2022**



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

This page is intentionally left blank.

# Table of Contents

	<u>Page</u>
<b>Forward</b>	5
<b>Conference Papers</b>	
<i>in alphabetical order of first author's surname</i>	
1. I am a Product Owner & I want to run a 'Load Test' - by Jerome Anthony, Eresha Spaulding, Thakshila Sugathapala, Hansika Muthunayake, and Oshara Amarasiwardena	9
2. Risk-based testing reduces time to market of dynamic EDA tools - by Mohamed Bahnasawi, Ahmed Khater, and Reem ElAdawi	31
3. Phased Testing: Testing Systems Undergoing Change - by Baubak Gandomi	40
4. Managing affirmatively through the "Great Resignation" - by Mark Bentsen	57
5. Fallacies of Data Driven Decisions - by John Cvetko	68
6. Hey Alexa - How To Test My Voice Assistance? An Automation Approach - by Juan Delgado, Lucaz Tutur, Piotr Wroblewski, Deanna Raven, and Christopher Schoppa	81
7. How I Learned Things and Influenced People with Visual Notes - by Moss Drake	91
8. Evolution and Future of Software Testing - by Mesut Durukal	100
9. 7 things I learned to build highly effective onsite/offshore teams - by Venkat Edagottu	110
10. Formal Technical Review Process Enhancement - by Eu Felix, Peh Wei Wooi, Ooi Mei Chen, and Liu Keping	117
11. Tri-Layer Testing Architecture - by Péter Földházi Jr.	134
12. Application of Usability Testing to GUIs in the Electronic Design Automation Industry – by Kirolos George, Marwa Adel, and Reem ElAdawi	143
13. Building Security into Your Apps One Story at a Time - by Bhushan B. Gupta	160
14. TestZeus: Automate the lightning and thunder of Salesforce Testing - by Robin Gupta	171
15. Testing IoT Systems Using V&V, Security Hacks, & Data Analytics - by Jon D Hagar	178
16. Agile Gets Physical: Slice-Based Integration - by Kathleen A. Iberle	189
17. Service confidence... because code coverage and CRAP scores just don't give you the whole picture - by Don Kidd	203
18. Software Quality Assurance Methodology for Hybrid Waterfall & Agile Development - by Liu Keping, Eu Felix, Ooi Mei Chen and Peh Wei Wooi	213
19. Test Case Prioritization Using Hybrid Deep Learning Approach - by Loo Willam and Yip Kin Choy	226

20. A Method to Select Tests Based on Code Coverage - by Jack Marvin and Trevor Hammock	242
21. Artificial Intelligence is the New Astrology of Software Quality - by Jack McDowell and Ying Ki Kwong	249
22. Role of Emerging Technology in Securing Rural Healthcare - by Sneha Mirajkar and Vittalkumar Mirajkar	260
23. How AI Can Help Accelerate Testing - by Chris Navrides	269
24. Driving Quality Improvement Through Root Cause Analysis - by Amol Patil	277
25. The Versus Framework: A File Comparison Standard for Software Testing - by Omar Mohamed Ragi, Ahmed Sayed Ali, and Reem Al- Adawi	291
26. Building a Smart High Quality Software Pipeline - by Richard Robinson	307
27. Active Feedback Loops in Software Testing - by Robert Sabourin and Chris Blain	315
28. Building a Modern Quality Program from the Ground Up - by Jeff Sing	337
29. Leading Successful Teams - by Vadiraj R Thayur	348

# Forward

This is the 40th anniversary of the Pacific Northwest Software Quality Conference (PNSQC). We celebrate this milestone by acknowledging the entire PNSQC community: our attendees, authors / presenters, volunteers, sponsors, and board members over the years. Thank you for your support over four decades!

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 2022 continues this tradition.

This year's conference featured speakers from around the globe in four tracks. During the three days of PNSQC 2022, our technical program consisted of keynote addresses, invited presentations, peer-reviewed papers & presentations, panel discussions, tutorials, workshops, and exhibits of our sponsors. For additional details, see [www.pnsc.org](http://www.pnsc.org).

This is the first year that PNSQC was held using a hybrid format; with a physical venue in Portland, Oregon and a simultaneous virtual venue via Zoom. We are pleased that 142 attendees participated in person, with an additional 183 attendees participated virtually. Among all attendees, 12% were based outside the U.S. Among attendees based in the U.S., 48% were from outside the Pacific Northwest (Oregon, Washington, and Idaho). There are significant lessons learned that will enable PNSQC management to improve the conference experience 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.

# I am a Product Owner & I want to run a ‘Load Test’

**Jerome Anthony**  
jerome@asiapacific.tech

**Eresha Spaulding**  
eresa@asiapacific.tech

**Thakshila Sugathapala**  
thakshila@asiapacific.tech

**Hansika Muthunayake**  
hansika@asiapacific.tech

**Oshara Amarasiriwardena**  
oshara@asiapacific.tech

## Abstract

Today modern software providers focus on improving the quality of their products from a non-functional perspective as well as a functional perspective. The quality of non-functional performance has become a crucial success factor for almost all online software applications. Even though immediate actions may not be taken, understanding how the application will perform under load should be a ‘**known fact**’ rather than an ‘**unknown fact**’. This metric will directly impact business decisions during certain critical business events and allow businesses to better prepare and predict behavior in the ruthless universe of online software. The standard practice for load testing is to run a suite of tests that simulate system load on the application. There are plenty of load testing tools available (free and commercial), to do a reasonable load test of an application. But the problem is that the majority of these tools require personnel with automation test development expertise to set up and run them. This paper discusses that there is an interest in non-technical users such as Product Owner's/ Business Analysts to carry out a load test easily and analyse the results themselves.

## Biography

*Jerome Anthony is the CEO of Asia Pacific Software Technologies. He has 20+ years of software solutions implementation experience. He is an AWS Certified Solutions Architect and holds a Bachelor's Degree with Honors in Information Technology from the University of Colombo School of Computing, Sri Lanka.*

*Eresha Spaulding is the Test Lead at Asia Pacific Software Technologies. She has over 20 years of experience in the QA Industry. She holds a Bachelor's Degree in IT from the Chartered Institute of Information Technology, UK*

*Thakshila Sugathapala is the Test Lead at Asia Pacific Software Technologies. She has over 6 years of experience in the QA Industry. She is an ISTQB certified tester and holds a Bachelor of Computer Science Degree from the University of Colombo School of Computing, Sri Lanka.*

*Oshara Amarasiriwardena is a Quality Assurance Engineer at Asia Pacific Software Technologies. She has over 2 years of experience. She is an ISTQB certified tester and holds a Bachelor's degree in Information Systems from the University of Colombo School of Computing, Sri Lanka.*

*Hansika Muthunayake is a Quality Assurance Engineer at Asia Pacific Software Technologies. She has over 2 years of experience in the software industry. She is an ISTQB certified tester and holds a Bachelor of Information Systems (Hons) degree from the University of Colombo School of Computing, Sri Lanka.*

## 1. Introduction

Online businesses have increased drastically in the past few years due to several factors such as the Covid pandemic, and economic and social reasons across the world. Since goods and services are available at a click of a button, online shoppers have become impatient and are eager to own products and services quicker than before when they would have spent time on the road to reach a store for example. Additionally, due to the ease of creating online applications, many companies are drawn toward creating websites and mobile applications which have resulted in stiff competition among online businesses.

Moreover, the demand for online applications to be more responsive and always available even in restricted conditions has increased too. Consumers simply expect applications to work, despite changing environmental conditions.

For example, consumers expect an e-commerce site to respond with the same level of responsiveness during a peak sale event like a black Friday sale, as it would during the normal non-peak period.

In another example, a social-networking site is required to respond seamlessly during a disaster event (natural disaster, war) more than it would during an uneventful period.

Consumers do not hesitate to move into other systems which are 'rumored' to perform better. An increase in consumer awareness has led to the online environment being an unforgiving landscape for a software business. (Huaji 2017).

However, in the rush to attract customers by having new features added to their websites and mobile apps, businesses often compromise on testing. The result is a website that crashes or an app that fails to work and in both cases, a disgruntled customer and loss of credibility.

Over the last few years, we have seen several instances of technical failure in websites and mobile apps of prominent organizations. Given below are a few such instances (this white paper relies on the research of publicly available information and assumes the authenticity of the articles as is.):

**1. Tickets for London 2012 Olympics Available for Purchase:** In June, the Olympic committee announced 2.3 million tickets were available for purchase for the 2012 London Olympics. Excited fans rushed to the site to find a "Sorry, we can not process your request at this time" message because the website could not handle the rush of visitors. London has gone to extraordinary lengths to win and host the Olympics, only to have its website crash when it opens its doors to the public.

**2. New York City Offers Government Website for Hurricane Irene:** The New York City website crashed amid overwhelming visitor volume encouraged by Mayor Michael Bloomberg who strongly urged the public to use the website for information on Hurricane Irene in August. More so than any other city in the world, you would think that New York City would be most prepared for extraordinary events.

**3. Target.com Launches Missoni Collection:** The much-anticipated launch of the Missoni Collection at Target was a huge failure when the Target.com site crashed due to high traffic, rendering the entire website inoperable for hours. Target experienced a repeat crash performance six weeks later when another rush of visitors hit the site. Target has stunning marketing, but surely having their website fail during such a high-profile event isn't part of their plan?

**4. Anatomy of Click Frenzy's failure:** Two days after the highly-anticipated Click Frenzy shopping event collapsed under the weight of its own hype, website performance analysts Compuware has released data showing just where the website failed and what the frenzy of clicking actually delivered to consumers.

**5. Black Friday website crashes in 2015:** Last year we tracked lots of performance issues – the most common of which related to slow performance and timing-out errors – 2015 Black Friday analysis of website failures. What went wrong? The challenge for most sites is, of course, handling the massive spikes in traffic so that customer experience isn't negatively impacted.

The above incidents are just a few of the many incidents which highlight the need for adequate and efficient performance testing in IT. While an increasing number of customers are moving online and mobile, it would be prudent to leave no room for an application collapse because of performance issues.

Now that we understand the importance of Load testing, let's see some of the benefits business gains by investing in load testing.

- Simulate real user use and behavior of the software application/platform.

This is the most important reason why we need to use load testing for a web application or any project. When testing your web application, website, or API endpoints under a load, we can simulate scenarios to check how it will perform when there are millions of users accessing it in real life. You can identify errors, bottlenecks, and limitations before they actually happen by understanding and analyzing the results of the load testing.

- Measuring how the software application performs under different load conditions.

In load testing, there are a few key performance indicators that we have to consider like response time, error rate, memory leakage, and CPU consumption. The software development team pays less attention to these factors since they are more focused on implementing the functional aspects. But a software application that could scale into thousands of users' access, should pay attention to doing load test analysis.

This would uncover system failures, problems, and loss of revenue.

- Understanding system failures that would cause customer dissatisfaction and negative reviews/ratings.

When a user/customer is using your online software to complete a workflow like e-commerce cart checkout or filling an application form, any crash due to a load condition in the software application, would lead to major annoyance and complaining. They may take the decision to report this system failure on social media or public forums. Also, a disappointed customer has a good memory which would limit revisits to your system or application. A failed system appears unreliable in consumer's mind even though the criticality of the failure may be minor or negligible from the platform's point of view. Load testing would let you capture these disasters before they happen whereas the alternative may be to spend time re-branding the system to recover from negative online consumer sentiment.

- Save your customer experience and money.

Software engineers with expertise or experience are required to conduct performance or security tests. An organization/business needs to invest money to get these services. Training in-house engineers or hiring experts to automate load testing will take up some investment. But it will be more effective than rebranding the system or your application or worse, restoring the system or application after any failures. Disappointed customers may look at your competitors due to these types of situations. Therefore load testing understanding system capacity beforehand can save money and customers and a lot of headaches.

The above points are just a few justifications for why businesses should pay attention to load testing their application/platform as it may have a direct impact on the success of their business. There are enough and more historical examples of disasters that have led to whole

businesses going into receivership, simply because they did not pay attention to real-life application usage and performance.

The good news is that many businesses have realized the downside of not giving preference for load testing and are slowly moving towards investing in it. The bad news is as mentioned earlier, many tools available on the market, either free or commercial require technical expertise which ends up doubling the cost for a company.

It would be fantastic if an organizational member, who is a non-expert, could run a sanity load test to make a preliminary assessment of the system behavior and performance under load conditions. Due to ever increasing demand for such testing and the lack and cost of experts to develop & run load tests, tools that any team member could intuitively leverage to perform some load tests would lead to better production systems.

## 2. Typical Load Testing

Functional testing has clear testing objectives as Pass or Fail criteria. Objectives are not so clear in the early stages of the software development lifecycle with non-functional tests. They usually get defined case by case as development iterates. There are many interpretations of load testing. One definition is: a process of assessing the behavior of a system under a load in order to detect load related problems. The load defines the rate of the different service requests submitted to the system under test.

The load test should be run against the production runtime environment rather than a design view or prototype view. This is the best form of validation as this would benchmark the live system on how much load it could sustain. Testing against any other environment may give misleading results unless this environment is a mirror of the production system. It is practically not a sustainable activity to mirror the production environment in a test capacity. This is a common issue, especially with services based on complex infrastructure and software. So testing against production ensures you are testing against a valid configuration.

Figure 1 shows how multiple users would access an online load balanced web application.

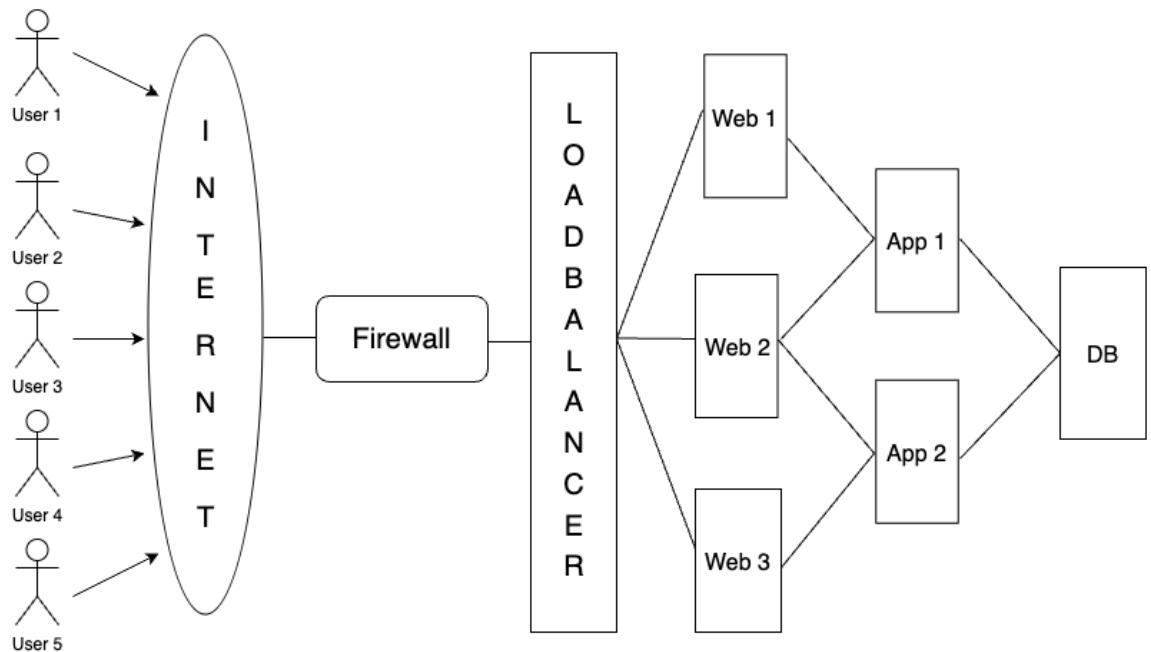


Figure 1: Load Balance behavior in a web application

Each user sends a request via the internet, which gets processed through the firewall. Then an application load balancer distributes the request to downstream web servers, which process the request at the application server and later to the database server.

For a system like this, load tests can be done manually (by enrolling a large number of concurrent users ~ which is not very practical) or by using a tool. More information about tools will be discussed later in the upcoming sections.

### 3. Outcomes from the load test

Once we have executed a load test for a system or application, it should provide detailed test results to observe whether the system or the application is in a good condition to go live. By analyzing the test results, you can assess the following facts;

- Check whether the system or application is able to handle n number of users or gain the expected scalable limits.
- Response times and average response time for each transaction.
- Component behaviors and responsiveness under various loads.
- Identify the server configuration level.
- Verify if the existing hardware is stable enough to handle the load.
- Identify CPU utilization, memory usage, and network delays.
- Derive cost estimates.

### 4. Load testing tools & strategies

Load tests can be run manually (for smaller requirements) or by using an automated test tool. Manual load testing requires manual coordination and execution every time there is a new version. There is no benefit from setup once run many times. The time and effort cost each time will be similar if not more. Coordination and aggregation of results are challenging and the accuracy of metrics is not precise.

Because of this, even for smaller requirements organizations tend to use load testing tools of various capabilities and capacities. A tool can be developed in-house as well, tailored to the organization's custom needs. Open source load testing tools or enterprise level tools provide more options and customizability but require experts (Kaushal 2020).

They all have pros and cons like other tools which have been used in the software testing field. Sometimes a combination of tools is required to fulfill the load testing objective.

Below are some pros generally prevalent in load testing tools:

- Provide accurate information about each load test level.
- Most of the tools allow you to load test specific servers rather than the whole stack.
- Easy installation.
- Some of the tools have a user-friendly interface.
- Ability to extract data from popular response formats like HTML, JSON, XML or any textual format.
- Readily available plugins, for example, visualization plugins for data analysis.

Below are some cons seen in current load testing tools.

- Some of the tools have a high learning curve.
- Need technical experts to do the load test.
- Output complex dashboards for test results which make it difficult to conclude.
- Some tools rely on a single programming language.
- Memory consumption is high in GUI mode which causes the tool itself to fail for a large number of users.
- Tool does not support multiple platforms.

## 5. My product manager wants to do the load testing

As most software businesses are wanting to run load tests, they have to seek expertise or experienced people to do the job. Most of the time this type of testing is carried out by the developers or the quality assurance team.

What if the product owner or any computer literate user with good intuition can run a load test for their system or application? This will directly translate to all organizations, big or small releasing better systems for their consumers.

Product/business owners are always well placed to make business decisions since they are domain experts. They will know what are the consumer events and what are the consumer needs. By being able to validate the system load capabilities, they will be able to make confident business decisions about the stability and runtime capacity of the application. If they want to have questions about how a certain workflow would perform under load they can validate it themselves without having to wait for an expert team to band together and run the test on the system.

Such a tool should;

- be simple to use and intuitive.
- have templates to launch typical test scenarios.
- allow for trial and error.
- require minimum setup.
- guide the user through the process.
- allow the user to save and repeat the tests multiple times.
- generate a generally understandable report with business meaning.
- be relatively quick to set up and run.

- be relatively low cost.

Allowing non-developers to assess non-functional aspects of the system in a timely manner would also provide independent review and confirmation of system readiness.

The industry needs to understand that involving business users in system non-functional testing will contribute to better domain validation and suitability of the software. Business users are more consumer empathetic and may review the software for readiness from these angles more than the engineering team. Making non-functional testing like load testing accessible to business/product owners will directly lead to better business software for consumers.

The challenge is the incremental introduction of such tools. A load testing tool that a business user can access may only have a few capabilities which whilst being intuitive gives a good sanity check regarding the load handling capability to the business users. Such a tool would also lead to the business users validating business critical workflows that they cannot afford to fail in a production environment.

## 6. Limitations of existing load testing tools

Load testing tools currently in use have several drawbacks. One of the main issues is these tools do not provide a friendly setup for non-technical users. Most widely used load testing tools mainly focus on providing a platform for developers and testers. These tools provide excellent scripting capabilities which would be done by developers and testers without much effort but which would be harder for someone who is not familiar with scripting or testing. As performance testing has become an essential factor to make sure the software is running smoothly, someone from the team who is not a developer or a tester but is in need of evaluating their system should be able to perform load testing. For example, Apache Jmeter requires certain knowledge to understand Jmeter objects, session management, etc (Abbas et al. 2017).

Another major issue with load testing tools is the steep learning curve. Having to put a huge effort into learning the tool and how it works, can be seen in many of the tools currently in use. The process of testing needs to be as simple as possible, minimizing the time and effort of learning.

Most of these tools consume a lot of memory and CPU. This can cause the tool to be frozen without any prior alerts. This consumption of hardware also needs to be minimized for a better performance of the tool itself. Popular tools such as Apache Jmeter, generate heavy load and test data (Jiang 2015). Prerequisites and installation processes are some other issues users face with existing tools. Most often in order to use the tool, users need to have to fulfill several other requirements and have to follow several installation steps.

### 6.1 Load test tool for Testers & Developers (Technical)

We surveyed to assess testers' and developers' challenges when performing load testing. This survey was conducted aiming at technical people (especially software developers and QA engineers). Following are the questions and responses we've posted/received.

**Q1.** Define the role of the user conducting the survey.

Most of our survey respondents were Test or Tech Leads. ([Appendix A](#))

**Q2.** Do you perform load testing on your product(s)?

77.4% of the respondents have stated they perform load testing on their products before releasing them ([Appendix B](#)). Although still, a significant amount does not consider load testing their products.

**Q3.** If yes, who performs the load testing?

Most often, the testing team is responsible for performing load testing. 66.7% of the survey results prove it. ([Appendix C](#))

**Q4.** How does your company perform load testing?

70.83% of the respondents use a load testing tool and 8.3% perform the testing manually while the rest use a combination of both tools and manual options to perform load testing. ([Appendix D](#))

**Q5.** Which environments do you load tests in?

The majority (58.3%) of the respondents perform load testing in their testing environment ([Appendix E](#)). 8.2% of the respondents perform testing in both production and testing environments.

**Q6.** If using a tool, what tool(s) are you using?

This question was asked to get an idea of the popularity of tools currently used to measure the performance of a product. (Ex: Load Runner, Apache JMeter, K6, LoadNinja). The most popular tool among them was Apache JMeter. The second and third most widely used tools were LoadRunner and Gatling respectively ([Appendix F](#)).

**Q7.** If load testing is conducted manually (without tools) then 'roughly' how long does it take to run for one project? (Assume one person does load testing)

The answers we've got from our respondents are varied. Most of them have stated it will depend on the requirement while some of them have provided an exact period of time like 5 hours, 1 day, or 2 days.

**Q8.** If you are using a load testing tool to perform the load testing then roughly how long does it take to run one project? (Assume load testing is done by 1 person)

While some said it takes, on average, about 2 to 3 days, and several others said it takes a few hours, about 32% of the respondents said it depends on the size of the application/ requirements & complexity of the project/ test scenarios.

**Q9.** Have you received complaints from customers of the site being unresponsive or crashing;

- during a specific time of the day.
- during specific periods of the month (e.g., end of month/weekend).
- during a specific period of the year (e.g., Christmas)?

The majority of the respondents (54.2%) stated that they have received client complaints ([Appendix G](#)) which reflects the possibility of such incidents being more usual than we might think.

**Q10.** If you are using or have used a tool for load testing what are the main difficulties you face using the tool?

55% of the respondents' main concern was that the tool eats up a lot of memory, the second highest complaint was that the tool required special hardware or high config machines to run it, and the 3rd highest complaint was that it needed a steep learning curve.

**Q11.** If load testing is not performed, what could be the reason(s) for your company to not perform load testing?

Most respondents said the tools were not easy to use, and others said the tools were time-consuming, resulting in not carrying out a load test ([Appendix I](#)).

**Q12.** Do you think your product(s) will benefit from load testing?

71.4% of the respondents thought the product would benefit from load testing, while a minority thought it was not beneficial.

**Q13.** Assume your company plans to purchase an online load testing tool to assist with load testing projects and the company asked you to do research about the current load testing tools, what are the features and qualities you would be looking for?

- How long will it take to learn the testing tool
- Does the testing tool have clear documentation about how to use the tool
- How long will it take to complete load testing for a project on average
- Does the tool have user-friendly interfaces
- Can the test results be understood by a non-technical person
- Is there a way to generate reports

While the majority focused on clear documentation, the second most essential feature respondents considered in a new tool was the ease of use and ease of interpreting results. ([Appendix J](#)).

**Q14.** Do you think your product owner can use the existing load testing tools?

One of the most significant statistics we obtained was that around 36% of the respondents thought non-technical members of a project would not be able to use existing load testing tools. Another 38% thought non-technical members may or may not be able to use existing tools ([Appendix K](#)).

## 6.2 Survey results & analysis (Technical survey)

The above survey was shared among technical personnel mainly targeting testers and developers in various capacities like engineer, or lead. By analyzing the responses we noticed several important insights such as

- Majority do carry out load tests for their products
- Although the majority uses tools for load testing, they have accepted the fact that the tools have pros and cons. For example, Although the testing time may take less than a day or two, many have accepted that most tools eats up a lot of memory, requires high configurations for the tool to run to name a few
- Many perform load tests on test environments rather than in production or production like environment for more accurate results
- Most of them have admitted that the tools are too technical and the product community may not be able to use them
- Many are giving priority to the ease of use, clear documentation and reports to be available when looking out for a new tool for their company.

[The Appendix](#) – shows detailed results of the above survey questions and summarised responses.

## 6.3 Load test tool for Product Owners (Non-Technical)

The product owner is the person who directly interacts with the stakeholders and the end users. They communicate with the client from the beginning of a project until the final version of the product is

handed over to the client. So product owners will receive first-hand feedback on the product directly from their clients.

Regarding the load testing, sometimes clients will complain that their released product is not working as expected when they are trying to perform an operation in the application at a specific hour of the day or during a specific time period of the year. Then the product owner needs to communicate with the development team (testers and developers) of the product. It will take some time to debug the issue and find a solution. To prevent this issue from happening in the first place, what if there is a tool that can be used not only by users who have a technical background but also by users from non-technical backgrounds. Hence the product owners also can contribute to doing the load testing of a product before it releases to production.

To find out more real-world data regarding this, a survey was conducted targeting product owners and IT managers. The goal of this survey is to identify whether product owners/managers receive complaints from the clients about performance related issues (load events related) and if they would consider running a load testing tool themselves to do some sanity testing rather than wait for a development or test team to conduct it.

It is also expected the survey results will outline the key features that the product owners are looking for in a load testing tool that they can use.

<b>Q1.</b> Define the role of the user conducting the survey.
Product Managers, Product Owners, Business Analysts, and Scrum Master have participated for this survey as non-technical respondents. ( <a href="#">Appendix L</a> )
<b>Q2.</b> Have you received client complaints such as they cannot perform an operation on the application, such as cannot do payments properly, or cannot submit a form during a specific time of the day or during a specific time period of the year? (Ex: They cannot do their payments for the purchased items in a Black Friday sale or they cannot submit their filled-out forms at specific hours of the day )?
60% of the respondents have said they received complaints about performance issues, while 40% have said they have not received them. However, the majority of products seem to be suffering from performance issues. ( <a href="#">Appendix M</a> )
<b>Q3.</b> Have you used a load testing tool previously to simulate a large number of users (Ex: more than 1000 users) concurrently to verify your site is able to function as expected?
60% of the respondents have not had any experience using a load testing tool. ( <a href="#">Appendix N</a> ) illustrates the responses to this question.
<b>Q4.</b> Do you think you need to perform load testing before releasing your product(s)?
All the respondents thought it is important to perform load testing on their products on their own, implying the need of a tool to perform load testing by themselves.
<b>Q5.</b> Do you think testing your site using a load testing tool is a good idea? If yes, why?
100% of the respondents thought using a tool for testing would be more beneficial than doing it manually. As the reasons for this selection, they have stated mainly manual testing would be time consuming, a tool may cover areas we might miss, and the benefit of being prepared early for system's unexpected behaviors.
<b>Q6.</b> As a non-technical person, if you were asked to do load testing using a tool, what qualities would you be looking for in the tool?

All the correspondents said they would consider how long it would take them to learn the tool as the main criteria followed by whether the tool has clear documentation, how long it will take to complete the project on average and how clear the reports would be.

## 6.4 Survey results & analysis (Non-Technical survey)

The above survey was shared among product related non-tech personnel mainly targeting product owners. 80% of the survey participants were product owners and 20% of them were product managers. By analyzing the responses we noticed several important insights which were not much explored previously. Most of the product owners seem to have limited knowledge of load testing. But the issue of customer dissatisfaction due to service outages and system crashes without any prior notice has hardly threatened them.

The Appendix – shows how many of them have received client complaints. It reflects the importance of product owners and IT managers being aware of load testing to minimize the issues customers face while they use the system.

## 7. Technical overview of the proposed load test tool

Let's consider implementation aspects when building a modern load testing tool. A modern tool may be a web-based application (SaaS) that can be developed using well-researched user experience design practices. Such tools could be developed targeting cloud platforms or cross/hybrid cloud platforms. The tool could execute load tests in containerized environments. Data reported could be stored in time-series databases. The metadata about the executing environment could be captured via advanced logging for further analysis and insight. Modern dashboards could represent drill-through views to inspect the test results. Integration with the CI/CD pipeline would allow for continuous validation against benchmarks or KPIs.

Following list details some key technologies prevalent in today's software engineering landscape that would allow for the development of a modern load testing capability.

### UX/UI development

We are going to use usability-focused Design methods along with agile development. We will use User research, personas, empathy maps, scenarios, user journeys, sketches, wireframes, prototypes, sitemaps, style guides, and usability reports with regular feedback and design iterations. As research methods, we can use user surveys for early findings. Once we create a working prototype, we can move on to heuristic evaluations also known as usability reports. It can be a low-cost method of checking early iterations against usability best practices. Once the prototype is tested against best practices, it's time to verify those findings with real users. Then depending on real user feedback, we can move on with design iterations.

- Scientifically approach UX development.
- Human-centric design.
- Research techniques

### Using containerization (Docker) for test execution/parallel testing

Containers usually can be used to maintain the parity between development, testing, and production environments. But also we can use containerization in order to reduce the time taken for

testing. The strategy involves executing multiple instances of fully isolated testing environments and executing tests in parallel.

Also, Dockerizing the test execution environment will allow us to run the load testing in any cloud provider and we could run any number of parallel testing with identical containers. Almost all cloud providers have the capability to run docker apps and we could use frameworks like Kubernetes for more control

The enhanced toolset of Docker makes this process simple and discreet, and Docker Engine, Registry, Machine, and Compose will work together to make the tests fast.

Docker is a tool that we can use to develop, bundle and run applications by using containers. Containers allow us to package the applications with all the dependencies and configurations into one bundle and deploy it. By doing so, we can ensure it runs properly without any issues. E.g., TestContainer

### **Record-playback (Playwright/Cypress)**

- Testing cases can be recorded and played back as part of load test execution.
- Test scripts for test cases can be auto-generated when you record the steps of the tests.

We could create a tool that users can use to record the steps user needs to use for the load testing and then use it in the load testing without having to worry about any automation frameworks or scripting language. The load testing tool then can generate the automation scripts using the recording itself which will finally be used to run the tests.

- Tester does not have to write the script from scratch. This is a great way to save time and make tests more efficient.

### **PhantomJS**

Leverage programmatic browsers for test development. PhantomJS can be used to run the tests in many different cloud providers because of its headless capabilities. Also since it's a scriptable browser it can be used for automation and testing

### **Serverless technologies**

- Leverage lightweight environments for test execution.
- For load testing, we need a lot of servers just for a few minutes, so having dedicated reserved servers just for running load tests is not the optimal solution as most of the time servers will be idle. Using serverless technologies will be a good solution in terms of cost as it only charges for the execution time.

### **Kubernetes**

We could use Kubernetes to manage container-based tasks at scale (orchestration). Setting up a Kubernetes cluster will help to easily scale the load testing executions up and down as needed.

### **Logging**

Capture and provide meaningful logging through useful logging capabilities. Proper logging is helpful to get detailed information of what happened in each execution and identify the issues if any. We could use tools like these,

- Grafana

<ul style="list-style-type: none"> <li>• Elastic search stack</li> </ul>
<b>Database</b>
<ul style="list-style-type: none"> <li>• Time series database for better reporting</li> <li>• NoSQL databases to capture and interrogate unstructured data.</li> </ul>
<b>Reporting</b>
<ul style="list-style-type: none"> <li>• Dashboard reports using elastic search stack</li> <li>• During the execution of the load testing we need to capture as much information as possible, then load it to a reporting engine like elastic search where we can generate any number of custom reports for the visualization. Users will also be able to create custom reports they need using these frameworks</li> </ul>
<b>CICD integration</b>
<p>Integrate various load testing into CI/CD pipeline to measure against benchmarks.</p> <p>If we can measure and compare the performance of each application build, that might provide a good insight about the product, also we can define baseline performance and check that on each build to ensure the product is meeting performance expectations as well.</p>

### Case Study

FootPrint (<https://loadtester.io>) is a cloud SaaS tool that is targeted at load testing and automation testing.

The tool tries to achieve the goals discussed in this paper implementing an intuitive tool that can be used by both technical and non-technical users leveraging better user experience design and providing simplified workflows.

## 8. Acknowledgements

We would like to express our sincere gratitude to all those who helped us immensely throughout the research. Especially Buddhiprabha Warnakulasooriya, Tech Lead, Asia Pacific Technologies for helping us with the technical details. We extend our special thanks to the developers of the FootPrint application. Finally, we are truly grateful for all the volunteers, who participated in our online survey.

## 9. Conclusion

Even though there are many tools available for load testing, most of them can only be used by technical people with proper knowledge of test automation development. Most companies tend to outsource the load testing to third parties because they don't need to do load testing frequently and having expertise in-house might be expensive. Cost concerns or time concerns due to engaging experts may deter organizations from testing non-functional aspects of the system which may lead to poor consumer satisfaction.

Organizations might opt to do non-functional testing like load testing **more** if they had access to intuitive testing tools which can be used generally by any computer savvy personnel in the business like the IT manager or product owner with little or no training. The industry needs to pioneer developing more test tools that have a broader audience who can leverage it to validate their business software before consumers find fault in real life. Wider software readiness validation would lead to better quality software all around the world.

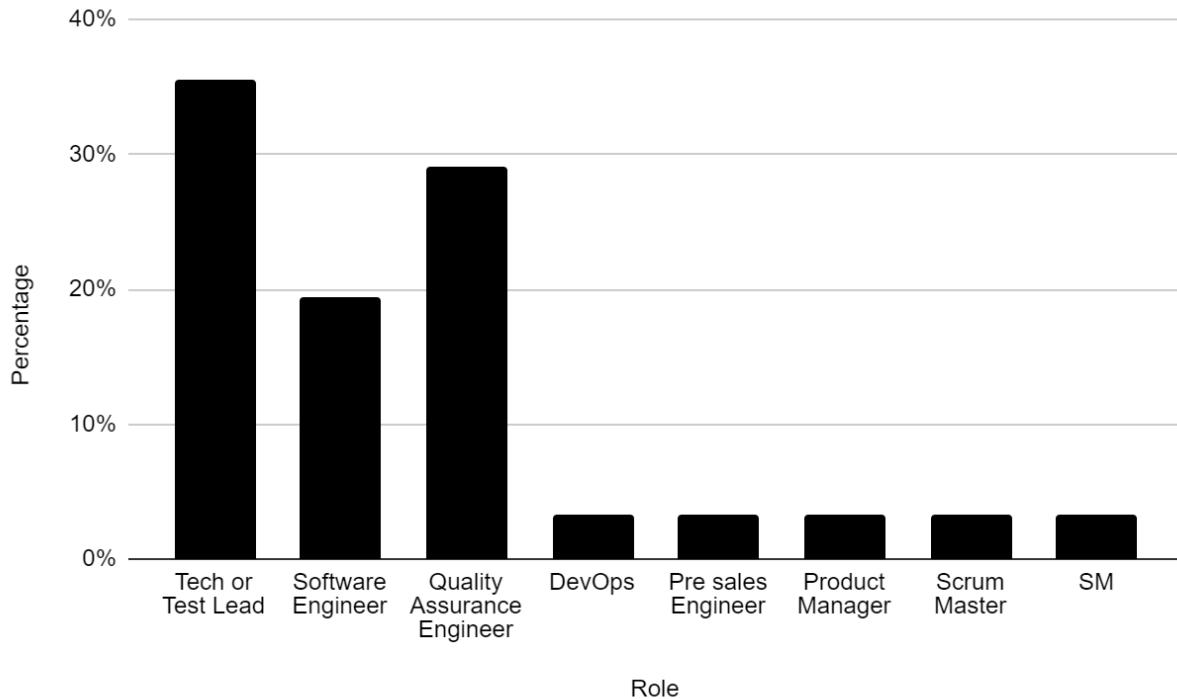
Through our paper, we have discussed the importance of performing load testing and the issue of the unavailability of a proper tool for stakeholders with zero experience in coding despite the large number of tools available for this purpose of load testing. Nonetheless, existing tools themselves have several drawbacks as well. We try to address these issues by introducing a novel tool for load testing to the community.

## 10. References

- “What Is Load Testing? Examples, Tutorials & More.” 2021. Stackify. February 5, 2021. <https://stackify.com/what-is-load-testing/>.
- Huaji, Zhu, and Wu Huarui. 2017. “Research on Web Application Load Testing Model.” 2017 *IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, December (December). <https://doi.org/10.1109/itnec.2017.8284961>.
- Project-Management.com. 2018. “Project Manager Roles and Responsibilities for Software Projects.” Project-Management.com. October 3, 2018. <https://project-management.com/project-manager-roles-responsibilities-software-projects/>.
- Kaushal, Shivangi. 2020. “LOAD TESTING ANALYZER for WEB APPLICATION.” *International Journal of Innovative Research in Computer Science & Technology* 8, no. 3 (May). <https://doi.org/10.21276/ijircst.2020.8.3.25>.
- Abbas, Rabiya, Zainab Sultan, and Shahid Nazir Bhatti. 2017. “Comparative Analysis of Automated Load Testing Tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege.” IEEE Xplore. April 1, 2017. <https://doi.org/10.1109/COMTECH.2017.8065747>.
- Jiang, Zhen Ming, and Ahmed E. Hassan. 2015. “A Survey on Load Testing of Large-Scale Software Systems.” *IEEE Transactions on Software Engineering* 41, no. 11 (November): 1091–1118. <https://doi.org/10.1109/tse.2015.2445340>.
- “Malfunctioning NHS App for Covid Vaccine Status Causes Travel Delays.” 2021. The Guardian. October 13, 2021. <https://www.theguardian.com/world/2021/oct/13/nhs-app-proving-covid-vaccine-status-malfunctions-causing-travel-delays>.
- ABC News. 2016. “Government, IBM Settle over ‘Census Fail,’” November 24, 2016. <https://www.abc.net.au/news/2016-11-24/government-ibm-settle-over-census-crash/8055784>.
- Technologies, Cigniti. 2014. “Classic Cases of Performance Testing Failures.” Blog by Cigniti Technologies. November 28, 2014. <https://www.cigniti.com/blog/2-classic-cases-where-performance-testing-failures-plague-large-organisations>.

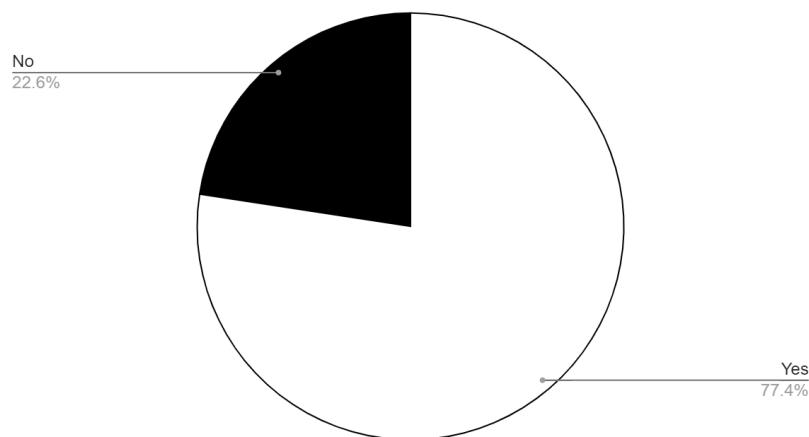
## 11. Appendix

### Appendix A - Technical Survey Participants

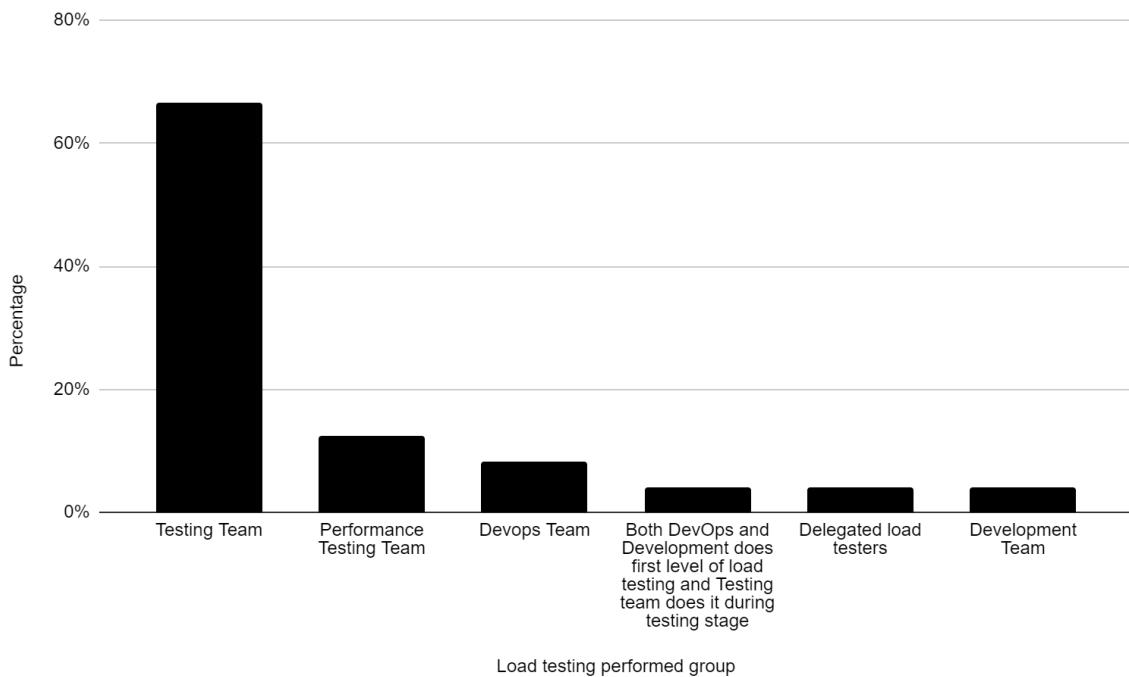


### Appendix B - Performing load testing on the current products

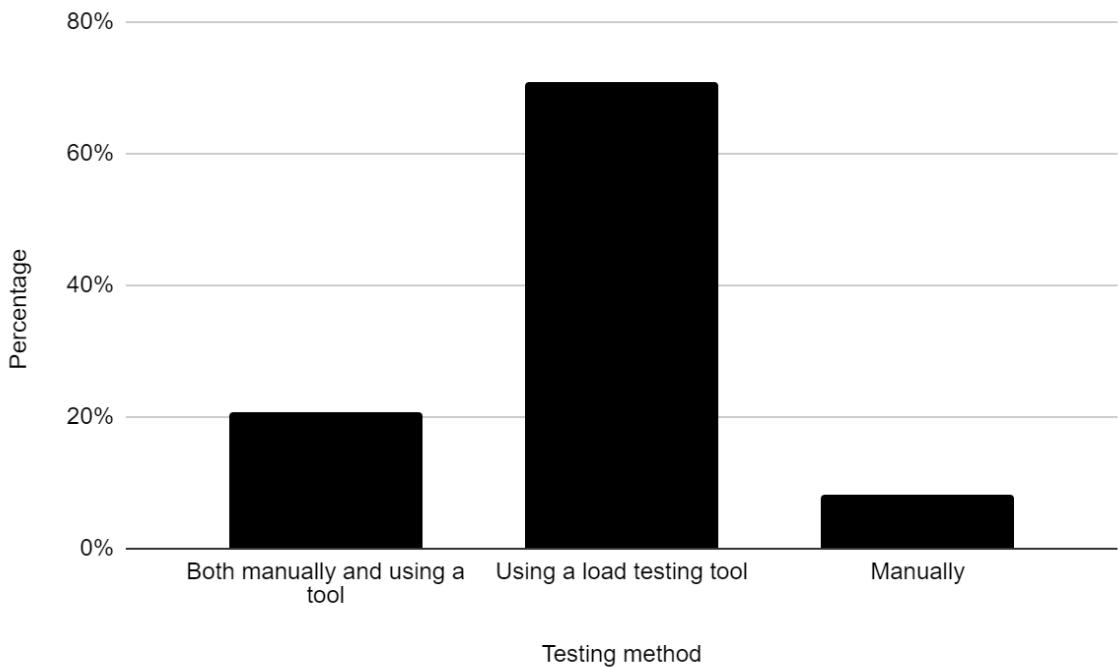
Do you perform load testing on your product(s)?



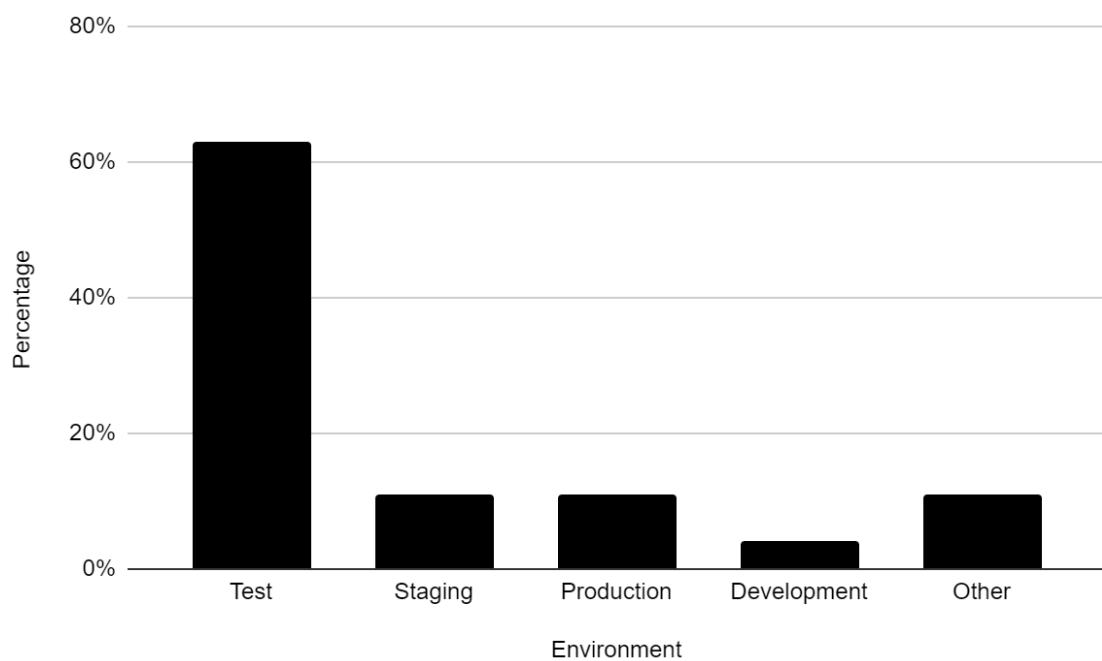
## Appendix C - Which team performs load testing



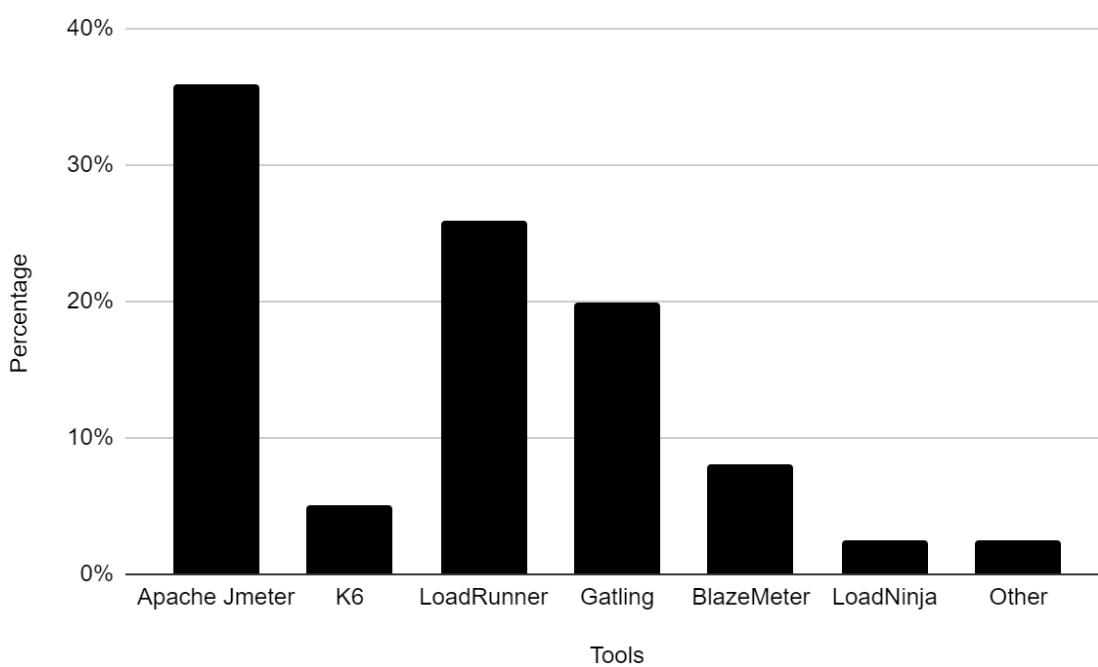
## Appendix D - Methods of load testing



## Appendix E - Environment of performing load testing

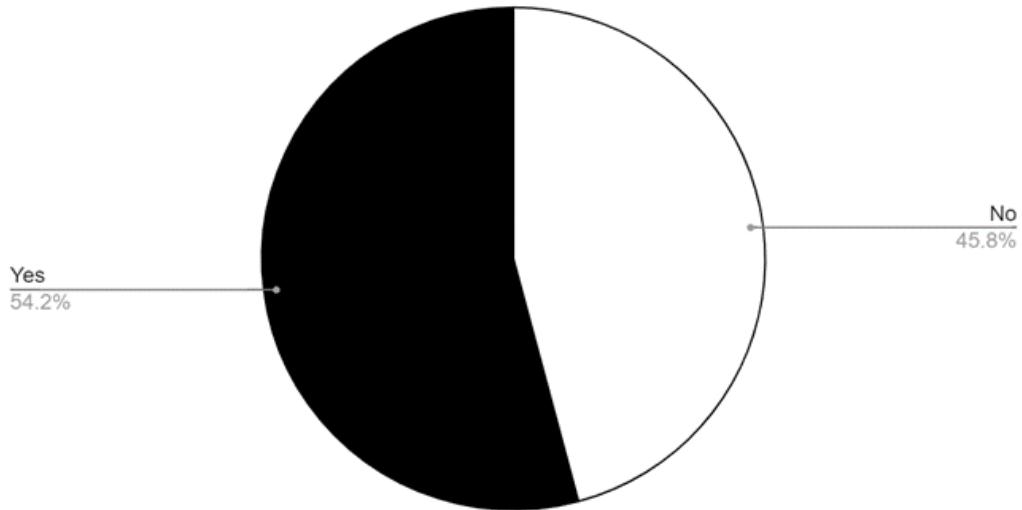


## Appendix F - Tools used for load testing

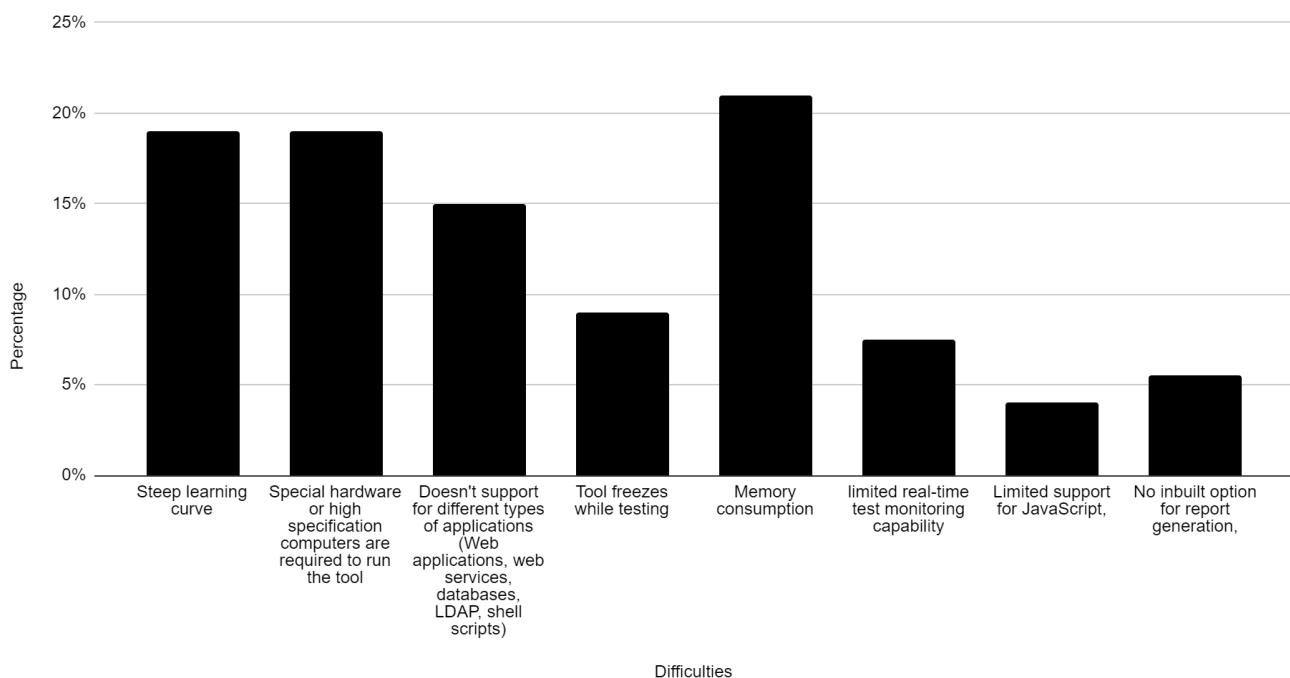


## Appendix G - Client complaints about the system crashing

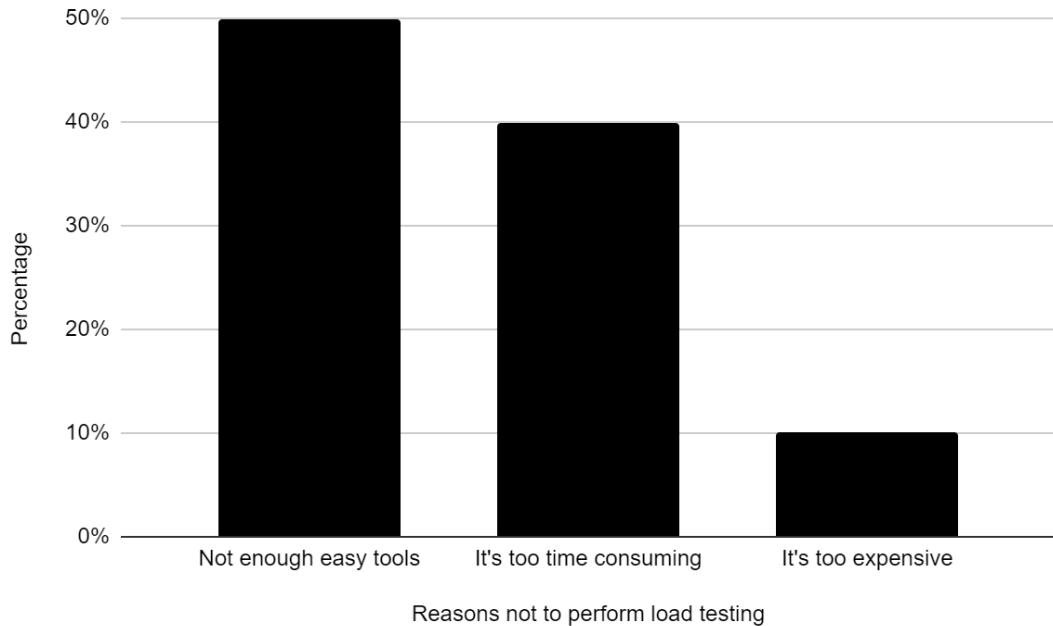
Have you received complaints from customers of the site being unresponsive or crashing;



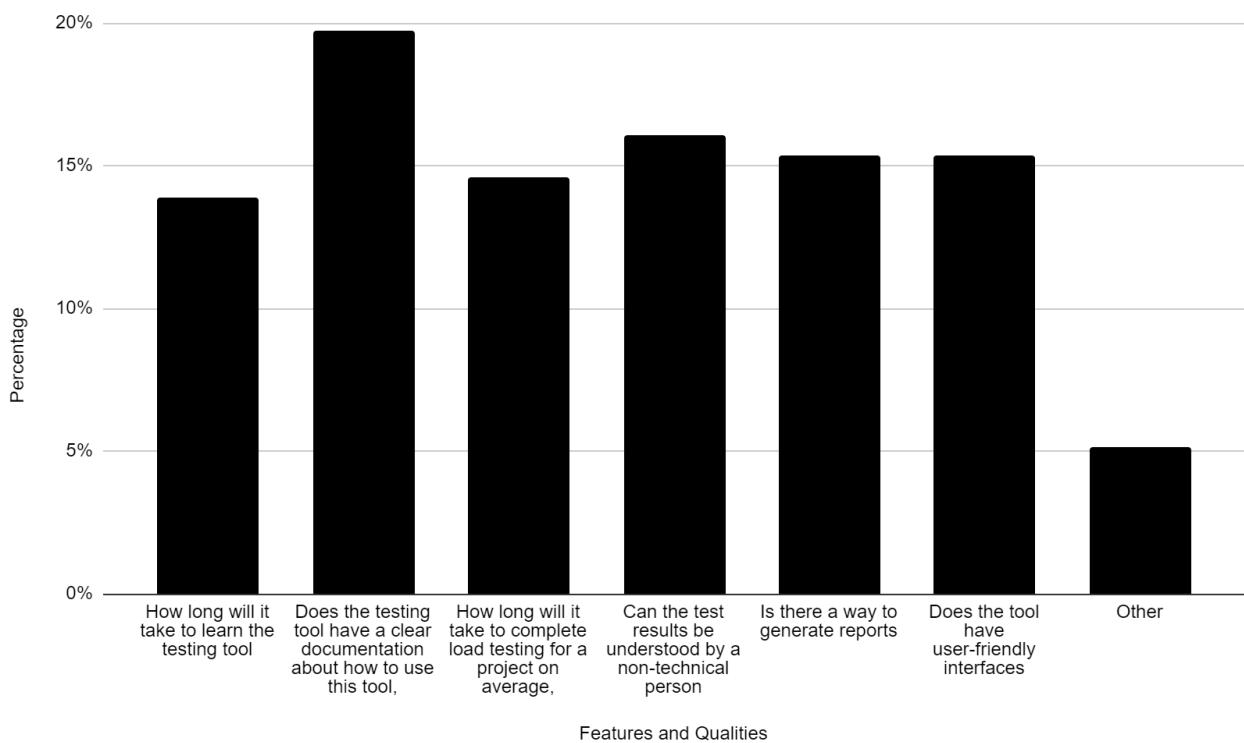
## Appendix H - Difficulties of using an existing tool



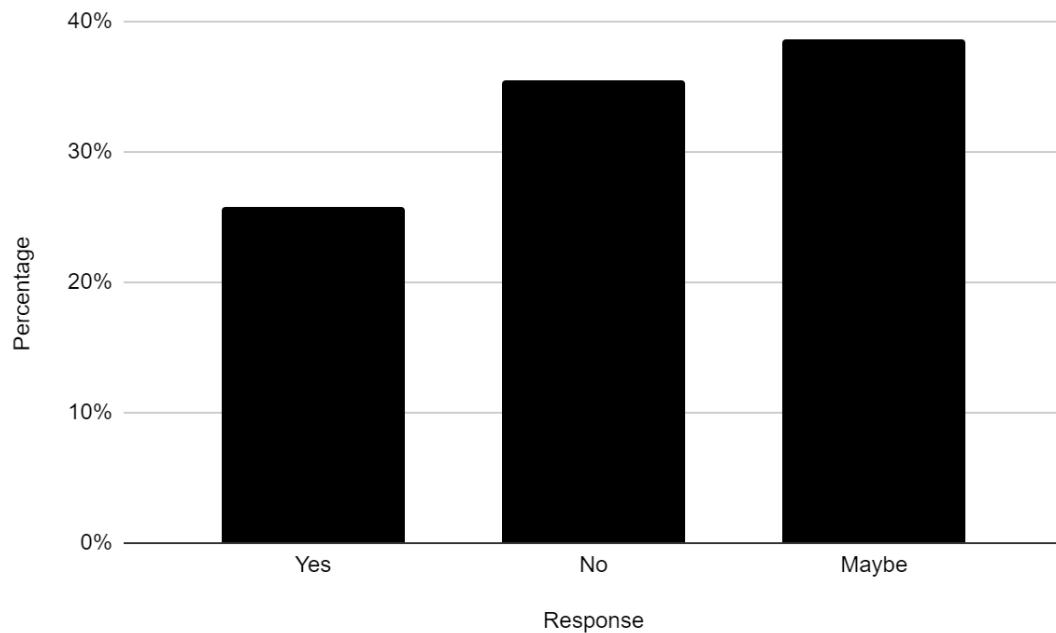
## Appendix I - Reasons for not performing load testing



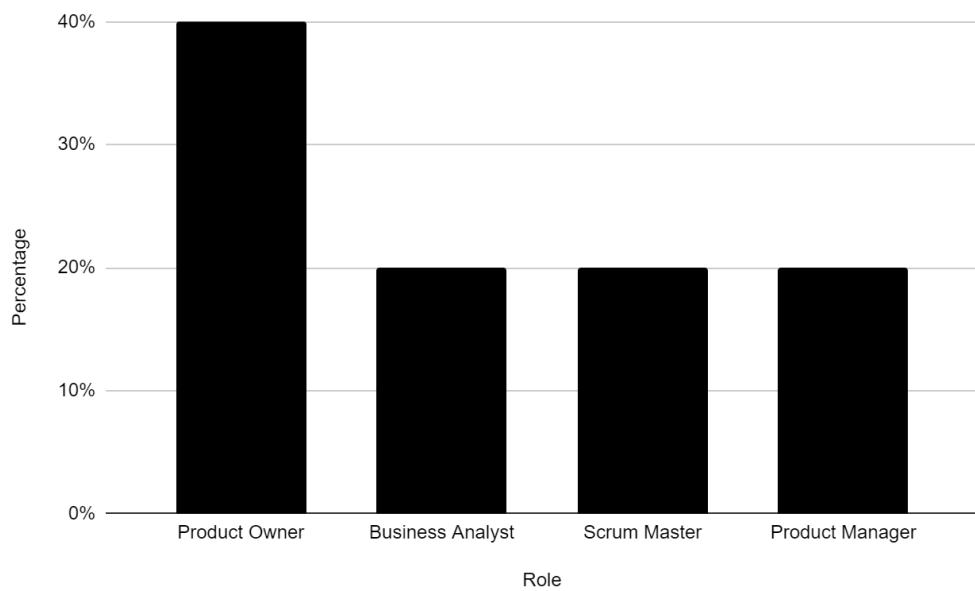
## Appendix J - Qualities and features of a good tool



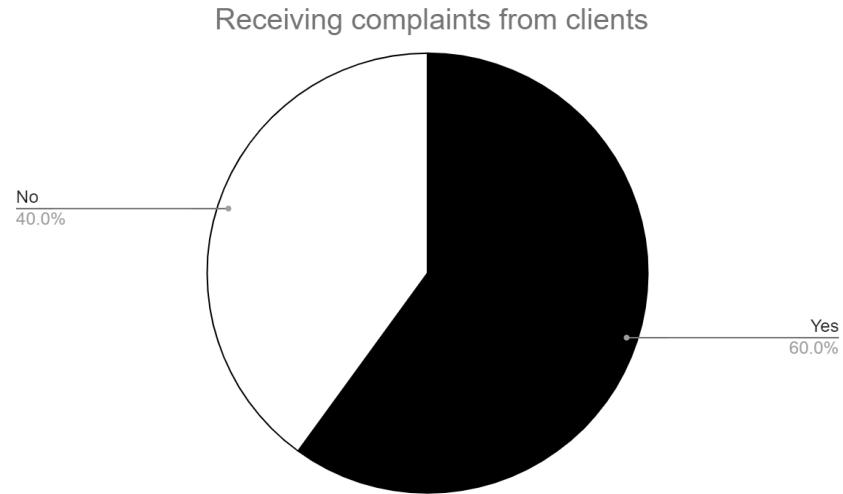
## Appendix K - Usability of existing tools for non-technical people



## Appendix L - Role of the respondent

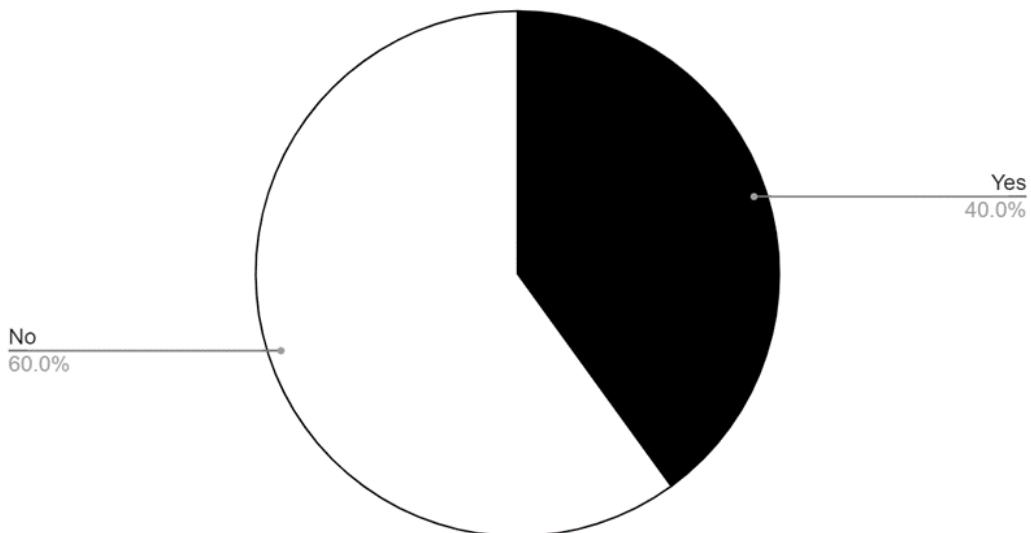


## Appendix M - Client complaints about system crashing



## Appendix N - Early experience of using a load testing tool

Have you used a load testing tool previously to simulate a large number of users concurrently to verify your site is able to function as expected?



# Risk-based testing reduces time to market of dynamic EDA tools

**Mohamed Bahnasawi**

Siemens EDA  
Siemens  
Cairo, Egypt  
[Mohamed.bahnasawi@siemens.com](mailto:Mohamed.bahnasawi@siemens.com)

**Ahmed Khater**

Siemens EDA  
Siemens  
Cairo, Egypt  
[Ahmed.khater@siemens.com](mailto:Ahmed.khater@siemens.com)

**Reem EIAdawi**

Siemens EDA  
Siemens  
Cairo, Egypt  
[reem.eladawi@siemens.com](mailto:reem.eladawi@siemens.com)

## Abstract

The integrated circuit (IC) industry uses electronic design automation (EDA) tools in its iterative cycle of analyzing, designing, and verifying ICs. Industry growth creates a need for improved and expanded functionality that performs processes with fewer resources in a shorter timeline with a very low risk of failures. The functional complexity of EDA tools continually increases as we strive to provide the most efficient and most accurate utilities.

When tool quality is critical, the risk of failure must be minimized. Risk-based testing (RBT) reevaluates the risks to steer test efforts in relation to customer priorities. With limited resources and time constraints, prioritizing test efforts is a must to catch critical bugs in early stages to fix them. Risk identification and analysis are critical components of targeted effort allocation.

In this paper, RBT is applied to “Create Random Array” (CRA) and “VIA Random Placement” (VRP), which are highly dynamic tools. We demonstrate that applying the RBT technique reveals critical bugs as early as possible, which can significantly reduce time to market while ensuring tool quality. The results include metrics for measuring and controlling the progress, efforts, and cost of test activities. We also show that RBT is applicable to any tool operating in a dynamic environment with limited resources.

## Biography

Mohamed Bahnasawi is a Senior Software QA Engineer at Siemens EDA with 5 years of experience in automation and testing.

Ahmed Khater is a QA Team Lead at Siemens EDA with 12+ years of experience in Software Developments and Testing Process.

Reem EIAdawi is a Test Engineer Director at Siemens EDA with 25+ years of experience in Software Development and Testing Process.

## 1 Introduction

Electronic design automation (EDA) tools have a long history of vigorous innovation, driven by the exponential expansion of the integrated circuit (IC) industry, which relies heavily on EDA tools during its long cycle of designing, analyzing, and validating results. To satisfy realistic design schedules and budgets, bigger designs require significantly higher designer productivity, creating a constantly evolving role for EDA tools. As a result of the increased functional complexity of EDA tools, particularly commercial ones, the necessity of quality process rises.

EDA development teams must find a balance between supporting new technology nodes and expanding the tools' capabilities when introducing new and expanded features to meet the expectations of EDA customers. Because of the intense competition among customers to support new technology nodes, EDA development teams aim for aggressive deadlines while maintaining staff skill levels. As new capabilities are introduced, the tools become more dynamic in dealing with more complex scenarios. At the same time, the customer's primary need is zero defects in the fabricated chips, which cannot be met without EDA tools with zero bugs. Even a few bugs in an EDA tool might result in irreparable and costly hardware defects. Some hardware defects cost hundreds of millions of dollars in the replacement process of defected chips (Thomas 2011).

Thus, high quality is not an option in EDA tools, as customers need qualified tools with almost zero bugs, within tight time restrictions. However, EDA tools have frequent upgrades. EDA software quality is also particularly difficult to evaluate because of its long runtimes, and the fact that the tools' outputs often contain tough iterated and numerical issues that are not always instantly verifiable. The quality process has also become more difficult because it must be completed, and results evaluated under time restrictions. However, regression testing is a must for each tool. Regression runs are standard for both hardware designs and EDA tools. Because very few modifications may occur in a single day, several bugs were discovered by using these runs. (Ng 2005)

Testing should unveil software defects that risk the product's mission-critical operations. However, software testing takes a significant amount of time. Testing can cost up to 40 percent of the overall of the total original cost of software development (Pressman 1995). Especially for EDA, due to its demanding time schedule, the testing process is done under extreme time pressure. In this environment, it is necessary to develop a strategy for prioritizing efforts and allocating resources to the software components that must be extensively tested to ensure that the software requirements are met.

Risk-based testing (RBT) uses risk assessment to drive all phases of testing process while reducing testing costs, such as time and resources, without affecting the quality of the tested tools. By focusing testing activities according to the risk assessment of the tool, which assumes that new, unclear, or undiscovered scenarios, and complex flows are more likely to fail (Felderer 2014). By identifying the risk factors in the tool's functional specifications and prioritizing the requirements based on these identified risks, the design and execution of tests become more efficient at covering critical situations in early stages, giving the development team time to fix detected defects without delaying the tool's delivery to customers.

This paper discusses the application of RBT in testing two EDA dynamic tools that have many testing scenarios to be covered due to their randomized flows. The paper will show the effect of using RBT in covering important flows and report their defects in early stages. The paper is organized as follows: Section 2 describes the approach of RBT and its main concepts and activities. Section 3 is a brief for the EDA tools that were tested using RBT approach. Section 4 presents the results and outcomes of the experiment. And finally, Section 5 summarizes the conclusions.

## 2 Risk-Based Testing Approach

Risk-based testing (RBT) is a testing technique that uses the risks of the software product as a guiding element to assist choices throughout the testing process. The tool's stakeholders should first identify the risks that will most probably impact the quality of the tested tool. And to be aligned with the concept of risk, the risk is a failure which hasn't occurred yet, but it may or may not occur in the future (Allam 2013).

The identified risks will be assessed, prioritized, and used in the guidance of the testing activities to be more powerful in reporting bugs and taking decisions early on, giving the development team enough time to fix any reported issues and then stakeholders reevaluate the risk of the tool to reassess used testing approaches and so on.

As shown in Figure 1 (Amland 1999), the RBT process has five key activities which are bounded in rectangles and they will be explained in this section.

### 2.1 Risk Identification

Risk identification is the activity used to detect any risks that can affect the project's ability to achieve its objective. This activity is a very early one as it is for reviewing the functional specification and product requirement documents. QA team performs this activity in parallel with the code implementation phase performed by the development team.

In this activity, it is necessary to review the functional specifications as developed by the development team to assess technical risks, detect any flaws, or unclear functionality, and to check whether there are any illogically supported flows.

Simultaneously, the product requirement documents should be evaluated to detect requirement risks such as illogical requirements, flows that conflict with the supported feature in the tool, and whether the functional specifications fulfill all the requirements.

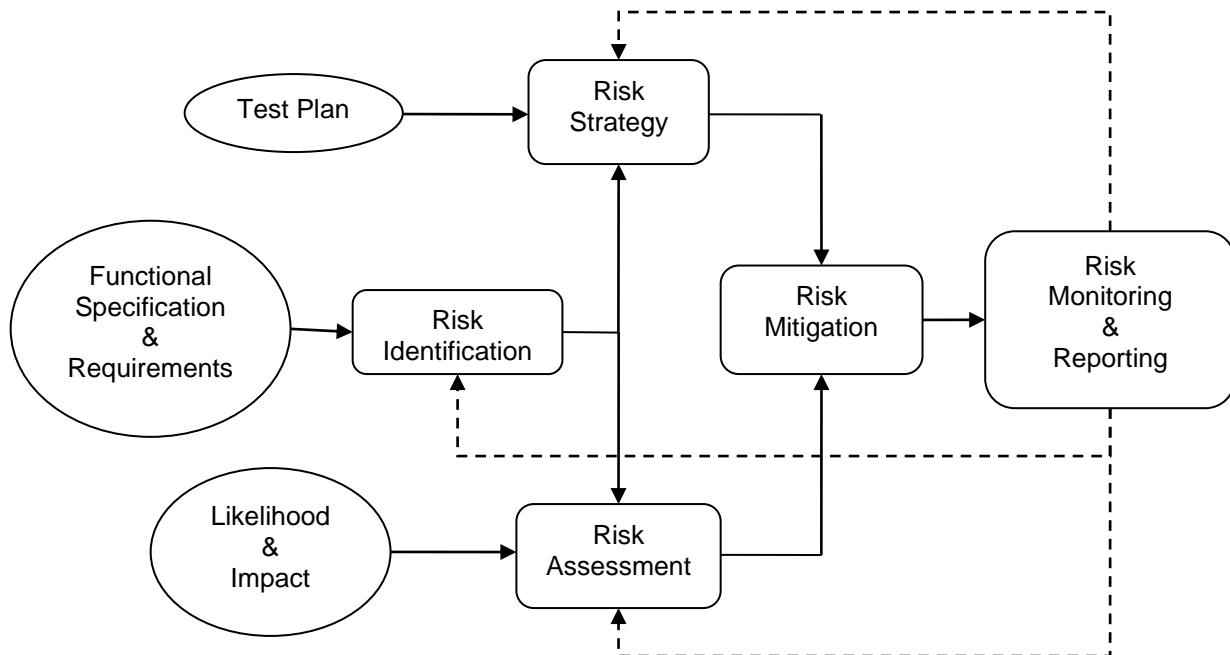


Figure 1: Risk Management Process

	3 Med	6 High	9 Critical
2	2 Low	4 Med	6 High
1	1 Low	2 Low	3 Med
	1	2	3
Business Risk (impact of defects)			

Figure 2: Risk Assessment by using grid 3x3

## 2.2 Risk Strategy

Before software testing begins, it is essential to think strategically about risks and understand what risk strategies are most likely to be successful, given the project context. Strategizing risks is used to develop testing procedures and alternatives for any contingent event. These plans will be used to guide risk management throughout software testing operations. During testing operations, four key risk strategies are employed: (i) Risk Avoidance is a technique of avoiding engaging in a feature or product because of the significant risk associated with it. It is also possible to disable or protect the code of this feature until all testing operations are completed. (ii) Risk Reduction which means taking steps to mitigate risks. This is accomplished by implementing tests provided in the test plan to expand the covered scenarios for the tested functionality. (iii) Risk Transfer, which is accomplished by outsourcing testing to a third party. This is because there is a lack of experience or resources. (iv) Risk Acceptance entails deciding not to deal with the risk or taking no action in response to it.

## 2.3 Risk Assessment

The identified risks will be assessed by the stakeholders of the tool in a brainstorming meeting. The testing team should describe the identified risks in detail with exampled scenarios. In this activity, the team provides a score for each risk from the perspective of (i) Likelihood of occurrence “Probability” (ii) the impact upon this occurrence “Consequence”.

The likelihood or the probability of the existence of a risk is focused on the technical risks. It comprises assessing the risk based on criteria such as software complexity, frequency of usage, potential defect locations and integration between legacy and new features.

The impact or severity of the risk is emphasized on business risk. As a result, the impact score is determined based on the criticality of the risk to the customer, the absence of solutions, and the customer's reliance on such a flow.

The levels or available scores for Likelihood and Impact differ from one study to another but in this case study, the grid of 3x3 (Figure 2) is employed to represent 3 levels for each of the two factors. By multiplying the two scores, the identified risks will be analyzed for prioritization.

## 2.4 Risk Mitigation

The testing team will define the testing strategy, the number of testing cycles for each risk level, the scenarios that will be tested for each risk, and the number of tests for each scenario at this phase. These

judgments are made depending on the available time frame and testing resources. The testing team will then begin the design and implementation phase.

## 2.5 Risk Monitoring and Reporting

These activities are for tracking and evaluating the tested risk levels with the stakeholders. Because the risk process has become an integral element of the development process, it is critical to assess its effectiveness. If there is any vulnerability because of the risk management process that is not monitored and corrected, it will serve as a risk. The results of risk monitoring methods can be utilized in the development of new tactics and update current techniques that have proven unproductive (Blancher 2013).

## 3 Case Study

The tools examined using the RBT technique will be presented in this section. "Create Random Array" (CRA) and "VIA Random Placement" (VRP) are highly dynamic tools. Each of them has its own functionality but both are generating random output and have a high level of complexity

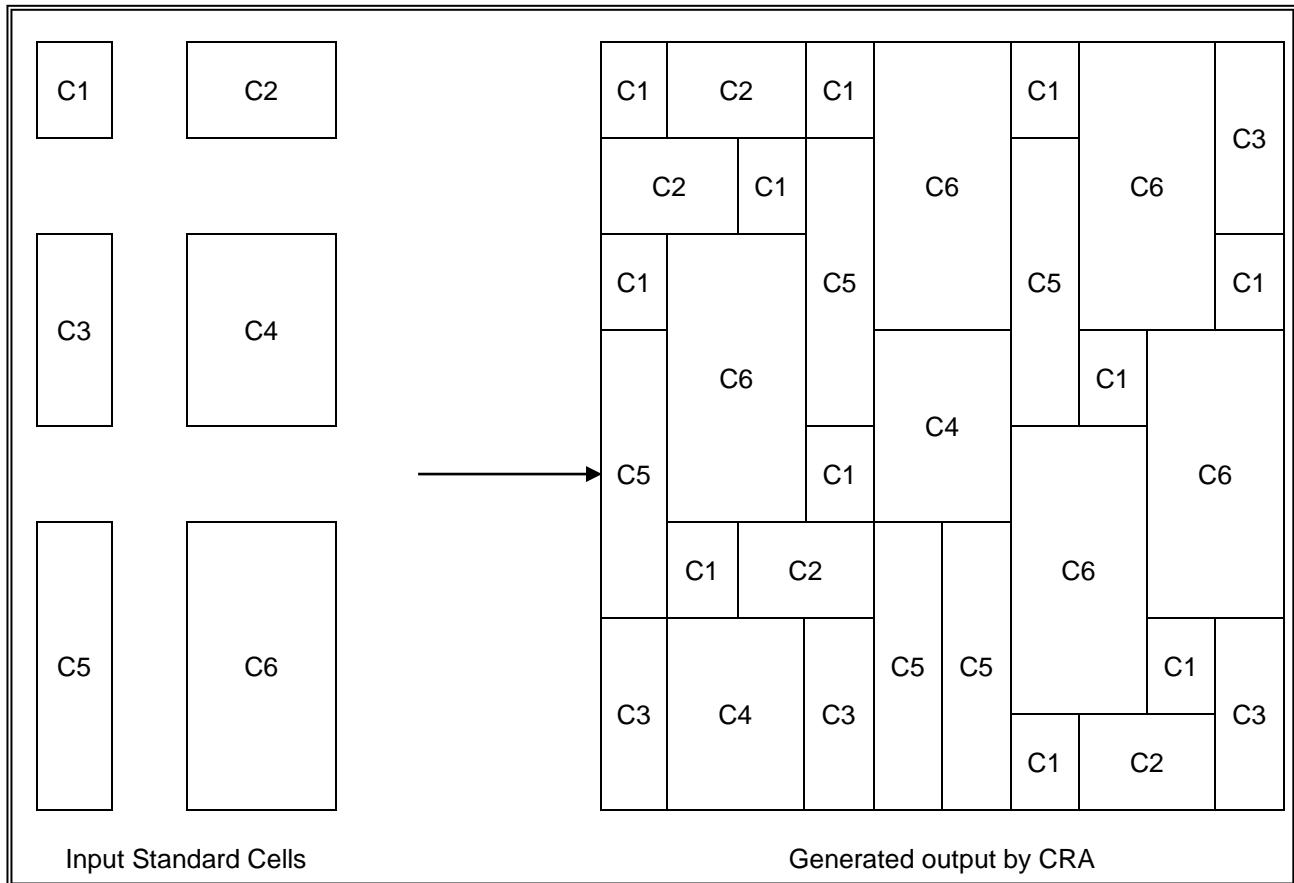


Figure 3: Example of input standard cells and generated output by CRA

### 3.1 Create Random Array (CRA)

For developing new technology nodes, it is necessary to have several layouts for process calibration. Specially for standard cell development teams as they need to make sure that any legal placement of their cells, with any combination of nearby placement or abutments, should produce a robust layout. Hence, they need to test a large variety of possible placements.

However, no realistic designs are accessible in the early stages. This highlights the significance of having synthetic layout generators. CRA is one generator that generates random arrays from provided cells. Where the user provides a tool list of standard cells and the quantity of random arrays to be created.

So as shown in Figure 3, CRA is a utility that creates random placements of standard cells. By tackling the difficulty of multi-height standard cell placement, this tool aids quality assurance of standard cell libraries including hundreds or thousands of cells. The CRA utility may be used by library developers to confirm that all placement possibilities of surrounding abutted cells are allowed.

CRA takes an input layout with certain standard cells to randomly place the multi-height standard cells into arrays. The adjacent standard cell arrays are "Design Rule Check" (DRC) compatible. This application generates a layout file for each standard cell array placed, which may be used to design a placement solution to enhance the standard cell library.

### 3.2 VIA Random Placement (VRP)

The "VIA Random Placement" VRP flow generates and places vias in the design layout at random using a set of input criteria. Using the VRP flow allows "computer-aided design" CAD engineers and designers to do early pattern analysis for "lithographic" LITHO simulation on designs with metal and through layers to identify probable sources of LITHO hotspots.

Running VRP flow necessitates the application of DRC criteria, such as minimum enclosure and spacing requirements, as well as specifications for the types and size of the created vias. The flow randomly inserts and positions vias based on the input, while adhering to the rule checks and via requirements. According to the regulations, through placement happens at places of overlap and intersections between two layers.

The VRP requires an input layout database along with files for the via spacing and enclosure rules. The output is a layout database with the generated random vias with respect to the spacing and enclosure rules. This layout can be merged with the original one which is used as input to VRP and then it will be used in LITHO simulation activities.

## 4 Results

In this section, the metrics of applying RBT on each of CRA and VRP will be shown in detail. The metrics includes the time spent for each activity in the RBT process, the number of identified risks, the type of testing for each risk and the result of prioritizing them.

### 4.1 Create Random Array (CRA)

By studying the CRA feature under test, the testing team identified 36 risks and Table 1 shows the results of Risk Assessment activity.

Table 2 shows the planned time for the testing activities of CRA and as indicated in the table, the overhead of using RBT in testing the feature is roughly 90 minutes, with the balance of the time planned for regular testing activities that will be used in RBT method or various approaches. So, the overhead time is less than 4% of total time.

Risk Assessment	Testing Categories				Total
	Functional	Integration	Performance	Negative	
Critical	4	5	3	3	15
High	5	0	1	0	6
Med	7	1	0	3	11
Low	3	0	0	1	4
Total	19	6	4	7	36

Table 1: CRA Metrics | Risk Assessment

Testing Activity	Time in mins
Risk Identification	~60
Risk Assessment	~30
Prepare the testing environment & input data	~480 (1 working day)
Testing scenarios for each risk including automation	~53-70 (Depending on scenario's complexity)
Total	~2370 (5 working days)

Table 2: CRA Metrics | Time planning of testing activities

RBT produced a priority ranking of potential bugs having the greatest impact to customers. During the Critical Risks testing, 5 bugs were discovered, all of them are found through testing the tool with the data provided by the customer and interrupt the main flow for customer scenarios. As a result of the early detection of these bugs, the development team had more time to fix them to meet the release cycle and improve the tool's quality for all usage scenarios that the customer had specified. Thus, during the critical risk testing phase, stakeholders desired to continue with the Risk Reduction Strategy as it was very essential to deliver these flows with high quality to the customer.

Due to time constraints, some lower priority features received only partial testing as some scenarios were broken by the discovered bugs. Since these were of lower importance and there was insufficient time to fully test them, the stakeholders preferred not to delay the release and to change the testing strategy for them from Risk Reduction Strategy to Risk Avoidance Strategy by protecting these flows under beta variable until all testing operations were completed.

Finally, in most testing approaches, performance testing is conducted in the latter stages of the testing process after functional testing is completed. Because performance was critical to the customer in this feature, there are some risks which are categorized under performance testing ranked as critical or high in Risk Assessment. So, these scenarios were tested before some scenarios under functional testing.

## 4.2 VIA Random Placement (VRP)

Regarding VRP feature under test, the testing team identified 28 risks and the same as CRA, Table 3 shows the results of Risk Assessment activity.

Table 4 shows the planned time for the testing activities of VRP and as indicated in the table, the overhead of using RBT in testing the feature is around 80 minutes which is less than 6% of total time.

During the Risk Identification process, there is a scenario that is not well stated in the requirements documents and functional specifications. This case revealed a discrepancy between the intended output of the code in implementation and the expected result by the customer. As a result, this is considered a bug, and it is detected at a very early stage of testing. As a result, the development team was given the opportunity to resolve this problem as soon as feasible.

There were no further bugs introduced throughout the testing scenarios for this functionality. As a result, the testing strategy utilized to test this feature is the Risk Reduction Strategy. This is accomplished by testing prioritized scenarios that have been validated through the Risk Assessment process to increase the tool's coverage and satisfaction.

Risk Assessment	Testing Categories				Total
	Functional	Integration	Performance	Negative	
Critical	6	5	0	2	13
High	3	0	0	6	9
Med	0	1	0	0	1
Low	1	0	2	2	5
Total	10	6	2	10	28

Table 3: VRP Metrics | Risk Assessment

Testing Activity	Time in mins
Risk Identification	~60
Risk Assessment	~20
Prepare the testing environment & input data	~240 (0.5 working day)
Testing scenarios for each risk including automation	~20-60 (Depending on scenario's complexity)
Total	~1440 (3 working days)

Table 4: VRP Metrics | Time planning of testing activities

It appears that RBT did not introduce any advantages over traditional testing techniques as the discrepancy between functional specifications and requirements documents is always detected in a very early stage in the traditional testing techniques, but in fact, the goal of testing is to increase the quality and satisfaction with the tool, not just to catch bugs. RBT was reporting at an early stage the satisfaction of the testing team with the tool quality, and it can be used in reallocating resources for testing other low-quality tools.

## 5 Conclusion

Using the RBT approach in these case studies demonstrated that the technique focuses on flows that are more likely to fail or are more vital to the customers. Consequently, it allows stakeholders to make early decisions to lower the risk of the delivered feature or product, hence shortening their time to market.

These case studies illustrate two distinct situations. A low-quality feature that RBT highlights concerns early on, and it gave the development team the chance to fix them and regarding some lower scenarios which there is not enough time to test them, the stakeholders preferred to use a different Risk Strategy since these scenarios are not important to the customer and not fully qualified. The second situation had only one issue in the provided documents and it was of good quality, thus the Risk Strategy adopted was to lower the risk by making early fixes to detected bugs and increasing satisfaction with the feature before going live.

The overhead in adding RBT to the testing process is around 4-6% of the total time planned for the full testing process. By applying RBT technique, major defects were usually reported at an early stage. Also, instead of blindly following traditional testing process, such as applying performance testing at the end, RBT raised awareness of the critical need for high performance over other features in this release and the order of testing became dependent on the customer's needs and the risky flows.

## References

- Nicely, Thomas. 2011. "Pentium FDIV flaw FAQ". trnicely.net. Archived from the original on June 18, 2019. Retrieved June 18, 2019. <http://www.trnicely.net/pentbug/pentbug.html> (Internet archive)
- Ng A, Markov IL. 2005. "Toward quality EDA tools and tool flows through high-performance computing". *Sixth international symposium on quality electronic design (isqed'05) 2005 Mar 21* (pp. 22-27).
- Redmill, Felix. 2004. "Exploring Risk-based Testing and Its Implications" *Software Testing, Verification and Reliability, Vol. 14, No. 1, March 2004*
- Pressman, R. 1995, *Engenharia de Software*. 1st ed. Makron Books, São Paulo, Brazil.
- Felderer, M. and Schieferdecker, I., 2014. "A taxonomy of risk-based testing". *International Journal on Software Tools for Technology Transfer*, 16(5), pp.559-568
- Alam, Md. Mottahir. 2013. "Risk-based Testing Techniques: A Perspective Study" *International Journal of Computer Applications (0975 – 8887) Volume 65– No.1, 2013*
- Amland, Stale. 1999. "Risk Analysis Fundamentals and Metrics for software testing including a Financial Application case study" *5th International Conference EuroSTAR '99, November 8 - 12, 1999, Barcelona, Spain*
- Blancher, Nicolas. 2013. "Systemic Risk Monitoring (—SysMo||) Toolkit – A User Guide" *IMF Working Paper (International Monetary Fund)*.

# Phased Testing

## Testing Systems Undergoing Change

Baubak Gandomi

[gandomi@adobe.com](mailto:gandomi@adobe.com) / [development@gandomi.com](mailto:development@gandomi.com)

## Abstract

In this paper we analyze the challenges involved in verifying and testing the success of system changes. These operations are often overlooked when testing, however as systems are moving more and more to continuous delivery, such tests become increasingly relevant. Some the more common system changes are:

- Database changes
- Application upgrades
- Server and Cloud changes

In our experience, validating that a system works after a system change is very complex. From the outset, two initial challenges stand out:

- How do we make sure that the problem is related to the change, and is not inherent to the product?
- How can a problem be clearly identified and reproduced?

In our view, traditional test approaches have limitations when testing for change. Instead, we present a new approach to writing tests that allows us to completely test how a use case is affected by a system change. We call this approach Phased Testing. Phased Testing allows our test scenarios to be leveraged for validating system changes. This is done by empowering our scenarios in two ways:

1. We allow test scenarios to be interruptive.
2. A test scenario is declared only once, but we are able to execute it with all the possible interruptions it may be subjected to.

## Biography

*The author, Baubak Gandomi, studied at the Mid-Sweden University, Universität Leipzig (Germany) and finally graduated from the university of Stockholm, Sweden with a degree in Computer Sciences.*

*He is currently a test automation architect at Adobe, a position he has held for the last 9 years. Apart from upgrade tests he is very interested in continuous delivery methodologies, cross-product/teams testing and finding measurement methods for assessing functional and transactional coverage.*

*When not solving Quality and Test related problems, Baubak practices the martial art of Ninjutsu, and promotes the ugliness of the world through his blog, Ugly City Guide.*

Copyright Baubak Gandomi June 6, 2022

# 1 Problem Statement

Given the popularity of continuous delivery processes, it is becoming increasingly important to ensure that application changes do not cause a loss of service. Some of the more common system changes are:

- Database changes
- Application upgrades
- Server and Cloud changes

The traditional migration scenarios usually are a combination of two cases:

1. System changes on a system with existing data generated by executing old scenarios,
2. The execution of scenarios on a changed system.

Our experience is that validating that a system works after a system change is quite hard. We see the following challenges:

- How do we make sure that the problem is related to the change, and is not inherent to the product?
- How can a problem be clearly identified and reproduced?

## 1.1 Complicated Data

Traditional testing is insufficient because it often does not consider the data prior to the system change. Systems performing such tests usually manage a database that is regularly updated. Managing such databases is quite complex and is hard to scale. We identify the following problems:

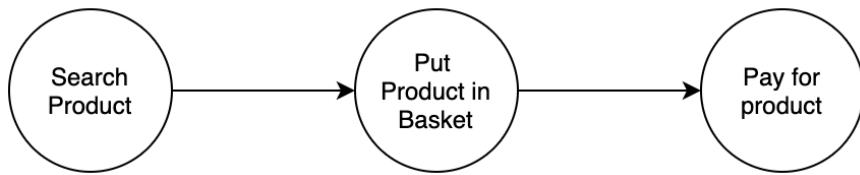
- Mapping data to a specific test scenario is complex
- Mapping data to a specific product/application version is complex
- Updating the data is cumbersome.

## 1.2 The Time Problem

Most test scenarios imagine a static system. They will perform a set of actions and verify that the end result is as expected. There are however, two problems with the traditional approach:

1. User Transactions are not usually atomic. The way we use services is rarely a sequence of events with a predefined time interval between each action.
2. We are relying more and more on remote services of various kinds. Services can be updated while you are in the middle of a transaction. This is especially true for systems using continuous delivery.

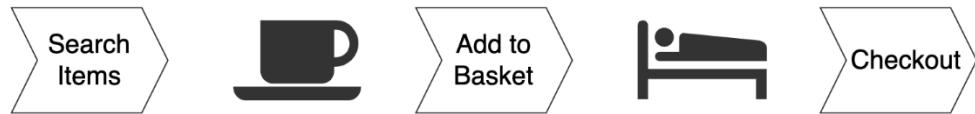
This has the consequence that a system change can happen during a transaction.



*Figure 1 : A simple e-commerce scenario*

An example is a simple retail site. As users, we will be performing searches, adding items to our basket, and finalizing the transaction. It is rare to perform all the steps in one go.

We usually add items to our basket, and even wait a few days before finally deciding on buying the goods we have selected. The full transaction, in reality, lasts a few days. System changes can arise at any point.

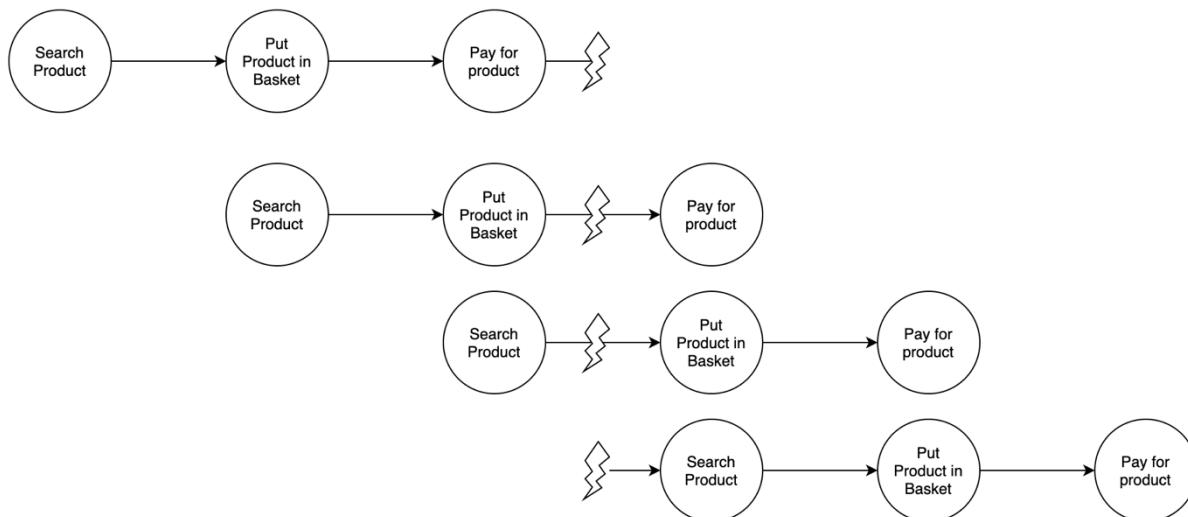


*Figure 2 : What can happen between actions*

Testing the example above would require us to first allow the scenario to be interrupted. Testing it in relation to a system change will require us to rerun the scenario four times in order to cover all of the possible cases within the transaction. This can be seen in Figure 3 below.

Traditional test frameworks were originally created for writing unit tests, and, because of this, they do not intuitively incorporate events or interruptions during the execution of a scenario.

In our first attempt, we were able to introduce interruptions in our test scripts by using conditional statements. We discovered that writing the scenario above would require us to rewrite the original scenario at least twice. This level of code copying grows with the number of steps. It introduces a complexity and a redundancy proportional to the number of steps.



*Figure 3 : The simple scenario regarding system changes. (The lightning sign indicates an interruption.)*

For example, the scenario in Figure 3 will require us to rewrite the original scenario in Figure 1 at least twice. Any subsequent changes to the original scenario will have to be carried over to the copied versions. We have added an example of this in [Appendix A](#).

## 2 Phased Testing

We developed Phased Testing to address the problems highlighted in the previous section.

Phased Testing allows our test scenarios to be leveraged, when needed, for validating system changes. By their very nature they also address the problem of test data. In this section we will be presenting the main concepts and building blocks behind this concept.

We have tested this approach by implementing it in the TestNG test framework. We will describe the implementation later in the section on [Implementation](#).

### 2.1 Phases of a Test

When a test validates the correctness of a change, also called a “Phased Event”, we make a distinction between the steps executed before and after a system change. Each group of steps is executed in its own “phase”:

- **Producer:** The steps executed before a system change.
- **Consumer:** The steps executed after a system change.

The choice of the names is regarding the data being generated by the steps. The steps in the “producer phase” will be generating data. The steps in the “consumer phase” depend on the data generated in the producer phase.

Another way to look at it is that the steps in the producer phase will put the user of the scenario in a state, in which we expect them to find themselves after a system change.

A test scenario may also be executed outside of the notion of a “Phase”. In this case, the scenario is executed like any other test scenario without interruption. This has the advantage that you do not have to write dedicated scenarios. Your scenarios will be used on a regular basis for testing the normal functioning of your application. It is only when you decide to execute the in phases that it will be interrupted.

### 2.2 Approach

The guiding principle in our approach is that test scenarios should create the data they need. This may not be new, but it is an approach we find necessary for the solutions we present here.

What we propose is to assume a different approach to writing tests. We present two main paradigms:

1. A test scenario should be interruptible.
2. A test scenario is declared only once, but we should be able to execute it with all the possible interruptions it may be subject to.

To make our point, let us imagine a simple scenario with three steps:



*Figure 4: Our scenario presented in steps*

The first paradigm is that we expect that the scenario to be paused at any time along the timeline of its execution. In the example above, it should be possible to interrupt the scenario at any point around steps 1, 2 and 3. The tester should be able to pause a scenario at any step and carry-on where you left off whenever you want. This constitutes what we call a **Phased Test**.

The second paradigm states that we define a scenario only once, but it can either be executed as a normal scenario without interruptions, or it can be executed with all the possible interruptions. This execution mode is called "**Shuffling**".

## 2.3 Execution Modes

A phased test needs to define how it is executed when it is in a phase. This is called execution mode. We currently identify two execution modes:

- Shuffle Mode: The test is executed with all the possible interruptions it may encounter.
- Single Run Mode: The test is always interrupted at the same step.

### 2.3.1 Shuffled Mode

Shuffled execution will execute the scenario with all the possible interruptions. A scenario with 3 steps will be executed 4 times. Each execution instance, uniquely identified by the interruption point, is executed within a context we call a "Shuffle Group". Henceforth we will use the notion Shuffle Group to designate the execution instance of a phased test scenario.

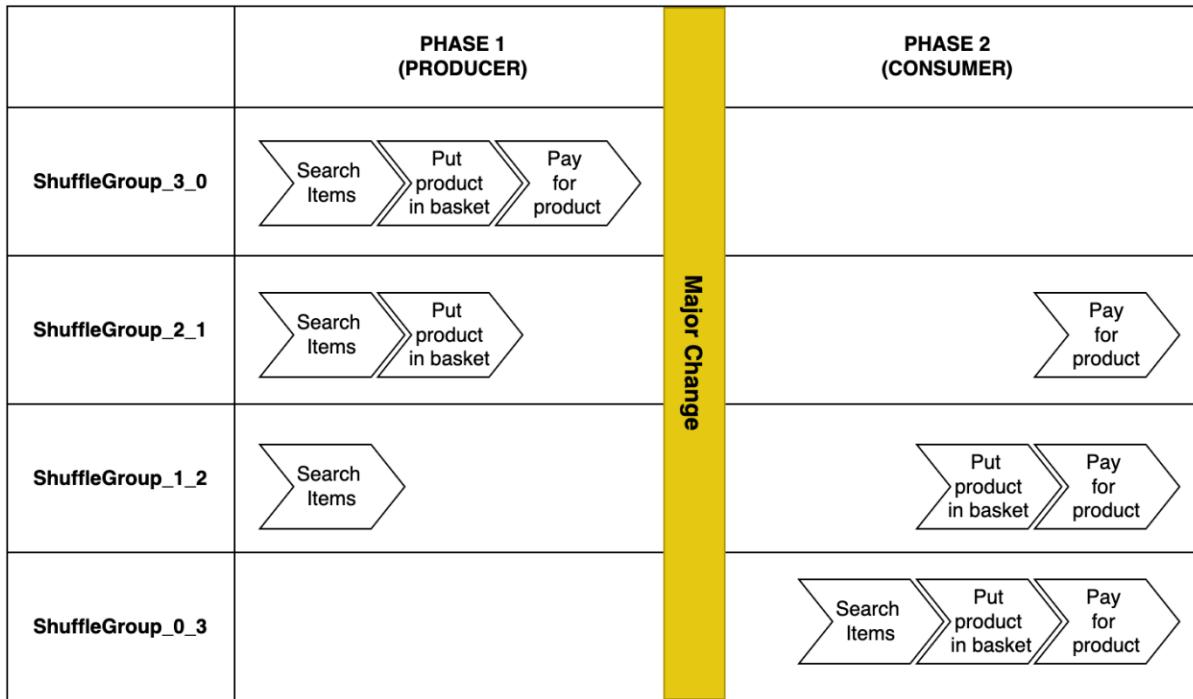


Figure 5: Shuffled Execution Mode

In Figure 5, although we have four executions, they remain the same scenario as in Figure 4: Our scenario . After the interruption, the test will continue where it left off.

We execute the scenario without interruption before and after the system for the following reasons:

- We can see if the scenario was working before the change.
- We generate data that will be used to test the change itself.
- We can see if the scenario can be executed completely after the system change.

### 2.3.1.1 Shuffle Groups

As mentioned earlier, each execution of a scenario with its interruption is uniquely identified by an execution context we call a “Shuffle Group”. This allows us to differentiate between the executions and to highlight the step in which the interruption occurred within the scenario execution.

We uniquely identify the Shuffle Group by the number of steps that are executed in each phase. For example, a scenario with 3 steps that is executed in Shuffle Mode, will yield the Shuffle Groups:

- 3\_0
- 2\_1
- 1\_2
- 3\_0

In the Shuffle Group 2\_1, we know that the interruption occurs after the second step, whereas in Shuffle Group 1\_2 the interruption happens between steps 1 & 2.

The number of Shuffle Groups in a Phased Scenario when executed in a Shuffled Mode is:

$$\text{Number of Shuffled Groups} = \text{Number of Steps} + 1$$

### 2.3.2 Single-Run Mode

Some use cases and scenarios are static in nature. By this we mean that the scenario will always be interrupted at the same step. The typical use case is when the interruption is due to a time-consuming process, like the heavy processing of data or the dependency on an external service. This requires an interruption at a specific step that will not change.

When a scenario is marked as “Single-Run”, it will only be executed once, and always interrupted at the same step.

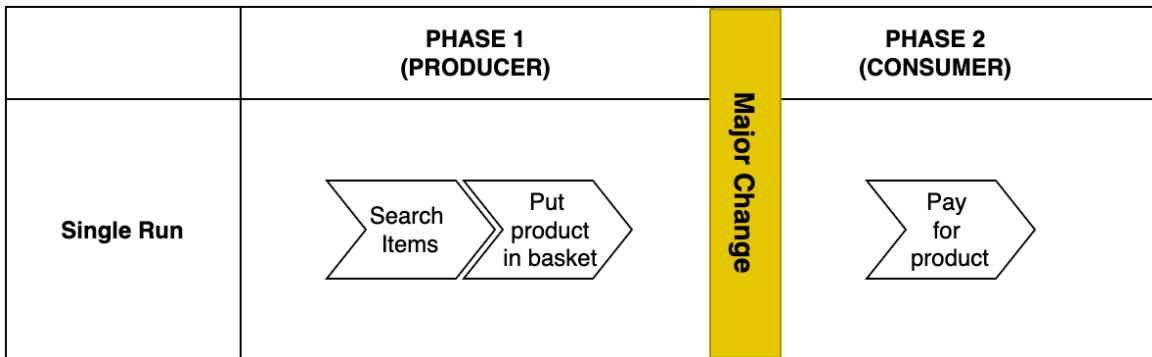


Figure 6: A single-run execution context

In Figure 6, the test scenario will always be interrupted after we put the product in the basket, and after the system change it will continue with the last step.

One of the main differences between the Shuffled, and the Single-Run modes is that in many use cases for the Single Run, the scenario cannot be executed in a non-phased mode.

## 2.4 Maintaining the Context

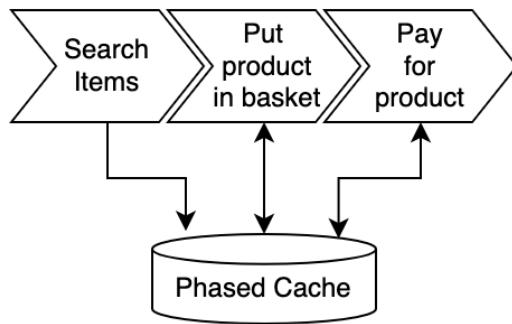
We need to set some mechanisms to ensure that Shuffle Groups can carry on wherever they are interrupted. These mechanisms concern the management and communication of data in Phased Tests. To implement these mechanisms, we need to enable two types of data communication in the system:

- Cross-Step Communication
- Cross-Phase Communication

One of the challenges when executing tests in Shuffle Mode is that the scenario is executed multiple times. We have introduced a notion of scope that requires that all data generated and retrieved are only visible within their Shuffle Group.

### 2.4.1 Cross-Step Communication

We need the steps to be able to carry on where the previous step left off especially if we have had to interrupt the tests to update the system. This means that they need to have access to, and to recognize the data generated in previous steps. Since our implementation of the Phase Testing anticipates an interruption in the test execution we cannot rely on global variables to pass information between steps. Consequently, each step stores the values needed for the following steps in a cache.



*Figure 7: Communication between steps*

We use two methods in a scenario to explicitly manage data in the Shuffle Group context:

- **Produce** allows a step to store data for future steps
- **Consume** allows a step to access data stored in previous steps

Because early steps will produce data, which is later consumed by the following steps, it is important to have a predictable order of execution between the steps. This is achieved differently based on the language and the used test drivers. We will describe our method in the section on Implementation and [Defining a Phased Test](#).

The Phased Testing framework must also store information regarding the state of the Shuffle Group. This includes information such as:

- The current state of the Shuffle Group.
- Errors in previous steps.
- Duration information.
- The phase in which the error occurred.

This implicitly stored information allows us to represent the Shuffle Group as a whole test execution, where the results reflect the whole execution, and the duration is that of the full execution of the Shuffle Group.

### 2.4.2 Cross-Phase Communication

We do not make a big distinction between steps in the same or different phases. Like in the case of intra-step communication, the steps following a failure will be skipped for that context.

There is one main difference, however. At the end of the “producer phase” the cache is exported to a file, which is later consumed when the tests are executed in the “consumer phase”. The consumer phase imports the cache file and makes the data available to the phased tests in that phase.

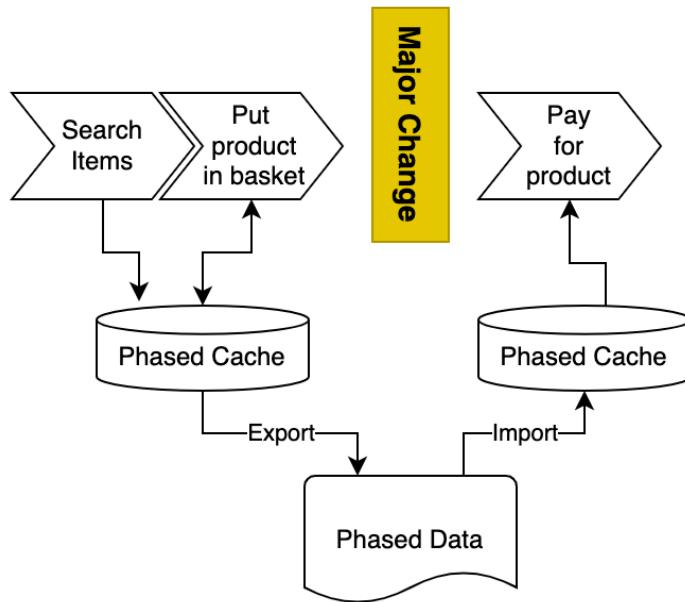


Figure 8: Communication between steps in different phases

### 3 Implementation

We implemented the Phased Test approach using the TestNG framework. The Data Provider notion used in TestNG, is particularly helpful, as it allows us to dynamically create a context in which all the steps of a Phased Test can be executed.

The Phased Test framework performs the “shuffling” when needed and interrupts the Shuffle Groups at the expected locations.

In our current implementation of the framework, the framework does not perform the system change itself. In our model and implementation, this is orchestrated by an automation/build system. We will cover more on this aspect in chapter 3.4 on [Orchestration](#). The system change, however, can be driven by the tool that is called from within this framework, but we do not include this aspect in this article.

#### 3.1 Defining a Phased Test

To implement Phased Testing we had to review the notion of a test in the test frameworks. In most frameworks the test is a method. However, in our implementation of Phased Tests, the test is a class. Each method in the class is a test step.

We declare a test as a Phased Test by setting the annotation `@PhasedTest` on the test class. By setting the annotation, we let the system know if the test should be executed in Shuffle Mode or not:

```
@Test
@PhasedTest(canShuffle = true)
public class ShoppingBasket {

    public void step1_searchForProduct(String val) {
        //Perform actions
    }

    public void step2_addToShoppingBasket(String val) {
        //Perform Actions

        //Perform Assertions
    }

    public void step3_payForProduct(String val) {
        //Perform Actions

        //Perform Assertions
    }
}
```

*Code Example 1: A simple Shuffled Test Scenario*

In Code Example 1, when executed in phases, the scenario “ShoppingBasket” will be executed 6 times and each time will be interrupted at a different step.

As described in the section [Cross-Step Communication](#), the order needs to be predictable as earlier steps produce data which is consumed in the following ones. Since in TestNG the tests are executed in an alphabetical order we use a nomenclature of prefixing the steps with their step numbers in our scenarios.

If we are in a Single-Run Mode, we set an annotation `@PhaseEvent` at the location where we expect the interruption to occur. The Phase Test framework will thus execute all the steps up to that point in the producer phase, interrupt the scenario, and, once restarted, continue with the steps following that annotation.

```

@Test
@PhasedTest(canShuffle = true)
public class ShoppingBasketSingleRun {

    public void step1_searchForProduct(String val) {
        //Perform actions
    }

    public void step2_addToShoppingBasket(String val) {
        //Perform Actions
        //Perform Assertions
    }

    @PhaseEvent

    public void step3_payForProduct(String val) {
        //Perform Actions
        //Perform Assertions
    }
}

```

*Code Example 2: A simple Single Run Test Scenario*

In Code Example 2, the scenario “ShoppingBasketSingleRun” is executed only once. In the Producer phase the first two steps, `step1_searchForProduct` and `step2_addToShoppingBasket`, are executed. In the Consumer phase only the last step, `step3_payForProduct` will be run, i.e. the interruption will occur before the method annotated with `@PhaseEvent`.

### 3.2 Storing Data Between Steps

As mentioned in the section [Cross Step Communication](#), we store data both explicitly in the scenario, but also implicitly through the framework.

In the scenario, variables are stored and accessed using methods called **produce** and **consume**.

```

@Test
@PhasedTest(canShuffle = true)
public class ShoppingBasket2 {

    public void step1_searchForProduct(String val) {
        //Search for product

        //Store value with key
        PhasedTestManager.produce("FoundProduct", "book A");
    }

    public void step2_addToShoppingBasket(String val) {
        // Fetch searched product
        String searchedProduct = PhasedTestManager.consume("FoundProduct");

        //add searchedProduct to basket

        //Store basket ID
        PhasedTestManager.produce("BasketID", "1");
    }

    public void step3_checkout(String val) {
        //Fetch basket ID using key
        String basketId = PhasedTestManager.consume("BasketID");

        //Fetch searched product
        String searchedProduct = PhasedTestManager.consume("FoundProduct");

        //Assertion
        //Check that the product is in the basket

        //Checkout
    }
}

```

*Code Example 3: Passing data between scenario steps*

In Code Example 3 above, we first search for the product “book A”, and we store the value by calling “produce” and using the key “FoundProduct”. This is later consumed in step2\_addToShoppingBag & step3\_checkout. We consume the value “book A”, by referring to the key with which it was stored, i.e., “FoundProduct”.

As mentioned in the section [Maintaining the Context](#), the data we store using produce and consume are only visible to the Shuffle Group in which they are executed.

There is a risk for conflicts with the data being created when running tests in Shuffle Mode. If the data being created is hard coded, then the other Shuffle Groups of that scenario will be creating the same data. We think that using random data is usually a good solution to avoid these issues.

In the Code Example 3: Passing data between scenario steps, when executed in Shuffle Mode, as described in the section on [Shuffle Groups](#), yields the following Shuffle Groups:

- 3\_0 The scenario is run with the change at the end.
- 1\_2 The scenario is run with the system change in the middle between steps 1 & 2.
- 2\_1 The scenario is run with the system change in the middle between steps 2 & 3.
- 0\_3 The scenario is run after the system change.

When “step1\_searchProduct” of the scenario is executed in the Shuffle Group 0\_3, the value “FoundProduct” is stored as property with the key:

Scenario name + storage key + Shuffle Group

The value “book A” is thus stored with the key:

```
demo.ShoppingBasket2(phased-shuffledGroup_3_0)->FoundProduct=book A.
```

This way of naming the values we store allows the data to be visible only in its Shuffle Group. This is done by using the Shuffle Group name in the data we generate. Below is an excerpt for the data as it is stored:

```
demo.ShoppingBasket2(phased-shuffledGroup_3_0)->FoundProduct=book A
demo.ShoppingBasket2(phased-shuffledGroup_3_0)->Basket=1
demo.ShoppingBasket2(phased-shuffledGroup_2_1)->FoundProduct=book A
demo.ShoppingBasket2(phased-shuffledGroup_2_1)->Basket=1
demo.ShoppingBasket2(phased-shuffledGroup_1_2)->FoundProduct=book A
demo.ShoppingBasket2(phased-shuffledGroup_1_2)->Basket=1
```

Another piece of information that is passed between the steps is the state of the scenario. This lets us know if the scenario is failing or not in its Shuffle Group.

### 3.3 Managing Context Between Phases

As mentioned in the section on [Cross Phase Communication](#), at the end of the “producer phase” the cache is exported to a file, which is later consumed when the tests are executed in the “consumer phase”. The consumer phase imports the cache file and makes the data available to the phased tests in that phase.

We realize that managing a file between phases can be complex. To address this issue, we introduced a functionality called the “Phased Data Broker”. By implementing the Data Broker, users of the Phased Tests can override the default way Phase Data is stored. When a Data Broker is implemented on a user system, it is used by the framework to store the Phased Data as per the wishes of the user.

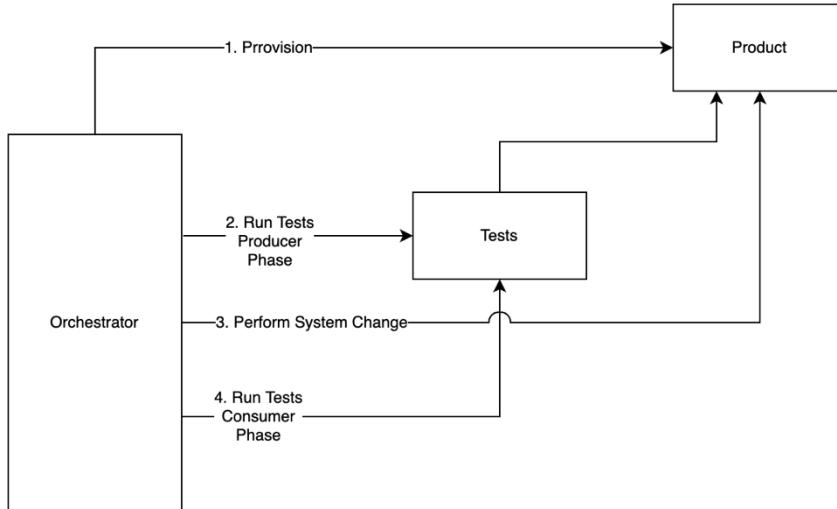
For example, in the current implementation, we store the Phased Data files on an SFTP server. This avoids problems when parallelizing tests.

### 3.4 Orchestration

As mentioned earlier, we chose not to manage the system change itself within the framework. This means that we use an external system to prepare a test system to manage the phases and to perform the system change. In our case this system is a Jenkins Pipeline.

A typical execution of a system change test campaign includes the following steps:

1. Provision: Setting the System Under Test to the state before the change.
2. Run the scenarios in the Producer Phase.
3. Perform the system change.
4. Run the scenarios in the Consumer Phase.



*Figure 9: The execution of a typical Phased Test Campaign*

The orchestrator, after having provisioned a test system, will run the tests in the Producer Phase by passing this as an execution property. After the Producer phase tests have finished, the orchestrator will perform the system change. Finally, when the change has happened, we will rerun the same tests, except this time they are in the Consumer Phase.

### 3.5 Results and Reports

In order to best reflect the results of Phased Tests, we report the results by Shuffle Group and Scenario.

The test report takes into consideration the full result of the scenario and its Shuffle Group. This includes the result and duration of the scenario and Shuffle Group spanning both phases. The consequence of this operation is that in most cases, we do not need to look at the results of the Producer phase, as it is already included in the report.

When a scenario Shuffle Group fails or is skipped, we provide the following information:

- The step in which the error occurred
- The phase where the error occurred
- The error messages

Since we include information about failures in the steps regardless of the phase, we usually focus mainly on the test report of the CONSUMER phase. The Scenario Shuffle Group result is based on the following cases:

PRODUCER Phase Result	CONSUMER Phase Result	Report
Passed	Passed	Passed
Failed	Skipped	Skipped with a description indicating in which step of the PRODUCER phase the Execution failed
Passed	Failed	Failed with a description of the step in which the scenario failed

## 4 Conclusions

The promise of Phased Tests is that of leveraging our test scripts to ensure that our application remains robust in the face of events. This will allow us to ensure that systems can carry on working as expected even after a major system change. As Phased Testing is a new concept, there is a lot of room for improvement.

We think that there is still work to be done in how we define and attach events to a scenario and its steps. For now, the events we have defined are interruptive and are executed before or after a scenario step. However, we would like to do research in the following domains:

**Defining multiple events**, will allow us to define a series of interruption types, so that we can test the system against each of these events.

As we mentioned before, our implementation of Phased Tests expects the scenarios to be interrupted, for our interruptive events to fold. We do, however, recognize that we have a class of "**Non-Interruptive Events**", that do not necessarily require the system to be interrupted to happen. Examples of this would be the introduction of heavy loads in a system, or in the case of a driverless car, the introduction of an obstacle.

While speaking of parallel events, one will necessarily start considering **Parallel event** executions, which is a notion that will allow us to assess the effects of an event on any step of a scenario. In our current implementation all events happen before or after a scenario step. We can imagine that an event happens during a scenario step.

Another area for further research in the field of Phased Tests would be to deduce the order of the steps by examining the "produced" and "consumed" data in each step. This would make the implementation more intuitive in the future. Although this would be of great interest, what is even more exciting is that the automatic detection of the step orders would then enable us to introduce a new test concept, namely the execution of a scenario with all the possible step combinations that would lead to the same end result.

Our experience is that we do not need to transform all scenarios into Phased Tests as it will make your System Change tests much longer. Good candidates for Phased Tests are the smoke tests or architecturally significant scenarios. Another good practice we have discovered in our research is to not add too many steps in the scenario as it increases the complexity, and scenarios will have a lot of Shuffle Groups as a result.

We see a lot of potential for Phased Tests, as it opens doors to testing the unexpected. We see an evolution where we can test a product against a large series of unexpected events. Today Phase Testing will help us predict problems related to major system changes. Tomorrow, this may go even further and allow us to predict stability regarding unexpected events.

## Acknowledgements

The author of this paper would like to express gratitude to the following people (in alphabetical order):

- Mihai Bilan, for his great insights, ideas and his attentive proof-reading.
- Jean-Baptiste Garcia, for many off his useful suggestions and ideas.
- Felix Meschberger, for validation of the document, and being a great inspiration.
- Roy Ogas, for his aid in the creation off this document.
- Marco Stankovich, for providing valuable feedback on the phased tests implementation.
- Charles McBride, for his kind and helpful remarks.
- Qingfen Zhu, for providing valuable feedback on the first generation of the producer consumer concepts.

## References

- Cédric Beust, Hani Suleiman. 2007. *Next Generation Java Testing: TestNG and Advanced Concepts*. Addison-Wesley Professional
- Marty Lewinter, Jeanine Meyer. 2015. *Elementary Number Theory with Programming*. Wiley
- Baubak Gandomi. 2021. "Automated Upgrade Testing: A Process to Test and Validate Software Upgrades". Adobe Tech Blog, entry posted March 2021. <https://medium.com/adobetech/automated-upgrade-testing-a-process-to-test-and-validate-software-upgrades-349e647f34f4> (accessed March 11 2021).
- The Phased Testing TestNG Implementation. <https://github.com/adobe/phased-testing>.

## Appendix A

Below is an example of the Shopping Basket scenario as a Phased Test, juxtaposed with the same scenario if we did not use Phased Tests.

The Shopping Basket Phased scenario. Here we only declare the scenario once. It will be used for both normal testing, and in Shuffle Mode:

```
@Test
@PhasedTest(canShuffle = true)
public class ShoppingBasket {

    public void step1_searchForProduct(String val) {
        //searchForProduct()
    }

    public void step2_addToShoppingBasket(String val) {
        //addToShoppingBasket()
    }

    public void step3_payForProduct(String val) {
        //payForProduct()
    }
}
```

The Shopping basket scenario in the case where we wish to implement all possible interruptions. (We have taken the liberty of still using Phases to highlight that we are in Producer/Consumer mode). If we chose to add a new step, we would have to add more methods, and update all methods.

```
public class ShoppingBasketWithIfs {

    //The normal test (0_3 and 3_0)
    @Test
    public void standardTest() {
        //searchForProduct()
        //addToShoppingBasket()
        //payForProduct()
    }

    @Test
    public void ShoppingBasket1_2() {
        if (Phases.PRODUCER.isSelected()) {
            //searchForProduct()
        } else {
            //addToShoppingBasket()
            //payForProduct()
        }
    }

    @Test
    public void ShoppingBasket2_1() {
        if (Phases.PRODUCER.isSelected()) {
            //searchForProduct()
            //addToShoppingBasket()
        } else {
            //payForProduct()
        }
    }
}
```

# Managing affirmatively through the "Great Resignation"

**Mark Bentsen**

[Mark.Bentsen@argodata.com](mailto:Mark.Bentsen@argodata.com)

## Abstract

Most of our QA team has been with the company for over five years and many over ten. Our team was impacted by the "Great Resignation" recently. Members found themselves being heavily recruited with offers of unusually high compensation and benefits. At the same time, QA needs upskilling for a future of increasingly technical and non-functional testing. Requiring a systematic approach, I developed a repeatable framework to navigate training the current QA team and hiring the quality engineering talent for future requirements. We have successfully made multiple recent hires with this framework. Also, it is actively directing advancements in QA's training.

**Learning Objectives:** Attendees will learn how to develop a customized QA skills gap analysis. The framework runs on basic six sigma structure. This incorporates the sum of existing individual strengths, organizational redundancy, and domain expertise. This is then analyzed with skills required for the success of the future QA organization. Inputs include known and anticipated projects, technologies, and capabilities. Attendees will learn how to create a key deliverable for their recruiting group. The process above requires defining skills of the 'as is' and 'to be' QA team. Attendees will learn how to design a visualization for communicating training objectives to the team. This tool is critical for the leader of the software quality organization.

## Biography

In 2015, became part the Advanced Research Center for Software Testing and Quality Assurance at the University of Texas in Dallas (UTD). Mark presents on QA leadership, KPIs, and root cause analysis in local, national, and international software conferences. Mark is a PMP & CTAL (Full) from ISTQB.

Mark & his wife Melissa are the two-time, past President Couple of 'Better Marriages Texas' and have been active in Marriage Enrichment since they said "I do" in 2001. Prior to working in technology, he worked in YWAM & Mercy Ships in Switzerland and Namibia. He lives in Allen with his wife and two boys, ages 14 and 18.

*Copyright Mark Bentsen, September 2022*

# 1 Introduction

Standing at the office door of our HR Manager she and I were discussing an alarming fact. It was the first time we could recall having five resignations in a single week from the software development group. It was in her explanation that I first heard three ominous words, "The Great Resignation". When asking about this trend, she explained how after the pandemic, employees were changing priorities around "work-life fit" or leaving the traditional professional model altogether. Technically skilled people are pursuing a different type of career model or lifestyle away from commutes and normal hours in an office, at a cube.

In that moment my selfish side thought, "Sounds tricky for the leaders who are dealing with this. Glad it's not me." Soon, I was joining these managers. It started with an employee giving a two week notice unexpectedly. The QA team had several years where turnover was zero. Most of the team had been at the company at least five years and a few for more than ten. I was caught off-guard and shouldn't have. My consolation, everyone in this industry seemingly is now in the same boat. If you are leading a technology work group right now, you're faced with talent leaving and doing your best to manage knowledge capital.

I cleared time on my calendar for putting together a strategy to manage affirmatively through the weeks and months ahead. A broader, repeatable strategy was necessary. A typical 'hire to replace' wasn't going to solve this.

# 2 Considerations

## 2.1 Managing knowledge capital in QA

Can you accurately measure the business-critical knowledge in QA today? If you have any doubts about measuring QA expertise and knowing where content is documented, you are unlikely to manage and improve this body of knowledge and the individuals with it. Don't rely on 'tribal knowledge'. Managing knowledge capital will be a key success factor in the weeks ahead.

### 2.1.1 Variety is best

The best approach is to have more than one methodology available for team members. Components of a strong methodology include recorded training sessions, checklists and detailed info on a wiki, and a contact sheet with the domain expert in each area. People learn differently and the level of detail drives the platform. The best, next step is to brainstorm ideas on how and what to document for training purposes. Let the QA team know you are looking for undocumented work process. Let them capture this over time and monitor progress. Follow up, follow up. Get the most exhaustive list possible. Don't leave undocumented work process off the list.

### 2.1.2 Prioritize

It's impossible to document every process gap. The goal is to ensure business-critical ones are first. Develop training you will use in nine months or less. Documentation can be a black hole. Spend your time and effort where you're confident the deliverables will get used. Find activities where documentation and planned work can be conducted in consort. Focus on absent checklists or undefined standards. The next occurrence of that work is the best time to capture and publish a deliverable.

Also, do not overlook the resources you have available today. Where feasible, use real world content for training. Leverage test assets. Test scripts are excellent as a training tool. Likewise, retesting legacy defects in a complex functional area is useful and pragmatic. This requires tests being accurate. If they are not current, this is the optimal time to update the test library.

## 2.2 Take a moment

Take a moment and consider this. What if you could not hire? You lost some people on the team and you have no choice but to find a way to make it work. How does this change your perspective, priorities, and objectives? Using imaginary constraints will require you to get innovative. Trying to find a solution with a non-standard method can shift your approach. Innovative thinking accelerates getting from 'as-is' and achieving a 'to-be' state.

This typically results in eliminating low value, reoccurring tasks, reports, and meeting redundancy. This step requires discipline. Making decisions and eliminating work feels risky. If you are like the vast majority of other SDLC work groups, more than 30% of your team's work falls in this category. Identify what can be delegated or combined with other work in the organization. Are standards for dev testing published? Are they being followed? How do you know. Now is the time to verify upstream quality checks are both in place and effective.

## 2.3 Work Intake

Review your work intake process. Can this be improved to eliminate waste and reduce risk? The answer is "Yes". Set expectations realistically. Don't agree to accomplishing the impossible. Be solutions oriented, outcome focused, and understand the mission. Chaos at the time of accepting work will compound. Be a supportive business partner, while at the same time, understand the success criteria for your team's accountabilities.

## 2.4 Hiring

Fortunately, this mental exercise of not backfilling is designed to increase the effectiveness of hiring for the QA organization. A critical next action is to acknowledge that you must determine the skills and characteristics to fill. Don't be convinced you already know exactly what the team needs.

# 3 Process

The objective is to identify the skills necessary to be in place across your team. Skills to effectively support the next twelve to eighteen months of quality assurance. The exercise will be critical path for QA's strategic training and any testing tool needs. This process requires you to take an inventory of your current state, objectively score the inventory, and outline future needs of the organization. Defining the "as is" state of skills across your team and then a desired "to be" state creates a visualization of your skills gap. A single visualization of your skills gap will be a critical tool as you communicate the training and hiring requirements to HR, peers, and executives.

Let the QA team know what you are doing and more importantly 'why'. This is a process to ensure you're hiring the right talent. For the current QA team, it is identifying training areas to target. Part of the communication is to inform the team that there will be a self-rating step. Do your best to announce this in person and answer questions. Your goal is to strengthen the team. You are being strategic. Bring the team into the process. When communicating 'why', schedule the meetings required for the remainder of QA's processes in the assessment exercise.

# 4 Data Capture

The individuals of the QA team will be rating themselves across skills. Specifically, the skills as defined for this exercise. The manager will also rate each direct report. Anticipate differences between individual self-rating and manager rating. The following process will normalize the data, visualize skills that are a gap, and provide a deliverable. The output communicates strategic hiring requirements in the QA organization. The **six sigma analysis** guides long-term insight for standards in performance management

activities and QA training plans. When announcing, be clear and upfront regarding the objectives, how the process works, and their part of making it successful for QA.

## 4.1 Process

This process requires reviewing the self-scoring and manager results and providing the visualization quickly back to the team. Two business days or less is ideal. When this work can be scheduled, begin the communication plan. You will need at least two weeks to forecast this activity and explain its value for hiring new QA members.

## 4.2 Skills and Definitions

Create a structure for your inventory. This may fall into categories such as quality assurance, testing, and process. The next level of detail is where you will be scoring the individuals on the team. It is recommended that the number of total skills areas be twenty or under. More is not always better for the consumer of this data. Having the primary key elements presented is most effective. Otherwise, the visualization may communicate all the team's strengths and minimize the necessity to fill a strategic gap.

There are skills and competencies specific to QA. There are more that overlap with the cross-functional team. The best practice is to have a balance, while including QA specific ones. Requirement's quality can be decomposed into multiple skills. Is it necessary for your purpose here? Where possible, do not decompose into too much detail. Remember, it is all about your audience. What do you need them to do with this information? How can you determine if you are being effective?

Definition. You have the wording, but that's not enough. Definition is critical. Document a concise, context relevant definition for each term. It will be used by those who score themselves as well as those assisting in training and hiring processes. To be clear, HR is unlikely to understand the difference between reliability and performance testing. Should 'assertive' be one of your areas, for everyone's benefit, be specific in how this is applicable to QA skills.

### Quality Assurance

Requirement Quality	Analysis of completeness of requirements for development and testing. Provide input that eliminates ambiguity, increases testability, and ensures a distinct expected result of the specifications.
Reliability	Non-functional performance, reliability, fault tolerance testing. Including proven expertise leading NF testing for Accuracy, Observability, Recoverability, Reliability, Responsiveness, Scalability, and Stability.
Security	White Hat pen testing, Burp Enterprise Suite, Authentication, Authorization, Nessus, Whitesource, SonarCube, Veracode
UI/UX	Usability testing, knowledge of tech behind it; HTML, CSS, JavaScript, JSON, React
Azure	Setting up environments, NF testing in the cloud, monitoring, analysis
Pure non-functional	Innovation of ideas that increase product quality. Identification of specialized needs to test product. Design or develop scripts, tooling, solution to meet this unique need.

### Testing

Functional	Basic testing to ensure product fitness to acceptance criteria, requirements, and other specifications.
Backend testing	(SQL Server Management Studio) Database / SQL Queries expertise. Proven competence testing relational databases (SQL Server, ORM Technologies, and Stored Procedures). For web services, (Ready! API, Postman)
UI Automation	Designing frameworks, developing scripts, analyzing results, troubleshooting failed, executing automation, Selenium, Cucumber Scenarios
Logs	Using logs to find errors and root cause. Review ARGO logs, traces & files (e.g. using BareTail Logs)
Virtualization	Create virtualized test APIs or other stubs to accelerate testing.

**Process**

SDLC	Methodology experience
Specifications	Business & technical requirements definition and understanding quality standards. Ability to accurately estimate work efforts for testing processes using requirements as basis.
Test Management	Leading testing processes by developing the test plan, accountable for scripts, execution, analysis and reporting status and risks
Assertive	Decisive, ability to make decisions and complete assignments independently. Team player aligned to project objectives and sense of urgency.
Integration	Cross-functional effectiveness. Design integration testing strategies and cross-disciplinary effects of new software solutions. Develop test scripts aligned to business value while also validating the backend. Focus on pragmatic usage by end-user.
Prob Solver	Strong analytical, reasoning, and problem-solving skills with an ability to visualize processes and outcomes. To identify root causes. Ability to escalate issues to appropriate levels of management.
Prioritization	Ability to prioritize work and manage multiple assignments within estimates, commitments, completion dates. Has strong follow through meticulous attention to detail.
Delegation	Ability to effectively communicate deliverables and expectations to individuals and customers. Ensure commitments are completed by others.
Eff Comm	Persuasive, articulate, and professional. Written and oral communication skills and the ability to present testing methodology and strategies to both customers and internal teams. Ability to train, upskill, and perform knowledge transfer to other QA members.

### 4.3 Self-Rating Prerequisites

Have managers reporting to you complete their personal self-rating beforehand. You will rate them and complete this at least a week before rolling it out to the QA team. Managers will better understand this process by going through it firsthand. This provides a process quality check. Anticipate making some modifications from insight acquired here.

The week prior to self-scoring, remind QA about the exercise, and provide the date when the worksheet will be sent. Ideally, communication can be in your weekly team meeting or other regular meeting or communique. Answer questions and prepare the team to receive definitions of skills they will use in the worksheet. Send the definitions directly afterwards.

By doing so, the team members will be given every opportunity to ask questions and understand the self-rating. Your communication and upfront efforts will pay dividends. Your goal is to have an efficient process while completing the self-rating process. Your goal in this phase is getting as much of the team onboard as early as possible.

After this meeting you are sending everyone the list of skills and corresponding definitions. Let everyone understand they will need to be familiar with the areas prior to self-scoring. Also, reiterate the process of self-rating they will have directly after the upcoming meeting. That meeting has been on everyone's calendar since you defined the 'why'. Now you've done your utmost in setting expectations.

## 4.4 Rating

Rating is divided into dual processes. The self-rating individual contributor step and the manager's rating. Managers score their direct reports. The rating categories below provide a heuristic. Use a skill rating scale best suited for your context. Skill rating scales using whole numbers 0 to 10 are recommended.

- 0      No experience
- 1 - 3    Low
- 4 -6    Moderate
- 7 - 9    High
- 10     Master

Consistency is the paramount success factor when selecting a rating scheme. This is the basis for the following two processes.

### 4.4.1 Manager's rating of QA

After defining the terms, score the team. Consider the job level, and you will be documenting your best subjective assessment of the QA team through grading all the members. The best practice is to do this once, efficiently, and think of a normal distribution.

### 4.4.2 Self-Rating

The self-rating process occurs after the launch meeting. In many ways, self-rating becomes the most mechanical. Below is a sample self-rating form.

	A	B	C	D	E	F	G	H
1	QA Skills Matrix							
2								
3	Quality Assurance							
4	Team Member	Requirement Quality	Reliability	Security	UI/UX	Azure	Pure non-functional	Functional
5	Your Name							
7								

Everyone has the definitions, the worksheet they receive following the launch meeting requires them to score themselves. Scoring is based off the skill rating scale.

#### 4.4.3 Launch Meeting

Walkthrough the expectations leading to this meeting. Provide the scoring scale and guidelines. Remind QA members to score against the definitions for the exercise.

In this meeting a deadline is set to complete the self-rating activity. Plan on one business day to complete their worksheet. The key is to have everyone perform the self-rating quickly. Providing more time does not improve accuracy.

Directly afterwards, send the self-rating form. QA members then individually complete and return the self-rated form to their manager

##### 4.4.3.1.1 Publishing results

Keep in mind, you have let the team know what you're doing. This is not a surprise. Give them 24 hours or less to complete their self-rating part of the process. Quickly distribute the combined, anonymous results you will calculate in the Standardized Data section. If QA sees the output within the time expectations set, this will be a positive experience for the team.

##### 4.4.3.1.2 Alternate Rating Option

This is an optional rating. If integrating the skills gap exercise with career planning, consider an additional rating dimension. The employee can choose to use this secondary rating scale to document their desired growth plan. Use the skills rating scale for X (Today) and Y (By next year).

X = Current employee skill

Y = Desired employee skill, where do you want to be a year from now

A	B	C	D	E	F	G	H	I	J	K
1	QA Skills Matrix									
2										
3	Quality Assurance									
4	Team Member	Requirement Quality	Reliability	Security	UI/UX	Azure	Pure non-functional	Functional	Backend testing	UI Automation
5	Your Name									
6	X/Y [Alt Scale]									
7										
8	Definitions									
9										
10	Quality Assurance									
11	Requirements Quality	Analysis of completeness of requirements for development and testing. Provide input that eliminates ambiguity, increases testability, and ensures requirements are clear and well-defined.								
12	Reliability	Non-functional performance, reliability, fault tolerance testing. Including proven expertise leading NF testing for Accuracy, Observability, and Scalability.								

## 4.5 Standardized Data

You have now collected all results for skills across the QA team. See an example below of data organized with scores collected from both manager and individuals.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
1	QA Skills Matrix																			
2																				
3	Mgr/Self Rating	Corporate Title	Task Alignment	Requirement Quality	Reliability	Security	UI/UX	Azure	Pure non-functional	Functional	Backend testing	UI Automation	Logs	Virtualization	SDLC	Specifications	Test Management	Assertive	Integration	Prob Sol.
4	Employee	Corporate Title	Task Alignment	Requirement Quality	Reliability	Security	UI/UX	Azure	Pure non-functional	Functional	Backend testing	UI Automation	Logs	Virtualization	SDLC	Specifications	Test Management	Assertive	Integration	Prob Sol.
5	Team Member 1	QA Supervisor	Product name	8	0	0	4	1	2	8	3	1	3	0	10	7	9	9	6	
6	Team Member 2	QA Supervisor	Product name	7	0	0	3	0	3	9	4	0	6	0	8	6	9	9	7	
7	Team Member 3	QA Sr	Product name	4	0	0	3	2	0	6	7	2	7	1	8	6	7	6	6	
8	Team Member 4	QA Sr	Product name	7	0	0	0	1	4	8	2	0	5	0	8	5	6	7	6	
9	Team Member 5	QA Sr	Product name	7	0	0	0	4	5	8	4	0	6	0	8	6	7	8	6	
10	Team Member 6	QA Sr	Product name	7	0	0	0	0	4	7	3	0	6	0	8	6	7	8	7	
11	Team Member 7	QA II	Product name	7	0	0	0	0	4	8	3	0	6	0	8	6	6	9	7	
12	Team Member 8	QA II	Product name	7	0	0	0	0	2	8	4	3	6	0	8	5	7	6	6	
13	Team Member 9	QA II	Product name	7	0	0	0	0	4	7	3	0	6	0	8	6	6	8	7	
14	Team Member 10	QA II	Product name	7	0	0	0	5	4	8	4	0	6	0	8	5	7	6	6	
15	Team Member 11	QA II	Product name	5	0	0	0	0	3	5	2	0	5	0	8	4	5	6	6	
16	Team Member 12	QA II	Product name	7	0	0	0	0	3	8	3	0	4	0	8	5	6	7	6	
17	Team Member 13	QA I	Product name	8	4	3	5	4	5	6	6	1	6	2	6	7	5	5	6	
18	Team Member 14	QA I	Product name	5	0	0	2	0	0	7	2	1	2	1	7	5	6	4	4	
19																				
20	Self Rating	Corporate Title	Requirement Quality	Reliability	Security	UI/UX	Azure	Pure non-functional	Functional	Backend testing	UI Automation	Logs	Virtualization	SDLC	Specifications	Test Management	Assertive	Integration	Prob Sol.	
21	Team Member 1	QA Supervisor	7	0	0	2	1	2	8	3	2	3	0	8	7	7	8	6		
22	Team Member 2	QA Supervisor	7	0	0	0	0	4	8	3	0	6	0	8	6	7	10	9		
23	Team Member 3	QA Sr	10	2	0	5	1	0	10	9	2	8	3	10	10	9	10	10		
24	Team Member 4	QA Sr	7	1	1	8	2	5	10	3	1	4	1	1	8	9	9	5		
25	Team Member 5	QA Sr	9	2	0	3	5	6	8	9	2	6	6	9	10	10	10	10		
26	Team Member 6	QA Sr	10	2	0	4	0	0	10	8	0	9	2	10	8	8	10	9		
27	Team Member 7	QA II	7	3	3	4	0	2	8	3	1	6	0	7	8	8	8	8		
28	Team Member 8	QA II	0	0	2	0	0	9	4	3	6	0	8	2	8	8	8	8		
29	Team Member 9	QA II	8	2	3	0	0	0	8	5	2	3	0	8	8	7	7	8		
30	Team Member 10	QA II	7	0	0	5	5	7	8	4	1	3	3	7	7	7	8	5		
31	Team Member 11	QA II	5	8	5	2	5	5	5	4	5	5	5	5	5	5	5	5		
32	Team Member 12	QA II	9	9	0	9	1	4	10	4	0	8	0	9	9	5	9	9		
33	Team Member 13	QA I	6	6	5	6	5	5	8	6	4	6	4	7	7	5	5	6		
34	Team Member 14	QA I	6	1	0	2	0	1	7	2	1	2	2	7	7	7	6	5		
35																				
36	Definitions																			
37																				
38	Quality Assurance																			
39	Requirements Qualit	Analysis of completeness of requirements for development and testing. Provide input that eliminates ambiguity, increases testability, and ensures a distinct expected result of the specifications.																		

This will be organized to provide a view for skills-based strengths and weaknesses. These will be describing the current skill levels in QA. These skills can be enhanced with training. The business-critical, strategic gaps can be filled via hiring. The gap can also be filled from lateral movement into QA from other areas of the company.

#### 4.5.1 Basic Mean Analysis with Pareto Visualization

Use a mean analysis when self-rating and manager ratings are similar in scoring. If scoring is +/- 2 for 80% of the areas, this method is preferred.

##### 4.5.1.1 Sum of Means

The objective is to calculate a mean from the two scores provided. One by the manager and the second through self-rating. A third table will reflect this.

A	B	D	E	F	G	H	I	J	K	
37	Analysis	Requirement								Backend testing
		Quality	Reliability	Security	UI/UX	Azure	Pure non-functional	Functional		
		7	1	1	2	2	3	8	4	
39	Team Member 1	QA Supervisor	7.5	0	0	3	1	2	8	
40	Team Member 2	QA Supervisor	7	0	0	1.5	0	3.5	8.5	
41	Team Member 3	QA Sr	7	1	0	4	1.5	0	9	
42	Team Member 4	QA Sr	7	0.5	0.5	4	1.5	4.5	2.5	
43	Team Member 5	QA Sr	8	1	0	1.5	4.5	5.5	8.5	
44	Team Member 6	QA Sr	8.5	1	0	2	0	2	8.5	
45	Team Member 7	QA Sr	7	1.5	1.5	2	0	3	8	
46	Team Member 8	QA Sr	4.5	0	0	1	0	1	8.5	
47	Team Member 9	QA Sr	7.5	1	1.5	0	0	2	4	
48	Team Member 10	QA Sr	7	0	0	2.5	5	5.5	8	
49	Team Member 11	QA II	5	4	2.5	1	2.5	4	5	
50	Team Member 12	QA II	8	4.5	0	4.5	0.5	3.5	3.5	
51	Team Member 13	QA II	7	5	4	5.5	4.5	5	3.5	
52	Team Member 14	QA I	5.5	0.5	0	2	0.5	7	2	

Calculate each value for this table.

Where a QA member has manager scoring in cell D5 and self-scoring D21. Use =(D5+D21)/2 for D39.

After all mean values are calculated, the average across the team can determined.

Managing affirmatively through the "Great Resignation"

Mark Bentsen

For PNSQC.ORG

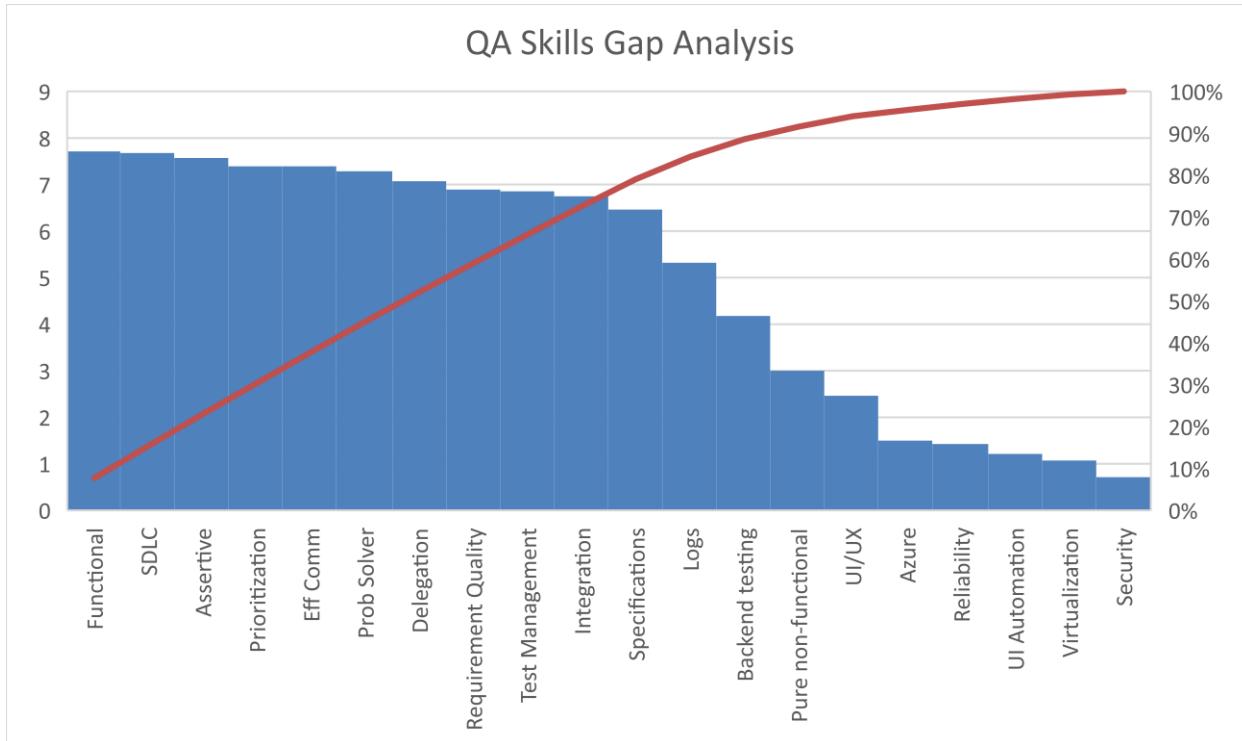
Page 8

With the mean values for team members listed on rows 39 through 52, use =AVERAGE(D39:D52) for D38.

The team's combined average becomes the basis for visualization of the gap analysis. This method is useful when scoring is similarly aligned between manager and team member.

#### 4.5.1.2 Pareto Chart

The output of this analysis is clear in Pareto chart format. This is not an 80/20% dataset that might typically be used with this visualization tool. Nonetheless, this format communicates to your audience. In one view, the skills gap is clearly identifiable.



#### 4.5.2 Advanced Standardization

##### 4.5.2.1 Overview of constituent calculations

The combination of manager and self-rating mean will be included with the sum and standard deviation. Then standardizing (Z-score) the data will provide what is necessary for identifying anomalies in the data. By doing this, data will be comparatively similar. This methodology eliminates subjectivity of individuals and increases overall objectivity of results. The results will not be whole numbers.

##### 4.5.2.2 Basis of mean data

Use the same data from the sum of means methodology previously described.

Taking a skill where column D contains its scoring. And where scores are on rows 21 through 34, use =AVERAGE(D21:D34)

##### 4.5.2.3 Standard Deviation

Producing the mean, standard deviation provides insight to distribution and confidence in the data. A standard deviation of two or less is excellent.

Taking a skill where column D contains its scoring. And where scores are on rows 21 through 34, use  
 $=STDEV(D21:D34)$

The best standard deviations are low. A one or two. Typically, there will be threes in the set with a low count of data. Any standard deviation beyond three is a signal the score is unrealistic for the expectations of the dataset. This will be amplified with the Z-score.

#### 4.5.2.4 Standardizing (Z-score)

This method of standardization provides a simple way to compare a data point to the norm. The prerequisites for using this method are the mean and standard deviation.

Taking a skill where column D contains its scoring. The value to normalize is D21, mean D36, standard deviation D37, Use  $=STANDARDIZE(D21,D36,D37)$

This provides the manager with a view of any data outside the expected values.

20			Requirement Quality
21	Team Member 1	QA Supervisor	7.5
22	Team Member 2	QA Supervisor	7
23	Team Member 3	QA Sr	7
24	Team Member 4	QA Sr	7
25	Team Member 5	QA Sr	8
26	Team Member 6	QA Sr	8.5
27	Team Member 7	QA Sr	7
28	Team Member 8	QA Sr	4.5
35			
36		Mean	6.96428571
37		Standard Deviation	1.26284607
38	Z-Score		Requirement Quality
39	Team Member 1	QA Supervisor	0.4242118
40	Team Member 2	QA Supervisor	0.028280791
41	Team Member 3	QA Sr	0.028280791
42	Team Member 4	QA Sr	0.028280791
43	Team Member 5	QA Sr	0.820142932
44	Team Member 6	QA Sr	1.21607400
45	Team Member 7	QA Sr	0.0282807

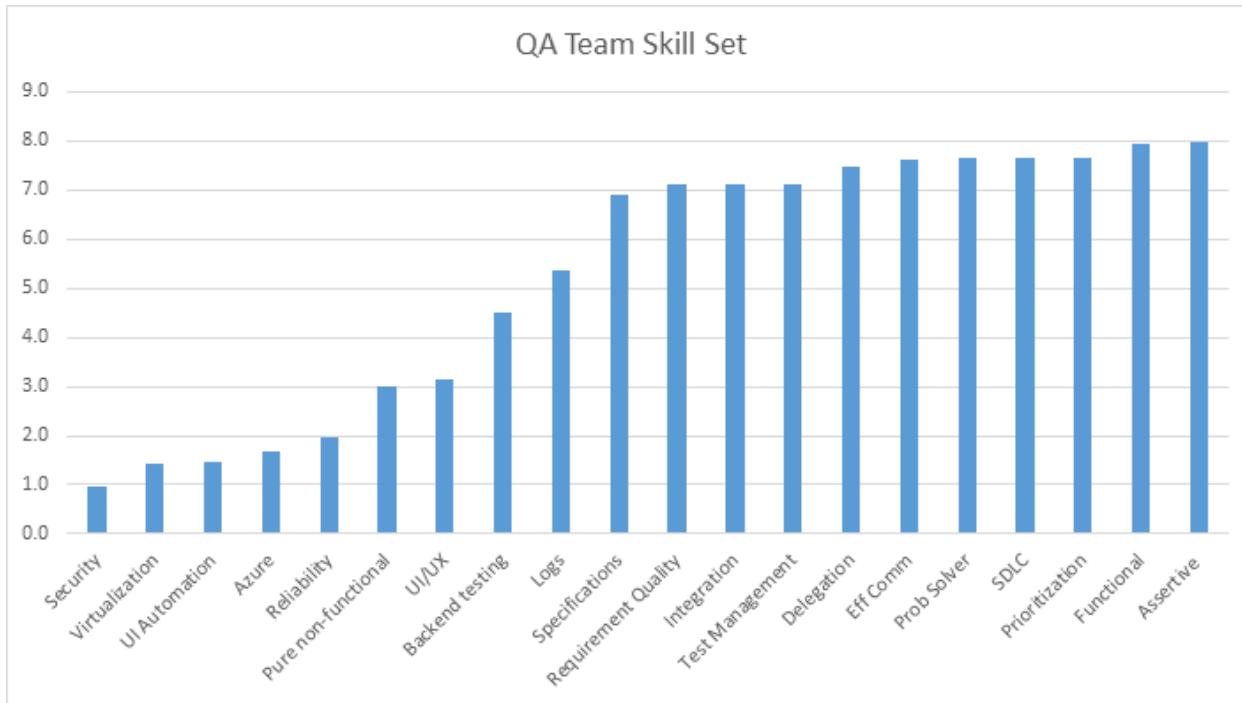
In this illustration team Member 6 is higher than the average. This may be valid and expected. It also can be an opportunity to leverage their under-utilized skills. Alternatively, this may show an area where the manager and team member are out of alignment with performance.

## 4.6 Presentation of Data

Consider the audience. What do you need them to do with this information? If HR, you might be communicating how to search LinkedIn or where to focus resume screening. If this work does not change the hiring process for you in a material way, rethink how you are defining your required skills. If the

audience is a peer manager and she is counting on you to support them in the SDLC, does this identify where core competencies are covered. While at the same time, clarifying your understanding to adapt QA to future challenges or new technologies. When presenting at a senior level, are you defining the situation, the complication, what can you do, and the plan you are selecting from various possibilities? Do you require a decision, or support, or are you simply wanting to build confidence that you are up to the task of solving this in a strategic manner? This is a communication tool. Having this visualization provides an efficient message to get support, build confidence you understand the need, and take the next actions to fill skill gaps.

## 4.7 Visualization of Normalized Data



## 5 Strategic Benefits

### 5.1 Hiring the right talent

In hiring processes ahead, use this as a blueprint for the needs of the current QA organization. Remember that this will change over time. Nonetheless, a quantification of team skills provides insight to filling the gaps and avoiding redundancy.

As hiring continues, consider how skill coverage has increased or new needs come to light. Depending on team size, in two to four hires most QA skill gaps can be covered.

### 5.2 Training

Training members of the team to cover skill gaps is often preferred. Training is normally planned across an annual period. This visualization is a key input when requesting budget. It is also likely training will be planned in other areas of the organization. Identify scheduled training that will cover the skills gap.

# Fallacies of Data Driven Decisions

**John Cvetko**

John.Cvetko@tekasc.com

## Abstract

Businesses and governments spend an enormous amount of time and money in search of data that informs their decision-making processes. The barrier for an individual to accumulate large amounts of data has decreased dramatically and can easily overwhelm their ability to make sense of it. This new reality has increased the importance of quality decision making skills at the executive, program/ product, and technical team levels of the organization.

For this reason, it is first important to understand the distinction between data and information. Data is a collection of facts, while information puts those facts into context. It stands to reason that the data from which a decision is based upon can be of high quality, but the context in which the decision was made could be lacking.

Decision-making processes and culture are foundational components of an organizations ability for meeting the needs of a constantly changing business landscape. An organization may tout a data driven mindset and great agility but lack the understanding of the reasoning or method from which a decision is made. Not understanding this key component can result in decisions that promote chaos rather than agility.

In this paper the author will explore performance optimization through the lens of quality, relative to decision-making and organizational adaptability.

## Biography

As a Principal of TEK Associates, Mr. Cvetko works with companies and government agencies to improve their organizations by helping them manage the IT challenges they face. He applies state of the art solutions to evolve business processes, creating more efficiency and productivity, all while improving quality. The last 10 years have been primarily focused on assessing and transforming large enterprise software systems for state governments. He has worked with the state governments of Colorado, Washington, Oregon, North Carolina, North Dakota, Mississippi, Utah, Kentucky, and Oklahoma. Earlier in his career he worked as a management/technical consultant for firms such as NIKE and Boeing, and in product development and program management for Tektronix, PGE/Enron and ASCOM.

*Copyright John Cvetko 8/25/2022*

## 1. Introduction

*“Fortune favors the data driven organization”* is the theme of 21<sup>st</sup> century. The volume of data available to the average employee has exploded, yet decision-making capabilities have not kept pace. The speed and fluidity of software development coupled with a well-educated and empowered workforce is a recipe for great performance or chaos.

At the turn of the last century management’s goal was to have factory workers do small tasks in a highly controlled process. In the modern era, staff are expected to work autonomously to solve complex problems daily, some of which have significant impact to the organization.

The nature of the problems being solved, linkage between disciplines, time pressures, etc., make even seemingly simple decisions ripple through an organization. All decisions have unintended consequences that create a new reality that present a set of problems which need even more decisions. Organizations see this as being “agile” however, if the decision-making process is inferior or misaligned, it is more akin to crashing back and forth between guardrails.

Many organizations profess they are data driven, however, having vast amount of data doesn’t guarantee a good or even an informed decision. A more appropriate tag line might be - *“Fortune favors a data driven organization with a culture that is born from robust and aligned decision making processes”*. While this might be a more realistic representation, as a sound bite, it is arguably less catchy.

## 2. Agile (Scrum) and the OODA Loop

The Agile Software Development Lifecycle (SDLC) is employed to provide a platform that supports the modern software development reality. It provides a highly skilled workforce the flexibility to manage the software development process. This flexibility is necessary to operate in a fluid environment with tools that are highly malleable.

Dr. Jeff Sutherland is a signatory of the Agile Manifesto, founder of the Agile Alliance, Co-creator of Scrum and Scrum@Scale. Dr. Sutherland began his career as an Air Force fighter pilot in Vietnam, and as part of his training he was drilled in a rapid decision-making technique called the “OODA” loop or Boyd cycle, see fig. 1. OODA which stands for Observe, Orient, Decide and Act is a mental process model that was developed in the 1950’s by an Air Force pilot, Colonel John Boyd.

Boyd began his Air Force career in the 1950’s just as the jet engine was being introduced. The move to jets increased the speed of air travel 3-fold and was seen as a major advantage to the military. Boyd’s time as a fighter pilot in the Korean war impressed upon him that the United States jet fighter design was fundamentally flawed. The enemy was utilizing faster, more maneuverable aircraft that were specifically designed for high-speed aerial combat. Not fully understanding the new technology and how it would be used in battle, the uninformed design decisions made by civilian engineers had lethal unintended consequence in battle.

Since he could not change the technology, Boyd was forced to change his way of thinking. Even though fighter aircraft were used extensively in WWII, combat tactics were considered an art rather than a science. The enemy had leaned so heavily on the technology that they did not reorient their way of thinking to maximize the advantage. Boyd’s mental adaptation resulted in leveling the playing field against the superior technology. His adaptation technique was the OODA loop.[Coram]

The OODA loop was one of many contributions of Colonel Boyd’s which are still utilized by militaries, governments, and businesses around the world. To get a fuller view of the breadth and depth of his contributions please refer to the books and articles listed in the reference section of this paper.

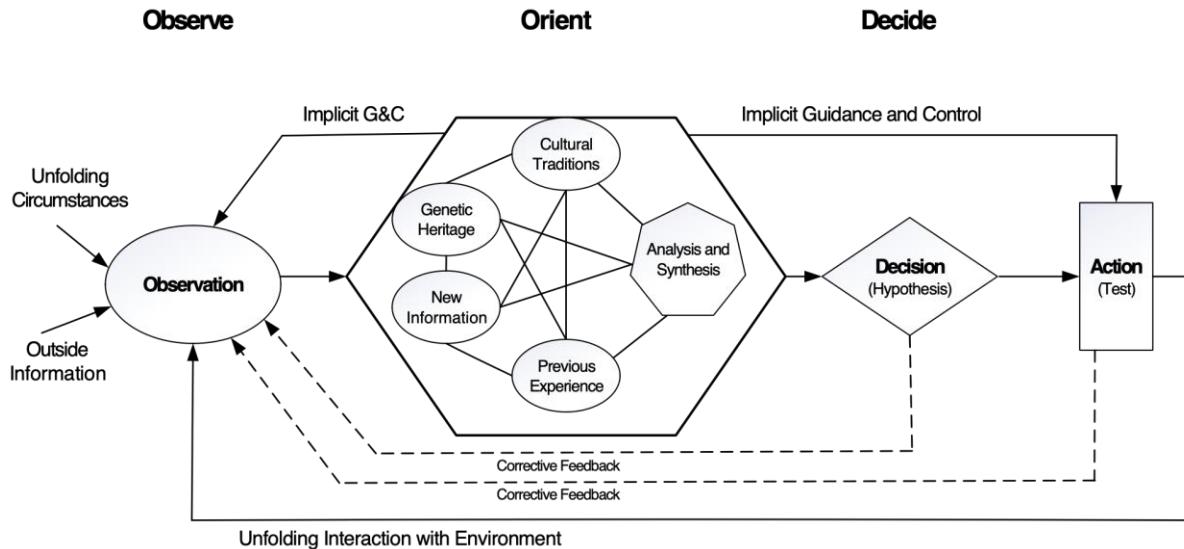


Figure 1. John Boyd's OODA Loop

## 2.1. OODA Loop Elements

Up until this time military personnel were generally told how to think through a ridged Command and Control structure. This approach assumes or takes the perspective that the only way to manage a large group of people toward a goal is through direct orders from the top. Henry Ford epitomized this with his “moving production line” where workers were required to stand in one place for 8-10 hours doing monotonous, repetitive tasks. The OODA loop is intended as a mental model that describes how knowledge workers consume and processed information. This understanding helps to better assess how to maximize the effectiveness of their actions toward a common goal. It is important to note that the Observe phase of the loop is on the left side or starting point of the diagram, this emphasizes that the OODA loop is an **information driven decision process**.

The OODA loop is better thought of as a series of loops that are continually switching back and forth as needed. The cycle can be thought of as three discrete loops and will be referred to as the Main Loop, Proactive IG&C and Reactive IG&C. Each loop has different processing speeds, mental energy requirements and corrective feedback emphasis. The OODA loop diagram in fig 2. is reformatted for discussion purposes.

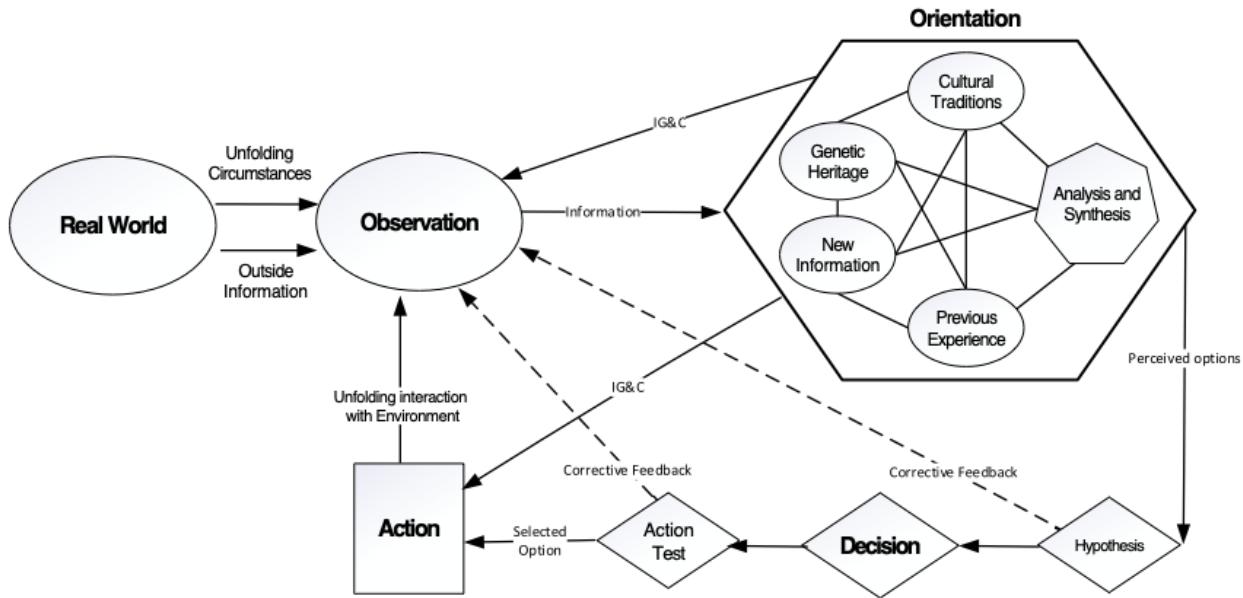


Figure 2. Reformatted OODA loop

## 2.2. Main Loop

When an individual receives information never seen before, they will process it through the Main Loop. They will attempt to identify patterns and orient themselves to understand its relevance, see fig. 3

### 2.2.1. Observation

The Observation stage is the point at which the information gathered by an individual is their perceived reality. The Outside Information feeding the stage represents the information being presented to the individual in time. This sensory input includes external sensory information (visual, tactile, auditory, taste, smell) as well as internal sensory information (position in space and time, internal cognitive reflection).

Unfolding Circumstances surrounding the individual refer to the dynamic operational environment in which the decision maker (individual) is operating. This environment includes the actions of other people and unexpected or evolving events (third party engagement) or other environmental changes (e.g., change in weather) that shape the observation process.

The Observation stage also receives corrective feedback pre and post the decision phase in the OODA loop. Corrective feedback acts to shape the observations according to the information needs and goals of the individual. [Richards]

### 2.2.2. Orientation

Boyd further developed the Orientation component of the OODA loop in the early 1970's and he discusses it in great depth. He viewed it as a crucial element where the information received will be interpreted by the individual from their inherent perspective. The Orientation phase includes all the internal cognitive reflection and information synthesizing needed to integrate new information with the individual's current understanding of the state of the world. According to Boyd, we make sense of the world by continuously un-structuring (differentiation) and restructuring (integration) our perceptions.[Boyd 1976]

As patterns are analyzed and synthesized, they produce impressions that shape an individual's perceptions and may be permanently added as new information to their heritage, culture, or experiences for future orientations. Once the information is processed, they begin to understand what options are available to them relative to their orientation. [Blaha]

### 2.2.3.Decision

The orientation process has formulated multiple options available from which to act upon. These goal-oriented options are presumed to be "good", "bad", incomplete, and limited to the capabilities of the individual. For example, the individual may have limited memory capacity, unique processing capabilities, emotional intelligence, analytical style, etc.

### 2.2.4.Action

The Action (Unfolding Interaction with the Environment) taken by the individual will change the individual's observation reference (perspective) in a manner that will require a reorientation for the next set of options and decision. Action is not always physical, for example the action may be to store the decision to be executed later. Additionally, an implicit option always available to the individual is to take no action if it is determined that more information is needed.

### 2.2.5.Feedback Loops

Boyd emphasized that the faster an individual could execute the OODA loop relative to their competition, the more apt they were to get inside the competitor's loop. He contended that once inside the competitors OODA loop, they could be out maneuvered to the point of breaking their loop. Speed can be emphasized over accuracy in the main loop because of the rapid feedback adjustments being made to the observation.

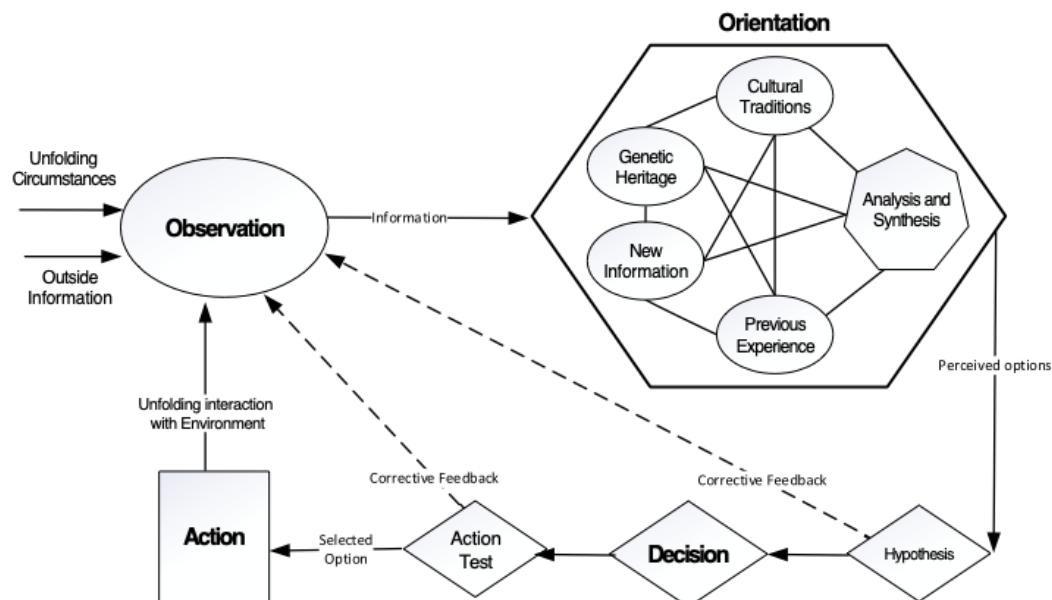


Figure 3. Main Loop

## 2.3. Internal Guidance and Control (IG&C) Loops

Internal guidance and control can be thought of as two sub-cycles that are more deliberative. As an individual develops experience from exposure, they move to the Implicit Guidance and Control loops which lessen the time and energy required to process information.

### **Proactive IG&C Loop (Fig. 4)**

The Proactive IG&C loop can be thought of as “goal-directed action”. Reflect on a time that you were presented information or a situation that you are unfamiliar with, as you process and reprocess the information it becomes second nature overtime. Eventually, when presented again with the situation, your previous processing can be leveraged to tune and amplify the signal in the noise. The amplified signal is processed faster with less mental energy. For example, a Business Analyst who skilled at gathering requirements meets with a business stakeholder to discuss their workflows. The BA knows this meeting is to gather requirements, so he arrives in a specific frame of mind. As they discuss the workflow the BA’s observation is tuned to amplify requirements he hears during the conversation. [Spinny]

### **Proactive IG&C Risk**

This cycle has inherent risk associated with it, if the individuals frame of mind is over amplifying a signal, the corrective feedback loops cannot adjust. This condition limits the ability to consider other potentially relevant and important information in the decision process. Using the same example above, the BA arrives at the stakeholder meeting with a specific idea of how the workflow should be crafted, he is biased towards a specific approach. Everything he hears from the stakeholder will be toward reinforcing his bias.

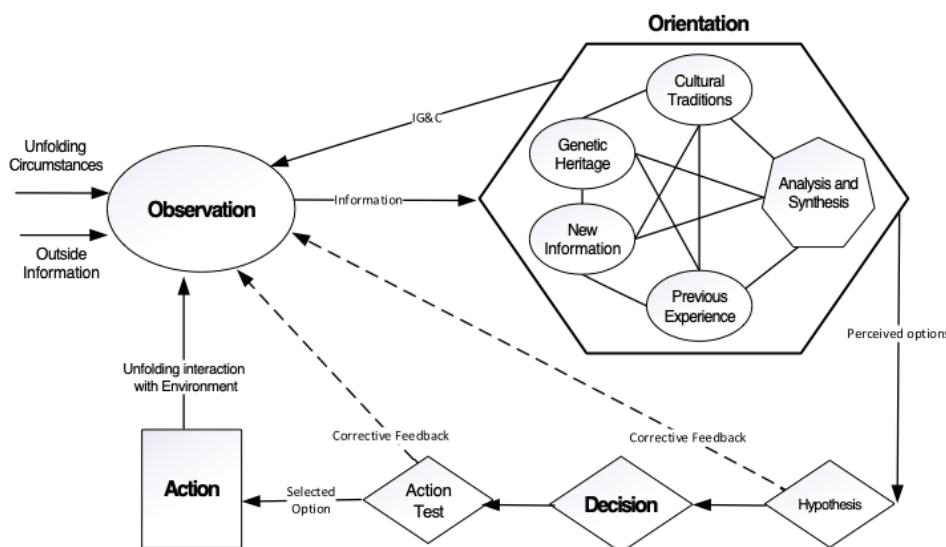


Figure 4. Proactive IG&C Loop

### **Reactive IG&C (Fig. 5)**

The Reactive IG&C can be thought of as muscle memory or being in the state of knowing. This ability is from an individual's deep understanding of a subject, activity, or situation. The energy and time to cycle through this loop is minimal. For example, a software developer that is not required to have her code peer reviewed, does it as a matter of course because of the way she was trained. Another example would be that of a Black Belt in Karate that blocks a punch reflexively.

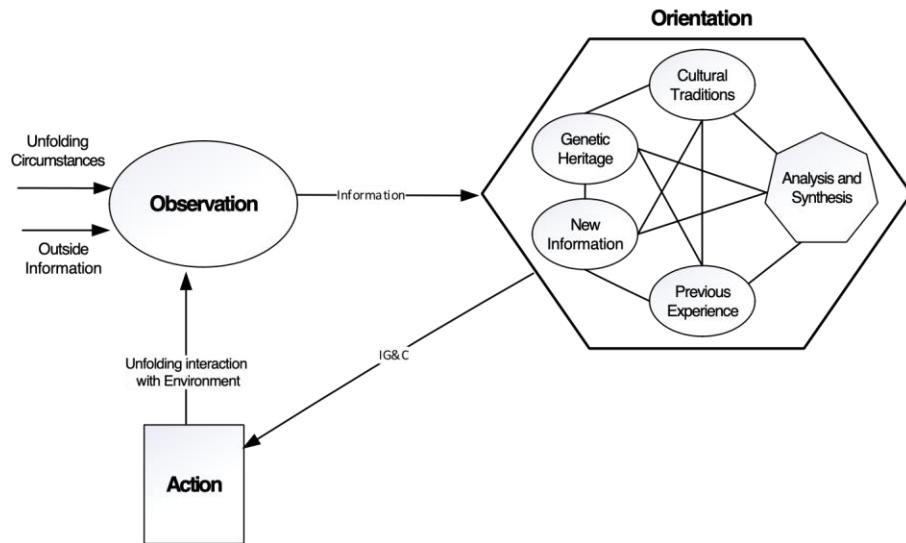


Figure 5. Reactive IG&C Loop

## 2.4. Organic Command and Control

To develop an organization mindset model, Boyd needed to understand how mindsets could be weaved together without compromising the speed and flexibility of the OODA loop. To put his scope into perspective the size of the five branches of the US military at the time was ~2 million individuals.

Boyd purposely did not create a “wiring” diagram to explain the idea of Organic Command and Control, he saw it as a shared mindset or culture that has implicit characteristics that make it fluid. Imagine trying to create a wiring diagram to explain the dynamic nature of the American culture. The diagram would require thousands of boxes and lines, that represent the multitude of perspectives, nuances, concepts, etc.

Boyd theorized that large organizations (corporations, governments, militaries) possessed a hierarchy of OODA loops at tactical, and strategic levels. In addition, he stated that most effective organizations have a highly decentralized chain of command that uses objective-driven orders, or directive control, rather than method-driven orders. This approach could harness the mental capacity and creative abilities of individual commanders at each level. Boyd argued that such a structure creates a flexible “organic whole” that is quicker to adapt to rapidly changing situations.

“A command-and-control system, whose secret lies in what’s unstated or not communicated to one another (in an explicit sense)—in order to exploit lower-level initiative yet realize higher-level intent, thereby diminish friction, and compress time, hence gain both quickness and security”. [Boyd 1987]

The use of the word “friction” was intended to describe a state of ambiguity, deception, uncertainty, but most of all mistrust. When friction is present the organization is spending energy looking inward instead of focusing on the desired goal. If an organization attempts to increase its agility and rapidity with friction present, it will amplify the friction causing confusion, disorder and ultimately chaos, lessening the chance to achieve the goal. He reasoned that if friction limits speed and agility then “harmony” could amplify it.

By viewing diverse individual orientations as a source of creativity, and exposing individuals to different skills and abilities, in a variety of situations where they can observe and orient to each other, a “harmony” is created. The harmony created by the bonds of implicit communications and trust become the culture of the organization. If the culture is tuned for harmony, then it will be able to maximize its speed, and agility.

## 2.5. Scrum

So why was the OODA loop and not another methodology (Deming, Shewharts, etc.) used by Sutherland to model Scrum? Because software is highly malleable, and requires a well-educated, adaptable workforce that can consume and act on large amounts of data and information quickly. Sutherland understood the dynamic nature of the challenge and the OODA loop design seemed like a logical fit. Like the OODA loop, Scrum is designed as an **information driven decision process** that emphasizes agility, rapid risk reduction, continuous feedback, and creativity.

## 3. OODA Examples

The following are real-world examples of decision-making successes and failures from the OODA loop perspective.

### 3.1. Medical Application

A large medical application was being modernized by an external vendor. Prior to placing the code into production, the customer was required to perform QA acceptance testing and certify the code. The contract with the vendor was created to facilitate an Agile SAFe delivery approach. Each release was comprised of multiple projects being combined into a single release every 4 to 6 weeks. This process cycle was called a “release train”, and like a real train it could not be easily stopped.

The vendor would scope, and cost estimate a project and then set a target for a specific release. The vendor would receive payment +/- 10% of the estimate upon delivery of the code into production. This contract structure incentivized the vendor to deliver as many features as possible in the shortest amount of time. The vendor code quality was to be measured by the number of defects found in production. Tracing specific defects back to individual releases, especially releases that occurred months prior was not practical. This set the stage for the vendor to sacrifice quality over time and scope.

To reduce cost to the client the vendor proposed to consolidate the System Integration Testing (SIT) and the clients Independent User Acceptance Testing (UAT) testing into a single environment. The initial concept was that once SIT testing was complete the UAT testers would then begin their testing.

The QA management not understanding the purpose of separate environments agreed to this approach. This decision reduced cost, however, it began to blur the perspective of the test teams. This blurring of environments eroded the independent perspective of the UAT test team to the point they became a functional arm of the vendor, rather than a verification and validation entity.

Over time the features slotted in each release grew, and delivery of testable code to the test environment would be late, yet the final release date did not follow suit. This placed tremendous pressure on the UAT test team to certify the quality of the release prior to executing all their tests. Over time the discussions between the vendor and the QA manager became contentious.

In reviewing code delivery slips into the test environment relative to the defects found by the UAT test team there was an odd pattern that emerged. As the UAT test time shortened, priority 1 (P1) defects would appear just prior to QA certification. The distribution of defects found in each test period resembled a “V” shape. Upon initial release several P1 defects were identified and fixed, then just prior to release another spike of P1 defects appeared even though no new bugs were found.

Through discussions with the QA manager, it was found that when the code was initially released into the test environment it was not uncommon for several P1 defects to curtail the testing effort. Once these defects were fixed then lower priority defects were identified. As the release deadline grew closer, the UAT manager would reprioritize the defects to priority 1. Per the contract this forced the vendor to fix all defects prior to launch. This action resulted in creating a “V” shaped curve of high priority defects over the testing period.

The QA manager, unable to convince his management of the vendor quality issues his team was absorbing, found an effective method to box the vendor in contractually. While unorthodox, he found a way to maintain the quality of the delivered product. This is a good example of the Main OODA loop in practice, see fig. 6. The QA manager analyzed and synthesized information that was unfolding over time.

In this example, the project was required to have an independent outside review, which identified the unusual pattern of defects. This led to discussions with the QA manager that uncovered the issue with the structure of the Agile contract. While the QA managers solution was creative it also highlighted his inability to effectively gather data and present his analysis to executive management in a compelling manner.

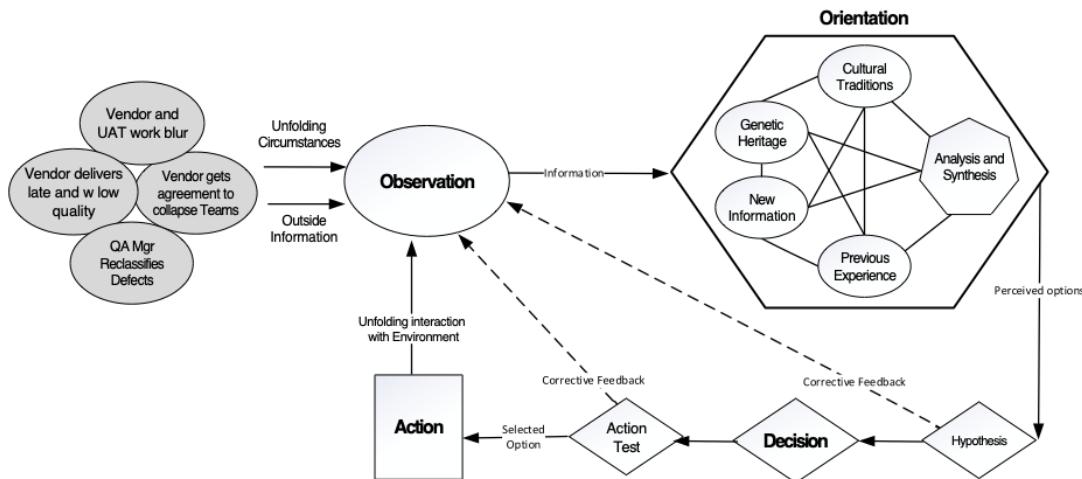


Figure 6. QA Manager Assessment and Decision Process

### 3.2. Commodity Contracts Application

An organization that managed commodity contracts required that a new system be implemented that would enable the management of more complex, long duration contracts. The business not being familiar with IT systems development, relied on the project and development teams to lead the effort.

#### 3.2.1. Alternatives Analysis Solution Decision

To “sell” a big project internally, development and business representatives need to create a business case that will often use the best-case scenario to demonstrate the benefit to the business. As part of the business case the project team produced an alternatives analysis that presented financial and technical information in an incomplete and biased manner that led the audience to a seemingly obvious conclusion.

In this case the analysis identified and rated three alternatives, of which the third was selected. They are as follows.

##### 1. Do nothing (Status Quo)

This option could be a serious consideration if an organization has decided to exit a line of business and the system would be decommissioned at a specific date. If the business is exiting a line of business the time horizon would be a key factor for the analysis. The cost of maintaining the system for the period specified, stability, security and business efficiency are all factors in the analysis. In this case the option was not valid and should not have been included as part of the business case presentation.

##### 2. Commercial Off the Shelf (COTS)

Procuring a COTS system or platform like that of a CRM or MRP is challenging when there is an existing team of developers, testers, and business analysts. The team felt threatened and were concerned they would have to downgrade their skillsets. To minimize these perceived threats the staff suggested that the business requirements are unique or too complex for a COTS/CRM solution. Uniqueness and complexity is in the eye of the beholder. Given the team was insular and at the edge of their skillsets, the tasks may seem complex when they are not. Additionally, it was presented that the risk of relying on a single vendor was too great.

There are three common reasons for the business and technical teams to exaggerate issues and rule out this option.

- A. Changes to business operations - COTS systems that directly target the market will be less customizable and the business may not wish to change their workflows to accommodate the new system.
- B. Unique and complex requirements – if there are existing commercial products that target this market then it is unreasonable to believe the requirements are unique or complex. Most likely this concern is related to #1.
- C. Cost – COTS systems will have higher upfront costs that will be prominent in the financial analysis. The price tag may be shocking at first, but the price is clearly quantified. When establishing the cost of the internal team to accomplish the task it was underestimated, left vague and/or conditional. By doing a total cost of ownership analysis, the cost can be more accurately estimated.

At the time the analysis was conducted, there was at least one COTS system in the market that was targeted specifically to this type of business. This vendor's product was not included in the analysis.

### 3. Internal development

The analysis of the COTS system option was used as the reference for the reasoning and cost for the internal development approach.

- A. Changes to business operations – none were expected because the development team would customize it to the existing workflow.
- B. Unique and complex requirements – the existing team knew the details of the requirements and their “complexity”.
- C. Cost – internal development costs were utilized to justify the price. These costs can be portrayed upfront as modest, however, there are hidden costs and risk that were not included either intentionally or unintentionally.

If the organizational structure is centered around internal product development, it is challenging to get an objective assessment because of self-interests. This bias is sometimes referred to as the “not invented here” syndrome and results in limiting innovation.[Katz] Since the business relies on development teams for technical guidance their bias will permeate through the entire organization as if it were fact.

In this case the information provided to the sponsor for their final decision was shaped (biased) toward internal development, see fig. 7. Subsequently the non-technical sponsor approved the analysis and recommendation from the development and project teams. Not being technical he approved it because of his bias to trust his team. The development teams Proactive IG&C loop amplified a specific approach that was continually being reinforced.

After the system was developed by the internal team, it was put into production with a fundamental architectural flaw. Over time this flaw severely limited business operations efficiency and growth.

The insular culture of the organization led to the perception that the requirements were very complex and could only be understood by the existing developers. Executive management's trust in the team was unfounded and the bias of

the team shaped the final decision to develop the system in house. Additionally, the governance committee was misled in the process, however, the purpose of governance is to guard against exactly this issue. At first glance it is easy to focus on the developers as the source of the issue, however, multiple post decision checks (formal feedback loops) had failed because of too much unwarranted trust at different levels of the organization.

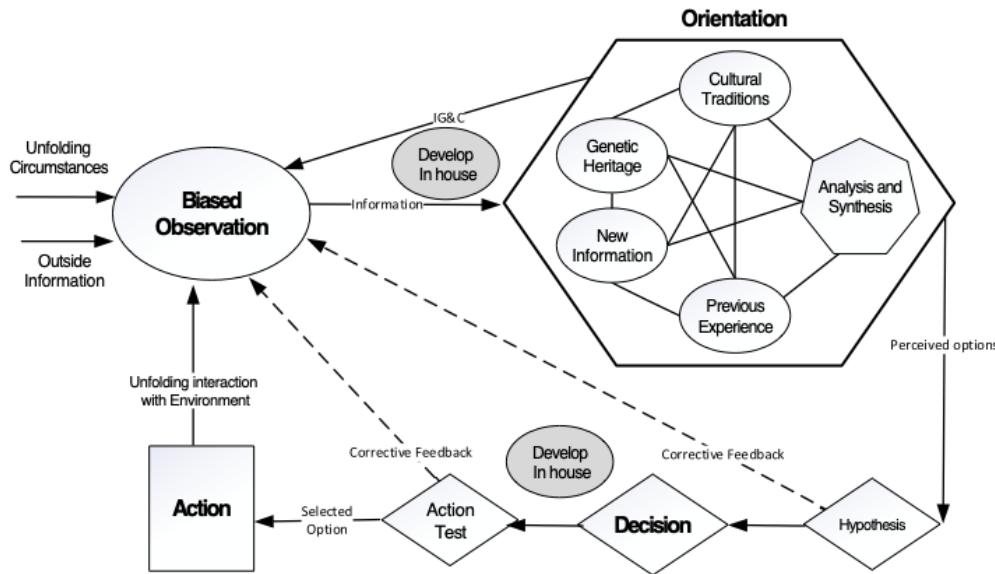


Figure 7. Internal Development Team Biased Recommendation

### 3.3. Product Manufacturer

A midsize company that developed and sold complex hardware and software was not releasing any significant features into the market for their existing products. The sales and engineering teams blamed each other and they both blamed product management. Through a series of interviews, it was clear the c-suite was unable or unwilling to prioritize and maintain a road map. The company was struggling to make revenue and the condition was being exacerbated by not delivering new features to the market. The team, under great stress, was folding in on itself.

If an individual salesperson were at risk of not making their quarterly revenue target, they would circumvent product management and contact the development managers and advocate for a specific feature to close a deal. This behavior was condoned by the VP of sales. Many of the features requested were significant and were not vetted relative to their desirability beyond the one customer. To support sales the development managers would deliver short term “sale closing” enhancements that did not close sales. Worse, if a feature required more than a couple of months to develop it ran a high probability of it being derailed at the end of the next quarter when new “sale closing” enhancements were requested. In an interview with a development manager, he expressed that they could “wait” sales out and just deploy their resources to longer term projects they “felt” were more important. It was clear that due to stress and pressure that the c-suite decision making processes were breaking down, refer to fig. 8. It was unclear as to how the situation began; however, it was certain that the culture of the organization was eroding, see fig 9 culture depiction.

Through a series of discussions/negotiations with the Sales, Development and Product Management VP's and Directors both individually and as a group, compromises were made, and a roadmap was established. In addition, a specific set of development capacity was left open to deal with simple tactical enhancements that could be turned around quickly. To support the approach the VP of sales committed to having all sales request go through product management, and the CEO understood there would be a short term hit to revenue as the organization was realigned. As short-term roadmap milestones were achieved and new features delivered, cooperation, trust and focus began to permeate the organization and accelerate the delivery of significant features in a more predictable fashion. The

company still struggled in a hyper competitive market, however by reducing the internal friction and reorienting the organization towards a set of goals it increased their capacity to explore new markets and revenue streams as a team.

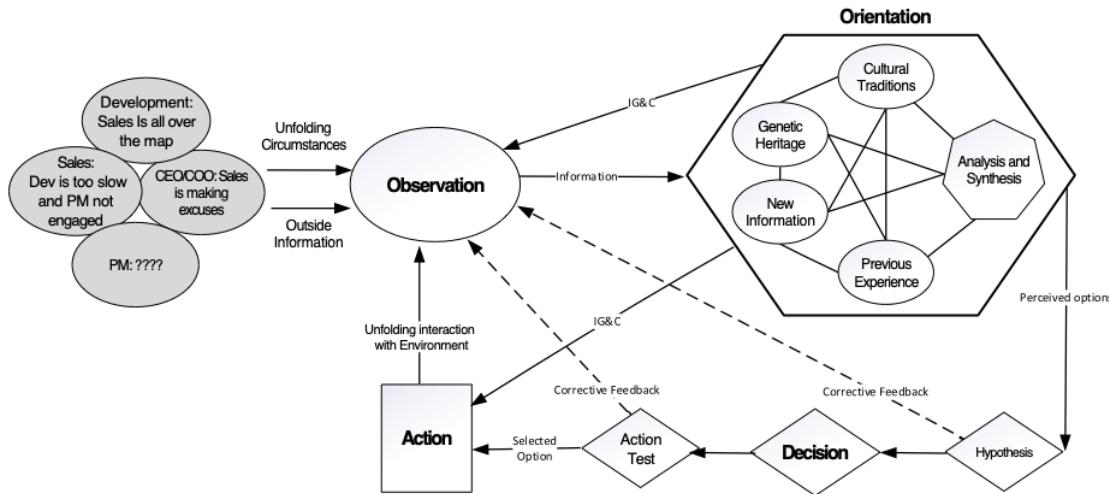


Figure 8. Multiple Individual Perspectives

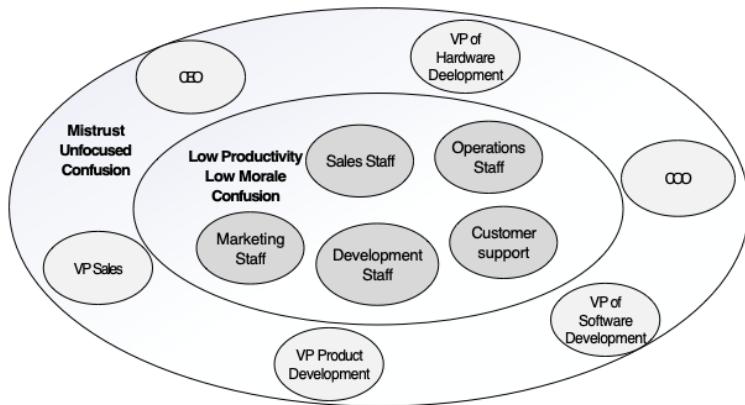


Figure 9. Organization Bonds

## 4. Conclusion

Most organizations focus on catching bad decisions through various control processes rather than spending the time and effort on making more informed decisions at the outset. This is akin to how an organization thinks about software quality. Some organizations rely only on the QA testing phase of the SDLC to control for defects, while others look to improve the quality of the final product by focusing on the source of the defects.

Being data driven and having the ability to “pivot quickly” or be “agile” is extremely desirable, however, without a mindset that supports these traits, the degree and fluidity of the agility will be minimal, ineffective, or even destructive to the organization.

## References

- Sutherland, Jeff, and Sutherland, J.J., 2014. *Scrum: The Art of Doing Twice the Work in Half the Time*. New York: Currency.
- Coram, Robert. 2002. Boyd: The Fighter Pilot Who Changed the Art of War. Back Bay Books.
- Richards, Chet, 2020. Boyd's OODA Loop. Necessity Vol 5, Issue 1. Sjokrigsskolen. Norway.
- Boyd, John R., "Destruction and Creation" (unpublished paper, Sept. 3, 1976), Defense and the National Interest, Dec. 6, 2007.
- Blaha, Leslie M. 2018. Interactive OODA Processes for Operational Joint Human-Machine Intelligence. NATO Science and Technology Organization MP-IST-160-PP-3 Bordeaux, France.
- Spinny, Chuck. 2019 Evolutionary Epistemology Talk. Expeditionary Warfare School (EWS) Quantico, VA
- Boyd, John R., "Organic Design for Command and Control" (unpublished briefing, May 1987), slides 26, 16, Defense and the National Interest, Dec. 6, 2007.
- Katz, Ralph, and Allen, Thomas 1982. Investigating the Not Invented Here (NIH) syndrome: A look at the performance, tenure, and communication patterns of 50 R&D Project Groups. Sloan School of Management M.I.T. Cambridge, USA.

# **Hey Alexa - How To Test My Voice Assistance? An Automation Approach**

**Juan Delgado**

**Co-Authors:**

**Lucaz Tutur**

**Piotr Wroblewski**

**Deanna Raven**

**Christopher Schoppa**

juan.delgado@mobica.com

lucasz.tutur@mobica.com

piotr.wroblewski@mobica.com

christopher.schoppa@mobica.com

deanna.raven@mobica

## **Abstract**

This paper presents an idea or a concept for test automation on gadgets to use voice assistance.

Nowadays the Voice assistants, like Alexa are taking over households, in the USA anyway, where one in four households owns at least one device. Those gadgets provide an increasing part of the added value of modern gadget systems and thus become more complex to test.

Where one of the most common general problems with the test automation framework working with hardware and software to test the voice assistance seems to be forgetting that any test automation project is a software project in its own right. Software projects fail if they do not follow processes and are not managed adequately, and test automation framework projects are not different. Of all people, test engineers ought to realize how important it is to have a disciplined approach to software development.

This paper focuses on developing a Proof Of Concept to approach an automation framework running a script on Python and using different libraries that strictly follow the QA best practices and coding to execute a common E2E Test Scenario, where the main goal is to approach a quality strategy to test the Alexa voice assistant on real android gadgets to validate "Hey Alexa" in different acoustic noise conditions and different utterances to validate how many times is successfully detected; looking to help on QA community sharing knowledge to foster a software testing quality.

## Biography

### **Juan Delgado**

*Juan de Dios Delgado has more than 15 years of experience as an engineer. Currently as Test Manager, with Mobica. With strong project management skills, key areas of expertise include understanding complex business requirements & formulating robust test strategies; developing automated test solutions with the ability to interface between Development, Project, QA & Test Teams to ensure execution of test strategy; and extensive software engineering skills.*

*He has participated in several projects, in the last years. He has worked on-site in the US and Europe; near shore from Mexico to the US, and offshore from Mexico and India to Europe; working strongly with Functional Testing focusing on Automated Testing for Mobile devices, Web Applications, and Web services.*

*Professor/Lecturer at Universidad Tecnológica de Aguascalientes and Universidad Panamericana in Aguascalientes, México.*

### **Lucasz Tutur**

*Lukasz Tutur has over ten years of experience in various projects for top companies. Firstly, system tests of Broad Band Access OSS systems, running both in Windows and Unix environments where his responsibilities included LSP and functional testing, using black/grey box techniques, incident reporting, test specification writing, and test list designing.*

### **Piotr Wroblewski**

*Head of Delivery Mobica US. 20+ years of experience in creating, defining, developing, and delivering innovative global telecommunications, defense, cloud, and software technology products, services, and solutions.*

### **Christopher Schoppa**

*Chris is Head of Solutions for Mobica USA. He has spent his career at the intersection of technology and business value across large volume web applications and services, automotive infotainment/digital cockpit, mobile devices, and IoT. I work to understand your product strategy, then come up with ways to use appropriate technology and deliver novel use cases that grow your business.*

### **Deanna Raven**

*Senior Program Manager/Senior Technical Project Manager with 12+ Years' experience across the software industry with a passion for leading change efficiently and an unwavering enthusiasm for solving business problems and uncovering new opportunities using the power of technology and data.*

## 1 Introduction

We are living in an age of smart devices, where voice assistant applications or devices have become the access point for controlling several functions daily. Some of these popular devices are Alexa by Amazon, Google Home, and Siri. Besides most new smart gadgets that are launched into the market nowadays also have inbuilt voice assistant features.

For example, Alexa is one of the most intelligent voice assistants that can solve diverse queries on specific gadgets which is a must need to do functional testing on the voice assistance under the most popular gadgets thoroughly in order can deliver an optimal user experience.

Test the voice assistance feature of the devices may seem simple on the surface, but in fact, many challenges determine the success of the skill

This paper is willing to show a proposed proof of concept with the quality approach testing the Alexa voice assistance under some gadgets as well as the technologies and methodologies implemented on the strategy besides the results and benefits.

## 2 Voice Assistance

Voice Assistant is a virtual assistant that uses speech recognition, natural language processing, and speech synthesis to take action on user utterances to help in our day-to-day and they are available in cars, household devices, smartphones, and several apps.

As mentioned, this paper will discuss how we can get into voice assistant testing, and we will propose a QA approach to develop a black-box automated functional test Proof Of Concept to test a voice assistant, specifically on Amazon wearables, then Hey Alexa! please let's start.

### 2.1 Background information

Much like the structured language used by humans, where a word signifies an object, digital assistants rely on a combination of utterances and intents to respond to voice commands.

- Utterances: An utterance is an input from the end user. It may be a sentence, such as “play baby shark on amazon music,” or a fragment of a sentence, such as “open amazon music”. Utterances are impacted by variations in user location, gender, age, etc.
- Intent: The intent is a purpose or goal expressed in a user’s input (utterance), such as playing baby shark on amazon music. An intent represents what the user wants the digital assistant to do.
- Entity: An entity represents detailed information that is relevant to the utterance. For example, in the utterance “play baby shark on amazon music on Fire TV” where Fire TV is an entity.

### 2.2 Alexa

It all starts with Amazon's wake word "Alexa", this voice assistant is part of the Amazon services that are built around the Alexa devices to support voice commands. Alexa voice service was first introduced with the Echo device. To interact with a smart device, a user must say Amazon's wake word “Alexa” and then say a certain command. The device sends then the user commands to the Alexa voice service which is the cloud service responsible for processing the voice commands and returning the response back.

On a generic workflow explained, a user can say Amazon's wake word "Alexa" plus a command (called "utterance") into a device. This utterance inputs are interpreted as voice commands that are detected and recognized. The actual output can be seen as a voice answer given by the device used to the user, or also can be a performer action on the device; as a Software developer in Test even could be an output value called "logs" shown on the Logcat window in Android Studio.



### 3 The Voice Assistance testing – The Big Picture

Voice assistance presents a huge opportunity for companies could engage with their users, however, such as is complex technology increases the difficulty of testing for that reason a correct quality strategy approach is essential for the best user engagement without negative impacts on their experience.

Common challenges for testing voice assistance include:

- User: A wide range of variables with different languages, accents, ages, and gender.
- Device: Lab with required devices as 1st party (Amazon Echo, Google home, etc.) and 3rd party (android devices, cars, smart gadgets, etc.)
- Environment: Different background noises.

#### 3.1 Proposal

In this paper, we will focus on black box testing to test the trigger of Amazon's wake word "Alexa" and to validate the performing actions with an in-house build python script besides using different tools and methodologies that support the most common aspects of quality approach in the proof of concept that will be shown using Amazon wearables that explore a real-life example to help you get started with voice app testing.

## 3.2 Goal

The QA approach strategy consists of building a test automation script that plays an utterance using Amazon's wake word "Alexa" on a computer or external speaker to be triggered on wearable Echo5 and performed on Android Mobile Phone or Fire TV apps.

Our potential solution is to define an approach based on an Automated test solution using python to perform the speech of Alexa and to read the Android logs in the wearable to validate if the trigger was successfully accurate and the action performed and willing to be shown in a demo as proof of concept.

# 4 Project Methodology

Let's look at some of the methodologies applied in our Proof of concept to achieve a successful demo.

## 4.1 Scrum

Working with Scrum methodology begins with identifying the problem and its core components, in this case, the team focused on the MVP. Knowing this we work in a series of sprints to achieve our goal.

The team was divided into three roles that ensure the workflow needed. First, we have the role of Scrum Master, this role is akin to a coach, who helps the team with its expertise. He focuses on improving the team's effectiveness.

The next role is the Development team, they work to deliver a potentially releasable increment of finished tasks at the end of every sprint. Their task is to give structure to the development project and determine the number of tasks given to each member. They also carry out the task of working on writing the paper.

Finally, we have the role of the Software Developer Engineer in Test, this person writes the tests proposed by the Development team, using all the given tools, for identifying the possible flaws of the web platform. This person analyses the outcome of each test and documents their conclusions. Since this is an automated framework, the Developer also integrates the tools needed for the scalability and automation of the Proof of Concept.

## 4.2 MVP

To define our proof of concept and build a demo to be shown to the PNSQC 2022 audience, we based our project on the true meaning of a minimum viable product (MVP) focusing on our python script with the Minimum set of features of being viable.

We based our solution on the following:

- Execute the speech of the Hey Alexa
- Validate the triggering of Amazon's wake word "Alexa"
- Validate the response of the triggering
- Validate action performed in the wearables.

## 4.3 Best practice testing

### 4.3.1 Testing

During the development of this automated framework, our team focused primarily on three main types of testing. Functional testing, black-box testing, and regression testing.

### 4.3.2 Testing techniques

To improve the quality of the tests there are various techniques during the design process of a test suite. Our design process was dynamic, a combination of specification based with experience-based techniques.

The specification-based techniques are primarily based on black-box testing which included techniques of its own such as use case, user story testing, state transition testing, boundary value analysis, and equivalence partitioning, among others.

Using experience-based techniques, such as exploratory testing or error guessing also is important to be considered in the test design process due the team might be composed of engineers with different skill sets and backgrounds and this could allow having a variety of perspectives and insights to lead a better robust test coverage.

For our Proof Of Concept and following the MVP, Scrum methodologies, and testing techniques, our testing will cover the following functional test cases that will be shown in our demo presentation.

- **Hey Alexa – Play a song!**
  - Functionality: When the Amazon Echo is connected to a Fire TV Amazon's digital assistant, Alexa, can help to play movies and TV shows from Amazon Prime Video, Hulu, and more using the wearable Fire TV. Also, can do most of the common functionalities such as stop, play, etc. with voice commands.
  - User story: As a Software Developer Engineer in Test, I need to validate that when Alexa wake-word is triggered should be detected by the Amazon Echo 3 to perform the utterance action on Fire TV.
- **Hey Alexa – Make a phone call!**
  - Functionality: When the Amazon Echo is connected to an Android device Amazon's digital assistant, Alexa, can help to make a phone call using the wearable mobile phone.
  - User story: As a Software Developer Engineer in Test, I need to validate that when Alexa wake-word is triggered should be detected by the Amazon Echo 3 to perform the phone call action from the mobile phone.

## 4.4 Best practice coding

There's no standard process when it comes to coding. Therefore, industries have been working on creating rules or tools that ensure that all code has the same structure and has an organic flow throughout the code structure.

Below we discuss a few best practices and techniques that we have used in the development of our demo to enhance the coding of the Proof of Concept.

### 4.4.1 ESLint

ESLint is a tool that finds and fixes structural and compilation problems in the code. ESLint is a pluggable tool that helps identify and report patterns found in JavaScript code, with the goal to make the code more consistent. You can make your own set of rules, or you use the rules that industries like Google or Airbnb employ.

#### 4.4.2 Code reviews

Code Review is the process in which a programmer consciously and systematically checks the code for mistakes or structural mistakes. The code reviews have proven that it helps to accelerate the streamlining of the development process for projects.

Code Reviews, when done right, can improve the programmer workflow, reducing the amount of time the Quality Assurance Team requires to check the code, therefore saving time in general.

Particularly in this project we employed the Over-the-Shoulder style of code review. We decided to use this technique because it's the easiest and more intuitive type of style to adopt. Once the code is ready, the supervisor of the code downloads the branch of GitHub that has the new code that needs to be reviewed. Once the code is reviewed and the team agrees with the changes, the programmer makes a merge from the review branch to the main branch of GitHub.

## 5 Framework

The voice assistance undergoes rigorous functional tests that ensure the correct functionality of Amazon's wake word "Alexa" and its performance. Usually, we have two types of testing: manual and Automation testing. By implementing the appropriate framework for automated testing, we can significantly increase the speed and accuracy of testing, providing a higher ROI from the project. Following the standards for a good Framework would achieve relevant automated tests, concise reporting, team consistency, implementation, and maximizing re-usability on the automation testing.

### 5.1 Tool

We will now talk about the main tools used in a particular way:

#### 5.1.1 PyCharm

PyCharm is the most popular IDE for Python and includes great features such as excellent code completion and inspection with an advanced debugger and support for web programming and various frameworks.

We will use PyCharm for the Test Automation project for running tests automatically, managing test data, and taking the results to improve the quality approach.

#### 5.1.2 Android Studio

Logs are very useful when Software Developer Engineer in Test is diagnosing an error or validating an expected value output from the Android tested, which includes several logs that deal with different parts of our performance of the testing,

The Logcat window in Android Studio displays those Android logs in real time and helps us to use it to collect and visualize those logs.

### **5.1.3 Audacity**

Audacity will help us with the different audio utterances on m4a file format that need to be recorded, edited, mixed, and added effects to be played during the test execution according to the test case.

### **5.1.4 GitHub**

A tool that as a programmer, the more you use the more you begin to depend on it due to its incredible ability for version control and management. It allows us to collaborate remotely with all team members, something that today has even more value due to the ongoing world situation. This makes it perfectly suited for our project. Thanks to our use of the agile Scrum methodology we will have a higher quality and security of deliverables for each sprint, which is combined with a software architect who will constantly help us have better code in our project.

## **5.2 Core (code structure)**

### **5.2.1 Suites**

This section contains the Tests files. Each file contains the tests that were identified as inherent for assessing the correct functionality of Amazon's wake word "Alexa" triggered.

- **Alexa-youtube.py:** This contains the following tests - Alexa plays a YouTube video on an Android device.
- **Alexa-phone-call.py:** Contains the following tests: Alexa makes a call on an Android device.
- **Alexa-firtev.py:** Alexa plays a video on Fire TV

## **5.3 Libs**

### **5.3.1 Adb Folder**

Run an ADB command and other functions that will execute specific commands like install apps, open apps, close apps, make a phone call, download logcat, etc.

### **5.3.2 My App Folder**

Create an object with the package name of the app and perform a specific action in the app.

### **5.3.3 My Device Folder**

Create an object with the ID of the device with main functionalities to perform

### **5.3.4 Utils Folder**

Contain the functions to help us to play an utterance on the computer.

## 6 Conclusion

### 6.1 Outcome

This project is just a glimpse of how to start an approach of quality on voice assistance functionalities that might evolve throughout the Software Testing Life cycle to create an Automated Framework.

Mobica being part of the PNSQC 2022 thinks that can contribute to the QA community by sharing experience and best practices so the community interested can start their own automation project.

## References

- <https://www.einfochips.com/blog/addressing-the-common-challenges-in-voice-first-testing-through-automation/>
- <https://www.codexa.net/alexa-skill-kit-beginner-tutorial/>
- <https://techwiser.com/fix-youtube-not-working-amazon-fire-tv-stick/>
- [https://www.slideshare.net/ApplauseMarketing/solving-the-top-problems-of-voice-testing?from\\_action=save](https://www.slideshare.net/ApplauseMarketing/solving-the-top-problems-of-voice-testing?from_action=save)
- <http://www.diva-portal.se/smash/get/diva2:1473113/FULLTEXT01.pdf>
- <https://alan.app/blog/voiceassistant-2/>
- <https://www.slanglabs.in/voice-technology-glossary/voice-assistant>
- [https://www.softwaretestinghelp.com/software-testing-techniques-2/#User\\_Story\\_Testing\\_Agile](https://www.softwaretestinghelp.com/software-testing-techniques-2/#User_Story_Testing_Agile)
- <https://www.audacityteam.org/>
- <https://www.jetbrains.com/>
- <https://developer.android.com/studio>

# How I Learned Things and Influenced People with Visual Notes

**Moss Drake**

[mxmossman@gmail.com](mailto:mxmossman@gmail.com)

## Abstract

High tech technologies and skills constantly change. In this whirlpool of new ideas, do you find yourself sometimes underwater -- struggling to organize the maelstrom of new information? If there were only a way to process information as you hear about it and organize it into accessible repositories for reference.

This was a problem that Moss Drake experienced until he discovered the practice of visual note taking or Sketchnotes. Sketchnoting is a method for taking notes that uses visual, kinetic and introspection to reinforce learning. Drake has ten years of experience using notes to learn and share discoveries from work, conferences, and meetups.

In addition to providing a way to take more effective notes, Drake has discovered a side benefit: sharing notes with communities is a way for a self-professed introvert to overcome barriers in networking, collaboration and communication.

## Biography

Moss Drake has over three decades of experience developing professional software with a focus on healthcare and insurance. He has a passion for improving team communication, discovering innovative ways to deliver software, and using agile methodologies. In addition to volunteering and being on the board of several organizations such as the Pacific NW Software Quality Conference, The Flying Focus Video Collective and Hack Oregon, Moss enjoys playing music, traveling and drawing.

*Copyright Moss Drake, 2022*

## 1 Introduction

Communication is hard. George Bernard Shaw observed that "the single biggest problem in communication is the illusion that it has taken place." In English we often use words imprecisely, and even when used correctly, simple words may have multiple meanings. For example, the Oxford English Dictionary cites over 600 meanings for the word "run." (Winchester 2011)

Yet, software development is all about communication. Peter Naur, of Backus-Naur notation fame, wrote an essay called "Programming as Theory Building" that says "...programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand." (Naur 1985) In my experience, agreeing on this theory is half the difficulty in developing software. The key to success is having a shared theory.

According to Karl Wiegers, "If you don't get the requirements right, it doesn't matter how well you execute the rest of the project...High-quality software development is based on high-quality requirements, elicited from the right people, crafted into usable forms, and clearly communicated to everyone who needs to know." (Wiegers 2021) A survey in 2018 found that 59% of U.S. workers say communication is their team's biggest obstacle to success (Conrad, Andrew 2018). Others have claimed that communication is "the nervous system of any organized group and the glue that hold organization together" (Weldearegay 2012). Given that, how may we improve communication chances for success?

Using alternate modes of communication can help improve alignment. In my career I have often leaned on drawings and diagrams to supplement written and verbal communication. Thus, I was primed for adoption when I discovered Sketchnotes, a framework for visual note taking. By using Sketchnotes, I have had a surprising number of benefits beyond improved communication

## 2 The Challenge

In the technology arena there are a lot of new concepts to learn. Some are fleeting, like Windows 8 Widgets or Facebook Home (aka the Facebook Phone) and others, like Java persist. Further still, some evolve over time and suffer obfuscation by marketers and opportunists. Regardless, technical presentations have a lot of details - It can be confusing to process and prioritize. In a swiftly changing environment, how much time should one spend on any topic?

A simple solution is to keep outlines of notes on various topics, with blocks of details when needed. Even so, notes can be forgotten or duplicated. Writing notes can become rote, so much that one ends up typing, verbatim, the details from the speaker, website or book. While written notes are searchable, they are dense to read or to quickly scan.

To further exacerbate the problem, many in the software industry are introverts, which means they find talking with people to be draining. Introverts may also find it difficult to initiate conversations. In a meeting or conference I would much rather follow my internal dialogue than to have one with a person in the room. Getting a dialogue with multiple people is sometimes not only tiring, but often unmanageable for complicated topics.

If only there were some way to organize ideas quickly to be memorable, quick to scan, easy to share and facilitate collaboration. This is my journey in discovering the benefits of visual note taking.

### 3 Discovering Sketchnotes

In 2010 I was sitting near a person at a conference. While the keynote spoke, the person used an iPad and a stylus to record their impressions of the presentation. I watched in interest as they combined symbols, text and quick doodles on the tablet to take notes of everything the speaker covered. Afterward, I spoke with Erin Dees, who was recording the session. They were using an app called Paper to record the Sketchnotes of the keynote.

While familiar with the book *Visual Meetings* (Sibbet 2010), this was my first inkling of the scope of possibilities using drawings for taking notes. Erin referred me to Mike Rohde's book *Sketchnotes* (Rohde 2012), which led to a number of simple concepts to get started.

The premise of Sketchnotes are

- Combining icons, drawing and text together to quickly document a topic
- By using multiple modes of thought, different parts of the brain are activated which helps store and process the concepts
- Similarly, whoever reads the notes also gains the benefit of triggering multiple pathways in the brain

The remainder of this paper discusses three cases of using visual note taking that improved communication and achieved other benefits.

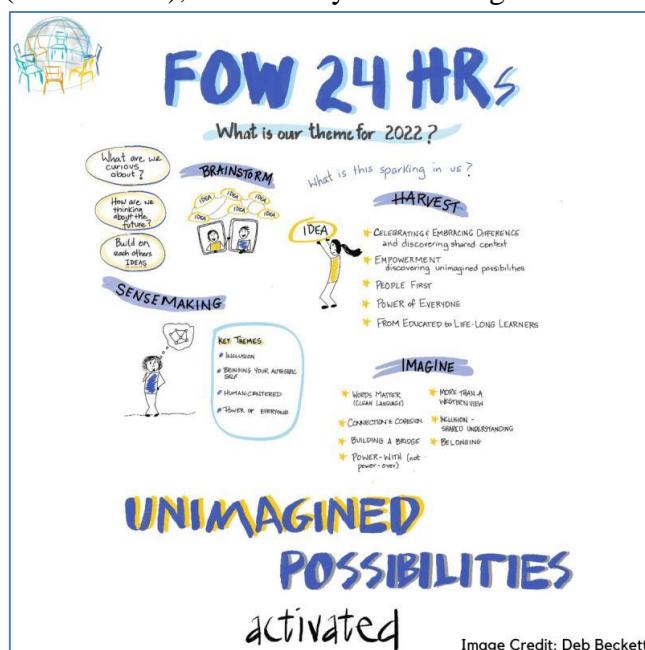


Image Credit: Deb Beckett

An example of Sketchnotes by Deb Beckett for the Future of Work (FOW) Conference

### 4 Case 1: Learning

I began by using Sketchnotes for my own personal reference. When attending a talk or a conference I would bring along my iPad, stylus and an open mind. There are several things that occur when getting ready to document a talk that reinforces learning. Looking up the author and the title of the talk, then writing it down in the upper left corner is a good way to internalize the information. Taking a minute before the talk begins to illuminate the title and author reinforces retention.

In the book, Mike Rohde suggests several general structures for taking notes:

- Linear
- Radial
- Vertical
- Path
- Modular
- Skyscraper
- Popcorn

Most of these structures are self-descriptive. The Popcorn structure can be thought of as spilled popcorn – a series of small bits of information that are not necessarily in significant order. The Sketchnotes book has numerous ideas for drawing, listening, organizing, and building one's visual processing skills. This paper is not a replacement for Sketchnotes. Rather than focusing on "how," this paper emphasizes "why" visual note taking is valuable.

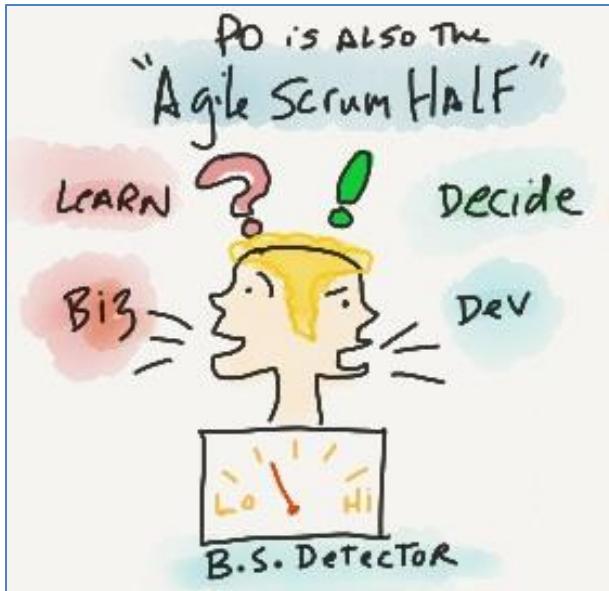
Drawing engages the brain in multiple ways: visually, cognitively and kinetically, making meetings more creative and memorable for everyone. A Wall Street Journal article began "Employees at a range of businesses are being encouraged by their companies to doodle their ideas and draw diagrams to explain complicated concepts to colleagues." (Silverman 2012) Drawing can be used to promote engagement, visual thinking, and enhance note taking.

Studies show that drawing enhances retention. In "Moonwalking with Einstein" Jonathan Foer devotes a chapter to discussing mind maps and how they correlate with memory palaces (Foer 2011). A memory palace is a visual mnemonic device used to help organize and recollect bits of information. As the team works with the image they inadvertently create spatial memory palaces of the system in their minds. Later, when writing code or testing, the image can help clarify details and provide a context within the larger system.

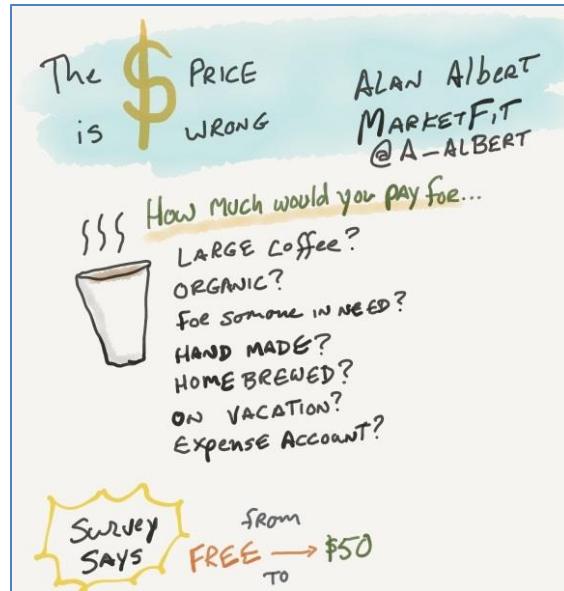
In my own experience, using visual note taking has the same effect. Where once my mind might have drifted during a talk, the process of doodling and translating from spoken word to notes helps with retention. I can immediately recall certain topics because of the drawings associated with them.



Detail of Sketchnotes by Moss Drake of the session "Putting Practice to Proficiency" from Agile Open NW 2015



*Detail of Sketchnotes by Moss Drake of the session "Being a Better Product Owner" by Ken Pugh*



*Detail of Sketchnotes by Moss Drake of the session "The Price is Wrong" by Alan Albert at Product Camp Portland*

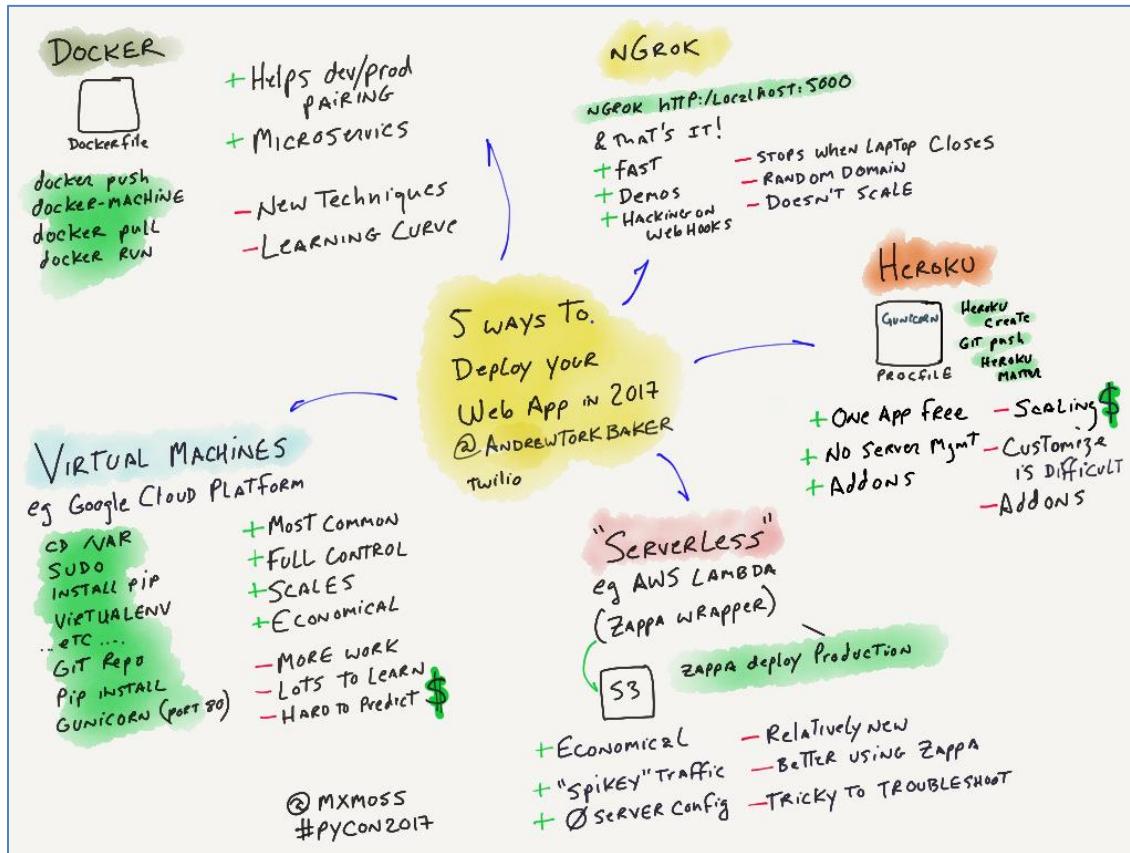
The illustrations on this page and the previous are examples of the *Pricing Discussion* led by Alan Albert at Product Camp Portland in 2019, the keynote on *Being a Better Product Owner* by Ken Pugh from PNSQC 2015 and Matt Plavcan's presentation on *Putting Practice to Proficiency* from Agile Open NW 2015. The coffee cup, the two-faced Product Owner, the pointed haired boss relating to programming, are all images that anchor the discussions.

And, when trying to find notes on a particular topic, it is easier to scan through a series of drawings than to read text. Pictures generally show superior recognition relative to their textual labels: an article from MIT claims that the brain can identify visual cues in as little as 13 milliseconds. I have found this advantage to be true personally as well as when working with others. (Trafton 2014) The end result of using Sketchnotes is improved retention and faster ability to refer to previous notes than when using plain text.

## 5 Case 2: Networking

In 2017 I had the opportunity to attend Pycon and made Sketchnotes of the talks I attended. Being new to Python, many of the talks were eye opening or discussed novel topics and technology. As part of my normal process, I used Paper on the iPad to take notes in real-time and then shared them on twitter with the #pycon hashtag. Consequently, the notes were seen in real-time by many of the people at the conference.

One of the presentations was “Five Ways to Deploy your Python Web App in 2017” by Andrew Baker. The talk had five main areas, so I chose a Radial design for the layout. The topic itself was technical, so it was hard to render most of it as graphical illustrations. Still, the segmentation of the topics, combined with quick checklists and some color highlights made the notes simple to



scan and understand. The tweet received 138 likes and 64 retweets. Andrew Baker, the presenter, mentioned it was “Clearer and more beautiful than I could have ever come up with!”

Additionally, the [tweet](#) started a thread about ways to improve some of the deployments in certain cases.

More surprising was the result from the Sketchnotes on Moshe Zadka's talk "Automate AWS with Python," which covered how to automate AMI builds, building Cloud Formation Templates and automating S3 bucket management. In this case, the details of both AWS and Python were unclear to me at the time so it was difficult to be sure how accurate the notes were. I posted the notes after the talk, and soon got a direct message from Moshe asking if he could meet me at 2 pm that afternoon. I began to wonder if something was wrong: did I make a mistake in the notes? Was his talk somehow proprietary? Will he want me to retract the notes from Twitter?

It turned out that Moshe wanted to thank me for capturing his talk in a succinct manner, and he also wanted to point out two things to add to the notes to make them more useful. We talked for a while and I understood the concepts better. As we parted, he gave me a business card and connected with me on LinkedIn.

The bottom line for sharing notes at a convention or on social media: people will appreciate the attention focused on their efforts and will reach out to you to provide more insight. In this case, the “normal” networking situation had been reversed: The person listening to the talk was approached by the speakers and provided with more information without having to initiate any

contact in person. This was a “perfect” solution for an introvert who finds it difficult to begin conversations with strangers in a crowd.

## 6 Case 3: Influencing

As yet, we have examined using Sketchnotes for capturing the gist of a talk. What about using visual note taking for taking notes during a meeting where the topic is still being explored?

One organization I worked for had a bear of a project that had been continually put off: The migration of a key system from a mainframe to a Windows server. The mainframe was expensive, many times more expensive than running a Windows server, and it was aging. The technology was limited and most of the system was written in COBOL. Unfortunately, the system touched several different departments and not a single manager wanted to assume responsibility for the project. I decided to create a shared vision of the project using visual note taking and then we could assess whether to proceed.

The culture of the company was conservative, so they weren’t ready to adopt agile methods for collaboration. I decided, however, to take a slight chance and use visual note taking to facilitate a vision for the migration. About 10 to 15 people attended the session. We started with a blank whiteboard and elicited the full image of the system, with dependencies and criticalities. As I facilitated the visioning session, I illustrated the subsystems and links between them, calling out specific data requirements, reports and other dependencies. The session lasted for a little over an hour and by the end we had a vision, a literal picture, in Sketchnotes, of the system to be replaced.

Having everyone in the room short-cut the time required for writing and reviewing documents. Since this was a conservative company, this session avoided any discussion of agile practices or perceived “woo-woo” collaboration techniques and jumped directly into designing the project plan from the shared knowledge in the room. Please note that I had enough history with both the system and the stakeholders that I was comfortable with taking this approach. An outsider to the company may have had to rely on a more conventional method for writing a vision document.

The process was actually an agile method for gathering the requirements, without having to speak the name “Agile.” Shared drawing, whether facilitated by one person or participated in by many, is an agile approach to capturing the zeitgeist of the room (the shared knowledge of the room). It follows many of the features of an agile process:

- Uses the Plan => Do => Check => Act cycle
- Proposes the simplest answer first, elaborate if necessary
- Uses short feedback cycles to avoid errors
- Involves collaborative design, if desired

The end result of this vision was that the project was greenlit and implemented in less than six months. The project was successfully received and is still in production. Without the shared vision of the system across multiple departments and zones of responsibility, this would probably still be a pending project.

## 7 Conclusion

It has been said that a picture is worth a thousand words. Using Sketchnotes will not only save time but brings many benefits, a few of which were covered in this paper.

As discussed, using Sketchnotes can improve personal notes by:

- Improving the ability to learn by processing in multiple modes
- Improving retention and recall
- Providing context for any topic
- Being easier to scan than dense text

Beyond personal improvement through Sketchnotes, the author experienced the external benefits of improved networking, higher visibility in crowds, and the ability to influence and facilitate successful projects through visual note taking.

This is only an introduction to the method of visual note taking. Hopefully you will be inspired to explore the topic more and take your own journey.

## 8 Further Exploration

This section includes people who have influenced me in thinking about Sketchnotes.

- Erin Dees can be found on twitter @undees
- Mike Rohde wrote the book on Sketchnotes and is on twitter at @rohdesign
- Zeger van Hese primarily sketches on paper with pencil and pen, then cleans up the notes after the talk and posts them to twitter @TestSideStory
- Julia Evans focuses on learning UNIX commands and programming. @b0rk
- When drawings are included, foreign languages become less of a barrier. Here are some non-English visual note takers
  - Luísa Diebold (Portuguese) @luisadiebold/
  - Hélène Pouille/J'ai Lu Ca (French) @helenepouille/
  - Sayo Demura (Japanese) @dem\_sayo/



@mxmoss  
2016

## 9 References

- Conrad, Andrew. 2018. *10 Need-to-Know Project Management Statistics*. Accessed August 28, 2022. <https://blog.capterra.com/10-need-to-know-project-management-statistics-for-2018/>.
- Foer, Joshua. 2011. *Moonwalking with Einstein: The Art and Science of Remembering Everything*. New York: The Penguin Press.
- Naur, Peter. 1985 . "Programming as Theory Building." *Microprocessing and Microprogramming* Pages 253-261.
- Rohde, Mike. 2012. *Sketchnote Handbook, The: the illustrated guide to visual note taking*. Peachpit Press.
- Sibbet, David. 2010. *Visual Meetings*. Wiley.
- Silverman, R. E. 2012 . "Doodling for Dollars." *Wall Street Journal*. Accessed August 28, 2022. <http://online.wsj.com/article/SB10001424052702303978104577362402264009714.html>.
- Trafton, Anne. 2014. "In the blink of an eye." *MIT News*. Accessed August 28, 2022. <https://news.mit.edu/2014/in-the-blink-of-an-eye-0116>.
- Weldearegay, Hailemicheal. 2012. "The role of communication in managing projects." *Umeå School of Business and Economics*. Accessed August 28, 2022. <http://www.diva-portal.org/smash/get/diva2:537872/FULLTEXT02>.
- Wiegers, Karl. 2021. *Sixty Software Development Pearls of Wisdom*. Accessed August 28, 2022. <https://betterprogramming.pub/sixty-software-development-pears-of-wisdom-22754a9a27db> .
- Winchester, Simon. 2011. *NPR*. Accessed August 28, 2022. <https://www.npr.org/2011/05/30/136796448/has-run-run-amok-it-has-645-meanings-so-far>.

# Evolution and Future of Software Testing

**Mesut Durukal**

durukalmesut@gmail.com

## Abstract

We all observe that software (SW) testing continues to grow, proving that it is a living organism. The product development approaches are being updated, which results in updates to testing approaches. As challenges grow day by day, various novel processes and workflows are embraced which will apparently further continue in the future.

Nowadays as the market dynamics are considered, replying to users and customers' needs requires intensive deployment activities. Basically, the most fundamental challenge is verifying and validating such a complex and complicated scope in a limited time with limited resources.

The improvements in Quality Assurance (QA) and testing activities can be classified under 4 basic categories:

- Test Automation
- Agile Testing
- Continuous Testing
- Leveraging Machine Learning (ML) in Testing

This paper explains how software testing has evolved by listing the improvements applied to cope with the difficulties. We will not only see how these improvements help to get rid of problems, but also how they raise new complications. Finally, we will take a look at integrating ML practices into software testing stages.

Various ways to use ML in software testing stages starting with the test definition until the maintenance stage will be discussed and we will see how it helps to reduce the manual effort and improve quality.

## Biography

*Mesut Durukal is a Quality Assurance and test automation enthusiast with experience in Industrial Automation, IoT platforms, SaaS/PaaS and Cloud Services, Defense Industry, Autonomous Mobile Robots, Embedded and Software applications. Along with having a 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.*

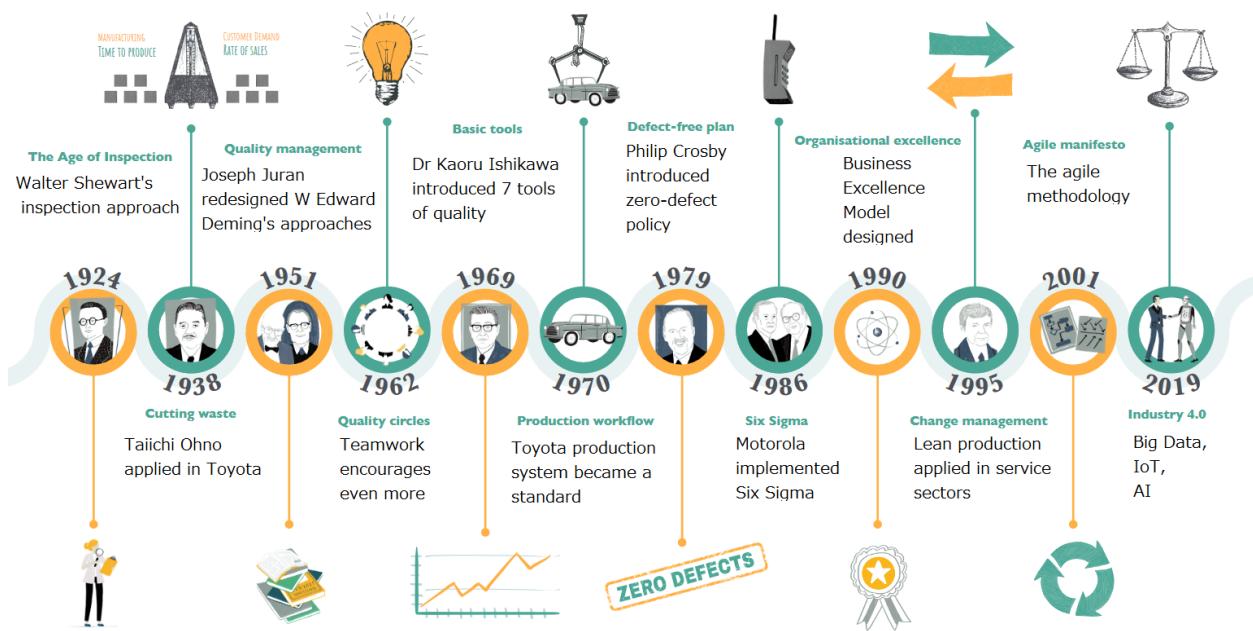
## 1 Introduction

We are living in a digital age. Software is everywhere in our daily lives. We have applications in our mobile phones, smart devices, televisions. These applications are connected to others and communicate with each other frequently. This means, when we are developing applications, firstly we must be aware that they will most probably run on various platforms or browsers. Secondly, it would have lots of interfaces including both user interfaces (UI) and the application protocol interfaces (API) used to send or receive messages to other applications or platforms.

The complex and complicated system under test may give an idea about the scope of the verification and validation activities. Considering the cost of completing tests, it is prominent to reduce both execution durations and the manual effort. As the tests slow down the deployment process, the deliverable will not be on time in the market and not be competent with the rivals.

On top of these difficulties, when the customers or end users' frequent change requests are taken into consideration, testing can be really a tough mission. Designing the best test definitions to properly cover the specifications is a key to reduce escaped bugs. Furthermore, whenever the requirements or specifications are changed, relevant tests should be quickly adapted.

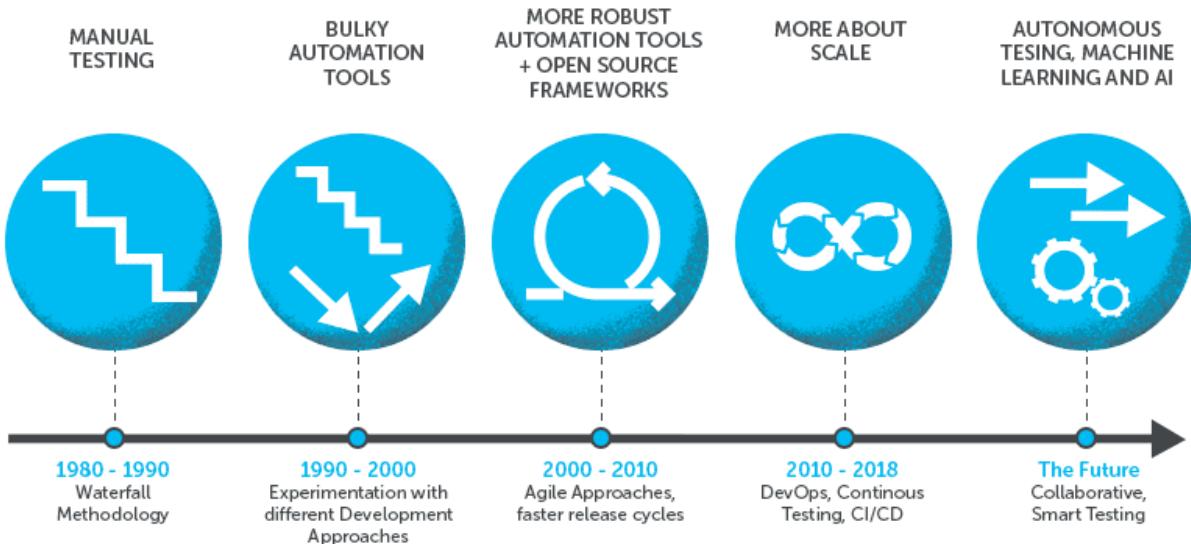
To cope with all these difficulties, various improvements are done in quality assurance processes. A brief summary of the last 100 years milestones was presented in the World Quality Day [1] as depicted below.



Going over this summary of the century, it can be seen that the quality assurance approaches start with basic inspection activities and after various modifications, industry leaders are still looking for further improvements to maximize the efficiency. The ultimate goal is to manage development and changes to the product with minimum waste and redundancy. Technology trends also contribute in shaping the updates.

## 2 Evolution of Software Testing

Focusing more into software testing, we can examine improvements under different titles as shown in the image below [2].



The fundamental milestones that have shaped today's testing activities can be listed as:

- Replacement of Manual Testing with Test Automation
- Testing in Agile instead of Waterfall
- Replacement of Big Releases with Continuous Deployment
- Leveraging Machine Learning (ML) in Testing

Comparing our daily tasks today with the testing activities we were doing a couple of decades back, we can easily see that SW testing significantly evolved and this may give us the idea that it will evolve even more in the future.

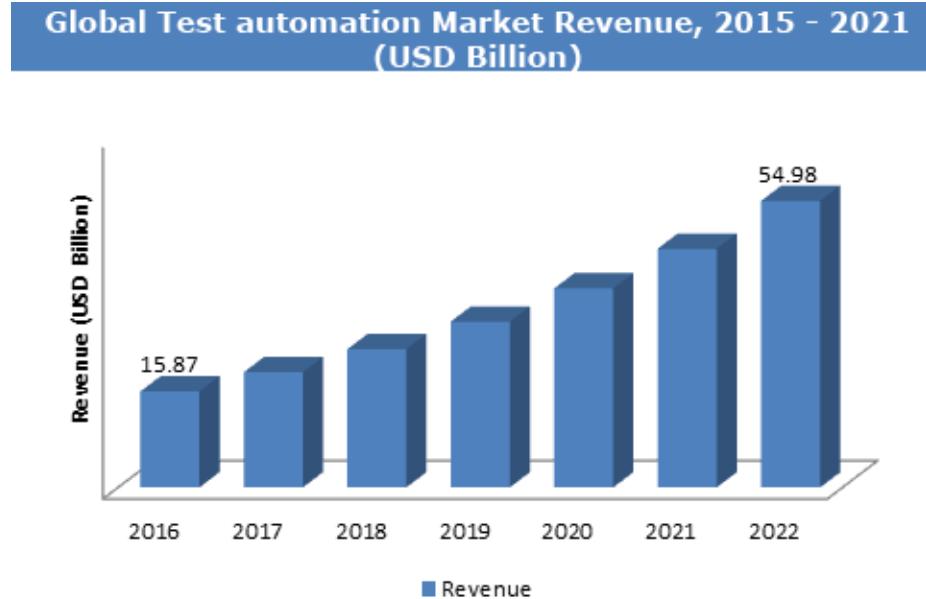
## 2.1 Test Automation

These days, most test engineers are involved in test automation. According to a study [3], automated testing in the world market size is projected to grow by 22% from 2017 to 2020. It has become inevitable to keep pace and sync with development activities. Benefits of test automation are straightforward. As human beings, we do not have infinite energy and processing capacity to execute test cases day and night.

A manual tester can execute a certain number of tests, but after some time or after some number of executions, they are very likely to overlook some issues or weaknesses in the product. In terms of reliability, test automation brings the power of machines into play. They simply do not get tired.

Coming to execution duration, no one would argue that machines outperform humans in calculating complex problems. Executing a single test case by a human may take minutes where it can be executed by automation in seconds. Moreover, we can run a lot of tests in parallel on machines with the help of automation.

Thanks to these obvious advantages, most of the key companies in the industry are embracing automation as part of their software development lifecycle. The image below [3] is supporting the idea that the investment in automation is growing more and more.



On the other hand, doesn't test automation have any difficulties at all? The answer is, predictably, yes. The most fundamental challenge raised by test automation is robustness and reliability.

There are test smells which can make our test automation framework not reliable anymore. Test smells are defined as indicators, observed during testing cycles, for potential problems [4]. Test smells may result in Silent Horror, which means overlooked failing features or false alarms which mean failing tests while the feature was working properly [5].

		<b>Correct Result</b>	
		<b>Pass</b>	<b>Fail</b>
<b>Execution Result</b>	<b>Pass</b>	No Problem	Silent Horror
	<b>Fail</b>	False Alarm	Real Bugs

Getting issues reported by customers although you have already developed and executed test cases for the relevant feature would be annoying. Just like being notified by the test results and figuring out that there is no real bug at all. These kinds of wrong test results will cause missed bugs and extra cost which is spent for the failure analysis.

A few of the most commonly seen test smells are:

- Flaky tests which are fluctuating and failing intermittently
- Fragile tests which are broken after parameter changes
- Eager tests which cover a scope wider than expected and are difficult to maintain
- Dependency issues which make generating test sets difficult
- Low understandability which stems from poorly documented test code

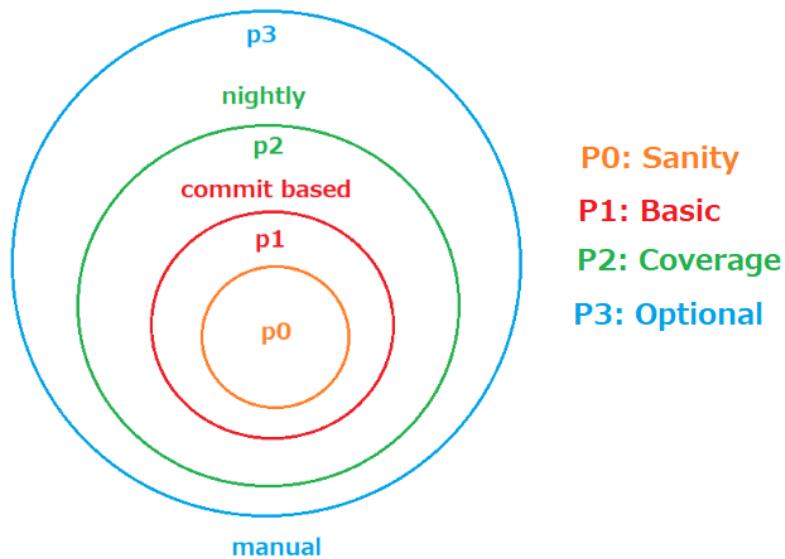
## 2.2 Agile Testing

Like the replacement of manual testing with test automation, agile testing is another milestone in software testing history. The way that we test completely changed after the introduction of agile practices.

In one of my first projects early in my career, I remember having requirement meetings for months. There were hundreds of pages for requirement documentations in different levels. After those meetings I did not participate in any task actively for a long while. And eventually I was supposed to test the product, but it was the last month before the final delivery. There was no chance to provide any feedback obviously.

Nowadays, that is not the case with agile practices. Agile encourages testing early. From the very first sprints, having a potentially shippable product is the target of agile management and for this purpose, testing should be performed along with the development.

But again, new challenges are waiting for us. In order to not fall behind the schedule, testing should be performed efficiently. Infinite testing is not possible as we all know. A good testing strategy with correct priorities and levels should be maintained.



The diagram above shows how we applied test priorities in various pipelines. We have taken visibility, criticality (impact) and complexity dimensions into consideration in order to decide test priorities.

Another important aspect of agile is strong communication, since it is required to maintain the processes in harmony. If the testing teams are not aware of what is being developed, the activities cannot be started in the meantime. Clarification of the features is a big target to be accomplished since we do not see hundreds of requirement documents anymore.

To handle this issue, working together with the product managers, we have added tags to all the tickets on our issue tracking system and tried to improve the traceability of them. Before this study, it was difficult to figure out the features of the product. We had to keep manual lists to track features. After adding categorical labels to features, it was possible to query all the features related to specific use cases automatically. In this way, we had a chance to build traceability and measure coverage.

## 2.3 Continuous Testing

Unlike the waterfall testing, test cases are not executed only once, but a lot of times a day in our recent approaches. Since lots of commits are being merged continuously, we must ensure that none of them are breaking the production. Considering this, the reliability of the tests is even more important.

Tests should not block the merges unnecessarily. If there are false alarms, the execution should be repeated which causes extra time and resources.

After several executions, various metrics can be collected to get insights about both tests and the product. Metrics related to execution durations, resource consumption, response times and others can base our monitoring activities to improve our activities.

By interpreting the results, various action items can be generated. For instance, in case a created object spike is detected in the environments, the cleanup tasks can be checked since it is likely that they have failed due to created but not deleted objects which were causing a load in the environments.



After generating graphs and dashboards, detecting spikes and anomalies are straightforward and alerts can be set up to notify the owners of the processes.

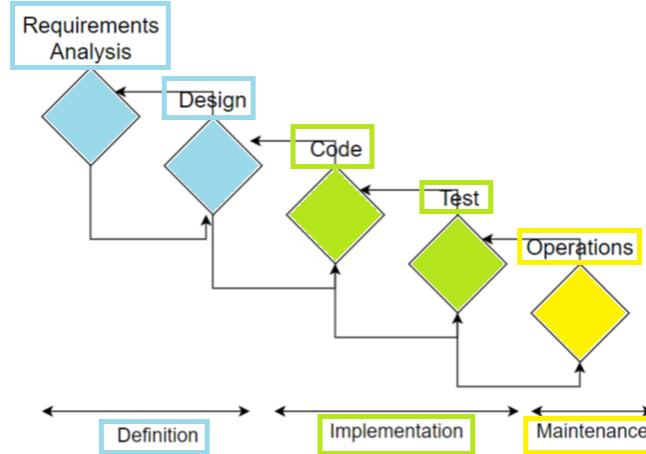
### 3 Machine Learning in Software Testing

Test automation, agile principles and continuous testing are all improvements embraced in software testing bringing great advantages. But they do not rule out all the difficulties and there is still room for improvement to be achieved in terms of testing efficiency.

There are still tricky obstacles to get rid of like quickly adapting tests after feature updates, healing broken tests and maintaining test code with a minimum manual effort.

To further improve all these processes, Machine Learning is a strong candidate to help testers. In cases where the strengths of machines are utilized, they can support testing activities.

ML algorithms can be integrated into testing in multiple stages. Software testing life cycle looks like:



In all these stages, ML can be used to assist SW testing.

### 3.1 Definition Stage

Since the general idea of ML is to understand the system dynamics by observing inputs and relevant outputs, the same applies in test stages. To define test cases, various user interactions are observed and the relevant behavior after each is analyzed. After all, a model is generated to predict the future outcomes. In this way, the expected behavior after similar interactions is predicted and the test coverage can be extended by adding more scenarios.

Going over examples, for an API under test, the system responses can be seen to understand the general approach to handle user requests. For instance, if it is observed that the requests without token or any other authentication related header field are responded to with 401/403 coded responses, the model may already realize that the API is an authenticated one. It may generate some more test cases by adding random tokens to see the actual responses. Or after observing the body of the responses returning from GET queries, relevant PUT or PATCH queries can be generated to test various endpoints.

Another way to generate test cases is to walk through the code under test. All the methods can be listed with the parameters supposed to be attached to the call and then the relevant unit tests can be generated in a similar approach with the past executions.

### 3.2 Implementation Stage

We can also support test implementation with ML. After the problem is defined with inputs and outputs, the needed operations are figured out. DeepCoder [6] follows the same approach. Here is an example of input and output in a scenario, in which negative numbers are filtered and listed in a reverse order after multiplied with 4:

For the input:

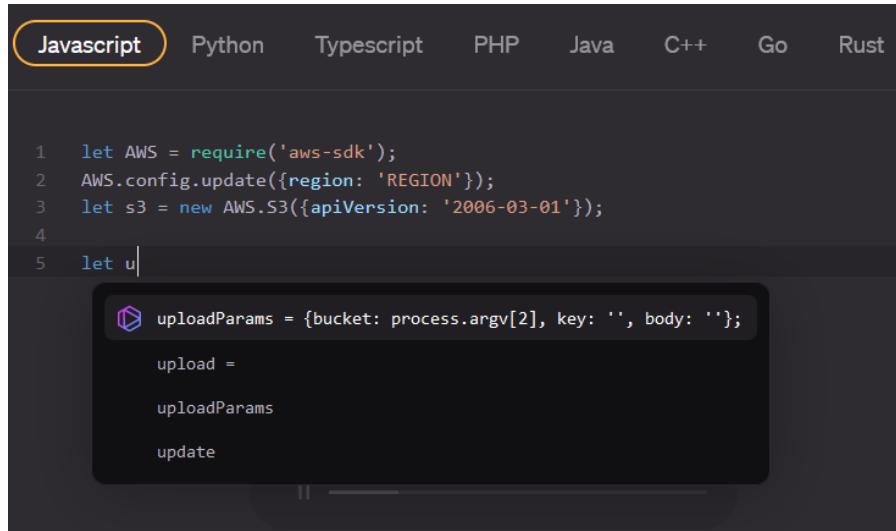
`[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]`

Expected output is:

`[-12, -20, -32, -36, -68]`

After figuring out that ‘filter’, ‘sort’ and ‘multiply’ operations are needed to build this algorithm, finding or generating the relevant code is not difficult.

Beyond generating code from scratch, another way to improve the speed and efficiency of writing code is to suggest completions. After the most frequently used patterns are learnt, ML proposes the subsequent code during implementation. Tabnine [7] is an application, which facilitates test implementation in this way.



One more way to support test code generation is replacing traditional UI automation with visual recognition approach. Locating elements on the page by selectors is prone to errors since they are frequently updated. But if somehow the buttons or icons are visually located, even if the selector like the ID or the path changes, the test still passes. Lots of similar images are provided for the training of the model and eventually it is recognized in the system under test. test.ai [8] is using ML to find the element:

```
async function find (driver, logger, label, /* multiple */) {
  const curSetting = (await driver.getSettings())
    .elementResponseAttributes;
  const needToChangeSetting = !curSetting
    || curSetting.indexOf("react") === -1;
  const confidence = getConfidenceThreshold(driver, logger);

  try{
    const els = await getAllElements(driver, logger);
    const screenshotImg = await getScreenshot(driver, logger);
    const elsAndImages = await getElementImages(els, screenshotImg, logger)
    return await getMatchingElements(elsAndImages, label, confidence, logger);
  }
  finally{
    // ....
  }
}
```

Finally, during the executions, ML algorithms can run to collect information and build a model to detect anomalies or generate expected results. For instance, after observing execution durations, if any of them takes more than expected, the model can notify users to check the execution to understand what went wrong. In this way, manual analysis effort can be reduced. Several applications can be seen in [9].

### 3.3 Maintenance Stage

The last stage in the software testing life cycle is the maintenance stage. Code review, healing broken tests and action items regarding failing tests are the activities performed in terms of maintenance.

Starting with the code review, if the best practices and the antipatterns are taught to the model, the code review can be done by machines. This reduces waste of time significantly since fixing comments coming from peer reviews requires a few round trips. DeepCode [10] is a tool for semantic code analysis. Additionally, there are various self-healing tools and platforms which analyze the broken tests and try to suggest healing actions. Finally, management of bugs can be supported by ML algorithms. Reported issues can be either classified or clustered for various purposes.

In my project, I have performed bug triage with ML assistance. Triage is very important since it directly affects the planning of the tasks. Adapting ML to bug triage would help to perform both more consistently and faster.

First, I prepared data by exporting reported bugs from the issue tracking system:

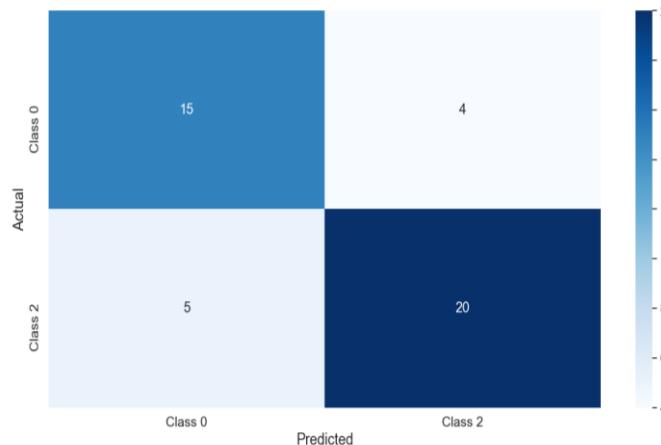
- 889 bugs exported from Jira
- 3 severity levels were defined in our project

So, the purpose for generating a model from those previous examples was to predict the severity level of the future bugs. When a bug is reported, it is supposed to be classified into one of 3 severity levels.

Second step after data preparation was feature extraction. Feature extraction is the conversion of data into numerical vectors. In my case, since the data is text values (bugs written in the English language), they had to be represented by binary vectors to be recognized by computers. I applied text related ML practices like Bag of Words feature extraction and TF-IDF normalization.

The next step was generating and evaluating the model by the learning algorithm. I tried several methods like Support Vector Machines, Decision Trees, k-nearest neighbors and some ensemble learning algorithms.

One more improvement I have done was getting rid of the bias. Most of the samples I used for training were labeled with Class 2. After I merged Class 0 and 1 together, data was more balanced. This made sense to me because in my project deciding whether a bug is Class 2 or not was important since Class 2 bugs were release blockers in our project. The confusion matrix on the test data looks like:



Evaluating the predicted severity levels by the ML model with the labels evaluated by humans, 82 % accuracy rate was achieved.

## 4 Conclusion

Software testing started with the software development because we want to see if it is working as expected just after we develop a product. Since the very early quality assurance applications, it has grown a lot and we can see various improvements like replacement of manual testing with test automation and waterfall approaches with agile. We also see that continuous testing is a part of our daily life nowadays. All these modifications help us to cope with challenges but do not resolve everything.

Beyond other improvements, integrating ML into software testing stages reduces waste of time and increases the stability and efficiency. From defining tests to classification of bugs, we can utilize ML in several stages. Even if all the manual activities cannot be replaced by machines, at least the effort and resources can be minimized to perform the verification and validation activities. We can foresee that ML will appear more in our software testing activities in the future.

## 5 References

- [1] SimpleQue. 2019. "100 years of Quality", <https://www.quality.org/file/16376/download> (accessed June 12, 2022)
- [2] Testim. 2018. "How AI is Changing the Future of Software Testing", <https://www.testim.io/blog/ai-transforming-software-testing/> (accessed June 12, 2022)
- [3] Zion Market Research, 2021. "Test Automation Market - Global Industry Analysis", <https://www.zionmarketresearch.com/news/test-automation-market> (accessed June 12, 2022)
- [4] G. Bavota, et al. 2015. "Are test smells really harmful? An empirical study," Empirical Software Engineering, 20: pp. 1052-1094, doi: 10.1007/s10664-014-9313-0.
- [5] M. Durukal, "How to Ensure Testing Robustness in Microservice Architectures and Cope with Test Smells." International Journal of Scientific Research in Computer Science, Engineering and Information Technology. pp. 167-175, 2019, doi: 10.32628/CSEIT195425.
- [6] M. Balog, A. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "DeepCoder: Learning to Write Programs," Proceedings of ICLR'17, March 2017
- [7] tabnine, "<https://www.tabnine.com/>" (accessed June 12, 2022)
- [8] test.ai, "<https://github.com/testdotai/classifier-builder>" (accessed June 12, 2022)
- [9] M. Durukal, "Practical Applications of Artificial Intelligence in Software Testing." International Journal of Scientific Research in Computer Science, Engineering and Information Technology. pp. 198-205, 2019, doi: 10.32628/CSEIT195434.
- [10] DeepCode, "<https://www.deepcode.ai/>" (accessed June 12, 2022)

# 7 things I learned to build highly effective onsite/offshore teams.

**Venkat Edagottu**

edagottu@gmail.com

## Abstract

If you are a supervisor or a team leader, you know how difficult it is to manage distributed teams with constant pressure to perform and deliver quality products/services on time. This is especially true for offshore teams in different time zones, regions, cultures, and working styles. If you do not put in the effort to develop effective strategies to manage these teams, it may eventually lead to low team morale, disgruntled employees, and, most importantly, poor performance. This is the reason why companies are investing millions of dollars in training, organizational, and team development.

From my 20+ years of experience working in tech and managing teams, I realized that as leaders, we not only need to consider the structure of the team but also the alignment and purpose of the team's work. This is to ensure that we have clarity on the value the team creates. In my opinion, a high-performing team flourishes in an environment where they know exactly how they contribute to the organization's purpose.

In this talk, I will share key strategies to improve the team's performance in delivering high-value products/services to customers irrespective of their cultural and geographical differences. I will share proven techniques that help build high-performing distributed teams. The tools and approaches I discuss will give the attendees a strategic and transparent way to manage team processes and communication. These concepts will reinforce how to implement the principles in a real-world environment working with onsite/offshore teams to solve the pressing challenges.

## Biography

*Venkat Edagottu is a Senior Manager at Hitachi Vantara. He has 20+ years of working in the IT industry across various domains that include - Banking, Healthcare, Insurance, Automobile, and Finance with globally distributed teams. He is passionate about aligning the team's goals with the company vision. This has helped him to have continued success within consulting and leadership areas focused on collaboration, operational excellence & organizational development. He loves inspiring team members and creating diverse and inclusive work environments. In his spare time, he is an active blogger, conference speaker, and passionate learner. He can be reached at edagottu@gmail.com, and you connect with him on LinkedIn - <https://www.linkedin.com/in/venkat-edagottu/>*

## Introduction

Suppose you are a supervisor or a team leader. You know how difficult it is to manage distributed teams with constant pressure to perform and deliver quality products and services on time. This is especially true for offshore teams in different time zones, regions, cultures, and working styles. If you do not put in the effort to develop effective strategies to manage these teams, it may eventually lead to low team morale, disgruntled employees, and, most importantly, poor performance. This is the reason why companies are investing millions of dollars in training, organizational, and team development.

From my 20+ years of experience working in tech and managing teams, I realized that as leaders, we need to consider the structure of the team and the alignment and purpose of the work the team does. This is to ensure that we have clarity on the value the team creates. In my opinion, a high-performing team flourishes in an environment where they know exactly how they contribute to the organization's purpose.

Currently, 16% of companies (Steward, 2020) are 100% remote, and 77% of remote workers say they're more productive when working from home. A Gartner survey of company leaders found that 82% plan to allow employees to work at least part of the time after the pandemic remotely, and 47% will enable employees to work from home full-time (Gartner, 2020). This being the case, building highly motivated teams across the globe becomes a huge challenge but is not impossible.

Here are seven strategies to build high-performing teams

### 1 Time Zones

One of the significant challenges of working in distributed teams is streamlining the collaboration process. Time zones become crucial in a distributed team environment to achieve high productivity.

Alternate between different meeting times when scheduling meetings across time zones. This lets your team know that everyone gets a chance to have a discussion that is convenient to their respective work regions. Also, make sure you plan.

Schedule critical meetings at an appropriate time suitable for onsite and offshore teams (Asia Pacific region, Europe, Middle East, and Africa). Working with remote teams is slightly different from co-located teams.

For example, if someone asks you to send a status report by 9 pm, mention whether it is IST, EST, or UK time. Be proactive in scheduling meetings. If required, you may need two-time slots for the sessions since not one-time best suits, everyone. Working with people around the world and traveling, scheduling becomes more complicated. Refer to the article "Six tips for managing meetings in multiple time zones with Google Calendar" for tips for working with multiple time zones (Wolber, 2015).

For example, to create a multi-time zone event. When you fly from Kansas City to Detroit, your flight starts in a one-time zone and ends in another. You can enter the flight departure and arrival times correctly for each time zone from Google Calendar in your browser. Create a new Google Calendar event, then select the "Time zone" link to the right of the event's end time. Check the "Use separate start and end time zones" box, then choose the time zones. Finally, enter the start and end times.

### 2 Communication

Lack of face-to-face interaction in the distributed teams causing communication problems while gathering requirements for one of my projects, I explain below the actual problem and how certain steps taken improved the common understanding to resolve this issue.

Use Messaging tools like Zoom, Google Meet, and Microsoft Teams as virtual workspaces to connect all team members. Set clear goals, deadlines, and expectations to work on tasks accordingly. Empower people to make decisions, and you check in periodically. Have regular 1:1 meeting to understand the problems/issues. Always welcome feedback and questions and listen carefully before you respond.

Understanding communication, cooperation, and trust are key. Boost team morale by giving recognition periodically. It will help if you recognize good work immediately. Having all-hands meetings on a quarterly basis is also very helpful. Poor communication also can cause friction in the team. For example, asking a team to complete a task in one week without clear expectations can cause frustration to team members due to a lack of clarity in communication.

Also, intercultural communication is essential, and you need to keep in mind the working styles in different cultures. Due to the hierarchical structure, some cultures may be hesitant to ask questions. As a leader, it is your responsibility to recognize this and encourage questions to ensure they understand it.

For example, when we say "Let's pull the plug on that project," a person who comes from a different cultural background may not understand the meaning of this phrase, and this could be open to different interpretations.

Here is one example how communication challenges impacted in delivery one of my projects. I was working on a critical financial solution project where my team provided end-to-end Testing services. There were issues regarding inaccurate specifications gathered by my team mistakenly. Due to the imprecise requirements, the test plan and test cases designed around the functionality had many gaps and were not usable. The client asked us to eliminate all the test cases and develop a proper test strategy to validate this more effectively within the agreed-upon release deadline. This put much pressure on our team, and there was no option but to make the requested changes.

To fill this communication gap with the client, I worked closely with various stakeholders and collected the requirements again. As a result, we could write a whole new test plan based on the new specifications. We were able to do manual and automated testing on the product and still meet the agreed-upon timelines.

### 3 Small work teams

Having small teams is good for managing and easily tracking progress. Many of the largest technology companies created their first successful products with teams of fewer than 10 people. Empower your team by giving them responsibilities and encouraging them. Break down goals and make sure all the team members understand them. Having a point person for each team is very helpful to know the status for tracking purposes easily. Also, diversity of thought and culture brings different perspectives which give a different dimension to the team composition.

Build cross-functional teams to understand dependencies from each other team. Form separate small groups for each work stream to discuss and chat about the deliverables related to the work stream. Breaking down goals into smaller ones and communicating them to different work teams is a good approach to achieve more progress. Also, remember that you do not need to have everyone in a meeting. Invite only those who are required to make decisions and have constructive discussions about the topic at hand. Communication channels will increase depending on the team size.

For example, in our organization, we were 100+ testing resources working on various clients. The problem was it was difficult for our leadership team to know which testers were working on what projects and also whether they were being 100% utilized. Because if they are not 100% utilized, we could assign them other high-priority tasks sitting in the backlog. I volunteered to solve this problem for our leadership team as it was leading to much confusion.

So, the first thing I did was, create a Testing Center of Excellence (TCOE). An independent body in the company that will oversee all testers. The next thing I did was, create four key areas of expertise to

classify all the testers under some logical category. Applications (ERP, CRM), Technology (Development, SOA/Integration), Performance Testing, and Test Automation. The third thing I did was schedule a meeting with Test Leads of all the projects to let them know what process I was going to follow to give everyone clear visibility of testers' work at any point in time. I made sure I answered all their questions.

Then I created a master Excel sheet for the Test leads where each one of them will update the resource name of their team and some bullet points of what tasks they were doing. Once I collected all this data, I classified the resources under the four key areas. I created an internal pool of consultants who can be assigned to various projects in the future based on their expertise. Finally, I asked the test leads to update the excel sheet at the end of each week and had a macro that scanned through all this data and gave the necessary information related to resource utilization.

This effort took me 2 months, but as a result, the leadership team had a better snapshot of resource utilization. The testing team started collaborating better as they all belonged to one group (the TCOE), and the bench resource utilization increased from 0 to 30%.

## 4 Seamless alignment

Misalignment happens when the team lead believes he has effectively communicated his message to his team but his expectations don't actually make it through to most of the team members.

Encourage teams to work towards common goals and objectives of the Project/Program. Make sure all the project artifacts are stored in a shared area to access by teams from onsite and onshore. You need to establish the same process for Onsite/Offshore teams while using the same set of tools, frameworks and resources. Make sure the onsite and offshore work goes seamless and place all relevant documentation in a shared folder to access for all Team members. Teams working in seamless environments do struggle with how much onsite work is completed and what tasks need to take up in order to continue the work towards completion.

Alignment happens on three levels:

1. High-level grooming with the whole team: Go through the requirements, make sure they have a common understanding, what the expectation from a testability standpoint and development standpoint and what the customer needs, what test data is needed...these all are things the whole team needs to align.
2. When the task has to complete: What are the expectations, everything can be set during the high-level planning meeting. During Product Refinement meeting, before developing a story, the business person, developer, and tester would meet together, go through the requirement and figure out whether the mockups are ready, requirements are clear, need test data, and whether everyone has the necessary resources to work on it.
3. Once the requirements are completed: show the demo to the business person to make sure the requirements were implemented according to what is needed.

## 5 Cross-functional Teams

A cross-functional team consists of people who have different skill sets or expertise, bringing all of them together to achieve a common goal. Backlog Refinement, Sprint Planning, Sprint Review, and Retrospective meetings for each sprint are very helpful to know what works well and what needs to be improved. Encourage the team to participate in all the team meetings and encourage them to speak up about their issues/concerns to get help from the team. Also, discuss the dependencies between various tasks that have to be accomplished.

For example, one of the biggest challenges I faced during big project teams was the struggle to work across silos. Having cross functional meetings in between multiple scrum teams where each team

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

PNSQC.ORG

Page 4

dependent on other team to integrate the product, having meetings on regular basis resolved most of the interdependencies people have while delivering features.

Here is one more example from my past experience. I once worked on a Data Strategy project. The problem was the client was struggling to estimate the size of the project in terms of various cross-functional teams involved. (Data, Governance, QA, Dev, BA, UX). Also, he could not figure out the amount of testing effort it would require. So, I as a QA manager had to figure out a solution.

The first thing I did was, I scheduled a meeting with all the cross functional team involved (Data, Governance, QA, Dev, BA, UX) to understand the work and create a road map based on high-level timelines available. Next, I created a high-level estimation efforts to align with the timelines. I then sent this to various cross functional team leads to get their feedback on it before sending it to the key stakeholders.

## 6 Accountability

Building a culture of accountability within the team takes discipline and practice. Though quality is everyone's responsibility, each team member is responsible for the task/activity they are performing and the outcome. Make individuals accountable for the deliverables and the final outcome should be responsible of the team. Encouraging new ideas and solutions increase team motivation and feel that their voice and ideas are valuable. Make sure team responsibilities are clearly spelled out. Use tools to manage task progress and set expectations that are always clear.

One great way to empower the team is to encourage peer review for each deliverable and provide feedback (Folkman, 2017). Accountability from a leadership standpoint is setting clear expectations, goals, and deadlines for the team. Divide the big task into granular ones and assign to the person, help them and make them accountable for it and coach them as and when needed behind the scenes.

For example, I was working on an SAP project for one of my Fortune 500 clients. We were tasked with doing data validations on legacy applications that were sending large volumes of direct and indirect procurement data to different interfaces. The problem was the data was so large and it was close to impossible to validate all the data manually within the given release deadlines. Also, no one had a clue on what teams had to be involved in this process, when the data needs to be sent to these interfaces and what data needs to be exactly sent. This led to our UAT cycles always running late beyond the fixed deadlines we had which led to wastage of time, effort and cost. So, I had to solve this problem as the manager of the team.

Data validation is the biggest challenge in an SAP procurement team, where legacy applications are sending large volumes of data each day with a lot of permutations and combinations. No clear direction of what teams are involved and when they are supposed to send the data to through interfaces. The SIT/UAT cycles are always running late due to the complexity of the data load and validation. It was taking a very long time to validate all the data loaded into the system manually. It requires a lot of resources and time to complete the testing cycles.

So first of all, I wanted to set clear expectations on how the SAP system works, what data needs to be sent, at what time to what interfaces. So, I called in my Tech Lead and SAP consultant to take collective accountability to solve this problem. We sat together for half a day and documented everything related to the system in an excel sheet. We put this in a shared folder for the entire team to see at any instant of time to prevent unnecessary confusion.

Next, we had to reduce the manual effort spent on validating all this data. So, I asked my automation engineers to create different automation scripts which ran automatically that validated different data combinations as soon as the data loading process started.

I wanted to have daily updates on this process so I worked with my automation engineers to configure some of the automated processes to run on a nightly basis and by the time we come to office the next morning, all the summary reports are delivered to our Inbox for final audit and confirmation.

This whole process took three months with three of my tech leads and two automation engineers to establish the process and develop the scripts and keep them ready for all upcoming SIT/UAT Testing cycles for various rollouts planned in the Program for next few years.

We saw some great results through this effort. First of all, the team was able to validate test cases with precise data inputs during UAT. We saved 25-30 % of testing time through this effort. This meant that the planned four-week testing cycle was able to be completed within 2.5 to three weeks. We got multiple client appreciation emails because of this.

Also, the reports generated throughout our automated process helped stakeholders make informed decisions to approve the testing cycle and move on to the acceptance and delivery phase. Now we were making strategic decisions based on data instead of assumptions.

Best of all, we reduced eight hours of data validation effort for each interface to 30 mins to one hour. This helped business teams a lot and they could focus their valuable time on other day to day business activities instead of just Testing and validating all the data in the system.

## 7 Celebrating success

Celebrating small wins at work boosts team morale. You need to send frequent "Thank you" notes and provide monetary gifts to ensure individual contributions are visible. You can also nominate people for internal awards or reward them with more responsibilities.

Remember, always celebrate small wins with the teams and respect different team cultures and background. Acknowledging good behavior is very important. I believe in one mantra we hire good people and provide an environment for them to be great.

As part of celebrating success and recognizing individuals to reward behaviors and performance that support company business goals, we have quarterly awards in different categories in Shining Star, Techno Wiz, Client Centricity, and Innovation and announce them in the all-hands meeting organized quarterly. These employee recognition awards encourage loyalty and engagement within the team, and they feel proud they are working for a product/service which solves customer problems and enhances their experience.

## Conclusion

In today's fast-paced world, focusing on distributed teams in delivering high-value products/services to customers irrespective of their cultural and geographical differences is the key. The above strategies helped me to strengthen the bonding between teams and transparency which translated into great collaboration and a healthy environment created. It helped to deliver the projects on time or ahead of time with great accuracy. The proven techniques discussed above helped build high-performing distributed teams.

By implementing all of the above strategies will help the team to improve overall performance and productivity, which in turn increases the team's velocity. You need to educate the team on these strategies and clarify if they have any questions and take suggestions if any other good ideas/practices they recommended.

## References

- Folkman. (2017). Retrieved from <https://www.forbes.com/sites/joefolkman/2017/03/02/the-6-key-secrets-to-increasing-empowerment-in-your-team/?sh=c98bb6377a65>
- Gartner. (2020). Retrieved from <https://www.gartner.com/en/newsroom/press-releases/2020-07-14-gartner-survey-reveals-82-percent-of-company-leaders-plan-to-allow-employees-to-work-remotely-some-of-the-time>
- Steward. (2020). Retrieved from <https://findstack.com/remote-work-statistics/#:~:text=16%25%20of%20companies%20in%20the,productive%20when%20working%20from%20home.>
- Wolber. (2015). Retrieved from <https://www.techrepublic.com/article/six-tips-for-managing-meetings-in-multiple-time-zones-with-google-calendar/>

# Formal Technical Review Process Enhancement

**Eu Felix, Peh Wei Wooi, Ooi Mei Chen, Liu Keping**

[felix.eu@intel.com](mailto:felix.eu@intel.com), [wei.wooi.peh@intel.com](mailto:wei.wooi.peh@intel.com), [mei.chen.ooi@intel.com](mailto:mei.chen.ooi@intel.com),  
[keping.liu@intel.com](mailto:keping.liu@intel.com)

## Abstract

The Formal Technical Review (FTR) is a software quality control activity with the purpose of ensuring the software fulfills specified requirements and predefined standards. In the past, many organizations have deprioritized the importance of reviewing feedback because they did not expect it to show a significant value added to software quality or bring any tremendous benefits to the organization. Despite the proven data that reviews significantly increase software release quality [Ref 1], they may be viewed as a redundant part of the software release process.

Based on years of experience in the quality assurance industry, an efficient FTR process has consistently proven its ability to improve the software quality in various aspects such as business requirements, architecture design documents, technical requirements, software release documents, test cases procedures, etc. The quantitative data in terms of cost-saving metrics and defects will be further explained in section 5. In a nutshell, the formal technical review is a key factor in meeting the quality release criteria and reducing the unnecessary cost (to fix the mistakes) incurred in the project, hence an effective FTR process and tool should be implemented to ensure the success of the project which leads to the success of the organization.

## Biography

*Eu Felix is a Software Quality Engineer at Intel Corporation based in Penang, Malaysia. He has held the Lean Six Sigma Green Badge since 2019, is a certified Software Quality Engineer (CSQE) from ASQ and holds a Degree in Computer Science from the University of Bolton in the UK.*

*Peh Wei Wooi is a Platform Validation Lead at Intel Corporation based in Penang, Malaysia. He is certified as the ISTQB tester and holds a Degree in Information Science from UKM, Malaysia.*

*Ooi Mei Chen is a Software Quality Engineer at Intel Corporation based in Penang, Malaysia. She holds a Degree in Computer Science from Universiti Tunku Abdul Rahman.*

*Liu Keping is a Technical Leader in Software Quality Assurance at intel Corporation based in Shanghai, China. She is a certified CMMI assessor, ISO internal assessor, ASPICE internal assessor, and CSQE, and gained 6 Sigma Orange Belt and CPMP certification in 2009. She holds a master's degree in Computer Science and Technology from Central South University in China.*

Copyright Eu Felix, Peh Wei Wooi, Ooi Mei Chen, Liu Keping 2022

# 1 Introduction

In this introduction, we discuss what Formal Technical Review (FTR) is and the importance of having it in the entire software development lifecycle by defining the objectives. We briefly go through the high-level overview of the different types of reviews currently available and move on to provide detail of the phases available in the FTR process.

In Section 2, we discuss the roles and responsibilities of each role that will be involved in the formal technical review. It is important to understand how each role plays its part and contributes to the success of the review. Section 3 describes the overall methodology for conducting an effective formal technical review, for example, setting up the meeting, record keeping, review reporting, review guidelines, and the recommended formal technical review tools which summarize the “best-known method” for the formal technical review process.

Section 4 will recommend several enhancements which we believe are useful based on past user experience, and how they can benefit organizations upon implementation. The two process enhancements are FTR Guidelines and FTR Tools. FTR Guidelines focus on how to reduce waste by implementing lean practices and avoiding process and roles duplication, while the introduction of the FTR tool brings more value by having a centralized repository for all the review comments, data consolidation, email notification, versioning & tagging, FTR dashboard and so on that enlightens exceptional and outstanding user experience. These recommendations are very important for an organization as they not only help to improve the FTR review efficiency (via processes) but also improve the velocity with the help of the FTR tool. In section 5, we provide some examples of the review metrics from various aspects which include user experience, establishing organizational benchmarking, and defining the quality data. Finally, in Section 6, we summarize and provide directions for future work and areas to research.

## 1.1 What is the Formal Technical Review (FTR)?

The Formal Technical Review (FTR) is a software quality control activity to ensure that the software meets the specified requirements and predefined standards. In the past, many organizations have deprioritized the importance of review feedback because they didn't envision it to show a significant value added to the software quality or bring any tremendous benefits to the organization. Despite the proven data that reviews level up software release quality, they may be viewed as a redundant part of the software release process.

Based on years of experience in the quality assurance industry, an efficient FTR process has consistently proven its ability to improve software quality in various aspects such as business requirements, architecture design documents, technical requirements, software release documents, test case procedures, etc. For instance, during the earlier stage of business requirement review, the customer and respective stakeholders will be engaged in the review and all the misconceptions or misalignment can be fixed earlier before flowing into the next stage which is technical requirement creation. In the nutshell, the formal technical review is a key factor in gauging the quality release criteria and reducing the unnecessary cost (to fix the mistakes) incurred in the project, hence an effective FTR process and tool should be implemented to ensure the success of the project which leads to the success of the organization.

## 1.2 Objectives

The high-level objectives for carrying out the formal technical review process are:

- To discover defects in function, logic, or implementation of the software.
- To ensure the software implementation meets the requirements and predefined standards.
- To implement and archive software in a uniform manner.
- To make projects more traceable such as the feedback and verification of changes are stored in a consolidated repository.

### 1.3 Types of Reviews

There are two types of reviews: Formal reviews and Informal reviews. Both types of reviews can be useful depending on the available resources.

Formal Reviews	Informal Reviews
Larger Group	Smaller Group
The review is scheduled beforehand. Sufficient time is given to the team members for preparation	The meeting is commonly scheduled at team members' convenience
Follow the process with specific formal agenda	Conducted per the need of the team with informal agenda
The review consists of a professional team that identifies and corrects errors in the software model	Equivalent to a buddy check. The main purpose is to detect defects, generate ideas/solutions, or quickly resolve minor problems
The review should not exceed two hours	This review is in between 1-2 hours

Table-1 Types of Reviews

#### 1.3.1 Different Types of Formal Reviews

The table below explains three different types of formal technical reviews which can be applied throughout the Software Development Lifecycle. Table-2 shows the overview of the different types of formal reviews currently available and their respective goals.

Type	Definition	Goals
Walkthrough	Led by the author to understand and collect comments. The content of the document is explained step by step by the author, to accomplish agreement on changes or to gather information. The participants are chosen from different departments and backgrounds. If the audience represents a broad section of skills and disciplines, it can give a guarantee that no crucial defects are "missed". It is helpful for higher-level documents, such as requirements specifications and architectural documents.	<ul style="list-style-type: none"> <li>• A preliminary review of the work product. Either first draft or clean code compiles.</li> <li>• To present the document to stakeholders both within and outside the software discipline, to evaluate conformance to a standard, and gather information.</li> <li>• To describe and gauge the contents of the document.</li> <li>• To initiate a basic understanding of the document.</li> <li>• To inspect and discuss the validity of suggested solutions and feasible alternatives.</li> </ul>
Technical Review	The review focuses on the technical content of a document and is led by a trained moderator or technical expert. The review is intended to identify discrepancies in specifications and standards and focus on defect	<ul style="list-style-type: none"> <li>• Determine the suitability of the work product for its intended use.</li> <li>• To assess the benefit of technical concepts and alternatives in the product and project environment.</li> </ul>

	identification based on referenced documents. Participants can be technically qualified personnel such as architects, chief designers, and key users. This is openly performed as a peer review without management participation.	<ul style="list-style-type: none"> <li>• To set uniformity in the use and representation of technical concepts.</li> <li>• To gain consensus and build confidence in the product.</li> <li>• To ensure at an early stage, that technical concept are used precisely.</li> <li>• To notify participants of the technical content of the document.</li> </ul>
Inspection	The most formal review type and led by a trained moderator to identify issues in a work product. The document under inspection is prepared and checked thoroughly by the reviewers before meeting by using the rules and checklists. In the inspection meeting, the defects found are captured.	<ul style="list-style-type: none"> <li>• Like Review but beyond that search for anomalies</li> <li>• To help the author to enhance the quality of the document by detecting potential defects.</li> <li>• To remove defects efficiently and rapidly.</li> <li>• To improve product quality by producing documents with a higher level of quality.</li> <li>• To create a common understanding by exchanging information among the inspection participants.</li> </ul>

Table-2 Different Types of Formal Reviews

#### 1.4 Different Phases of Formal Technical Review

An FTR process generally breaks down and is implemented into different phases to ensure process integrity. Figure-1 below shows the FTR principally takes place in the well-thought-through approach that includes 6 different steps that are essential to assure and inspect software quality, efficiency, and effectiveness.

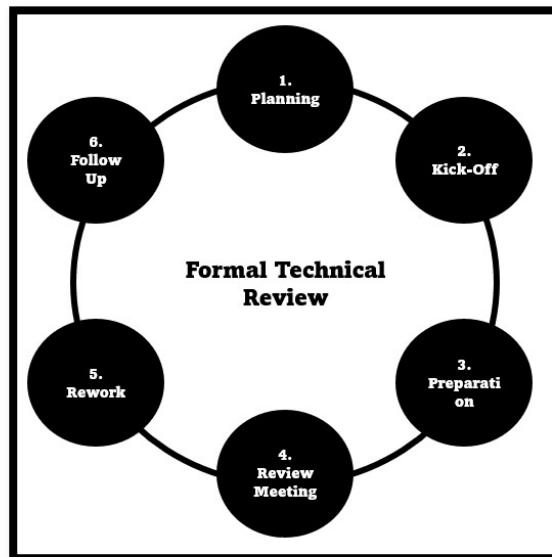


Figure-1 Formal Technical Review Phases

Phase	Descriptions
-------	--------------

Planning	Commonly starts with a 'request for review' by the author to a moderator. The moderator also performs entry checks and exit criteria.
Kick-Off	The main purpose is to get all participants in sync and understand the intention of this kick-off. The entry result and exit criteria are talked through as well in this meeting. This phase brings a greater level of understanding of the team about relationships between documents under review and other documents. During the meeting, all documents under view, source documents, etc. will be distributed.
Preparation	The participants (reviewers) will work solitary on the document under review. They mainly identify or inspect for any defect/error, and offer their opinion, which is later logged or captured. Spelling or cosmetic errors are also tracked on the document under review but not mentioned during the meeting.
Review Meeting	This phase commonly involves three different activities: logging, discussion, and decision.
Rework	The author primarily revises the document that is under review based on the defects that are detected and improvements being proposed in the review meeting. Changes that are made to the document must be easy to identify during the follow-up phase, consequently, the author is required to specify what changes are made.
Follow-Up	Moderator to ensure all satisfactory actions need to be taken on all logged defects, improvement suggestions, and change requests. Ensure the exit criteria are met.

Table-3 Details of the different phases in Formal Technical Review

## 2 Roles and Responsibilities

The finest formal technical review commonly comes from well-structured teams being led by moderators or review leaders that are professionally trained. For the review process, there are several roles and responsibilities being defined and required within the review meeting. Table-4 describes the discrete roles and responsibilities for each profession that will be involved in the formal technical review. It is important to understand how each of the professions plays their roles and its contribution to the success of the review.

Role	Responsibilities
Author	An individual or a team who has written the "document under review". The author needs to <ul style="list-style-type: none"> <li>• Ensures the readiness of work product for review.</li> <li>• Assists moderator in planning for review.</li> <li>• Prepares for briefing in the meeting.</li> <li>• Fixes all defects identified and resolves all issues raised during the meeting review.</li> <li>• Identify the right subject method experts for the work product review.</li> </ul>
Moderator	Known as review leader. Organizes and leads the review process. <ul style="list-style-type: none"> <li>• Work closely with the author to schedule the meeting review.</li> <li>• Coordinates with the author and checks the entry and exit criteria for review.</li> </ul>

	<ul style="list-style-type: none"> <li>• Ensures the review discussion is focused on the work product and not the author.</li> <li>• Briefs inspection team roles.</li> <li>• Distributes inspection work product to reviewers.</li> </ul> <p>The individual needs to be trained to become a certified moderator. The moderator training course can be either enrolled in-house or public.</p> <p>In-house: The certification is only recognized within the organization. The training is customized by the organization to fit its own needs only.</p> <p>Public: The certification is recognized by one or more organizations. The training is commonly used by most organizations.</p>
Reviewer	Scrutinize the document in accordance with the business specifications, standards, and domain knowledge. Verifying the work product's completeness, and correctness, also make suggestions.
Reader	A person who reads the work product in the review meetings.
Recorder/Scribe	A person who records the defect and suggestions/feedback during the review meetings after consensus is reached. Read back recorded defects to ensure the information logged is clear, complete, and correct.

Table-4 Roles and Responsibilities in FTR Process

### 3 Conduct an Effective Formal Technical Review

An effective FTR ensures the improvement of the software quality by detecting the defects upfront (either due to miscommunications or human errors) to reduce the risk of extra efforts and costs that might potentially be incurred to the project.

#### 3.1 The Review Meeting

- Every review meeting should be conducted by considering the constraints below:
  - ❖ Involvement of people in the review.
  - ❖ Sufficient advance preparation is allocated (Approximately 2 hours of work per person). If the reviewers do not have adequate preparation prior to the review date, the recommendation is to postpone the review meeting.
  - ❖ Review meeting duration should be 2 hours or less.
- Review focuses on a small group of modules rather than the entire design.
- Review focuses on the work product, not the author/designer.
- At the end of the review meeting, all participants must:
  - ❖ Accept the work product without any modification.
  - ❖ Reject the work product due to crucial error (Requires another round of review).
  - ❖ Accept the work product provisionally (Minor errors require correction, but no additional review is needed).

NOTE: The moderator plays an important role in driving the participants and closing all the misaligned opinions. This is to ensure that all participants have the same consensus on the above decisions. Conducting approval voting (The highest votes speak for the majority decision) is one of the common techniques to be used during the review meeting when having a disagreement.

### 3.2 Review Reporting and Record-Keeping

- During the review meeting, the recorder actively captures all issues and feedback that have been raised.
- All these issues and feedback are consolidated in a review list after the review meeting.
- A summary of the review findings list for formal technical review now is ready to be shared.

### 3.3 Review Guidelines

A clear and well-defined review guideline should be established and distributed to all participants in advance (1-2 weeks) before the formal technical review. The expectation of participant behavior and process for review should be clearly stated that all the participants agreed upon and then followed. This is to set accurate expectations for the review process during the review.

Roles	Responsibilities
Author	<ul style="list-style-type: none"> <li>• Limit the number of participants and insist upon advance preparation           <ul style="list-style-type: none"> <li>❖ Only the related participants are invited.</li> </ul> </li> <li>• Create a checklist           <ul style="list-style-type: none"> <li>❖ The checklist is to ensure the review meeting is conducted more structurally and only focuses on critical issues.</li> </ul> </li> <li>• Assign resources and time           <ul style="list-style-type: none"> <li>❖ Ensure the formal technical review is scheduled as a must-have task in the software development lifecycle.</li> </ul> </li> </ul>
Reviewer	<ul style="list-style-type: none"> <li>• Review the work product, not the manufacturer/producer           <ul style="list-style-type: none"> <li>❖ Defects should be pointed out constructively.</li> <li>❖ The attitude of the meeting should be loose and beneficial.</li> <li>❖ Avoid finger-pointing, insulting, or belittling.</li> </ul> </li> </ul>
Moderator	<ul style="list-style-type: none"> <li>• Set the agenda accurately           <ul style="list-style-type: none"> <li>❖ Try to keep the review on track and on schedule. Control the timing if the meeting starts drifting.</li> </ul> </li> <li>• Don't attempt to solve every problem           <ul style="list-style-type: none"> <li>❖ A review is not a problem-solving session. Separate the problem areas, acknowledge the problem, and solve the problem after the reviewing meeting.</li> </ul> </li> <li>• Restrain argument and rebuttal           <ul style="list-style-type: none"> <li>❖ When an issue is raised by a reviewer, there may not be general agreement on its impact. Record down the issue for offline discussion instead of wasting time debating the question.</li> <li>❖ Involve relevant parties to follow up on the issue discussion after the meeting. If the involved parties can't come to an agreement, assign it to the Program Manager to make the final decision.</li> </ul> </li> </ul>
Recorder/Scribe	<ul style="list-style-type: none"> <li>• Write the notes           <ul style="list-style-type: none"> <li>❖ It is recommended for the recorder to write down the notes on a (physical or virtual) whiteboard so that wording and priorities can be assessed by other reviewers as information is recorded.</li> </ul> </li> </ul>
Program Manager	<ul style="list-style-type: none"> <li>• Provide relevant training to all reviewers           <ul style="list-style-type: none"> <li>❖ To ensure the reviewing meeting is effective, formal training should be provided to the participants</li> </ul> </li> </ul>

Table-5 Review Guidelines for Each Role

## 4 Recommendations for FTR Process Enhancement

There is no doubt that implementing the FTR process has helped the organization to gain throughout the Software Development Lifecycle (SDLC). The FTR process is not something that is carved in stone and different people will have different interpretations and expectations. The FTR process enhancement discussed below is based on actual work experience which we believe will further assist the organizations to streamline their resources, time, and budget in software project management. We focus the process enhancement on two specific areas, i.e., FTR Guidelines and FTR Tool.

### 4.1 FTR Guidelines

#### 4.1.1 Review Checklist

- Create separate templates for review checklists for documentation and source code reviews
- It is recommended to make the review checklist a mandatory activity for each review.
- Reviewer can fill up the checklist to ensure nothing is forgotten in the review process.

#### 4.1.2 Merging Roles and Responsibilities

- To keep the review meeting participants as lean as possible, it is advisable to combine some of the FTR roles and responsibilities with the premise of no conflict of interest.
- The moderator, reader, and recorder roles can be handled by one person.
- The author and moderator should not be handled by the same person.

#### 4.1.3 Avoid Misjudged and Achieve Uniformly Consensus

- The author is not allowed to close the defects and comments by themselves.
- The defects and comments must be verified and closed by either the submitter or the other reviewers. This is to ensure no bias and misjudgment by the author.

#### 4.1.4 Define Specific Role Coverage for Reviewer

- Upstream reviewer
  - ❖ Roles: Software Manager, Project Lead, Quality Manager, etc.
  - ❖ Ensure the specification meets the customer's requirements.
- Downstream reviewer
  - ❖ Roles: Developer, Feature Architect, Peer, etc.
  - ❖ Ensure the software implementation meets the specification.

### 4.2 FTR Tool

To conduct an efficient FTR, besides the well-defined guidelines, an FTR tool is also important to assist and strengthen the FTR review efficiency which led the path to better quality release criteria. The organization should understand its needs and objectives to achieve in the project, then pick the right tool to accomplish the goals.

An effective FTR tool requires the following capabilities:

- Able to schedule the review meeting (Date, Time, Location, Path, etc.), assign the R&R for the participants accordingly, and send out the email notification.
- Able to capture all the defects/comments logged by the reviewers and arrange by review priority in one place.

- Able to capture all the defects/comments logged by the participants and review efforts in one place.
- Able to indicate which defects/comments need to be discussed or skip.
- Able to backtrack all the review records in one place.

#### **4.2.1 Types of FTR Tool**

- Off the shelf
  - ❖ Commercial software tool which provides software solutions for the mass market.
  - ❖ Pros: Quick to implement, can try before buying, updates are included, support is included, etc.
  - ❖ Cons: Expensive and may be impossible to customize according to the needs, upgrade/support costs, etc.
- In-house development
  - ❖ Developing the software tool using your own company experts you have on hand.
  - ❖ Pros: Able to customize and suit the needs, less expensive, higher level of control, etc.
  - ❖ Cons: Slow to implement, lack of expertise, no upgrade or support if staff dismissal, etc.

A high-quality FTR tool should have the following features that will help make the work progress run more efficiently and help participants in the process quickly understand and align their thinking and the work to be done. FTR tool creates values by having a centralized repository for all the review comments, data consolidation, email notification, versioning & tagging, FTR dashboard provides users with the analysis capability by utilizing the consolidated review comments.

#### **4.2.2 The Ability of Consolidation**

The FTR tool should be able to group the defects or feedback based on severity (from highest to lowest), similarity (Help to identify duplication), and deprioritize the “don’t need to discuss” defects (Cosmetic issues like spelling mistakes or format errors). This will help to shorten the review meeting duration and only fully focus on those important items.

No	Defects/Comments	Priority	Need to Discuss	Similarity
1	Defect A	P2	Yes	#2 – 85%
2	Defect B	P2	Yes	#1 – 85%
3	Comment 1	P3	Yes	N.A
4	Defect C	P4	No	N.A
5	Comment 2	P4	No	N.A

Table-6 Defect Consolidation

#### **4.2.3 The Ability of Notification**

The FTR tool should be able to trigger an email notification when the review is scheduled. All new changes such as R&R, date, etc. should trigger another email notification as an update.

Email for the Participants:	
Email Address	Roles & Responsibilities
<a href="mailto:AlexGrey@gmail.com">AlexGrey@gmail.com</a>	Moderator
<a href="mailto:ChristineLoo@gmail.com">ChristineLoo@gmail.com</a>	Author
<a href="mailto:DesmondChung@gmail.com">DesmondChung@gmail.com</a>	Reviewer
<a href="mailto:ArifKurf@gmail.com">ArifKurf@gmail.com</a>	Reviewer/Recorder

Date:   Send email to the participants [Add](#)

Time:  Set reminder [15mins](#) before

Location: Click here to join the meeting"/>

Work Product Path: Link to download"/>

[Setup Meeting](#)

Figure-2 Email Notification

#### 4.2.4 One-click Report Generation or One-glance Dashboard

The FTR tool should be able to present the work progress status (% open, % closed, % work in progress, % not an issue, etc.) via one-touch report generation or a one-glance dashboard. This will help to create quick insights or a summary of the overall status of project reporting.



Figure-3 FTR Dashboard

#### 4.2.5 Versioning and Tagging Capability

The FTR tool should be able to tag the versioning accordingly based on milestone release and be able to compare the delta between two different versions with just a few clicks. This will provide a quick summary of what has been changed between the current and previous releases.

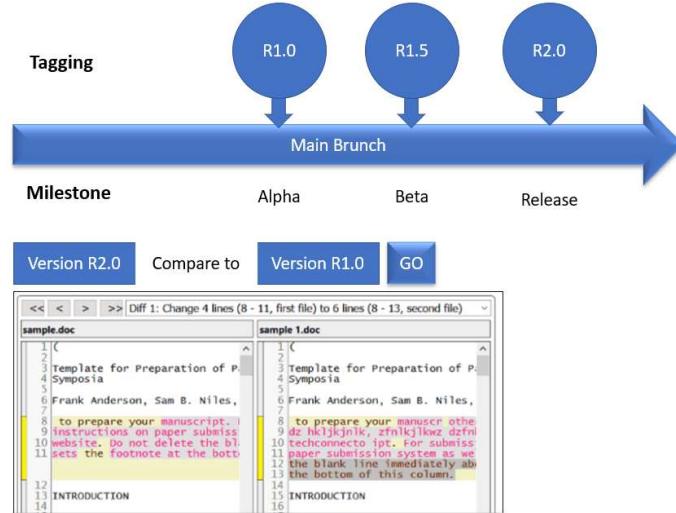


Figure-4 Versioning and Tagging

#### 4.2.6 Log the Reviewing Efforts

The FTR tool should be able to capture all the reviewer's review duration. This will provide a reference point and help to estimate the total effort needed for a similar review in future.

No	Name	Review Effort (In Hours)
1	<a href="mailto:AlexGrey@gmail.com">AlexGrey@gmail.com</a>	1.5
2	<a href="mailto:ChristineLoo@gmail.com">ChristineLoo@gmail.com</a>	1.0
3	<a href="mailto:DesmondChung@gmail.com">DesmondChung@gmail.com</a>	
4	<a href="mailto:ArifKurf@gmail.com">ArifKurf@gmail.com</a>	0.7
5	<a href="mailto:AlexGrey@gmail.com">AlexGrey@gmail.com</a>	1.2

Total: 4.4

Figure-5 Time Logged for Review Efforts

#### 4.2.7 Grammar & Spelling Checker

The FTR tool should be able to detect any spelling or grammar errors. This will help to improve the organization's professionalism.

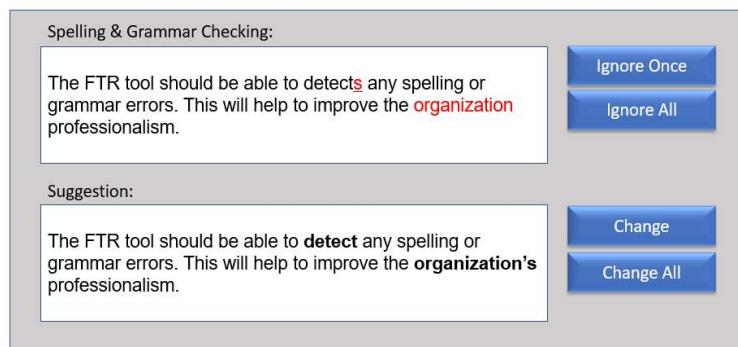


Figure-6 Grammar Check

#### 4.2.8 Plagiarism Detection

The FTR tool should be able to detect plagiarism or copyright infringement within the work product. However, this is a 'nice to have' rather than an essential feature. This will help to avoid any legal repercussions or destroy the company's professional reputation.

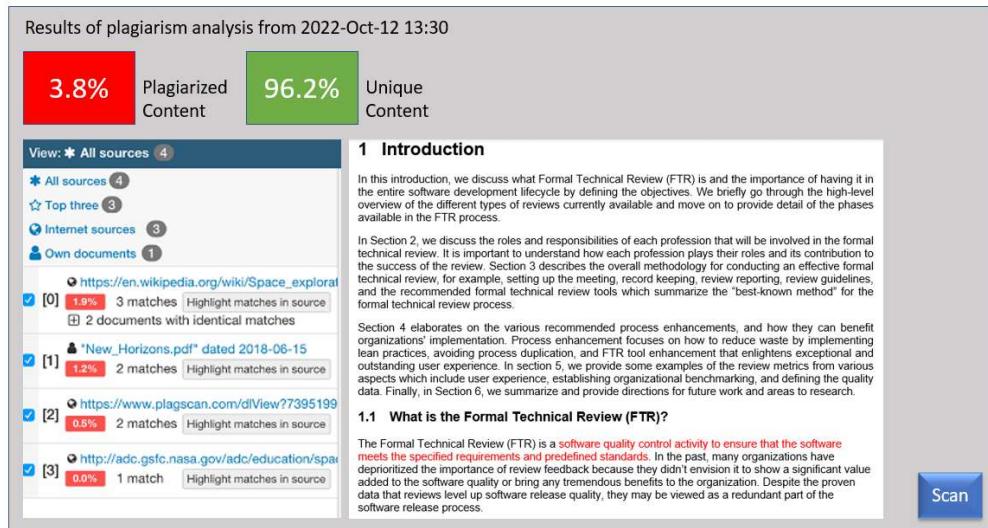


Figure-7 Plagiarism Detection

#### 4.2.9 Auto Archiving Capability

The FTR tool should be able to archive the work product and the review summary report automatically and have the capability to backtrack if needed. This will help to store as review evidence for quality auditing purposes.

No	FTR	Archived Date	Work Product Link	Details
1	FTR Meeting A	Jan 2022	<a href="#">LINK</a>	<a href="#">View Details</a>
2	FTR Meeting B	Feb 2022	<a href="#">LINK</a>	<a href="#">View Details</a>
3	FTR Meeting C	May 2022	<a href="#">LINK</a>	<a href="#">View Details</a>
4	FTR Meeting D	Aug 2022	<a href="#">LINK</a>	<a href="#">View Details</a>
5	FTR Meeting E	<input checked="" type="checkbox"/>	-	-

**Archive**

Figure-8 Records Archiving

## 5 Review Metrics

A review can be considered effective only if the FTR identifies the gap at the right time. If it fails to achieve this will indicate that the review is not effective. Therefore, it is important to track and record gaps that do not get identified at the right time in a retrospective review.

Data collected for improvement purposes expresses progress over time and inspires the team to further improve, refining their products to achieve intended outcomes. Review data requires time to accumulate to define powerful metrics to strengthen the review process.

The recommended metrics are defined as follows:

- Mean time per issue logged.
- Total reviewers' efforts in a specific review.
- Total escaped issues by time.
- Total escaped issues by cost.
- Issue Priority.

The mean time per issue logged over review time metric measures the average time spent per issue logged per reviewer. Adding more reviewers to each review could expedite the review process but it depends on the reviewer's expertise and skillset.

The total reviewers' effort metric is to measure the effectiveness of the review process based on the total time spent and the total number of issues logged in the tool per reviewer. The greater number of reviewers being invited into the review may result in more review comments being logged.

The total escaped issues by time metric are to measure whether each assigned reviewer(s) has the expertise to perform an effective review. However, the work product with fewer defects does not mean the quality is excellent. It could be due to the reviewer who does not have sufficient knowledge or expertise in this review.

Issue priority metric, as the name implies, is to compare the issue arrival based on the priority. It is critical for the author to address and resolve the issues with the highest priority over the other issues.

As a result, having these metrics is to highlight how the review process might be adapted for future improvement based on the data collected and the trending from each of the metrics identified. The team should allocate sufficient time to analyze each metric, identify room for improvement in the FTR process and predict future trending.

Samples of the above-mentioned metrics and charts are shown below:

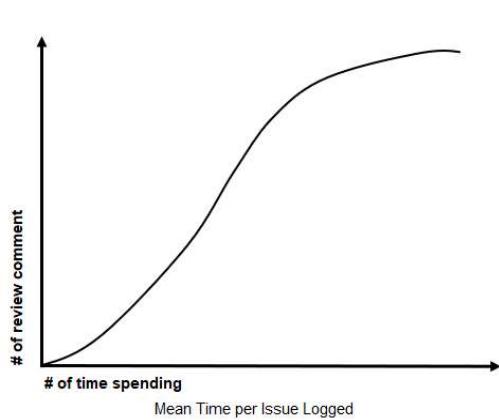


Figure-9 Mean Time per Issue Logged

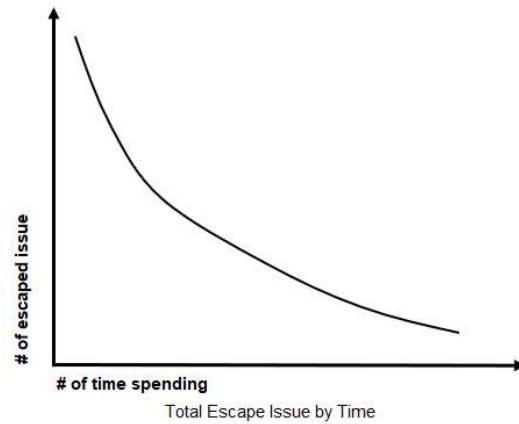


Figure-10 Total Escape Issue by Time

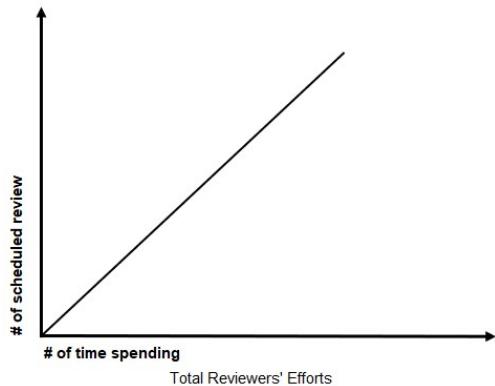


Figure-11 Total Reviewers' Efforts vs Scheduled Review

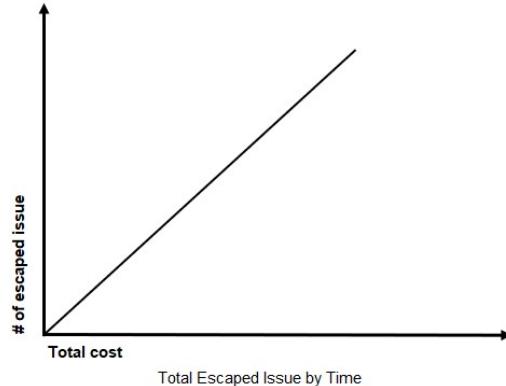


Figure-12 Total Escaped Issue by Cost

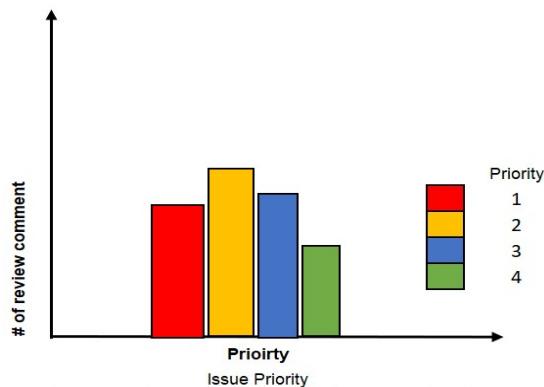


Figure-13 Issue Priority

Table-5 below shows the high-level summary of the FTR metrics based on the actual information captured on software architecture and design document review and improvement. In general, an effective FTR will adhere to the below guidance:

FTR Metrics	Improvement
Total defects captured =< 50	If the defects captured exceed the limit, it indicates that the work product is not ready. The FTR meeting should be called off, instead conducting one or more informal reviews as needed to ensure the total defects are reduced below the limit
Total reviewer time is =< 3 hours	Focus on small group of modules rather than the entire design
Escaped Issue =< 20%	Define clear roles and responsibilities for all participants and ensure the reviewers with correct expertise are invited into the review meeting to ensure the review coverage

Table-7 Sample FTR Metrics

## 5.1 User Experience (UX)

Evaluating the user experience (UX) on the FTR tool and other FTR-related guidance at an organization is important and should not be missed. A strategy without metrics is like flying blind without indicators,

hence clear metrics should be identified in our use case like review feedback, issue logged, and time spent in review. This is because these indicators represent the actual user experience that summarizes both good practices and the potential gaps.

The customer experience tracks the journey of touchpoints the customer may have with products and services. Similarly, the user experience includes all the touchpoints the employee has with the organization throughout their journey including onboarding, engagement, development, and growth. User experience metrics mainly focus on ease of use. Usability is familiar territory and something that user experience teams do well, so this makes sense as a starting point. The most used metrics are performance measures which we have described in section 5.0. These are objective measurements that record what people do.

The employee experience in FTR covered the review performance of the FTR guidelines and FTR tool in different areas of reviews throughout the Software Development Lifecycle (SDLC). The information gathered from the review indicators turns them into measurable metrics for further process improvement and enhancement. One of the methods to collect employee experience is to provide a short survey. For example:

User Feedback	Excellent	Very Good	Good	Fair	Poor	Remarks
Overall Experience	✓					
Sufficient Info to Kickstart the Review		✓				
Sufficient Review Duration		✓				
The roles and responsibilities are clear		✓				
The Review Meeting is Efficient	✓					
The FTR Guidelines is Clear	✓					
The FTR Tool is Easy to Use	✓					

Table-8 User Feedback sample

Another source of data on the user experience might be creating a focus group of participants in a retrospective type of activity to reflect on the FTR process so that feedback trends or abnormalities can be further analyzed.

## 5.2 Benchmarking

Benchmarking is an organizational tool to drive continuous improvements using best practices. This can translate into increased efficiency and create competitive advantages. Data collection from the FTR process may help organizations to develop a process for benchmarking their FTR work by accumulating and comparing the data and turning it into an acceptable value. Establishing the benchmark values for each metric of the review process is critical to ensuring the effectiveness of the review performance for each review meeting. However, the benchmark values vary from one review to another review and are highly dependent on the complexity of the software deliverables. For most cases, an effective review meeting should have a minimum number of review comments logged as part of the process despite where the issues coming from.

For instance, the review metrics are built on top of the benchmark values and provide an initial measurement of the review process. Gap analysis is performed when the actual review output is mismatched to the benchmark values and could be far from the acceptable range. A deep dive into root cause analysis (RCA) should be carried out when the review's metrics diverge from the benchmark and continuously drives further process improvement. It is recommended to continue to fine-tune the benchmark values so more accurate benchmark values can be archived.

### 5.3 Quality Data

The below information is captured based on the actual FTR meeting from the Feature requirement. The chart shows a total of 3 FTR meetings conducted for 3 new features named FTR1, FTR2, and FTR3.

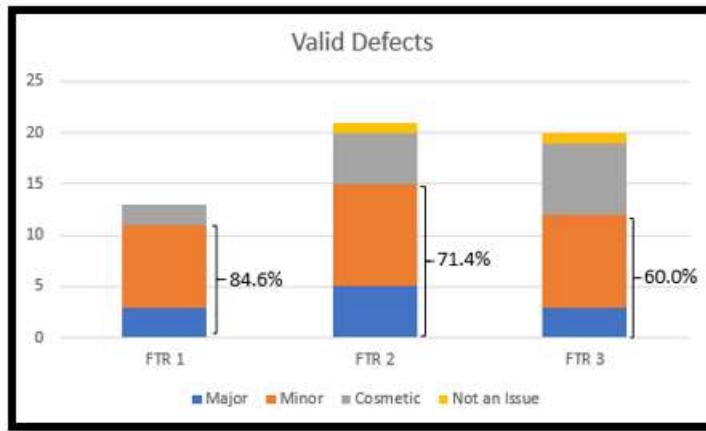


Figure-14 Valid Defects Summary on FTRs

Formal Technical Review	Major	Minor	Cosmetic	Not an Issue	Valid Defects (%)
FTR 1	3	8	2	0	84.60%
FTR 2	5	10	5	1	71.40%
FTR 3	3	9	7	1	60.00%

Table-9 Valid defects are based on the sum of Major and Minor

Based on the actual data captured from the FTR review from different projects, we can confidently set a disclaimer that without an FTR review being conducted, at least 60% of the valid defects might escape without awareness and release to market. Consequently, an additional cost will be imposed to address the escaped defects which will increase the overall cost of the project for issue fixing and introduce a poor-quality impression to the customer. In the nutshell, prevention is always better than cure, it's easier to stop the defects happening in the earlier stage than to repair the damage after it has happened.

## 6 Conclusion

In this paper, we provide a detailed explanation of the FTR which include the overview, objectives, types of reviews, roles & responsibilities, etc. Meanwhile, we also listed the recommendations for the FTR enhancements which we believe can elevate the FTR capability to a new level. The key benefits of FTR to the organization are clear: to improve the work product quality and identify the potential defects upfront to avoid any unnecessary costs being imposed in post-release defect fixing. In short, to develop an organization with strong quality culture, we must adopt new changes and view them as an improvement opportunity. It is more about managing costs more intelligently. We need to invest the time upfront to do these analyses up front, rather than spend a lot more time doing rework when defects are identified late in the validation process. FTR has proven its ability, and now it is up to you to still view it as a redundant process or a quality improvement opportunity.

## References

1. Design for instrumentation: high quality measurement of formal technical review, <https://link.springer.com/article/10.1007/BF02420943>
2. Measuring Effectiveness of a Review, <https://www.ijsrp.org/research-paper-0315/ijrp-p3969.pdf>
3. Three Employee Experience Must-Have Metrics, <https://elearningindustry.com/three-employee-experience-must-have-metrics>
4. Formal Technical Review in Software Engineering, <https://www.geeksforgeeks.org/formal-technical-review-ftr-in-software-engineering/?ref=gcse>
5. Traceability of Implementation to Design and Requirements Specs: A FTR Method (Reverse Engineering Tool), <https://www.computerscijournal.org/vol8no2/traceability-of-implementation-to-design-and-requirements-specifications-a-formal-technical-review-method-reverse-engineering-tool/>
6. Different Phases of Formal Review, <https://www.geeksforgeeks.org/different-phases-of-formal-review/>
7. Roles and Responsibilities in Review, <https://www.geeksforgeeks.org/roles-and-responsibilities-in-review/#:~:text=Formal%20review%20generally%20provides%20various,early%20in%20software%20development%20process>
8. Difference between Software Inspection and Technical Review, <https://www.geeksforgeeks.org/difference-between-software-inspection-and-technical-review/?ref=gcse>
9. Software Review and Formal Technical Review, <https://mechomotive.com/software-review-and-formal-technical-review/>

# Tri-Layer Testing Architecture

**Péter Földházi Jr.**  
 Quality Architect @ EPAM Systems  
 peter\_foldhazi@epam.com

## Abstract

Test automation engineers traditionally have designed their test automation frameworks with essentially two separate layers: Test Scripts and Test Libraries. This has been successful for some of them, but has shown to cause problems for many others.

The problem with this Bi-Layer architecture is that it does not allow the engineers to scale their solution to support multiple teams and especially multiple projects and organizations. As the application dependent libraries are intertwined, it becomes very difficult to reuse components for building new frameworks, to accelerate test automation in other teams and to scale test automation inside an organization.

That is how the idea of the Tri-Layer Testing Architecture was born: why not add a 3rd layer, for all the libraries that can be easily reused?

In this paper, I will talk about the main differences between the Bi-Layer and the Tri-Layer architectures. The three layers in the Tri-Layer Architecture are: Test Scripts (runnable test scripts), Business Logic (all the application specific libraries) and Core Libraries (independent, reusable libraries). I will provide examples from my experience at big financial technology, sportswear, online gaming and other companies, where I successfully implemented the Tri-Layer Testing Architecture and drastically lowered the test automation costs for these big companies.

## Biography

I have been working at EPAM since 2012 and moved to the USA in 2019 where I have been working as a Test Automation Consultant and as a Quality Architect.

I have experience in mobile, web and desktop testing on all levels of test automation in the financial, gaming, fitness and other domains.

I am the first European and one of the first people in the world having successfully taken and passed the ISTQB Test Automation Engineer module exam. I am actively helping the ISTQB (International Software Testing Qualifications Board) through the Hungarian Testing Board by reviewing and authoring syllabi of foundation and advanced levels.

I am also an active speaker, having spoken at meetups and software engineering conferences such as STAREAST, HUSTEF, UCAAT etc. I used to be a guest lecturer at 3 Budapest based universities: Óbuda, Pázmány and the ELTE universities. Brewing beer and planting chilis are some of my hobbies.

Copyright Péter Földházi Jr. 06/15/2022

## 1. Introduction

Test automation has become more and more prominent in recent years. While test coverage and reducing the number of issues leaking to production have been some of the key metrics for indicating the effectiveness of test automation; cost effectiveness, maintainability, scalability and modularity are also key aspects of a good solution.

That is why it is so important for test automation engineers to acquire architecture knowledge. Unfortunately, people tend to over complicate their diagrams, which will result in lengthy implementation time and higher maintenance cost. The concept of the Tri-Layer Testing Architecture helps address these complexity issues.

The three layers that I will discuss are:

- **Test scripts layer**, with the runnable tests
- **Business logic layer**, with the application dependent libraries
- **Core libraries layer**, with the application independent libraries.

## 2. Test Automation Architecture

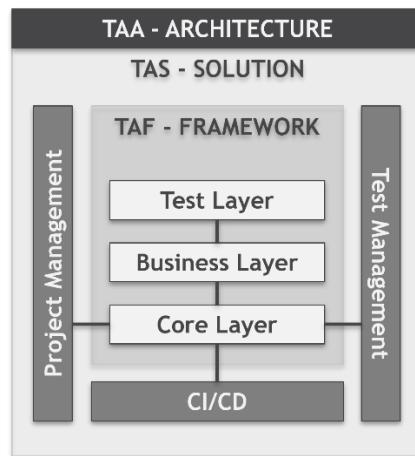
### 2.1 Framework expectations

Before going into details with the Tri-Layer concept, I want to make sure that readers are all on the same page when it comes to test automation architecture.

Let's first think about what the typical expectations are from a test automation framework:

- Automated testing
- Detailed logging
- Generated reports
- CI/CD integration
- Easy to use
- Low maintenance costs
- etc.

## 2.2 Using the right terms in test automation design



Next, I want to talk about some of the terms that people sometimes confuse when talking about test automation design.

Test Automation Architecture (TAA) is the high-level architecture design of the solution we plan to build. It is basically a blueprint of that solution.

The Test Automation Solution (TAS) is the realization of the architecture. It includes the framework, the integration to CI/CD pipelines, the project management and test management tools as well.

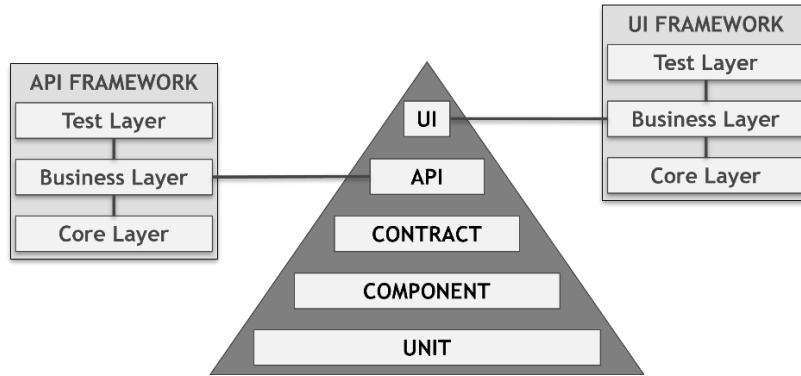
The Test Automation Framework (TAF) is the basis of the whole solution, without it, no tests would run. Therefore, it includes the test harness (typically JUnit, NUnit, XCTest etc.), test libraries, test scripts and test suites that are found inside the boundaries of the framework layers.

Framework layers define distinct borders of classes that have similar purposes. Your test scripts would fall under the very same layer. Page Models, Flow Models, an app dependent BasePage or BaseTest class file would make another layer that are application dependent.

## 2.3 Integration into software systems

Typically, simple frameworks have only one System Under Test (SUT). If you wish to support multiple applications or unrelated services with the same framework, the result is many times messy.

System level end-to-end testing involves functional validation of multiple applications, services, databases and their generated log files. The different levels should still be covered with different frameworks. Some of the basic libraries could and should be shared in the core libraries layer. If your only goal is end-to-end testing, then of course it is better to build just one all-purpose framework.



### 3. Bi-layer testing architecture

#### 3.1 Linear scripting

One of the reasons why test automation projects fail is the lack of understanding of design concepts or the lack of any type of subject matter expertise in test automation. In such circumstances, companies typically come up with a simple solution that only includes test cases, where all custom test automation libraries are absent. Scaling and maintaining such solutions is quite challenging to say the least.

Following this type of approach is not recommended in general as none of the elements of your test steps are reusable. Just think about a successful login flow. If you need to log in to an application in 50 different test cases, then you are essentially copy-pasting these steps into all the 50 tests. Whenever the login flow itself changes or the structure of the software under test, you will have to manually incorporate those changes in all the 50 test cases, instead of making that change in one place.

However, there are certain project setups or phases, when it is actually good to select linear scripting. Firstly, if you cannot afford bringing in test automation expertise, and only a couple of test cases need to be automated for a system that rarely changes, it is absolutely fine to create or generate your test cases without building an extensive framework.

Another case is when you wish to do a proof of concept before selecting the tool for building your test automation solution. You do not want to spend too much time on this activity, your goal is to compare a couple of tools and quickly make a decision, so that you can move on to building value for your team. Once the tool selection is done, you can start building a proper framework.

To give you an example from my professional life: a couple of years ago I was testing 2 iOS applications for one of my financial company clients. I wanted to see which test automation tool to select for building the test automation framework with. So I did a proof of concept (POC) with Apple's XCTest UI library and with Appium. As I needed to by-pass the biometrics (TouchID), I selected Appium, as at that time only that tool was able to do the by-pass. For covering these POC activities, I didn't need to build an advanced framework just yet. I wanted to first prove that test automation is possible, and then to prove which tool is better. In this case it was easy to do the selection, as only one tool remained by the end of the POC.

#### 3.2 Structured scripting with two layers

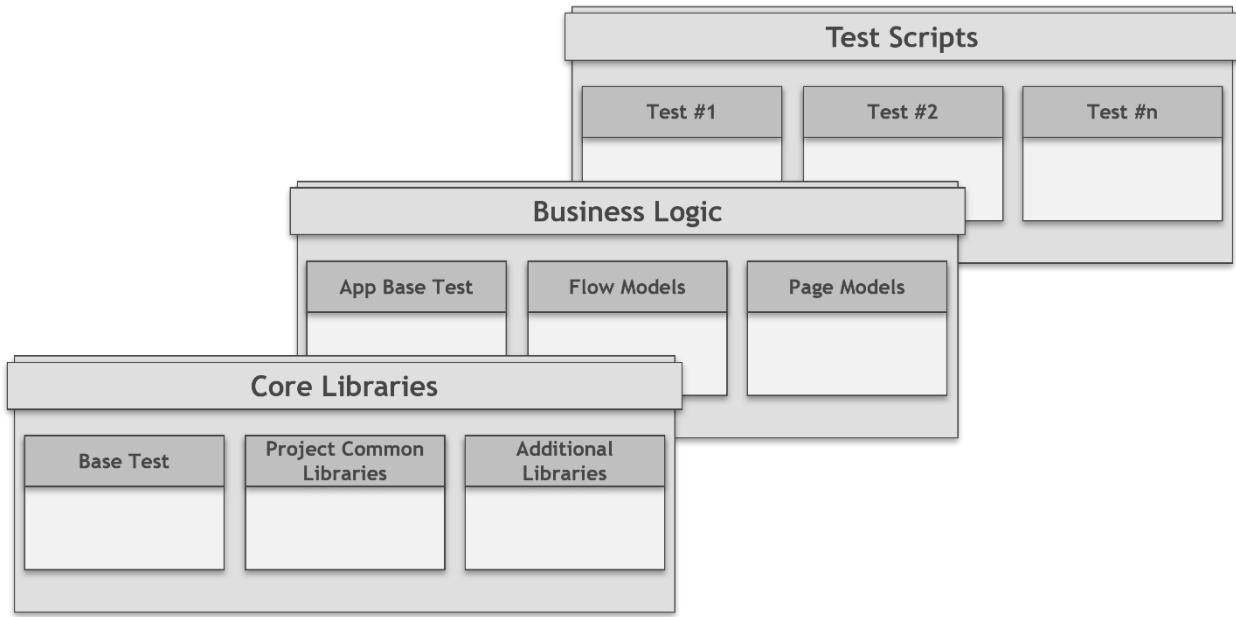
Once we introduce libraries, it allows us to scale the solution and better maintain our test cases and data. In the linear scripting approach, we had only one layer: the test scripts. The bi-layer approach introduces

the test libraries layer. This is where you find page models, flow models, utilities, step definitions, loggers, alerts etc.

This is already a decent architecture design to follow. It is maintainable and can scale much better than linear scripting does. But for bigger organizations, it is not scalable and there's circumstantial reusability holding back collaboration between teams and projects. By wiring in all the application and service dependent logic into your generic libraries (such as a Base Test, Utils or Alerts), you turn these libraries into app and service specific ones. Although it is necessary to have these configurations for your test solution to work, for others to reuse your libraries, they first have to decouple your product specific code snippets and only keep the generic ones. This may take days or even weeks to complete. And you also risk potentially giving away protected code or data that only your project is allowed to see.

In a DevOps and continuous testing world, this is a huge drawback. That is why I created the Tri-Layer Testing Architecture.

## 4. Tri-layer Testing Architecture



In 2013, I was working for one of the biggest sports clothing brands in the world. Initially, I was responsible for the test automation of one of their iOS applications. Eventually, I had to pick up the automation development for another related iOS app as well. That is when I came up with the concept of the Tri-layer Testing Architecture.

What I did was the following: identify the libraries that needed to be reused for building the framework of the second app. After the identification process, I did a pruning process in which I got rid of all the product dependent pieces of code and refactored them to make sure both frameworks will be able to leverage the same set of libraries. I started calling these common libraries as the core of the framework.

I moved the application dependent parts to new libraries that were inherited from the core libraries. This is what I started calling as middle layer at the time, but later I renamed it to business logic, as it made more sense.

Then I built a base framework for EPAM with only a core layer, which was then reused by multiple iOS projects' engineers. They were able to leverage this core layer to build their own framework on top of it. This allowed all our teams to kick-start their projects saving a couple of weeks of framework development work. The architecture design became easier as well, as instead of identifying 5, 6, 7 or even more layers (e.g. tests, feature files, steps, utilities, runner, reporter, logger etc.), my colleagues started identifying only 3 distinct layers, and that was enough on all of their projects. This again resulted in time saved. We were able to easily create maintainable automated test cases in the first couple of days of our very first sprint. We were able to mentor junior engineers to pick-up test automation knowledge quickly and also be productive during the first week.

Another example was when I designed an API test automation framework together with a test automation engineer colleague that he built for our financial client. Our goal was to implement a base framework that dozens of other teams could include as a dependency in their projects. Our business logic and test scripts layers served as the examples on how to build on top of the core layer, which was developed in a standalone repository. Once we demoed the solution, all the teams that were interested in replacing their manual API testing efforts were able to include the base solution right away. This is a modular way of building your own framework. Think of it as playing with legos and you get to choose which building blocks you get to use. The same type and colour of legos exist, and everybody can reuse that exact same lego in their own lego building. Just like software libraries.

#### **4.1 When to use and when to avoid using the Tri-Layer Testing Architecture?**

Designing your architecture with this Tri-Layer approach is most beneficial when you plan to start your test automation project from scratch or when you plan to do a complete rehaul of your current solution.

If you are doing a re-architecture of an existing framework, then it takes some time even with a Tri-Layer design. Moreover, someone with architecture knowledge first has to do an assessment of the current solution and the system under test, to identify the scope of automation. Without a proper assessment, you will just create more mess, no matter how useful the architectural approach is that you are following.

In some projects, your team may be 100% certain that your test libraries won't be reused anywhere else and it's not that important to build a core layer right away. If that is the case, then it's totally fine to follow a Bi-Layer approach. However, the problem comes when managers do not want to give a couple more days for designing and building a core layer, and then it turns out that other teams' work could have been accelerated, had you created solution independent test libraries.

My suggestion for engineers is to still implement a core layer, as it does not take much more time to do so. In case in the future you do have to reuse your base libraries somewhere else, then you save some time for yourself or for a colleague. And if you can go open-source with your solution, then that's even better!

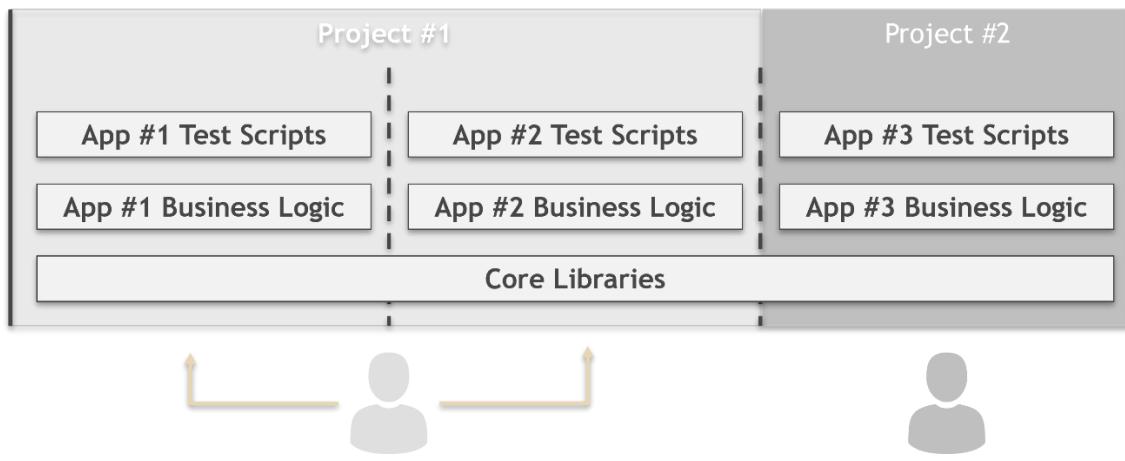
To sum it up, the test scripts layer stores all the runnable automated test cases along with the test suites. The business logic layer holds all the application dependent libraries, data and configuration. The core libraries layer consists of product independent, widely reusable components such as base test, base page, base validation, user actions and so on.

The core libraries may also include libraries that are specific only to certain projects, domains or technologies, such as a fake GPS location generator, gadget mocks, bank cards, cryptography, special app alerts etc.

## 5. Examples

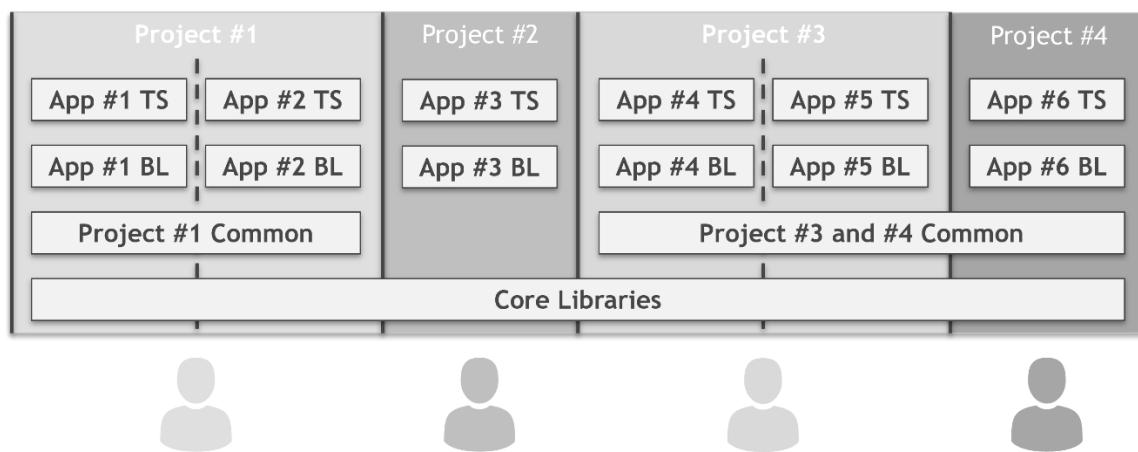
### 5.1 Example of reusability

This first example is identical to my first usage of the Tri-Layer Testing Architecture, when I built a solution for one iOS app, then I built another iOS test automation framework based on the core libraries. Later, other projects were able to reuse those libraries as well.



### 5.2 Project, domain and technology common libraries

In this next example, on top of the core libraries there are project-common libraries as well. Project #1 has its own libraries that are company specific (e.g. common way of handling users, or integration of hardware gadgets such as stride sensors). Project #3 and #4 share libraries that are domain specific: establishing connection to and processing data of public API's such as locations on a map or the schedule of a city's public transportation.

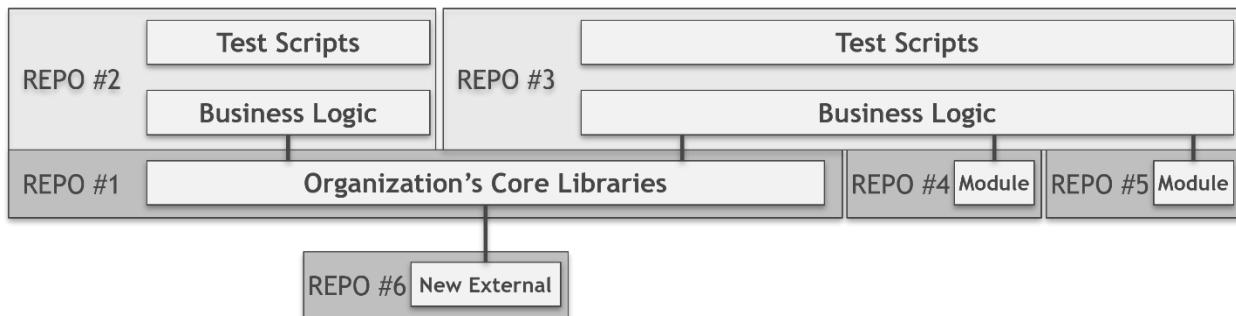


### 5.3 Repositories

The test scripts and business logic should be stored in the same repository. They could as well be in the same repository with the system under test.

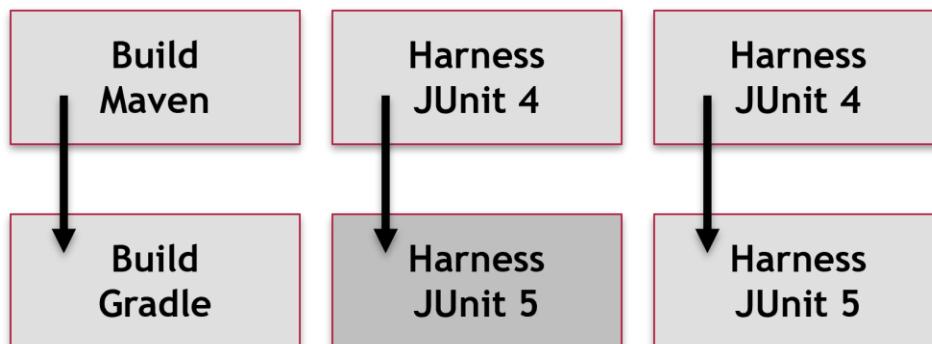
The core layer should be separate from the other two layers. Some of the modules that are considered as part of the core layer, can be managed in separate repositories. Typically, these are libraries that were developed by one team, then later picked up by other teams as well, but as not everyone want to sign-up to using these modules, they are not integral part of the organization's core libraries, but still part of the core layer for those teams that include them as dependencies.

New external libraries can be loosely coupled to the core. This allows easy replacement by pulling in an alternate, similar purpose library. External libraries are the ones that were not developed by the company's engineers. Typically, these are open-source solutions.



### 5.4 Modernization

As time goes by, some of the tools and frameworks will see major updates. In such cases, your organization could decide to adopt the newest version on organization level. If for any reason, that takes more time, then teams could individually decide to do the updates by overwriting the core functionality in the business logic layer. Reasons for not being able to merge such updates in a short time could be due to other teams not responding to change quickly. In such a case, updating it incrementally for each team makes sense. Eventually, all of the teams will be using the new version, and the dependency would be updated on core level instead of on business logic/team level.



### 5.5.1 Summary

The Tri-Layer approach provides a smooth design experience for those who wish to build test automation solutions that can be scaled easily, and which include libraries that are reusable for other teams as well. These advantages are present on project level, for the whole organization or even globally through open-source accelerators. This architectural design concept modularizes the framework into smaller libraries, which allows quickly building the framework itself and to easily update, replace or expand any of the incorporated libraries.

I hope that my readers will be able to improve their test automation after finishing this article.

## References

### Special thanks to:

Brian Crumrine, Gergely Ágneusz, Adam Auerbach, Philip Soffer, Andrew Pollner, Patrick Quilter, HTB (Hungarian Testing Board), EPAM North American QA Consultants

### HUSTEF 2014 - Péter Földházi Jr. – Effective Mobile Automation:

<https://www.youtube.com/watch?v=l2Y7VGI-iYE>

### ISTQB CTAL Test Automation Engineer 2016 syllabus:

<https://www.istqb.org/downloads/send/48-advanced-level-test-automation-engineer-documents/201-advanced-test-automation-engineer-syllabus-ga-2016.html>

<https://www.istqb.org/certifications/test-automation-engineer>

### My personal website

<https://www.peterfoldhazi.com>

# Application of Usability Testing to GUIs in the Electronic Design Automation Industry

**Kirolos George**  
**Marwa Adel**  
**Reem ElAdawi**  
**Siemens EDA**

kirolos\_george@siemens.com  
marwa\_adel@siemens.com  
reem\_eladawi@siemens.com

## Abstract

A graphical user interface (GUI) enables software users to setup, initiate, and control software execution using visual elements, standardized task menus, and simple data entry to replace tedious and error-prone command line data entry. A GUI should enable users at all experience levels to select run conditions and run the software quickly and accurately. The usability testing goal is to determine how well a GUI satisfies three parameters: effectiveness, efficiency, and satisfaction. Usability evaluation helps product engineers, developers, and quality control engineers learn how all users use the GUI, determine user experience satisfaction, and identify opportunities for improvement. We focus on GUIs used with electronic design automation (EDA) tools (software programs) that help design companies and manufacturing foundries design, simulate, and verify both the physical construction and performance of integrated circuit (IC) chip designs, and predict the outcome of IC fabrication processes. We discuss the work in progress on EDA GUIs, and how usability testing techniques can be applied and automated to improve software quality and user experience (UX) to augment manual usability testing. We evaluate a GUI used to invoke an EDA tool, gather feedback, evaluate results, and prepare recommendations to enhance the UX. We employ usability test conventions and formal usability testing by monitoring 34 participants while they use the GUI. These activities result in the identification of 56 enhancement opportunities which, when implemented, improve both the perceived UX and the usability of the GUI, satisfying the three usability parameters of effectiveness, efficiency, and satisfaction.

## Biography

*Kirolos George: Senior software quality assurance/software testing engineer for Siemens Digital Industries Software, Calibre Pattern Matching team. Received his B.Sc. in Electrical Engineering from Communications and Electronics department Ain-Shams University, and MBA from ESLSCA Business School.*

*Marwa Adel: Senior quality assurance engineer for multiple products for Siemens Digital Industries Software. Received her BSc. from the Computers and Systems Engineering department of Ain Shams University.*

*Reem ElAdawi: Test engineer director at Siemens Digital Industries Software. Received her B.SC., M.Sc., and Ph.D. from the Electronics and Communication department of Ain Shams University.*

# 1 Introduction

## 1.1 GUI usability

A graphical user interface (GUI) enables software users to setup, initiate, and control software execution using visual elements such as buttons and icons, standardized task menus and options, and simple data entry, replacing tedious and error-prone command line data entry. To reduce the risk of setup errors and simplify tool invocation, a GUI should enable users at all experience levels to select the appropriate run conditions more quickly and run the software accurately. While functional testing ensures that a GUI for an application under test performs exactly as expected using its designed functionality, usability testing determines how efficient, usable, intuitive, and learnable the GUI is in real-world conditions. The usability testing goal is to determine how well a GUI satisfies the three usability parameters: effectiveness, efficiency, and satisfaction.

Usability testing and evaluation are important tasks in the GUI design process and are considered among the more important aspects of software testing in this era of focus on the user experience (UX). Usability evaluation helps product engineers, developers, and quality control engineers learn how both expert and novice users use the GUI, determine how satisfied users are with their product experience, and identify and prioritize opportunities for improvement.

# 2 GUI usability in EDA

## 2.1 EDA industry

As digital transformation becomes the main theme and driver of many industries, software and hardware in the form of integrated circuits (ICs) now sit at the heart of product functionality. Advancements in the semiconductor industry allow IC designers to add more electrical circuits that support the new and expanded functionalities that are the main drivers of this digital transformation. The design of analog and digital electronic circuits, prototyping, testing, and production are the foundation of electronics engineering (Alajbeg and Sokele 2019).

In 1960, researchers for leading industrial and academic labs developed the first computer-aided design (CAD) tools to help and support engineers creating the layouts of electrical circuits (Brayton, et al. 2015). Since then, electronic design automation (EDA) tools continue to support and enhance IC engineer productivity through the dramatic reductions in IC size coupled with an equally dramatic growth in complexity and functionality. Many EDA tools exist to help electronic design houses and manufacturing foundries design, simulate, and verify both the physical construction and the designed performance of IC chip designs, and predict the outcome of IC fabrication processes, all before delivering a design for fabrication. These verification and simulation processes ensure that designed chips are constructed in compliance with foundry requirements, will perform according to design specifications, and will meet desired production yield targets. A billion-transistor chip can now be designed, debugged, and tested far more easily and quickly using sophisticated EDA tools that reduce time to market while ensuring product quality. Without the EDA industry, it is unlikely we would have the electronic devices and systems that are now a part of our daily lives (Lavagno, Martin and Scheffer 2006).

## 2.2 EDA GUI usability

Historically, EDA software engineers have been more concerned with developing the functional software that helps simulate highly sophisticated IC designs using complicated and difficult algorithms than focusing on the UX. This bias toward functional accuracy typically meant engineers were more likely to use technology-oriented methods rather than user-oriented methods (Nayebi, Desharnais and Abran 2012), which often resulted in GUIs that were not intentionally designed or optimized for usability.

Because usability testing in the EDA industry is still not commonplace, our focus is on graphical user interfaces (GUIs) used with EDA tools. We discuss and illustrate the work in progress on EDA GUIs, and how usability testing techniques can be applied and automated to improve the software quality and UX, and augment manual usability testing. We apply formal usability test conventions to evaluate the usability of the GUI in the Calibre Pattern Matching tool (Calibre Pattern Matching n.d.) from Siemens Digital Industries Software. Our research uses qualitative methods such as the 10 Nielsen usability heuristics evaluation and walkthrough. We monitor 34 participants while they use the Calibre Pattern Matching GUI, gather feedback, evaluate results, and prepare recommendations to enhance the UX with the current GUI. These activities result in the identification of 56 enhancement opportunities which, when implemented, improve both the perception of the UX and the usability of the Calibre Pattern Matching GUI, satisfying the three usability parameters of effectiveness, efficiency, and satisfaction.

### 3 Usability definitions and techniques

Usability as defined by ISO 9241-210 is the extent to which a software product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use (ISO 9241-210:2019 Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems 2019). Usability testing is concerned with examining and evaluating the user interface (UI). A UI contains all the components of the software product that provide information and controls for the user to accomplish certain tasks through the system. The objectives of usability testing are to:

1. Evaluate whether the usability requirements are satisfied or not
2. Discover any usability problem that requires correction
3. Measure the usability of a software product.

To deliver a high-quality user-friendly software product, a set of usability evaluations are performed iteratively:

- a. **Formative** or **exploratory** evaluation is conducted early in the development cycle (typically during the design and wireframing or prototyping phases) to get ideas and guide the design
- b. **Summative** evaluation is done just before or after implementation to measure the usability of the software product. It is a quantitative method, and it focuses on gathering measurements about the effectiveness, efficiency, or satisfaction of a software product.

All these evaluations are done to discover any usability problems, defined as defects that may result in difficulty when performing tasks through the UI. These defects impact users' ability to achieve their goals effectively, efficiently, and/or with satisfaction. Usability problems can lead to time waste, user dissatisfaction, confusion, delay, or failure to complete some tasks.

#### 3.1 Usability components

Usability is counted as a quality attribute that evaluates how effective, easy, and user-friendly a GUI is. For our project, usability is defined by five quality components (Nielsen, Usability 101: Introduction to Usability 2012):

1. **Learnability**: Is the software easy to learn the first time users encounter it?
2. **Efficiency**: Once users have learned the software, how quickly can they achieve their goals in such software?
3. **Memorability**: When users return to the software after a period of not using it, can they easily remember how to use such software?
4. **Errors**: How many errors do users make, and how does the software help users recover from errors?
5. **Satisfaction**: How pleasant is it to use such software, and how likely are users to recommend such software to others?

There are many other quality attributes (Seffah, et al. 2006) (Umara and Ghazalib 2014), but one key additional attribute we include is (Nielsen, Usability 101: Introduction to Usability 2012):

6. **Effectiveness:** Can users achieve their desired goals using such software? Does the software do what the users need?

Our usability evaluation activities include usability reviews, usability testing and user surveys (Certified Tester Usability Testing (CT-UT) 2018), as shown in Fig. 1.

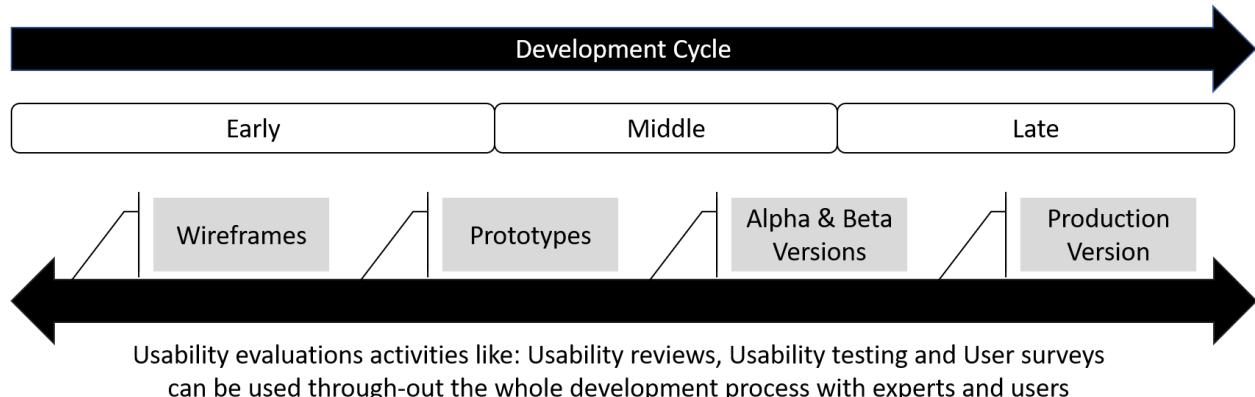


Fig. 1. Usability evaluation activities that can be used throughout the whole GUI development cycle to assess GUI usability.

### 3.2 Usability reviews

Usability reviews consist of three types:

- a. **Informal** usability reviews, which can be conducted by anyone. An informal usability review is a usability review based on the judgment of one or more reviewers who examine or use a software product to identify potential usability problems. Informal usability reviews are often based on opinion, personal experience, and common sense.
- b. **Expert** usability reviews, which are conducted by usability experts or subject matter experts (UX Expert Reviews 2018).
- c. **Heuristic evaluation**, which is a usability review in which a small group of evaluators (preferably usability experts) inspect a software product and judge its alignment with well-recognized usability principles (heuristics), identifying where the product does not follow those heuristics. A usability heuristic is a generally recognized rule of thumb that provides reliable and useful guidance to a reviewer during the usability evaluation of a software product. Jakob Nielsen created a widely recognized list of 10 heuristics to ensure ease of use and maintenance. Complying with these heuristics ensures the software product meets the three main specified goals of usability testing which are: effectiveness, efficiency, and satisfaction (Nielsen, 10 Usability Heuristics for User Interface Design 1994). Jakob Nielsen recommends using three to five evaluators to lead to the maximum benefit, as one evaluator typically discovers only 35% of usability problems (Barnum 2020). Heuristic evaluations are considered cheap, quick, and easy to do, and can usually identify approximately 50% of real usability problems (Au, et al. 2008).

It is common to combine informal and expert usability reviews with heuristic evaluation when evaluating usability.

### 3.3 Usability testing

Usability testing is likely to identify most major usability problems, as it invites real users to execute real tasks under observation, where usability issues can be seen and recorded. Users are encouraged to verbalize their actions, which aids discovery of even more usability problems. Studies show that six to eight test users are sufficient to uncover real usability problems (Au, et al. 2008). Usability testing watches a single person using the software product or a prototype to detect what and where pain points occur that confuse and frustrate the user. Results are then used to propose solutions for such usability problems (Krug 2014).

Usability testing has four steps (Barnum 2020):

1. Preparing usability test
2. Conducting usability testing sessions
3. Analyzing the findings
4. Communicating the findings with stakeholders.

Usability testing consists of a series of usability test sessions. In each session, a usability test participant performs representative tasks on the software product, or a wireframe or prototype of the software product. A test session is conducted by a moderator (a neutral person) and observed by observers. Usability testing should be done under conditions as close as possible to those under which the software product will be used, such as a mock office, a lab, or a home office environment. When possible, observation of usability test sessions should be done from a neighboring room so that observers and/or stakeholders can observe the effect of the actual software product on real people without influencing the test. Preparations for a usability testing begin with writing a usability test plan, which is a short description of the key tasks to be tested. A usability test script specifies a sequence of actions for the execution of a usability test. It is also a checklist used by the moderator of a usability test, containing the following information:

1. Activities for preparing the usability test session before the test participant arrives
2. Briefing instructions
3. Pre-session interview questions
4. Usability test tasks
5. Post-session interview questions

During the usability test session, the note-taker records usability observations, usually by writing them down. Usability observations reflect both events that cause problems or that have a positive effect on effectiveness, efficiency, and satisfaction. After all usability test sessions have been completed, the moderator and the note-taker each separately extract the usability findings from their observations. These findings reflect the observations that they consider most important. The moderator and the note-taker then meet to discuss their findings and create a usability test report consisting of a merged list of the usability problems and findings. These findings are shared and discussed with stakeholders (product development and product owners). The moderator logs the agreed-upon findings in the defect tracking tool, tracks the problems to resolution, and reviews the implemented solution. If the implemented solution represents a risk, it should be subject to another usability test.

### 3.4 User surveys

A user survey is a usability evaluation activity whereby a representative sample of users are asked to report subjective evaluation into a questionnaire based on their experience in using the software. User surveys can be used to evaluate the levels of user satisfaction with a software product. Surveys are useful as they allow users to participate anonymously, and they are generally a cheap and easy tool. Surveys can include various demographics of people to compare their answers (Marsh 2016).

## 4 Usability testing challenges in the EDA industry

Usability testing is a very challenging activity in the EDA industry for numerous reasons:

1. Customers use the EDA tools when simulating their designs, which are confidential and cannot be exposed outside their companies or disclosed or shared with anyone
  - a. Cannot easily access real users in their normal environment (Bezerra, et al. 2014)
  - b. Sometimes we can't know the exact way users use our tools
2. Some users are used to traditional methods of using EDA tools and are reluctant to adopt new practices
3. Product development team members usually focus on functionality and performance rather than usability (Nayebi, Desharnais and Abran 2012)
4. Can be difficult to justify investments in usability and UX efforts to management (Sauro 2016)
5. Cannot build code within the software to track most-used features
6. Pressures of time and budget considerations (Bergstrom, et al. 2011)
7. Difficult to recruit and bring in real users to participate in usability testing
8. Multiple personas (e.g., foundry engineers vs. design engineers) may use the same EDA tool in different environments and conditions

## 5 Introduction of new UX Process

### 5.1 Adopting usability testing

A GUI consists of components (widgets) that provide information and controls for the user to accomplish specific tasks with the software tool. We have been performing GUI functional testing and running automated functionality tests for years. We have functional specifications for all tools, so we begin by writing a functional test plan, then perform functional testing. This kind of testing determines if the GUI functionally behaves as it is designed to do.

Testing how easily or intuitively a GUI performs for users is usability testing. A software product can work exactly as intended in the specifications and still have serious usability problems, such as:

- Complicated/unclear error messages
- Unclear use model/steps of operations
- No enabled/disabled buttons to help guide the user
- Bad/poor look and feel
- No undo/redo to correct user mistakes
- Extra/unnecessary steps that confuse the user

While most product design teams address usability at the end of the product development life cycle, we wanted to address usability throughout the product development life cycle, as shown in Fig. 1. We started by adding usability testing to our test process. We used the three usability evaluations activities—usability reviews, usability testing, and user surveys—to gather feedback from both internal team members and real users to evaluate the usability of the GUI used in the Calibre Pattern Matching tool (Fig. 2).

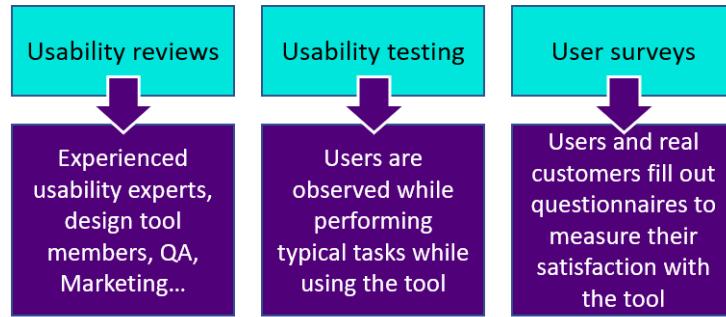


Fig. 2. A variety of processes were used to collect feedback on GUI usability.

The usability test is a black box technique for measuring the overall UX when using the tool. The outcome of a usability test is a report or list of findings that includes:

- Measurements of how easy/difficult, intuitive, and learnable the tool is (based on the usability components discussed in 3.1)
- Action items and bug/enhancement reports to define changes to the tool based on the feedback received

Our GUI usability testing was based on:

1. UI design patterns/usability standards (for example, input feedback, Undo/Redo actions, and usage of module tabs)
2. Usability requirement checklist

The usability requirement checklist we adopted (Table 1) contained 10 heuristics based on the Nielsen heuristics discussed earlier, in addition to a check to ensure the development team provides the whole team with wireframes or low-fidelity prototypes to assess.

Usability heuristic	Desirable characteristics
Visibility of system status	Visible, relevant, timely feedback to keep users informed of system operations and status.
Match between system and real world	System messages use words, phrases, and concepts familiar to users, and display in a natural and logical order.
User control and freedom	System provides clear and simple undo and redo functions to enable users to correct inadvertent usage mistakes without complicated or lengthy steps.
Consistency and standards	System employs consistent usage across all operations that complies with platform standards.
Error prevention	When irreversible or potentially problematic events cannot be eliminated from GUI by design, system provides review/confirmation option before users commit to the action.
Recognition rather than recall	System makes objects, actions, and options visible across all operations to simplify user recognition. Instructions/help for system use are visible or easily retrievable when needed.
Flexibility and efficiency of use	System provides “accelerators” that can be accessed by expert users to speed up interactions and tailor frequent actions, while still supporting novice users with full guidance.

Aesthetic and minimalist design	Dialogues contain only the minimum of language needed to achieve the task.
Help users recognize, diagnose, and recover from errors	Error messages are expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
Help and documentation	Support/help information is easy to search, is focused on the user's current task, lists specific steps to be executed, and contains complete but minimal language.

Table 1. Usability heuristics

We adopted an informal UX testing approach in our overall testing techniques, using the following activities:

1. Include UX testing in functional test plans.
2. Validate the usability of the GUI under test against UX components and standards.
3. Add automated test scripts to simulate user actions in the GUI tool and output the results data as part of the test validation.
4. Regularly run automated usability testcases against the GUI to guarantee consistency across versions and updates.
5. Review wireframes and low fidelity prototypes provided by the development team for evaluation and assessment.

## 5.2 Usability testcases automation

Evaluating the usability of new features can be done using multiple processes. We can manually walk through the different user scenarios of the feature, use batch application programming interface (API) functions to access its different functionalities, or add automated testcases that simulate the required tests.

Testing the usability of an application under test (AUT) is now performed as part of the regular GUI functional testcases automation. To support our GUI usability evaluation process, we added usability checks to our automated testcases that address the functional testing for our products for the flows we want to check for usability. These usability checks simulate user actions in the GUI, such as clicking on different widgets to perform steps that must be executed to reach or perform a specific functionality, then dump their content and state (enabled/disabled) to evaluate the results.

The main goal of adding usability checks to these automated testcases is to ensure we consistently test the GUI tool against the UX conventions and the 10 Nielsen heuristics to confirm its usability remains intact. In the event of future code changes, the previous tool behavior acts as a baseline to enable us to determine if the code changes result in a “failure” of usability compliance that requires remediation.

We also added a dedicated usability testing section inside our test plans for tool features to address the level of testing that needs to be done to ensure tool use is effective and intuitive.

For example, testing a new dialog includes the following steps:

1. Check that the new dialog invocation is accurately placed
2. Add automated testcases to dump the default values of the dialog
3. Dump the tool tips of all widgets to check text consistency from run to run or identify text changes
4. Dump and check the state of the widgets, (selected/unselected) and/or (enabled/disabled)
5. Check the number of steps required to perform or reach target functionality to look for unnecessary steps
6. Dump all warnings and error messages to check that text is comprehensive, clear, intuitive, and correct
7. Check and verify that generated messages include updated data for any values that are changed
8. Dump the font type and font colors of the dialog widgets and check against usability standards

9. Check the background color for the drawing area to ensure the text is easily readable
10. Take screenshots of the drawing area and save them for image comparisons
11. Use the undo/redo functions of the tool to ensure dialogs retrieve previous input(s) correctly
12. Ensure keyboard actions result (like pressing the tab buttons) result in the desired function (including forward and backward, where applicable), and that keyboard shortcuts work properly

We run these automated testcases as usual on different platforms/environments, as part of the complete functional test suite regression. They can fail if any of the usability conditions are not met, or if unexpected code changes were applied. These tests can be updated as needed with GUI enhancements.

Clicking the widgets and dumping their content is performed using scripts (typically Python). During the testcase execution, all Python commands are performed in the same order, and dumps are compared after the testcase is fully run. The testcase will fail in any of these conditions:

- The script failed to access any widget because it's invisible, disabled, or moved to a different frame/dialog
- The content of any dump is different from its saved (baseline) version
- Any messages or output of the test are changed
- Any GUI behavior has changed

### **5.2.1 Mapping UX heuristics with automated test scenarios**

We try to measure our usability heuristics against all new features, using automated usability tests to avoid any unneeded or random outcome of the GUI. With reference to the heuristics checklist in Table 1, we continuously add automated scripts to check and verify usability.

<b>Usability heuristic</b>	<b>Automation</b>
Visibility of system status	Dump AUT messages and warnings. Dump AUT status bar.
Match between system and real world	Verify all labels, error and warning messages, tool tips use simple, real-world language. Verify all error and warning messages provide clear guidance for accurately performing task. Dump label names. Dump error and warning messages.
User control and freedom	Verify tool dialogs include specified or required user controls (ex: Cancel, Apply, Undo, Redo buttons). Dump user control state status (e.g., enabled/disabled). Verify user controls are enabled and clickable when needed.
Consistency and standards	Verify labels, tooltips, messages, and warnings, generated by the GUI are comprehensive, correct, and compliant with Calibre product standards. Verify similar functionalities in various Calibre products use the same shortcuts and are placed in similar drop-down menus.
Error prevention	Verify unneeded widgets are disabled. Verify tool prevents bad data (e.g., special characters in line edit fields).

	<p>Verify tool provides clear warning/opt-out message to users before performing any deletion action to prevent data loss.</p> <p>Verify dialog widgets include only specified/standard actions (e.g., Apply, Cancel, Close buttons).</p> <p>Verify provided error messages provide corrective guidance and prevent actions likely to fail.</p> <p>Verify warning and error messages ask for confirmation of risky/irreversible task before it is performed.</p> <p>Dump all widgets' status and verify they are only enabled in correct order of testcase execution.</p>
Recognition rather than recall	<p>Verify user choices are recognized and saved for future checks.</p> <p>Dump tooltips and verify they provide accurate, clear guidance to users for performing specified scenarios.</p> <p>Dump tool transcripts and verify user choices are printed correctly to minimize the need for user memory recall.</p> <p>Verify indicators provide users with flag if anything is edited/changed (Fig. 3).</p> <p>Dump dialog fields before and after user actions and verify AUT remembers user's choices.</p> <p>Dump selected menus and verify the last-edited items appear first in the list.</p>
Flexibility and efficiency of use	<p>Dump selected menus and verify the last-edited items appear first in the list.</p> <p>Use AUT shortcuts and verify they function correctly and invoke the correct dialogs.</p> <p>Test the search and filter features to verify user can easily and quickly reach intended data.</p> <p>Dump dialog fields and verify default values are set correctly to speed up the process and enable user to perform intended goal efficiently.</p>
Aesthetic and minimalist design	<p>Dump widget frames and windows sizes to verify GUI provides the same view on different environment (OS, machine resolutions, platforms).</p> <p>Dump font styles and file window types (native or modified) to verify GUI provides the same view on different environment (OS, machine resolutions, platforms).</p> <p>Verify users can use menu button(s) to select which dialogs to view, keeping unselected dialogs hidden (Fig. 4).</p>
Help users recognize, diagnose, and recover from errors	<p>Dump messages and warning messages and verify they provide accurate error correction guidance.</p> <p>Verify selected actions can be performed using multiple alternative methods in the GUI.</p>
Help and documentation	<p>Dump GUI help messages and options and verify they are comprehensive, accurate, and easy to read.</p> <p>Dump tool tips and help buttons and verify the information accurately guides users to correct tool use.</p>

Table 2. Automated usability verification

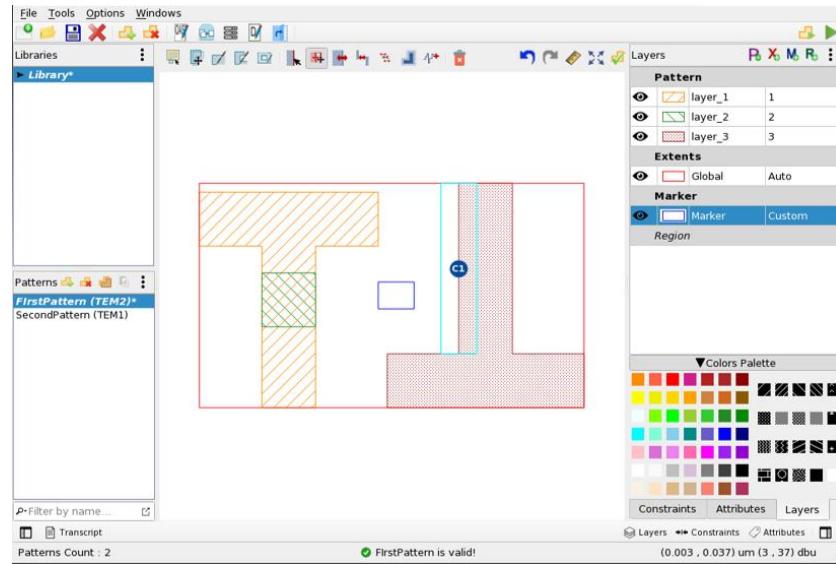


Fig 3. Verify flags are used to indicate item is edited but not yet saved. Under Libraries, “Library” is displayed in bold and italic format and has an asterisk. In Patterns, “First Pattern (TEM2)” is displayed in bold and italic format and has an asterisk. These flags indicate those items have changed, but have not been saved.

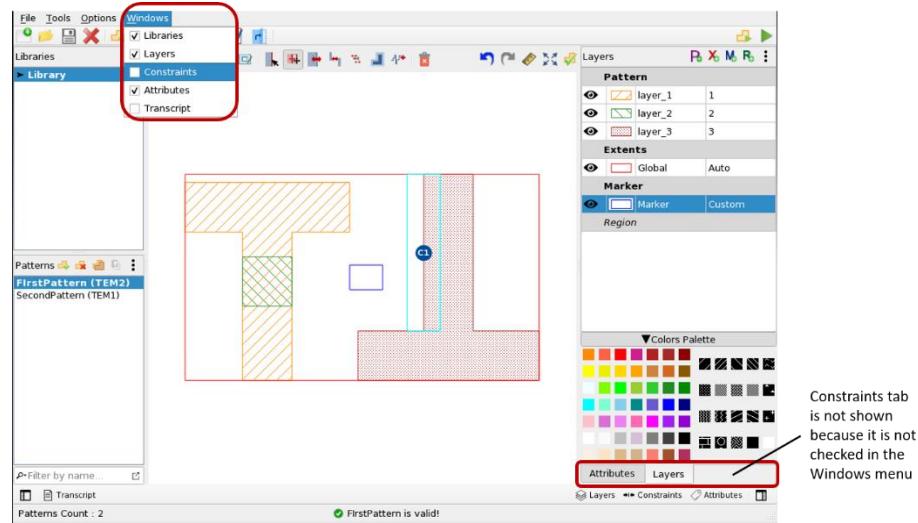


Fig. 4. Verify user can choose which widgets to view. The constraints tab is not displayed because it was not checked in the menu options.

## 6 U2U success story

### 6.1 U2U usability testing

Siemens product usability labs are part of a formal usability testing event held during the annual Siemens User-to-User (U2U) conference (User2User n.d.). During the U2U usability labs, product teams can directly engage customers to collect feedback on new product enhancements or usage flows.

### 6.1.1 Usability test lab

Setting up and running a usability test lab involves multiple steps:

1. The product team (including quality assurance/software tester(s), developer(s), and product owner) identifies a product/features to be tested for usability.
2. The product team prepares exact usage scenarios for the product/features.
3. The product team prepares a set of questions about the feature(s) for which they want to gather customer feedback to measure user satisfaction with the GUI. These questions are all related to the product UX while performing the scenario steps.
4. Computers are set up with the tools, executables, test data, and usage scenarios.
5. Two staff members are assigned to each station. One serves as a moderator, and the other is available to answer in-depth technical questions and troubleshoot if issues arise with the product itself.
6. Users from different expertise levels are invited. Some are customers actively using the product, while others are potential customers, or individuals with no previous experience with the tool.
7. Users are instructed to perform the specific tasks in the usage scenario:
  - a. Using a specified order of steps
  - b. Using specific test data
8. Users are asked to provide their feedback to session moderator using the think aloud technique.
9. Users can ask the technical person questions about the product or any issues they encounter.
10. Users agree to answer specific questions related to the exercise/task they performed.
11. Both staff members observe users and take notes as the users go through the lab and ask questions

### 6.1.2 Data collection and analysis

After the U2U usability lab event:

1. Test feedback is collected, and data is extracted and analyzed.
2. A final report is prepared containing the findings of the usability test, including difficulties encountered by users, recorded observations, questions asked by users during testing, and user feedback.
3. The report is presented to the product team.
4. Product team uses report data to identify and implement GUI changes to improve the UX.
5. All the usability test labs are added into the GUI test suite as automated testcases to automate and perform the exact steps users were asked to perform in the U2U lab event.

## 6.2 Usability testing results

Three usability evaluations activities—usability reviews, usability testing, and user surveys—were used to gather feedback from internal team members and real-world users to evaluate the Calibre Pattern Matching GUI. Mixing quantitative and qualitative methods helped provide an understanding of how the UX affects brand attitudes, letting us measure what people think against what they do. Using the information gleaned from usability evaluations contributed to identifying several usability gaps and opportunities in the Calibre Pattern Matching GUI, resulting in 56 distinct enhancement tasks. While implementing the enhancements, we were able to watch participants interact with the wireframes (a basic blueprint for the software product) (Marsh 2016) and prototypes, as well as the final product, to determine what worked and what did not (Bergstrom, et al. 2011) (Norman 2013). Implementing these changes contributed to a 28% increase in UX satisfaction, as calculated by dividing the number of the filed usability items by the number of filed functionality items within a certain time span.

### 6.2.1 Findings caught by usability testing and the ten Nielsen heuristics and informal usability review

A comparison of the previous pattern capture tool, in which users had to select which capture mode to apply, to the new pattern capture tool after deploying the usability evaluations enhancement is shown in Fig. 5. The new view satisfies Fitts' law (Interaction Design Foundation n.d.), which states that the amount of time required for a person to move a mouse to a target area is a function of the distance to the target divided by the size of the target. Therefore, the longer the distance and the smaller the target's size, the longer it takes. The pattern capture tool was redesigned to add a visual display of capture mode options inside a much larger button, making it easier, more intuitive, and faster for users to review options and select the desired capture method. In addition, we reduced the potential number of clicks by setting a default value (the current directory) for the output file path. If the default value is acceptable, the user can proceed with the minimum number of clicks; otherwise, the user has the flexibility to either select a certain area or enter the desired coordinates and the required cell name.

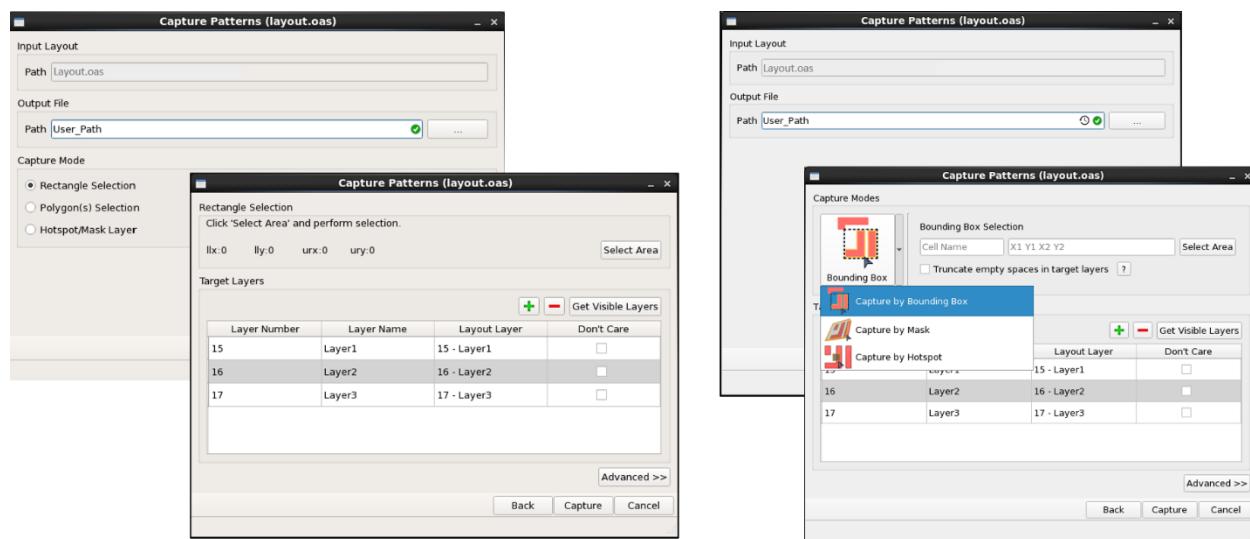


Fig. 5. Comparison of the old pattern capture GUI (left) to redesigned GUI (right). Visual selection options replace text-based options for faster, easier selection.

### 6.2.2 Findings caught by the Nielsen heuristics

Using the Nielsen heuristics to analyze usability revealed that the previous Calibre Pattern Matching main content viewing window required users to click either a polygons tab or constraints tab to choose among set of tools to use. By enhancing this content window to minimize the number of clicks and provide users with more control and freedom, the enhanced GUI provides users with a visual and intuitive set of frequently used tools, eliminating the secondary need to choose between polygon and constraints tools as shown in Fig. 6. This change better satisfied two of the Nielsen heuristics:

- Recognition rather than recall
- Aesthetic and minimalist design

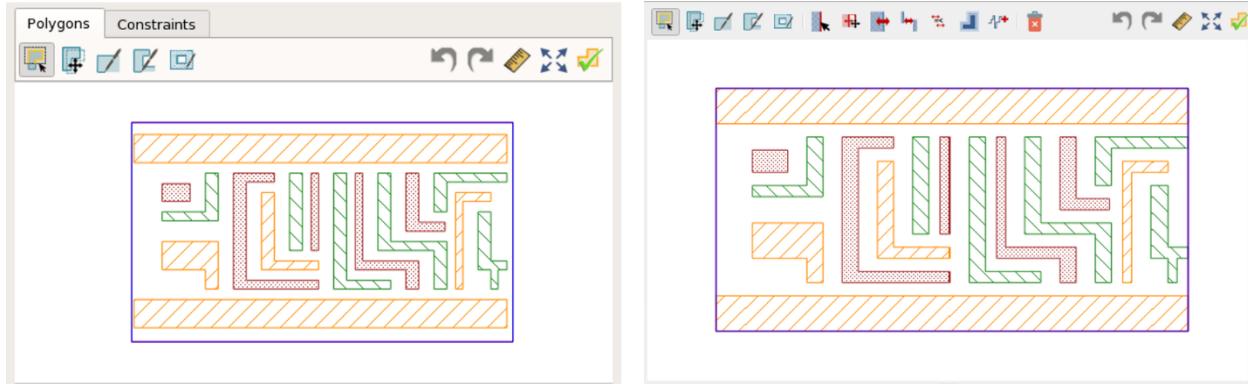


Fig. 6. Compared to the previous Calibre Pattern Matching main content viewing window (left) that required multiple clicks to select tools, the enhanced version (right) lets users select any tool quickly and easily using visual selection options.

### 6.2.3 Findings caught by usability testing

In the previous Calibre Pattern Matching Compile Library viewing window, users had to remember not to close the results window, so they could access the results multiple times without having to re-run to reach the results again. This usability bug was caught through the User2User Usability labs while the QA, development and product engineering team members were monitoring real users while they used the GUI as shown in Fig. 7. After fixing this usability bug, a “Results” tab was added as shown, allowing users to choose whether to edit the run setting from the setup tab or to access the results from the Results tab.

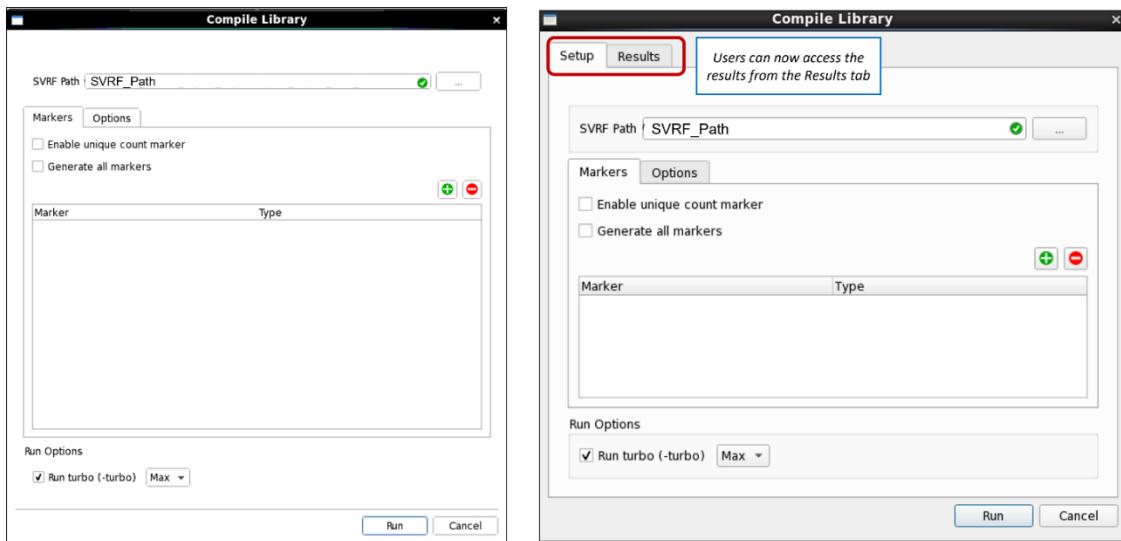


Fig 7. The Calibre Pattern Matching Compile Library window was enhanced to simplify results access based on findings obtained from usability testing.

### 6.2.4 Contribution of each usability evaluation method

The contribution of each usability evaluation method in identifying new enhancements for the Calibre Pattern Matching tool is shown in Fig. 8. Our usability testing activities contributed to the delivery of a

premium quality GUI that eases the workload of users by helping them achieve their goals with a more intuitive UX (Walia and Casey 2021).

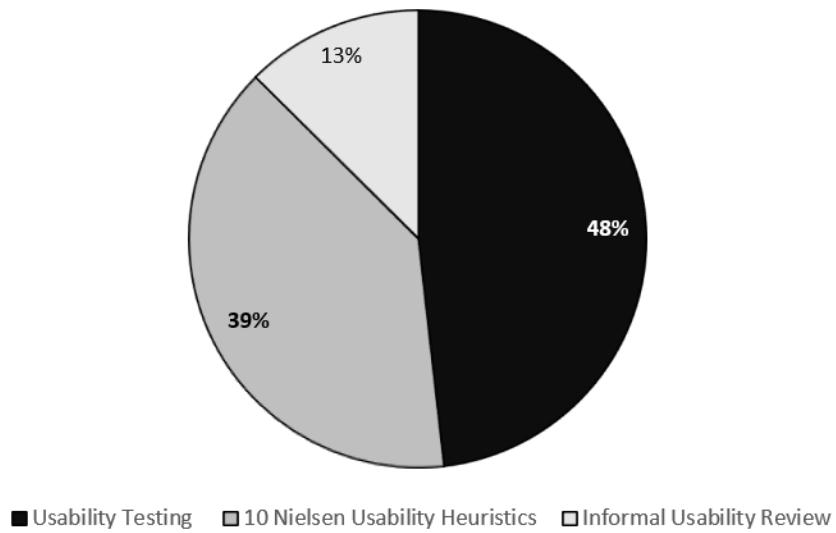


Fig 8. Percentage contributions of each usability evaluation method to improvements in UX.

## 7 Conclusion

Usability evaluations are an important component of today's software testing process, especially in the EDA industry. EDA tools contain sophisticated functionality, and their complexity increases with each new semiconductor technology manufacturing node. Poor usability not only affects brand image, but also lowers the productivity of users and blocks them from achieving their desired goals in an effective and efficient manner. Automating usability tests helps organizations deliver a high level of ease of use by providing user-friendly software that contributes to a positive holistic experience of users during and after their use of the tool. It also improves testing efficiency and makes it easy to integrate usability within the regression suite and the development process.

Enhancing the usability and the UX of software is not only the responsibility of the software testing/quality control/quality assurance teams, but also the development, software testing, and product owner team members. It is a common vision and effort that contribute to enhancing usability, starting from questioning any and all aspects of the tool design, to evaluating the software product with real users and gathering their feedback using multiple usability evaluation methods, preparing enhanced wireframes and prototypes and testing these in accordance with the usability heuristics, performing usability testing activities and analyzing the output, until launching the production version of the enhanced software. Usability analysis requires the use of suitable methods to understand user behavior and perception and to identify user needs and challenges to deliver a useful, effective, and efficient software product. Implementing and deploying a software process that includes usability analysis within the software product life cycle helps identify and eliminate potential usability problems early in the software development cycle.

We used multiple usability analysis methods to evaluate the usability of the Siemens Calibre Pattern Matching GUI, resulting in 56 distinct enhancements that resulted in improved performance, ease of use, and product perception. By developing and employing an automated usability testing framework within our complete functional test suite regression, we can quickly update our usability requirements and checks as needed in the future to ensure all our products satisfy the three usability parameters of effectiveness, efficiency, and satisfaction.

## References

- Alajbeg, Trpimir , and Mladen Sokele. 2019. "Implementation of Electronic Design Automation software tool in the learning process." International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Opatija, Croatia: IEEE. pp. 532-536. doi:10.23919/MIPRO.2019.8757096.
- Au, Fiora T. W., Simon Baker, Ian Warren, and Gillian Dobbie. 2008. "Automated usability testing framework." *The Ninth Conference on Australasian User Interface*.
- Barnum, Carol M. 2020. *Usability Testing Essentials: Ready, Set...Test!* San Francisco, United States: Elsevier Science & Technology .
- Bergstrom, Jennifer C. Romano, Erica L. Olmsted-Hawala, Jennifer M. Chen, and Elizabeth D. Murphy. 2011. "Conducting Iterative Usability Testing on a Web Site: Challenges and Benefits." *Journal of Usability Studies*.
- Bezerra, Carla, Rossana M. C. Andrade, Rainara Maia Santos, Mourad Abed, Káthia Marçal de Oliveira, José Maria Monteiro, Ismayle Santos, and Houcine Ezzedine. 2014. "Challenges for usability testing in ubiquitous systems." *IHM '14: Proceedings of the 26th Conference on l'Interaction Homme-Machine*. 183–188.
- Brayton, Robert, Luca P. Carloni, Alberto L. Sangiovanni-Vincentelli, and Tiziano Villa. 2015. "Design Automation of Electronic Systems: Past Accomplishments and Challenges Ahead [Scanning the Issue].," IEEE. pp. 1952-1957. doi:10.1109/JPROC.2015.2487798.
- n.d. *Calibre Pattern Matching*. Accessed July 7, 2022. <https://eda.sw.siemens.com/en-US/ic/calibre-design/physical-verification/pattern-matching/>.
2018. "Certified Tester Usability Testing (CT-UT)." Accessed July 17, 2022. <https://www.istqb.org/certifications/usability-tester>.
- Interaction Design Foundation. n.d. *Fitts' Law* . Accessed August 12, 2022. <https://www.interaction-design.org/literature/topics/fitts-law>.
2019. *ISO 9241-210:2019 Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*. July. Accessed July 7, 2022. <https://www.iso.org/standard/77520.html>.
- Krug, Steve . 2014. *Don't Make Me Think, Revisited A Common Sense Approach to Web Usability*. New Riders.
- Lavagno, Luciano , Grant Martin, and Louis Scheffer. 2006. *Electronic Design Automation for Integrated Circuits Handbook*. CRC Press.
- Marsh, Joel . 2016. *UX for Beginners*. Canada.: O'Reilly Media.
- Nayebi, Fatih, Jean-Marc Desharnais, and Alain Abran. 2012. "The State of the Art of Mobile Application Usability Evaluation." *Canadian Conference on Electrical and Computer Engineering*. IEEE. doi:10.1109/CCECE.2012.6334930.
- Nielsen, Jakob. 1994. *10 Usability Heuristics for User Interface Design*. April 24. Accessed July 17, 2022. <https://www.nngroup.com/articles/ten-usability-heuristics/>.
- . 2012. *Usability 101: Introduction to Usability*. January 3. Accessed July 7, 2022. <https://www.nngroup.com/articles/usability-101-introduction-to-usability/>.
- Norman, Don. 2013. *The design of everyday things*. Basic Books.

Sauro, Jeff . 2016. "The Challenges and Opportunities of Measuring the User Experience." *Journal of Usability Studies*.

Seffah, Ahmed, Mohammad Donyaee, Rex B. Kline, and Harkirat K. Padda. 2006. "Usability measurement and metrics: A consolidated model." *Software Quality Journal*. doi:10.1007/s11219-006-7600-8.

Umara, Muhammad Aminu , and Masitah Ghazalib. 2014. "Investigation into Usability Attributes for Embedded Systems Testing." *International Journal of Engineering and Technology*.

n.d. *User2User*. Accessed August 1, 2022. <https://events.sw.siemens.com/en-US/u2uconference/>.

2018. *UX Expert Reviews*. February 25. Accessed July 17, 2022. <https://www.nngroup.com/articles/ux-expert-reviews/>.

Walia, Ritu , and John Casey. 2021. "QA Best Practices: GUI Test Automation for EDA Software." *PNSQC* . PNSQC .

# Building Security into Your Apps One Story at a Time

Bhushan B. Gupta  
 Gupta Consulting LLC.  
 Bhushan@bgupta.com

## Abstract

In the agile software development, a story is the smallest element of your application and setting an appropriate security threshold dictates the security of your application.

This paper discusses the implementation and validation of security controls in the lifecycle of a story in the agile software development environment. The three 'Ws'; what, when, and who are emphasized with reference to WHAT security controls to implement, WHEN to verify and validate the implementation, and WHO should assure that the security control provides the intended safeguard. The role and time of engagement of product owner, security engineer, quality engineer, and test engineers are explained as a story progresses from one stage of the lifecycle to the next.

With the help of examples, the paper demonstrates the necessary security controls at the product definition. Once the security controls for a story are defined, the implementation needs verification by the security engineers and the product owner. The test team is then responsible to validate that the security controls are working as intended in the context of the application and without degrading the customer experience. The paper also highlights the post deployment security related activities and measures that should be taken for an uninterrupted operation.

## Biography

A proven champion for quality, well-versed with software quality engineering, and a WebApp security researcher, Bhushan is the principal consultant at Gupta Consulting, LLC. In WebApp security, his research areas are; infusing security in SDLC, OWASP Top10, Risk Analysis and Mitigation, Attack Surface Measurement, and Static and Dynamic Application Security Analysis. As a leader of Open Web Application Security Project (OWASP) Portland Chapter, he is dedicated to driving the web application security to higher levels via technical education and training. Bhushan often provides training workshops and presentations to corporations and non-profit organizations. He is also an invited speaker and a panelist in discussions for both application security and agile software development. Bhushan serves as a Program Team member for the Pacific Northwest Software Conference and has been a member of the Program team for the Global AppSec Conference 2020 organized by OWASP.

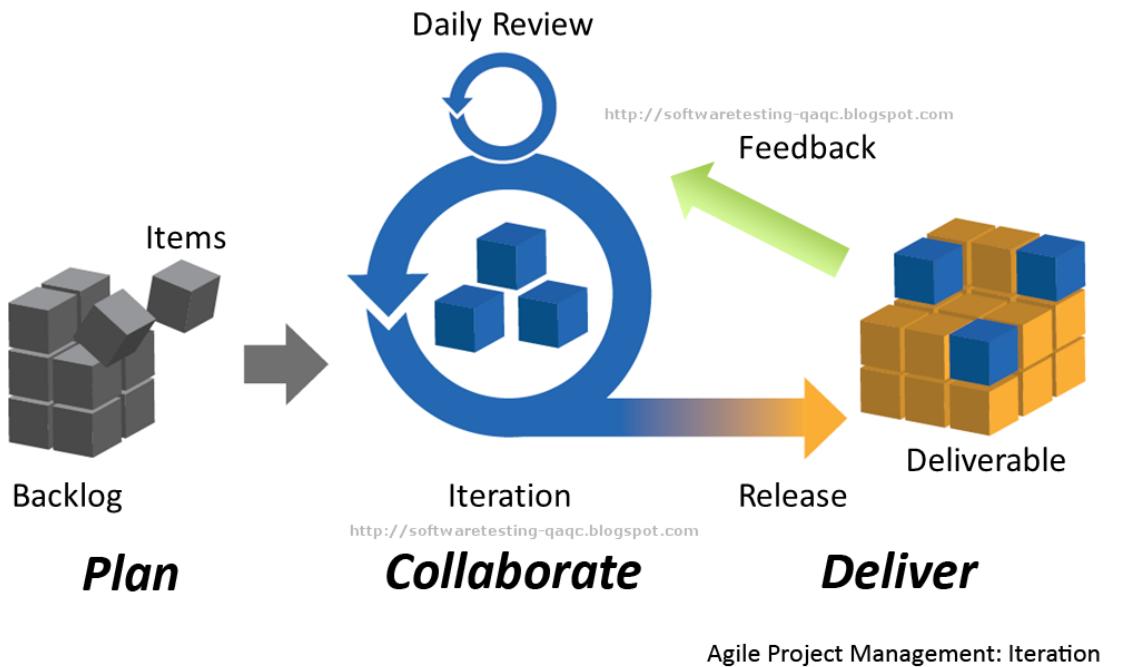
Bhushan has been a Certified Six Sigma Black Belt (American Society for Quality and Hewlett Packard) and possesses deep, but broad experience in solving complex problems, change management, and coaching and mentoring. Bhushan has a MS in Computer Science (1985) from New Mexico Tech and has worked at Hewlett-Packard and Nike in various roles. He was also a faculty member at the Oregon Institute of Technology, Software Engineering department, from 1985 to 1995 and is currently an Adjunct Faculty member.

*Copyright Gupta Consulting, LLC.*

## Introduction

Web application security is becoming increasingly important as the extent of breaches has increased. The software development community is also realizing that application security should be given due consideration throughout the development lifecycle and not be an afterthought. The Open Web Application Security Project (OWASP) has now ranked insecure design as the 4<sup>th</sup> in its Top-10 vulnerability list (OWASP 2021). In his recent book, Kohnfelder (Kohnfelder, 2022) has emphasized the importance of secure design along with threat modeling, mitigation, and design patterns.

Although not the only software development paradigm, the agile software development life cycle (SDLC) is widely used in the software industry. The basic element of the agile SDLC is a story and a collection of stories makes an epic. A story goes through a well-established transformation throughout the development as shown below:



**Fig 1: Lifecycle of a Story in the Agile SDLC**

Like any other attribute of software quality, such as usability, security can be and should be integrated at the earliest point possible in development. The following discussion will focus on how to integrate security in each iteration and ultimately deliver a minimally vulnerable web application.

## Scope

Multiple factors influence web application vulnerability. Some of the important ones are the operating system, the platform and the framework the application is built on, cryptography, the hosting environment (local vs. cloud), and the system configuration. They all introduce their own risks and some may have lower risk than the others. For example, the Django development framework is considered more secure than others.

The scope of this article is limited to the stories, the functional aspects of the web application, and not the platform the application is running on, the development stack, and how it is hosted. However, there are scenarios where a component of the hosting environment such as the memory management scheme

provided by the operating system, effects the application vulnerability and should be given due consideration in the design and development of the story.

## **Story Characterization**

Not all stories impact the application security at the same level, some are more susceptible to breach than the others. This section is an attempt to classify stories based upon their vulnerability.

### **High Vulnerability Stories**

These stories are directly associated with application security. For example, "As a user of the application, my personal data should be protected". This story explicitly dictates that the data should be protected at all levels. It is not specific to any particular data entity. Another story, "As a user, I should be the only entity to login to my account" is also a high vulnerability story. This story is also about data protection but specifically refers to a specific set of data. Stories that are more specific to security are more tangible and actionable as we design a secure solution.

### **Medium Vulnerability Stories**

In an online banking system, a typical story would be "As a user, I should be able to make a deposit". The typical process for this story will be login into your account, follow the deposit process and submit your transaction. A man in the middle attack may redirect the money to her/his account. This may cause the loss of money but the privacy of the account is not compromised for further data breach.

### **Low Vulnerability Stories**

"As a user I should be able to add an item to the cart" is a story that has a very little risk associated. There is a potential that a hacker may change the price of the item. But otherwise the action will be safe.

### **Threat Modeling**

Realizing that not all stories have the same level of threat, it is important to analyze the threat level of a story. Several threat modeling techniques exists (Shevchenko, 2019) and the one most applicable should be used to assess the risk. A popular techniques is STRIDE (an acronym for **S**poofing Identity, **T**empering of Data, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege) and has been widely published. It is possible to enumerate STRIDE for a set of stories by assigning a value to each element in STRIDE and establishing the overall threat as a numeric value. Appendix A provides examples of how to enumerate the threat for a few selected stories.

The threat modeling is helpful in evaluating the security risk associated with the story. In these times of "first to market", a good technique to prioritize your vulnerability helps you focus on the most vulnerable stories and thus minimize the threat.

## **Security Controls**

In the literature (Harris, 2012), the security has been defined by a triad, **C**onfidentiality, **I**ntegrity, and **A**ccessibility (CIA) and it is important to strike a balance between these elements. An extreme example will be, your application will be very secure if you lock it up. But, it will not serve any purpose because no one will be able to use the system. This section will describe the security controls specific to each element.

## **Confidentiality**

Confidentiality implies that the data is exposed in an authorized manner during its entire life time – at creation, in motion, and while stationary. If a story does not involve sensitive data, confidentiality is not important. For example – adding items to a shopping cart does not include any sensitive data. It may be tempting to treat every data element of a story “confidential” but this approach warrants securing the entire data. In this scenario, it will be necessary to develop a robust acceptance criteria, a secure design, coding, and verification, and finally an exhaustive validation of the story.

Confidentiality is preserved by access management which is enforced mainly by two security controls identification and authentication. NIST (National Institute of Standards and Technology) article 800-53, REV. 5, (NIST Task Force, 2020) recommends a set of controls that should be given due consideration while evaluating a story for confidentiality.

## **Integrity**

Integrity is defined as keeping data safe from unauthorized modification and removal. While we tend to think of integrity as a threat from the external elements, due consideration must also be given to malicious and disgruntled internal users. The integrity of the data may also be compromised by an unintended action of a normal user. Some of the necessary security controls are using secure socket layer to transmit data, encrypting data both in motion and while at rest, certificate management, and controlling authorization including privilege escalation.

## **Availability**

Is your web application available for use during the intended hours of operation? DOS (Denial of Service) vulnerability can be very damaging to your organization. You not only lose revenue, it also drives customers away causing further revenue impact. Security controls such as “no single point of failure” are more applicable at the systems level. This control requires an existence of an additional instant of the data that can be activated when the existing data instant is impacted by a DOS attack.

# **Testing Evolution in a Story Development**

This section will elaborate on how a story progresses through its agile life cycle and the actors who play important roles in each phase. RACI framework has been utilized to simplify the discussion and keep it organized. RACI is a responsibility assignment matrix and an acronym for different roles in the matrix as described below:

- R – Responsible: those who do the work to complete the task
- A – Accountable: the one who is ultimately answerable to complete the task
- C – Consulted: those whose advice should be sought to develop the solution
- I – Informed: those who are kept up to date about the progress of the task

From the RACI perspective, in an agile environment, the scrum master, product owner, development engineer, security engineer, and test engineer are the most active roles. In the RACI matrix, the informed could apply to numerous roles and therefore it is not critical to highlight unless it is absolutely necessary in the context of the story. A manager role has been used where it is more reflective than the scrum master. Otherwise, the scrum master can very likely be treated “accountable” for all activities.

## **Story Context**

To illustrate the essential elements of story development, we will consider the following story; it is generic in nature and essential for almost every commercially intended web application.

### Story: Sign Up

As a customer, I want to setup an account, so that I can login to the portal and make a purchase.

This story is a high impact story as it contains not just the personal identifiable information (PII), but also has login credentials, data in motion and finally at rest at the server side.

In a software development organization that follows best practices, this story will go through the following agile iteration steps:



**Fig 2: Typical Story Progress in an Agile Iteration**

While the first three steps are self-explanatory, in the web security community, static testing refers to any type of testing that is performed when the code is raw or compiled and is termed as SAST (Static Application Security Testing). Dynamic testing is when the code is in the executable state and is known as DAST (Dynamic Application Security Testing). The rest of the section details each iteration step with using the RACI matrix.

### Acceptance Criteria

Establishing an acceptance criteria is an important activity as it defines the boundaries for the story and sets the user expectations. In the context of security, it establishes which data elements should be protected and what security controls should be used to protect it. We will be using a list format for documenting the criteria and bear in mind, this list is not exhaustive by any means.

Criteria ID	Criteria Description
01	Collect only essential PII from the user
02	Hide sensitive data at submission (login ID, password)
03	Protect user data exposure when in motion
04	Protect user data exposure when stored
05	Protect user data from unauthorized changes when in motion and stored

**Table 1: List of Acceptance Criteria for Sign Up**

The RACI matrix to establish the acceptance criteria is:

R	A	C	I
Product Owner	Scrum Master	Security Engineer	Need based

Once established, the security team can add the security controls to the acceptance criteria. The updated list looks as follows:

ID	Description	Security Control
01	Collect only essential PII from the user	None/security policy
02	Hide sensitive data at submission (login ID, password)	Mask LoginID and Password at the data entry
03	Protect user data exposure when in motion	Encryption and Secure Transmission
04	Protect user data exposure when stored	Encryption

05	Protect user data from unauthorized changes when stored	Prevent privilege escalation Implement lowest level of privilege
----	---	---

**Table 2: List of Acceptance Criteria with Security Controls for Sign Up**

This sets up the security requirements for this story. The RACI matrix for this activity will be:

R	A	C	I
Security Engineer	Scrum Master	Product Owner	Need based

These security controls will provide guidance to the developers on how to design and develop the story and to the test engineers on how to validate it. The test case design team is ready to engage at this point.

## Design/Code Development

These two activities, although serial in nature, are performed by the development team. When it comes to security there are some best practices that the team should put to use. Some of the common best practices with examples are listed below.

Best Practice	Example
Defense in depth	Two-factor authentication
Fail secure	System lockup if an unsafe activity is suspected
Protect weakest code	
Principle of least privilege	Allow lowest level of privilege
Use of prepared statement	Building database queries

**Table 3: Best Practices for developing a Secure Story**

The responsibility matrix for this phase is:

R	A	C	I
Development Lead Engineer	Development Manager	Product Owner, Security Engineer	Need based

## Static Testing (SAST)

As described earlier, static testing is performed when the code is raw. The simplistic approach for SAST is to perform code reviews and compiling code with debug on. The typical vulnerabilities that can be detected from this method are buffer overflow, verification of encryption and secure transmission algorithms which would take an extra-ordinary amount of effort if performed by executing the code. A story may warrant a specific system configuration (also refers to as system hardening) which can be efficiently verified using the SAST approach. This approach is particularly valuable to verify individual stories such as the one being considered in this discussion.

The SAST, when performed manually, is time consuming and requires a subject matter expert to be effective. OWASP (OWASP, 2022) has provided a list of numerous open source tools that are capable of performing SAST. Commercial tools are also available with a limited history. While considering a tool, one should be aware of false negatives that will be harmful. The responsibility matrix for the SAST is:

R	A	C	I
Development Engineer (s)	Development Lead Engineer	Product Owner, Security Engineer	Need based

## Dynamic Testing (DAST)

Dynamic testing is the validation of a story while executing the code. The story may be independently verified or may need other functional stories. As an example,

As a customer, I should be able to pay by credit card or PayPal.

The pre-requisite for the story will be another story that will prepare the cart to be ready for checkout. DAST testing is a combination of manual and automated testing. There are open source tools such as ZAP or Burpsuite that will facilitate a significant part of this testing. The manual testing is absolutely necessary to achieve a high level of confidence. There also exist commercial tools that can be deployed for DAST. A systematic approach to evaluate an open source DAST tools has been described by Brian Myers (Myers, 2019). These tools look for prominent vulnerabilities and are not meant for catch all type testing.

The responsibility matrix for the DAST is:

R	A	C	I
Test Engineer(s)	Test Manager	Product Owner, Security Engineer, Development Engineer	Need based

The following figure brings these aspects together for an agile development practice.

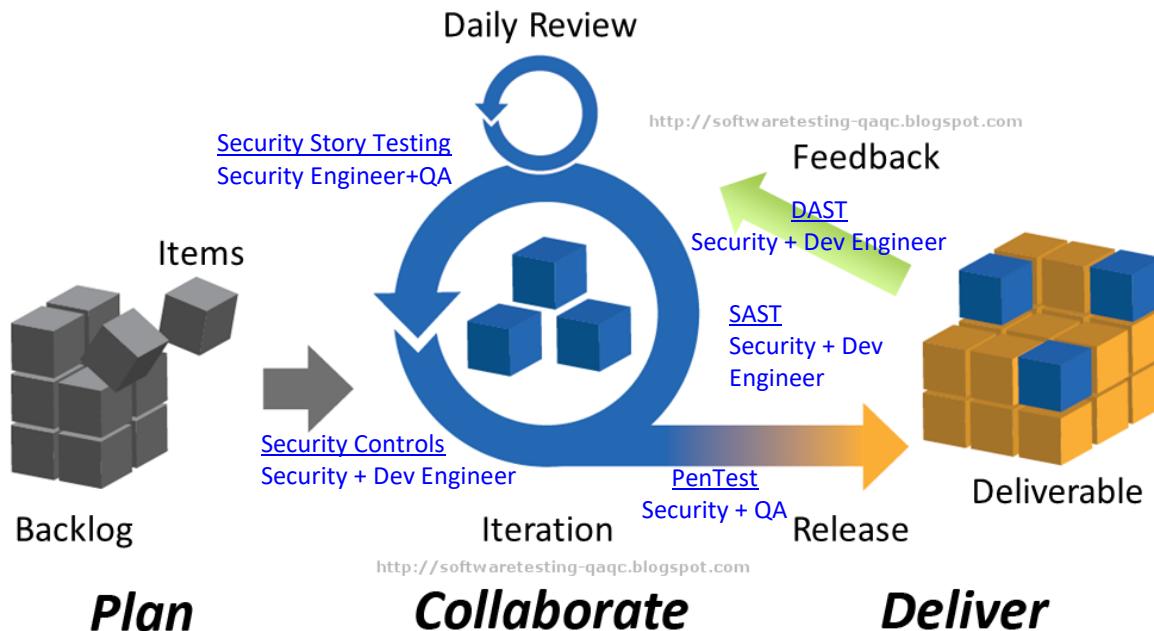


Figure 3: An Agile Iteration Cycle for a Story

## Test Planning

The test planning for any story can begin as soon as the story acceptance criteria has been finalized. This section describes the essential elements of the test planning.

### Test Case Development

The initial step will be to develop test cases, one or more, for each acceptance criterion. For example, test cases for the acceptance criteria 02 will be:

Test 01	Verify that the entry of login ID is not human readable
Test 02	Verify that the entry for the password is not human readable
Test 03	Verify that the login ID can be set to be human readable if desired
Test 04	Verify that the login password can be set to human readable if desired

Of course, the elaboration of the story at the design time should include the human readable requirement.

### Type of Testing

A serious consideration should be given to what type of testing will be most effective and efficient for a story. If a story requires a specific system configuration such as access control at multiple levels, static testing will be most effective. Another example will be SQL injection where a code review can verify that a “prepared statement” has been used to build the SQL query and the query has not been created from the user input.

### Efficient Use of Dynamic Testing

Dynamic testing is an effective way to gauge an overall understanding as it gives you a fairly detailed view of the vulnerabilities that exist in the system. A test plan should be enhanced to retest these vulnerabilities both using the dynamic and manual testing to eliminate any false negatives.

### Testing Priority

The testing priority for a story in an iteration is driven by the vulnerability of the story. Needless to say, the high vulnerability stories must be given the top priority. The prudent consideration while prioritizing stories for testing is: the developer’s confidence level. Any prior history of breach for a similar story either by the organization or the industry will also influence the testing priority of a story. Testing order for the stories in an iteration can be best achieved by a strong collaboration between the product owner, development engineer, and the test engineer. Here is the responsibility matrix:

R	A	C	I
Test Engineer(s)	Test Manager	Product Owner, Security Engineer, Development Engineer	Need based

## The Team

Security testing function has evolved over the past few years given the damaging impact of a breach to an organization. With the evolution of different levels of testing, the organization of test teams has also changed over time. Described below, are the test teams that help achieve a high level of confidence in building a secure product.

### **Red Team**

The concept of a red team was the first evolution of a security test team. The team attacks the application from the hacker perspective and its main objective is to breach the system in a realistic way. The team members are skilled in reconnaissance and have a deep knowledge of test tools and techniques. They are normally referred to as attackers.

### **Blue Team**

The second evolution of test team organization was a “blue team” and includes security personnel (defenders) who work with the development team (builders). The developers know the code and joined with the expertise from the security engineer, this team can effectively explore vulnerabilities.

### **Purple Team**

Purple team is a relatively new trend in security testing and it's a cooperation between the blue and the red team to find vulnerabilities.

There have been thoughts in the industry about other teams such as yellow and green teams (Miessler, 2021). The functional aspects of these teams are not yet well established.

## **Conclusion**

Multiple factors influence web application vulnerability and the development team has limited control on these factors. The most important of these factors, is secure development which requires an understanding of security controls and their implementation at the proper stages of development in the SDLC. Developing a secure web application is a team effort and cannot be left entirely up to a security engineer or security team. Actors such as product owner and development engineer along with the quality assurance engineer play an important role at various stages. Testing methods and tools are constantly evolving and organizations such as OWASP are providing guidance that, when followed, will yield web applications with a high level of security confidence.

## References

- OWASP, A04:2021 – Insecure Design, [https://owasp.org/Top10/A04\\_2021-Insecure\\_Design/](https://owasp.org/Top10/A04_2021-Insecure_Design/)
- Kohnfelder, Loren, 2022, Designing Secure Software, no starch press, San Francisco
- Shevchenko, Nataliya, 2019, Evaluating Threat-Modeling Methods for Cyber-Physical Systems, <https://insights.sei.cmu.edu/authors/nataliya-shevchenko/>
- Harris, Shon, 2013, CISSP, McGraw Hill Education, New York, USA
- NIST Task Force, 2020, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf>
- OWASP, Dave Witchers et al, [https://owasp.org/www-community/Source\\_Code\\_Analysis\\_Tools](https://owasp.org/www-community/Source_Code_Analysis_Tools)
- Myers, Brian, Starting a Security Program on a Shoe String, Pacific Northwest Software Quality Conference, Portland, OR, October 2019
- Miessler, Daniel. 2021, The Difference between Red, Blue, and Purple Teams, <https://danielmiessler.com/study/red-blue-purple-teams/>

## Appendix A – Enumerating Threat with STRIDE Example

Story ID/ Vulnerability	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privileges	STRIDE Score
Browse to Web Site	No	No	No	No	YES	No	1
Create user Profile, login, security questions	Yes	No (data stationary)	Yes	Yes – social engineering	No	No	2
Submit form	Yes	Yes	Yes	Yes – social engineering	Yes	Yes	5
Receive Confirmation	No	Yes	Yes	Yes (MitM)	No	No	3

# TestZeus: Automate the lightning and thunder of Salesforce Testing

**Robin Gupta**

Robin.gupta@provartesting.com

## Abstract

Software test automation for a SAAS (Software As a Service) is inherently complicated at various levels. While the development team does functional changes, the platform also evolves with time and poses exponential challenges for software testing teams with not only building out automation scripts, but also maintaining them over a period of time. Salesforce is a CRM (Client Relationship Management) SAAS offering and is a classic example of posing the aforementioned challenges to software development testing teams. Salesforce releases change the HTML, JavaScript, and CSS that make up the user interface. These changes break traditional automated tests and lead the testing teams down the rework spiral. This paper demonstrates the implementation of TestZeus an open-source framework, which uses the publicly available APIs to bridge the gap between test automation scripts and Salesforce platform, by reducing the reliance on the standard locator mechanisms.

## Biography

*Robin is a multi-talented engineering manager with close to 15 years of metrics driven approach to challenging assignments. He is successful at managing all aspects of delivery vis a vis Engineering and QA. Robin has a polymorphic leadership style with fluency in BFSI, EdTech, Retail, Healthcare and E-commerce domains. He has hands on experience working with startups and enterprises across geographies and time zones. Besides work he is a mentor at ADPLIST and Plato, and contributes to opensource via Selenium and TestZeus.*

Copyright Robin Gupta 14June2022

## 1 Introduction

Software testing has been an effective approach of ensuring the quality of web applications. In practice, software testers construct manual and automation scripts for testing out the web application. These automation scripts are created using open-source tools like Selenium, Playwright or Puppeteer etc. and work on the UI (User Interface) interaction layer to simulate end user behavior. These UI (User Interface) test automation tools identify the web elements (dropdowns, links, input boxes etc.) on the application under test, using locator strategies like identifying the xpath (XML Path Language), CSS (Cascading Style Sheets) or Ids. However, even with the advent of these tools, the test automation for SAAS (Software as A Service) applications like Salesforce remains a bane for the software testers due to the ever-changing nature of front-end code, as per platform level upgrades. Quarterly and annual service upgrades cause these web element locators to break and the testing team to focus on maintenance for the automation scripts. Additionally, waiting for the page to load in a multi-tier web architecture can add extra layers of failure for a test automation framework. Therefore, in a large test automation base, the maintenance of these frameworks can exceed the release window of the application itself, rendering the coverage useless.

In order to solve this trident problem of accelerating automation development, stabilizing the locator strategy and reducing the maintenance effort, we explored the dynamic creation of locators via the TestZeus framework using Salesforce's publicly available UI API and observed at least 30% gains in execution times, along with 60% reduction in test creation and maintenance effort.

## 2 Salesforce UI API

Salesforce UI API provides data and metadata around layouts, field-level security, fields and types in a well formatted JSON response. This is the same API which is consumed by the Salesforce front end system (Lightning Design System) to render the UI at run time. Along with the layout and design information about the front end, UI API also provides record level information. For example, if we query the UI API for an ACCOUNT object (Example: RecordId as 001R0000003GeJ1IAK):

GET /ui-api/record-ui/001R0000003GeJ1IAK

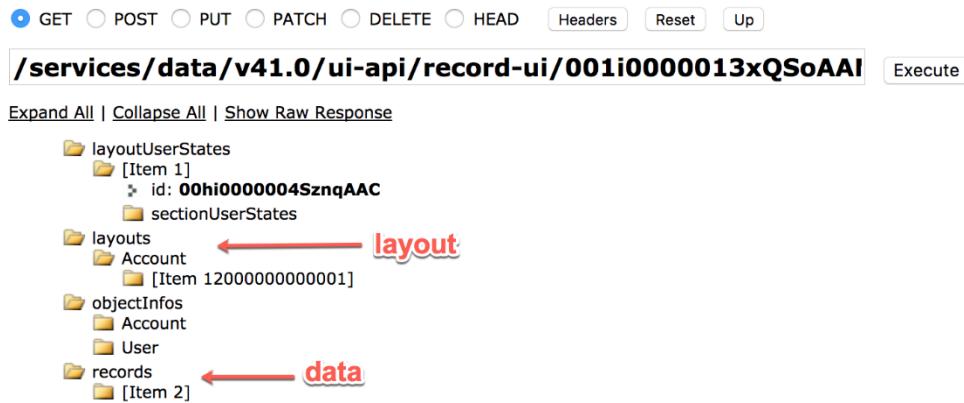
We will get the below crucial information in a neat JSON response:

1. Fields on the ACCOUNT object like First Name, Last Name, Phone, Address etc.
2. Placement and layout related information of the aforementioned fields such as which field lies on which section of the page layout.
3. Field types of the various fields on the page
4. Exact values of the fields for the record id passed in the API call. For example: First Name as "Robin", Last Name as "Gupta", Address as "Bangalore" etc.

There are many use cases where customers build custom mobile or web apps that use Salesforce data and metadata—sometimes they want to just replicate all or partial Salesforce UI inside those apps. But Salesforce UIs are complex. The correct UI for any given user depends on various rules, permissions, and page layout configurations. All of these dependencies can be easily changed by an admin (or anyone with appropriate permissions) at any time to better fit business needs. It used to be very hard to build dynamic UIs and display them in custom apps that take care of all the rules and updates whenever changes are made.

That's where the UI API comes in. Its main purpose is to make it simple to replicate all or part of a Salesforce UI according to rules and permissions inside a given custom app. Since UIs also allow the users to edit, update, or delete the data, the UI API further provides endpoints to even perform CRUD operations corresponding to each layout with ease. This data is intended for developers to build a dynamic UI for native and mobile apps. It can also be consumed for designing and implementing custom applications which interact with Salesforce databases.

Examples of the response from UI API are listed below in Diagram A and Diagram B.



(Diagram A).

The above diagram depicts the high-level structure of the response received from UI API for a sample record. Notably, it contains the layout information and the data points for the record.

```

"lastModifiedBy" : {
    "displayValue" : "Deanna Li",
    "value" : {
        "apiName" : "User",
        "fields" : {
            "Id" : {
                "displayValue" : "005R0000000IEDsIAO",
                "value" : "005R0000000IEDsIAO"
            },
            "Name" : {
                "displayValue" : "Deanna Li",
                "value" : "Deanna Li"
            }
        },
        "id" : "005R0000000IEDsIAO",
        "recordTypeInfo" : null
    }
}
  
```

(Diagram B).

The above diagram shows the exact displayed values for the record as received from a sample UI API response.

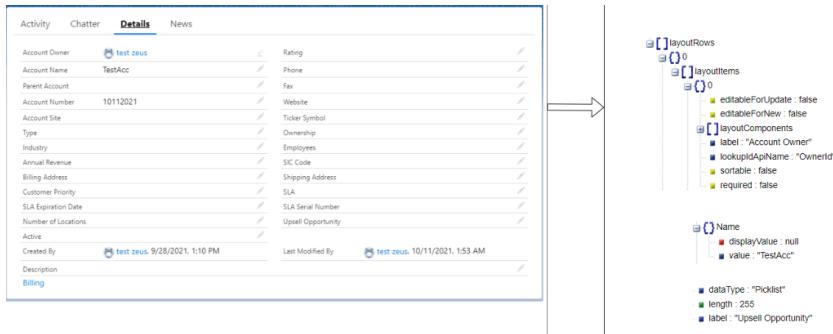
### 3 TestZeus

TestZeus is an open-source test automation framework built specifically for the Salesforce platform.

Based on the information from Salesforce's publicly available UI API, TestZeus creates the locators automatically at run time to exponentially cut down the build and maintenance times for test automation scripts. In addition to the bespoke locator strategy, TestZeus also supports waiting mechanisms built specifically for Salesforce and custom components to handle API testing, Email integrations and Page object implementation.

### 3.1 Auto-Locators

The core of TestZeus works on automatically creating web element locators based on the response received from UI API for a certain object or record. TestZeus parses the UI API and processes the JSON response for labels, datatypes and sections to create the contextual actions and locators for the UI elements on the fly as depicted in Diagram C below:



(Diagram C).

TestZeus scrapes the sections, labels and datatypes from the UI API to create the stable locators and based on the datatypes, contextually provides actions to interact with the UI.

Here is the flow of events in programmatic fashion:

1. Hit the UI API via a connected app
2. Get the list of sections for the object
3. Recursively get the field types and labels for all the sections on the page
4. Create a data structure for the user interactions based on the test flow

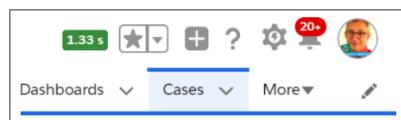
As the locators are created at run time based on the above data, user need not create page objects explicitly for the test case.

These locators are passed onto standard Selenium based web driver to accurately perform the user interaction based on the test flow.

Also, the UI API can be utilized for CRUD (Create, Retrieve, Update and Delete) operations, therefore testers can not only view and assert the data, they can modify the field values using TestZeus as well.

### 3.2 Experienced Page Time

Experienced Page Time (EPT) is a performance metric Salesforce uses in Lightning to measure page load time. EPT measures how long it takes for a page to load into a state that a user can meaningfully interact with. The EPT is measured as the time from the page start to when no more activity occurs for at least two frames (~33 milliseconds). The two extra frames help to avoid false positives due to asynchronous calls. These calls include any XHR activity, any storage activity, or any user interaction or client-side work of any kind in the main JavaScript thread. Example EPT of 1.33 seconds is shown in the Diagram D below:



(Diagram D).

### 3.3 Intelligent Wait Mechanism

In addition to utilizing the UI API for dynamic creation of locators, TestZeus also utilizes the EPT (Experience Page Time) as described above to intelligently wait for the web page to load. Basically, TestZeus continuously polls the EPT value for 20 seconds and waits for it to be a value greater than zero, meaning that the web page has loaded completely, and is interact-able as published by Salesforce's front-end system. This has been implemented using the value retrieved from console as part of the browser interaction.

### 3.4 Example Usage

TestZeus can be utilized as the boiler plate framework for starting test automation or can be imported as a MAVEN dependency in an existing test automation framework. Here's a sample test case built using TestZeus, which logs into a Salesforce instance, and creates a new account without having to write explicit page objects or web element locators:

```
//Create a new instance of the SFPageBase class
SFPageBase pb = new SFPageBase(driver);

// Use methods from TestZeus as below for Navigation to login page
pb.openHomepage("https://testzeus2-dev-ed.my.salesforce.com");
pb.maximize();

// Or Use the webdriver implementations: Example for Submitting user id,
// password and logging in
driver.findElement(By.id("username")).sendKeys("Ulusername@gmail.com");
driver.findElement(By.id("password")).sendKeys("Ulpassword");
WebElement loginbutton = driver.findElement(By.id("Login"));
pb.safeClick(loginbutton);
pb.appLauncher("Account");

WebElement newbutton = driver.findElement(By.xpath("//a[@title='New']"));
pb.safeClick(newbutton);

// We fetch all the labels and datatype from UI API here for a certain record
String recordid = "0015g00000S9lfUAAR";
pb.uiApiParser(recordid);
```

```
// Form data can be passed directly on the new sObject creation screen
pb.formValueFiller("Account Name", "AccountCreatedOn : " + pb.getCurrentTimeStamp());
WebElement savebutton = driver.findElement(By.xpath("//button[@name='SaveEdit']"));
pb.safeClick(savebutton);
```

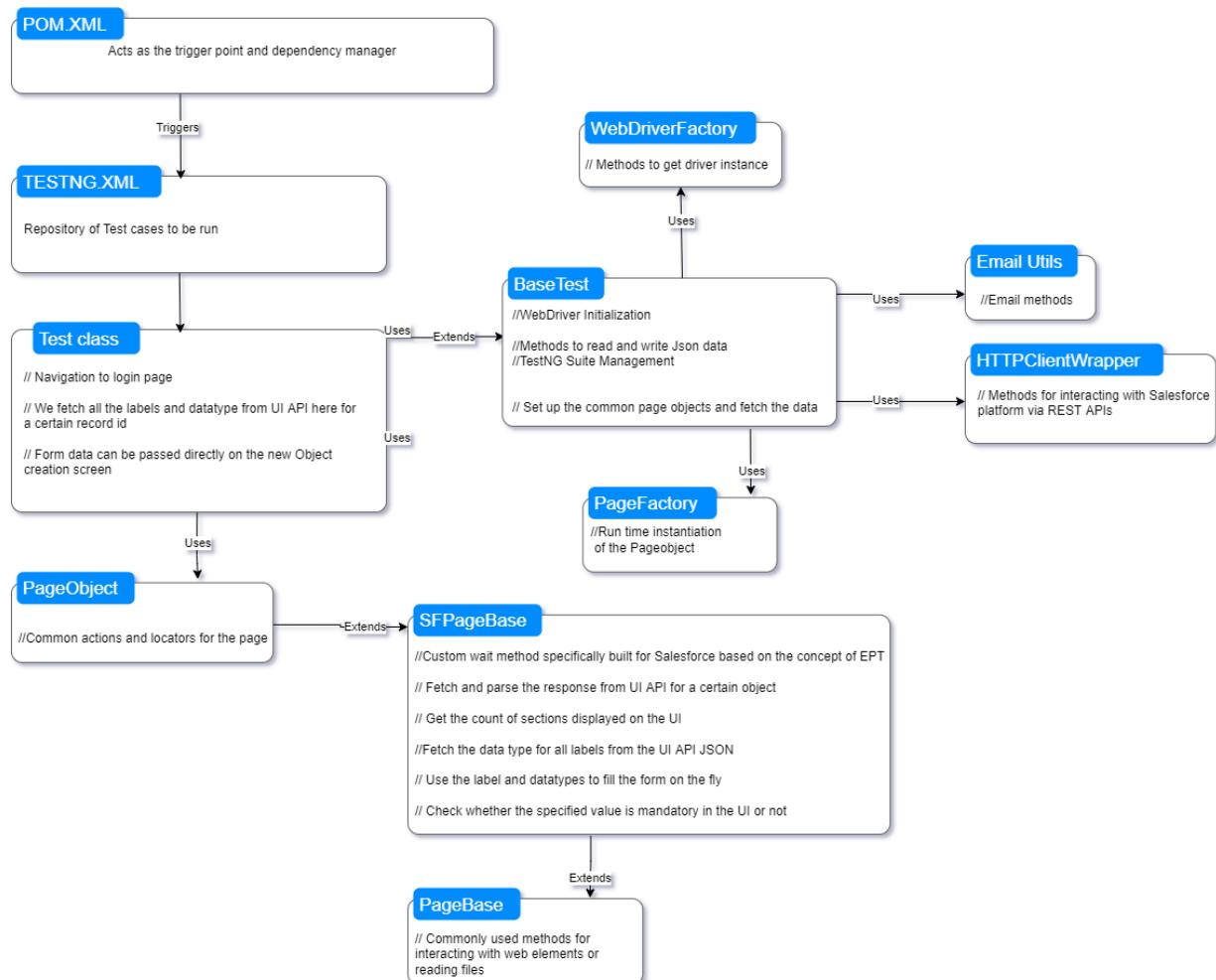
### 3.5 Results

TestZeus brings value and results at three levels as below:

1. Accelerates the Test automation effort by removing the need for writing explicit locators
2. Adding stability via automatically waiting for the page to load
3. Reducing the maintenance effort for feature and platform level changes.

### 3.6 Architecture

Diagram E below shows the high-level architecture of the TestZeus framework:



## 4 Challenges and Limitations

As with any technological advancement, TestZeus is also limited by its coverage and Salesforce's publicly available APIs. Specifically, auto-locator strategy was briefly touched in the Section 3.1 above, it is nowhere near the point of hundred percent test coverage for the whole Salesforce ecosystem encompassing the Lightning Design System, Lightning Web Components, Standard and Custom objects, Installed packages and Visualforce pages. A paradigm shift in application testing focused on more granular validations combined with more simplified test case creation may result in a convergent satisfaction of this limitation.

## 5 Conclusion

In this paper we present a novel approach to test automation for Salesforce ecosystem – called TestZeus. TestZeus uses an API first approach to generate web element locators, which are dynamic in nature and change automatically based on the page layout changes. Internally, TestZeus also intelligently waits for web pages to load and consumes Selenium Webdriver APIs to drive the browser and mimic user behavior. There were numerous other observations and feedback received from the technical implementation. We believe there is scope to considerably improve the coverage of the Salesforce ecosystem by modelling the out of the box objects and features offered by Salesforce. The concept and the implementation of TestZeus were very well received by the software testing and open-source community. Current efforts are in deploying TestZeus in other scenarios within multiple projects and improve the tool's capability particularly around various kind of web systems built using Salesforce.

The same approach of using metadata level APIs can be utilized in other SAAS applications like Oracle, Workday, JIRA and ServiceNow as well.

## References

Selenium Webdriver, <http://www.seleniumhq.org/projects/webdriver/> (Online, retrieved 15 June 2022).

Salesforce UI API, [https://developer.salesforce.com/docs/atlas.en-us.uiapi.meta/uiapi/ui\\_api\\_get\\_started.htm](https://developer.salesforce.com/docs/atlas.en-us.uiapi.meta/uiapi/ui_api_get_started.htm) (Online, retrieved 15 June 2022).

Salesforce EPT, [https://help.salesforce.com/s/articleView?id=sf.technical\\_requirements\\_ept.htm&type=5](https://help.salesforce.com/s/articleView?id=sf.technical_requirements_ept.htm&type=5) (Online, retrieved 15 June 2022).

TestZeus, <http://testzeus.com/> (Online, retrieved 15 June 2022).

Apache Maven, <https://maven.apache.org/> (Online, retrieved 15 June 2022).

# Testing IoT Systems Using V&V, Security Hacks, & Data Analytics

**Jon D Hagar**  
**Grand Software Testing, LLC**  
**Hot Sulphur Springs, Co, USA**

[jon.d.hagar@gmail.com](mailto:jon.d.hagar@gmail.com)

## Abstract

The Internet of Things (IoT) systems have become a hot growth area in tech, which in part will drive the modern economic world. Engineers are busy creating a connection to the internet for nearly every electronic device, meaning that many “things” or devices that were never before connected to the internet or other devices or networks, will now be, and that presents a “boundary issue” of where to stop testing for testers as well as management. Many IoT managers and stakeholders may wish to limit the testing of these devices or their connections to control costs by only covering the IoT device software. Their rationale being the “thing” or “system” may be well known, and only the IoT software is “new”. This limited testing may be acceptable for a small percentage of products, but lower, system-level integrated testing may leave other IoT fielded projects with severe problems—especially with regard to security. This paper considers several important IoT system-level test activities and the technical skills to support the maturing world of IoT system testing.

In many cases, the IoT device, software, edge interfaces, and finally, the cloud elements of the IoT system will need to be tested. Simplistic, manual testing of IoT needs to evolve into complete Verification and Validation (V&V) during the system lifecycle while often including levels of independence in the test team organization. Testing activities such as: analysis, reviews, inspections, modeling, structural tests, functional tests, etc., will need to be appropriately budgeted and then allocated and detailed in testing plans and strategies in order to address the risks of IoT systems testing beyond just the IoT device itself.

Traditional manual testing will have limited application in this new domain compared to test automation or using Artificial Intelligence (AI) with data analytics (“a method of logical analysis,” Merriam-Webster) to drive the IoT evaluations. To support these new levels of testing, testers from more traditional software will benefit from learning and practicing advanced engineering skills. Testers will have advantages when building on traditional software test concepts, as well as expanding their use of newer software, hardware, and system engineering concepts. Test practitioners that can master advanced testing skills will be in demand and may find the IoT world very challenging, exciting and fun.

## Biography

*Jon Hagar is a systems-software tester consultant supporting software product integrity, verification, and validation, specializing in embedded, IoT, and mobile software systems at Grand Software Testing in Colorado. Jon has worked in software engineering, particularly testing, for over 40 years. He has managed test teams and built test labs using test automation. He teaches classes at the professional and college level. Jon regularly publishes with over 100 presentations/papers, best paper STARWEST 2011, a contributor to three books, and author of three books: “Software Test Attacks to Break Mobile and Embedded Devices” (CRC Press), a children’s book, and “IoT Testing” (due out in Sept 2022). Jon is an IEEE/ISO editor and author on various international testing standards. Finally, Jon is a skier, biker, builder of a one-of-a-kind passive solar, thermal mass home, and heavy equipment operator. Projects he has supported include: control systems (avionics and auto), mobile-smart devices, spacecraft, and ground systems information technology (IT), as well as consulting on attack-based testing on smartphones and IoT systems.*

## 1. Introduction to IoT and challenges testers now face

IoT puts computer processing and communications in every imaginable device, including things that were not originally electronic or connected. Figure 1 presents a partial listing of common IoT systems, including consumer, industrial, and items in the middle between these two. This figure is not a complete classification of all IoT “things” but presents the large scope of possible IoT systems.

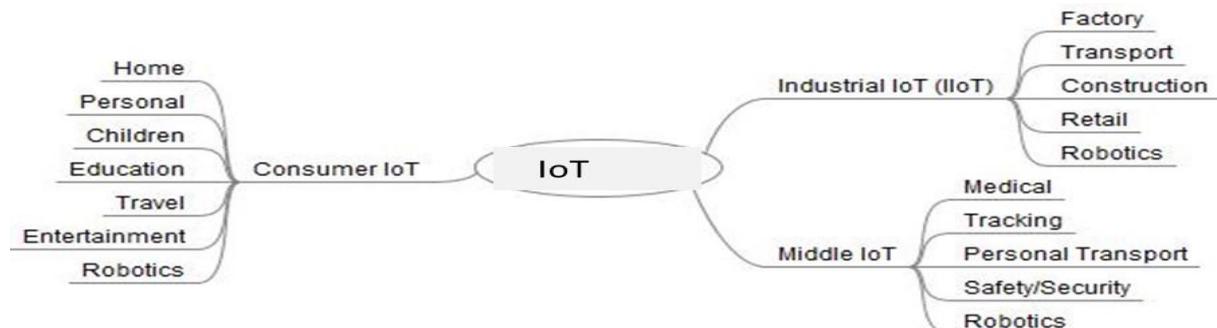


Figure 1: IoT will be in everything and everywhere [1]

Questions arise with IoT, such as: “How and to what level do projects test such devices, such systems or even collections of IoT systems?” Many projects and testers are used to simply testing the software in isolation on a PC or web page, leaving the complex system space to users and stakeholders. This paper provides pointers to classic testing of the software device itself. However, it is my position that just testing the device in isolation is only a beginning. The real problem for IoT systems will be going beyond the device to the higher levels and interfaces. In addition to classic testing, a key point for IoT systems will be the need for system and software level testing and assessments, which some teams may wish to minimize.

When there is limited system-level testing, I advise development to disclose the level and type of testing in the product information. For example, disclose that the IoT device was tested at the functional level of assessment in defined limited environments. This disclosure will allow later customers and stakeholders of the IoT device to incorporate the IoT device into their extensive system and to understand when they may need to be conducting additional system-level Verification and Validation (V&V [2,3]) with testing to ensure the whole system is viable.

### **V&V/IV&V, testers need to understand these terms.**

*Verification is assessing a system to ensure it meets requirements and design specifications.* *Validation assesses a system to assure it meets stakeholder needs beyond the published specification [3].* While verification can use checking against things like requirements during testing, validation is more open and uses the understanding of the engineers doing the work. The use of independent test, V&V, and/or IV&V organizations have been used in many areas of industry [3,4] for years when system level assessments are warranted. Using this approach aligns with much of the historic IT world (companies, large organizations, and governments), where large and complex software, hardware, and systems are subject to integrated systems testing by the owning stakeholders or their representatives before deployment to users. For example, many companies have IT departments evaluating or testing products before integrating them into their corporate IT systems or on their networks. Testing beyond an individual IoT device should be considered because many issues and faults will only be detected at an integrated system level. Industrial, government, and commercial stakeholders will understand such limitations because they have dealt with integrating IT systems for years. Users, even testers, used to simple “plug and play” may not understand the importance of testing a fully integrated system nor the limitations or consequences of testing just the device.

Such users may be disappointed in an IoT device, stop using it (so much for product viability), or worse many users will publicly complain about the system on social media (bad product advertising). While this does not sound like a significant problem at first, complaint reviews can kill a product with a few bad posts from unhappy users in these days of social media. IoT companies wishing to stay competitive should be aware of defining where and how to finish the assessment of IoT device qualities. Testers may want to recommend and advocate when more extensive system-level testing

may be needed to satisfy customers. Stakeholders building complex IoT-based systems will need to consider this information when expanding testing beyond the IT department into their IoT world. This may lead those procuring IoT systems and doing integration toward the concept of Independent Verification and Validation (IV&V [3])

I assume the IoT development team will use some common testing practices [1,7,8,9,10,11,12,13] on the device itself and beyond the device to a limited extent. However, many IoT devices will become part of a system of tens (in a smart home) or even thousands of connected devices (in a smart city). In these cases, the idea of an independent test team beyond the developer testers may come into play. A classic concept of IV&V [3], or an independent test organization [4], has a history in the tech industry. IV&V has a long history in critical government systems.

This paper continues next considering areas of IoT system V&V/IV&V assessment. A broader picture for IoT system testing/V&V is presented in [1]. Next, the impacts of AI and data analysis as part of testing are presented. There will be many opportunities for testers and companies in the IoT domain for these independent test areas.

The present economic society of Technology-Capitalism [5] is driving companies, IoT teams, and testers to add activities and new skills. This paper includes some of the disciplines of test technology used by skilled engineers. Additionally, the paper briefly considers the impacts on generalized IoT stakeholders. Testers will particularly need to think about the impacts of AI, DA, and software test architectures (STA [6]), as well as classic testing concepts. Even IoT advocates have a hard time envisioning the whole nature, scope and expanse of IoT system impact on stakeholders.

Note: STAs are defined here as the hardware, software, test, and system elements that support test execution. They are part of test plans and include test strategy, very high-level test design approaches, models, and practices. They are analogous to the system, software, and hardware architectures yet are not universally accepted by the industry.

## 2. IoT Testing and IV&V Assessment Beyond the Basic Device

I expect that most testers will understand the importance of test planning and having a comprehensive strategy. Planning is often considered an essential first step considering test cost, schedule, risks, staffing, environments, techniques, and documentation. More information can be found in [7,8,9,10,11,12,13]. As a start, most testers plan to target the IoT device itself. However, I recommend due consideration of aspects found in Figure 2. This figure shows a smart home that contains IoT devices. The devices are connected to an edge system, which could be a PC with a router, a cell phone or another edge device. Edge devices provide the first line of advanced processing for IoT, particularly in AI, DA, and early user feedback. IoT devices are often small and do not perform much advanced data processing. The edge devices are the first point where such processing can actually be performed, and hence why *system IoT testing is important*. After edge processing, the IoT data and information will be passed into the cloud for additional processing and analytics. Therefore, the team must ask an early question in test planning: where does their testing start and end? It is possible that just testing the device will leave issues in the edge and cloud undetected resulting in unhappy stakeholders which may eliminate product viability or worse, violate security. In the IoT test planning, the team should consider a test strategy where they weigh concepts such as: test automation [14,15], environments, models, tools, risk-driven testing, and what test approaches or techniques should be used. The team should ask what the needed product qualities are including: IoT security, performance, safety, reliability, etc. [16]. Will any project be successful if the team restricts itself to just the device's structural and functional tests and does not address these other topics? The elements of Figure 2, test plans/strategies, and concepts form a sample IoT STA. An STA can assist in answering some of these questions.

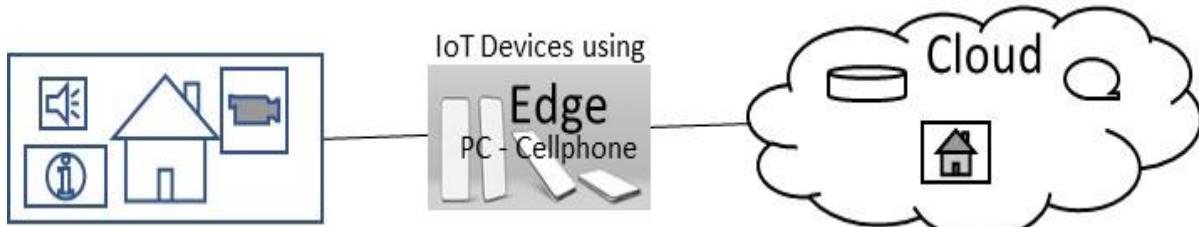


Figure 2: The IoT System from the device to the edge to the cloud [1]

There is no one answer for IoT test planning and STAs. Risks, costs and schedules must also be considered. Stakeholder needs must be factored into the test planning. The disclosure that only the device has been tested under set plans and strategies may force customers or users to decide what other test activities may be advised. For example, a medical device used at home for only personal information may not need IV&V. However, a similar medical device that feeds data to an AI or doctor for monitoring a critical condition (across networks) may need IV&V as well as regulatory approval (e.g., FDA and [1]). Of course, the cost and planning for each configuration would be quite different. Unfortunately, customers may not fully understand the IoT system risks and may not be in a position to make informed decisions.

This kind of IoT situation should lead one to consider system-level IV&V and V&V, as well as independent, from the development team, third-party testing in IoT planning.

### 3. IV&V/V&V assessment at system and software levels

A key point here is that for IoT evaluation and quality, test activities need to extend beyond the IoT device to the internet systems or other networks to which the device is connected. There will need to be industry and societal considerations of ownership of this extension. In their literature, individual device manufacturers may rightly claim that their quality ownership responsibility *stops* at the connection to the edge or, more extensively, to the internet/network. This leaves an open question of where this responsibility falls and an opportunity for test organizations to assume ownership. For example, I have a semi-smart house that generates much of its own power and heat. We see problems with the connections inside the house, between the solar panels, and the outside power grid. The solar panel manufacturer was little help, leaving the ownership to us. The problem lies with the connection to the cellular network for the panels to “call home” to report energy generation. And that is where the vendor had not tested since they did not plan, have an STA, or test for that possibility. We have figured it out, but we are technologists. Most home customers will not want this ownership. The problem becomes more prominent when we look at examples such as medical devices or IoT systems to control smart cities [17]. For these more extensive areas, we see the rise of “test cities,” [1, 17] V&V, and IV&V organizations to conduct the quality assessment of the systems or system-of-systems. Organizations, companies and testers will need to step into these gaps doing IV&V/V&V. An example allocation of activities is shown in Figure 3.

The names of these activities are defined in texts such as [1,3,7,8,9,10,11,12,13]. Readers unfamiliar with them should use these references to gain a better understanding. Also, the allocation can be done differently based on context. An example is the IoT world that Aerospace has moved into, where pretty much all of these are practiced. When consulting with large Aerospace companies, I was surprised to see very familiar test plans, system architectures, computers, techniques, and tools between the organizations that had minimal communication but were solving common IV&V/V&V allocation problems. This commonality tells me that while Figure 3 is an example of allocation to IV&V, for IoT industries looking to solve system-level quality test problems, the commonality is a good starting point. In critical, large, and/or complex IoT systems, many of the activities in Figure 3 should be considered.

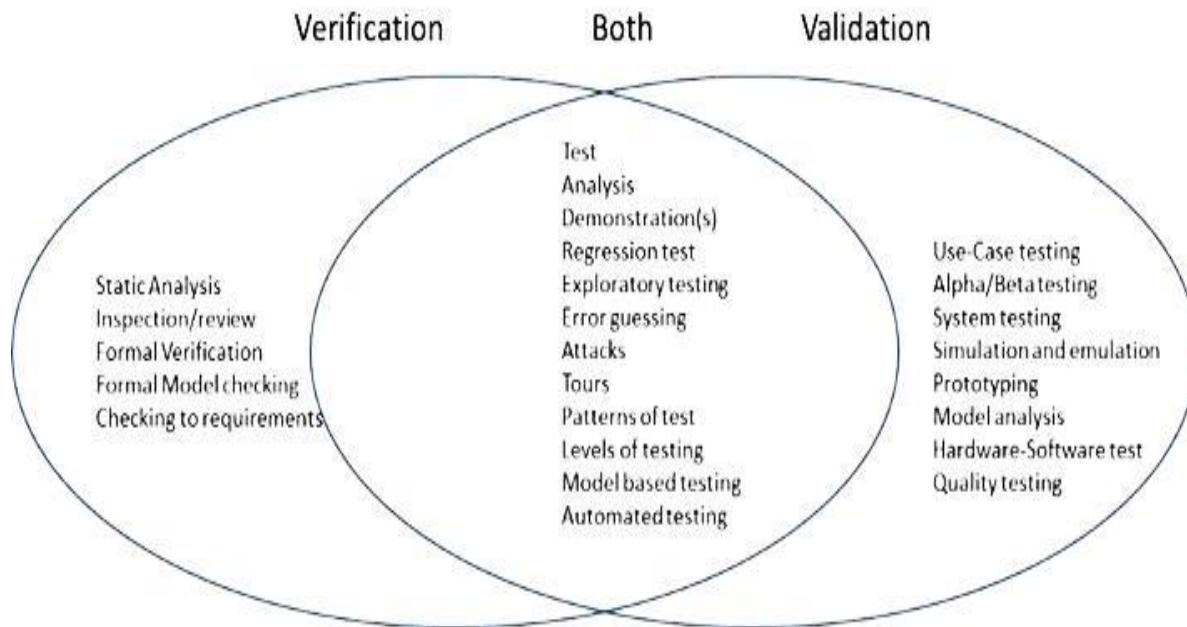


Figure 3 – Example activities of V&V/IV&V [1]

For IoT, the key V&V activities from Figure 3 are: static analysis and reviews of the code, testing, analysis using modeling, exploratory testing, attacks to break the system, model-based testing with automation, quality(s) assessment with security testing, simulation, and system assessment with hardware in the loop driven by use-cases or acceptance driven testing. Verification teams use these activities to focus on functional and structural assessment. For IoT validation, teams are assessing the system using V&V activities addressing the stakeholder needs. A skills list for these V&V activities includes [19,20,21] and those found in the following list:

- Hardware engineering and testing/assessment
- Software engineering and testing/assessment
- Systems engineering and testing/assessment
- Operations and deployment of particular AI and data analytics
- Identification of potential project/test risks and impacts
- Documentation and technical communication of testing/assessment
- Working knowledge of programming as related to test automation and analysis
- V&V processes of [3]

## 4. Security testing and assessment will be a top task

Having a security plan, outlining risks and mitigation plans is an excellent start for any project. A variety of articles have cited that IoT systems are ripe for hacking. While there are many important qualities to plan for and test, currently, security testing should be near the top of the list. Figure 4 gives a starting point for security test plan assessments and designs.

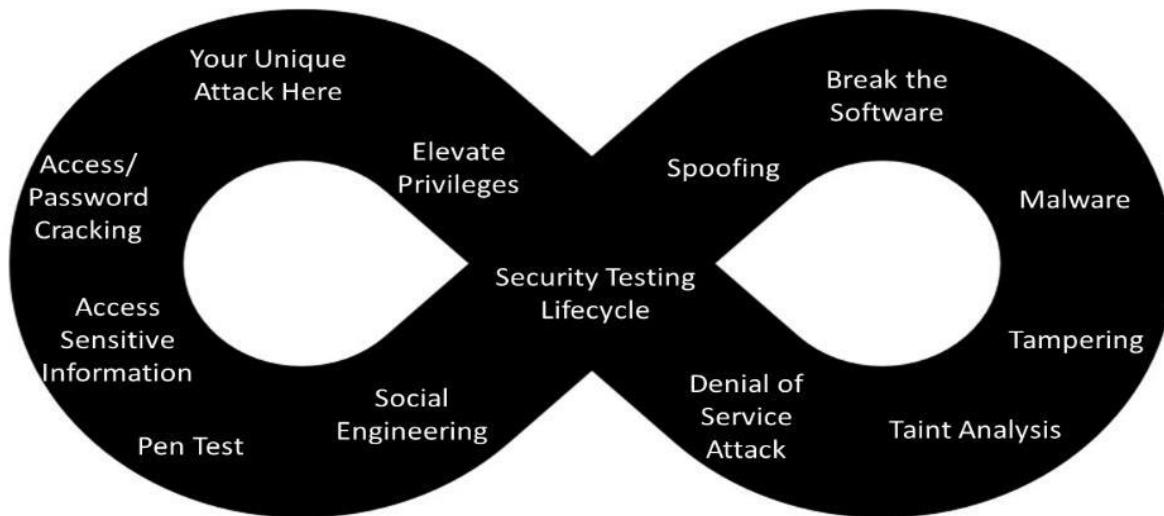


Fig 4 The security infinite test diagram – you are never done with security [1] testing

The tests in Figure 4 are common security test approaches [1,18]. Security testing is a specialized area that IoT teams will need to plan for ahead of time. The bad guy hacker depends on development teams or IV&V teams not doing a good job on security testing. Hearing your company's name broadcast on the nightly national news is certainly not good for any business. Given the "internet" connection part of IoT, the door is always open for a hacker, and much more so than older non-connected, embedded cyber-physical systems. The security test planning starts at the IoT device, but as shown in Figure 2, testing includes the edge and cloud. The use of AI and data analytics will help the security testing, but all of these levels of testing and analysis will need to work together to secure IoT. From here, I define the topics of Figure 4 beginning where the hackers might.

#### 4.1 Social Engineering [1]

Testers need to think and act like the "bad guys" in social engineering. They ask: "How can we trick the users and system?" Testers must think the way the hackers do, but before the bad guys can cause problems. An excellent model to follow is the cyber kill-chain model [1]. *I remind you to be legal and ethical* (white hat). Also, readers will note that in the IoT security model of Figure 4, blanks for "your unique attack" may need to be customized to your local threats and risks found during social engineering.

#### 4.2 Penetration (Pen) testing [1]

Pen testing is where the security tester attempts to penetrate the IoT system's security measures. Pen test results include testing software, issues, reports on what was done, and prioritization or guidance on untested areas. The tester should think of this security Pen testing as a service on software and a collaboration between the teams doing development. When the Pen results come through, developers can then seek clarification and advice from testers and V&V and the security team on the issues and how best to resolve them. I recommend outside testing groups or V&V conduct Pen testing when risks justify it.

#### 4.3 Access Sensitive information [1]

This activity is a continuation of social engineering and is part of gathering information to support security risk assessment. The hacker-tester tries to access information about the system for which they are not privileged to access. They may "play" the role of a user with limited access and then try to gain privileged access or even play an outside hacker trying to get in. Testers should try to access the IoT device, edge, fog, development areas, and even the cloud. Testers also look for data and information used in later testing and attacks in this information gathering phase.

## 4.4 Access control and password cracking

A favorite process that is a very productive subset of Pen testing is password cracking or gaining system control access. To do this, there are tools such as PKcrack [1]. Even worse, many IoT systems use default passwords and tools such as those found on the website Shodan™ [1], exposing these systems to anyone looking to do a hack. Why? Because the bad guys will do this.

## 4.5 Elevate access privileges [1]

After the Pen or cracking testing works and the tester-hacker is within the IoT system, they should gain elevated access privileges. This will expose vulnerabilities and can provide yet more information to use in security attacks and testing. Again, the bad guys will try to do this, so the hacker-tester should try to do it first.

## 4.6 Denial of Service (DoS) Attack [1]

IoT systems are very attractive because many IoT devices make a large attack surface, and many of the security features of IoT devices are lacking. Bad guy hackers will attack all IoT systems; therefore, you will want your system to respond as “securely” as possible (safe-secure engineering is a design issue that can be addressed in an STA [1, 16, 19, 20]). The bad guy hackers will start by attacking the company’s systems, including IoT devices. Hackers will start with easy kills, like DoS on IoT. A DoS attack attempts to block the IoT site operations by sending many requests that exceed the network bandwidth and system responsibility.

## 4.7 Security Taint Analysis [1]

Security taint analysis is associated with malware, code faults, and a DoS to some extent. It is the process of staff checking the flow of user input in the IoT inputs to determine if unanticipated input can negatively affect program execution during each functionality step. The staff doing this analysis can include developers, security staff or testers. The analysis staff will “play” different roles and attempt to detect disruption of normal input processing flow. Taint analysis can help check the IoT attack surface and define new avenues of attack. The test staff should know the following kinds of information, which can be tainted: inputs, data values, sequence of inputs, messages output, file outputs, response actions, and links to other elements of the IoT system, such as the edge, fog, and/or cloud. During the analysis, the staff “user” tries to corrupt (taint) the inputs and then reviews outputs for improper actions and outputs. The output could be reviewed in real time, but the output must also be post-processed. Post-processing can be enhanced if a dynamic runtime taint monitoring or AI system [1] is included in the taint configuration.

## 4.8 Tampering Attack [1]

In tampering, testers try to “break” into the hardware and/or software to gain access. This is closely related to password cracking but takes other approaches such as inserting code, data or hardware into an IoT configuration (for configuration management (CM)/software configuration management (SCM) tools and/or processes). The IoT tampering attack can modify parameters exchanged between the device and edge, fog, or cloud. The goal is to manipulate IoT data such as passwords, output values, permissions, etc. The tester is trying to understand the following:

- Does the external CM/SCM system or internal version checking detect tampering?
- Does the system itself notice and inform stakeholders or provide notification of the attack?
- Can bad data be input, output or used?
- Malware and Security Attacking the Off-the-Shelf (OTS) Software for Malware [1]

Most IoT systems will include OTS software from open sources or procured from third parties. OTS introduces security risks. The use of tools and analyses to support testers when conducting this attack is a good idea. When you do not trust the OTS software, conducting these scans and analyses may help, in addition to running security test attacks. In this analysis, the tester will need access to the

source or binary code using a tool for the OTS. The OTS code should be visually checked during an inspection meeting by a team, but humans will still miss things, so tools are also recommended.

#### **4.9 Spoofing [1]**

A spoofing attack uses the information to impersonate an authorized device or user. Once in the system, spoofing can: steal data, insert malware, or bypass access control systems for later access. For IoT systems, connections can be spoofed, including:

- Website or Domain name for IoT edge, fog or cloud services
- IP addresses
- Address Resolution Protocol
- GPS spoofing of the location
- User (man, system, hardware)-in-the-middle
- Identity of user or login
- Report how hard or easy it is to read the file (unformatted clear ASCII text is bad; encrypted data is much better since it is another stumbling block for a hacker to overcome).

### **5. Break the Software**

For test methods to “break the software” security see the classic book references [4, 8] which provide many security test attacks.

### **6. Security testing needed beyond a single IoT device**

These security assessment elements include the hardware, system, and other IoT devices, edge, fog, and cloud when dealing with vulnerabilities. These assessments are common with a single IoT device, but I advocated that the security threat assessment must continue at a system level for IoT. While ownership of these other levels of IoT security testing may not be clear, the device IoT test team should at least bring risks forward so stakeholders can make informed decisions after the single device is assessed.

### **7. Skills for Security Assessments**

A sample skills list for security assessments includes [1,16,18,19,20,21] and the following:

- Software engineering
- Systems engineering
- Partners and suppliers of support products from a security risk viewpoint
- Test governance, standards, certification, and guidance
- Testing of quality characteristics, which are essential to the security of IoT systems
- Maintainability
- Performance
- Interoperability and integration
- Reliability
- Security attacks
- Dependability

### **8. AI and DA Testing Assessments for IoT**

AI and DA will be essential engineering areas for IoT systems. The first use many people think about for AI and DA is helping users and improving sales (for marketing). However, this simplistic view overlooks the important areas of development and test activities. The IoT device will feed massive amounts of information into AI and DA. We already see this in the web world. The edge and cloud will collect and analyze data and feed AI and DA. Testers that want to improve their assessments. Testing and product quality areas are advised to learn and use AI and DA. Since the data values will be large, tools, STAs, AI and DA must be incorporated into an integrated system, as shown in Figure 5. Ever-

improving AI and DA should drive the testing and assessments over the lifecycle. Testers need to do the top-level test planning and product quality assessments. Manual testing will become a minority activity as test automation, AI and DA use increases.

In Figure 5, AI and DA are part of a constant flow of information from the branches of the IoT system, device, edge, and cloud, into development, test, and as needed, V&V/IV&V. Continuous lifecycle development, integration, test, and deployment is part of this really important model for IoT. Testers and companies that do not learn new skills and improve the IoT systems based on the AI and DA feedback will become like the programmers of FORTRAN and Cobol systems i.e., dinosaurs (if you are not sure what those terms are, search the Internet).

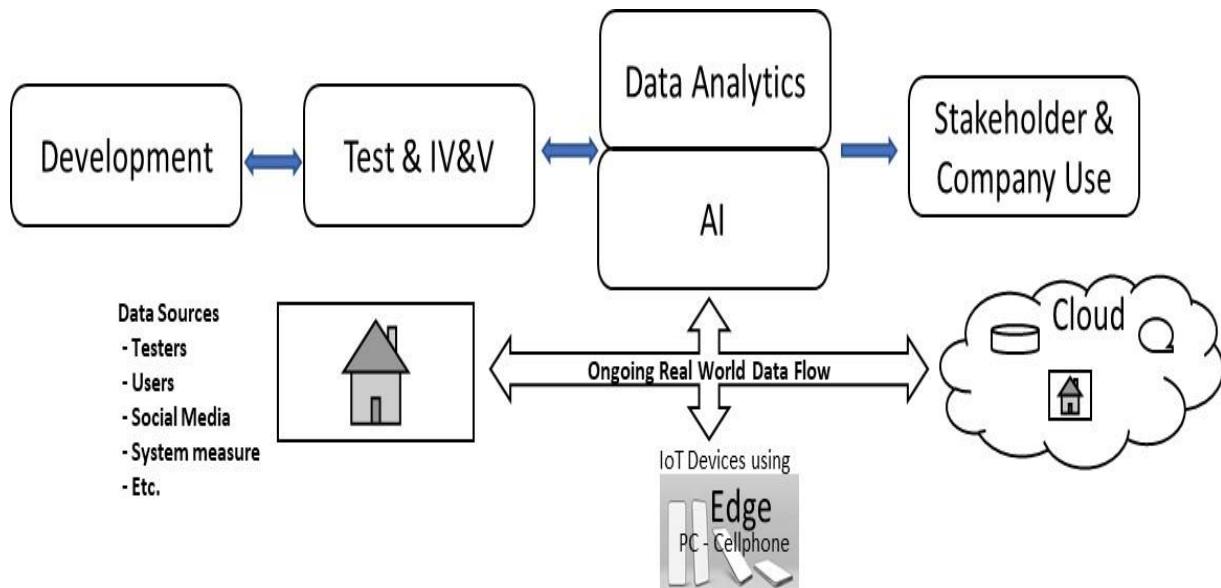


Figure 5: AI and data-driven development and test

A continuous process of improvement and development will rely on the information coming from AI and DA. The massive data millions of IoT devices and systems can generate exceeds human capabilities. The data flow from the real world of local IoT devices (the home), the edge, and the cloud requires constant analysis. While based on statistical tools and concepts, DA will need AI support. Some of the flow of AI and DA information to stakeholders and companies will be for things like improving sales and help desk support. At the same time, teams cannot forget the flow of AI and DA information into development efforts as well as to testing, V&V and IV&V. IoT products will need to be iterative as updates and new product features are implemented. Many IoT teams may forget to include development and test in the information coming from the AI and DA flow by choosing not to update or improve products. This will be a mistake. In particular, test teams doing assessment and V&V activities will need to ensure they are part of this information flow while learning new technical skills. It is important to consider that V&V and IV&V will need to include AI and DA modelling, as well as product improvement beyond what the basic, initial requirements dictate.

For IoT systems, these assessments will need to move beyond just running regression tests. For example, as the test architect on a project, I used data from the test issue database and help desk reporting systems to conduct system-level reliability and security analysis during product usage. This allowed the development team to estimate when failures or threats might occur and identify areas for improved development and testing. While development management did not like to hear about these analysis predictions and increased costs, it allowed them to allocate more resources toward process and product improvements while staying business.

Skills list for these AI and DA activities include [1,19,20,21]:

- Systems engineering
- Knowledge and familiarity with test framework tool sets (e.g., unit test; automation; AI, data analytics, STAs with continuous integration, test, and deployment frameworks)
- Statistics and data analytics math concepts as applied to IoT testing
- Understanding AI processing, tools and analysis

- Creating software test architectures (STAs)

## 9. Future work

The nature of IoT will unfold and evolve the way general IT and the web did. The impacts of AI, DA, system-level V&V/IV&V/testing will evolve as assessments of IoT come of age, and the changing skill base of the development and test team evolve too. Teams must use analytics and AI to conduct process and product improvements. The available information needs to be applied, analyzed and, as needed, used to make new IoT systems. The ideas in this paper should be expected to be updated by analyzing home, personal, medical, societal and industrial usage of IoT.

## 10. Conclusion

This paper advocates the need for IoT software system-level testing beyond just the IoT device, because software IoT systems will need to go beyond the basic testing of the device to satisfy stakeholders and security requirements. Although testing the device is important, testing *only* the IoT device software is insufficient. To achieve this, testing and V&V/IV&V, particularly at the system level, will be optimal for many IoT systems. Testing/V&V should include focus areas such as security assessments, use of AI, and data analytics.

## References

1. Book: IoT Testing by Jon Hagar, scheduled for publication Sept 2022, Springer Nature
2. Software Verification and Validation (V&V) by Deutsch, 1997
3. "IEEE Standard for System, Software, and Hardware Verification and Validation," in IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1-2017), vol., no., pp.1-260, 29 Sept. 2017
4. Software Test Attacks to Break Mobile and Embedded Devices by Jon D. Hagar, 2013
5. Educating Students on Techno-Capitalism's Impact to STEM, by Jon Hagar, 2021
6. Software (Test) Architecture Elements Applied to Software Test: View, Viewpoints and Containers, by Jon Hagar, InSTA conference, 2022
7. The Art of Software Testing by Myers (the grandfather of them all), 1979
8. How to Break Software: A Practical Guide to Testing by James Whittaker, 2004
9. A Practitioner's Guide to Software Test Design by Lee Copeland
10. Domain Testing Workbook by Cem Kaner
11. Lessons Learned in Software Testing by Kaner, Bach, Pettichord
12. Testing Computer Software by Kaner, Falk, and Nguyen, Van Nostrand Reinhold, 1993
13. Systematic Software Testing by Craig and Jaskiel
14. "IEEE/ISO/IEC 29119, ISO/IEC/IEEE International Standard - Software and systems engineering - Software testing" –Parts 1 to 5 series
15. Software Test Automation by Fewster and Graham
16. IEEE 982.1-2005 IEEE Standard Dictionary of Measures of the Software Aspects of Dependability, in revision as of 2022
17. Smart Santander: IoT Experimentation over a Smart City Testbed, Luis Sanchez et al, 2020
18. How to Break Software Security by James A. Whittaker and Hugh Thompson, Addison-Wesley Professional, 2004
19. ISO Standard 15288 Systems and software engineering — System life cycle processes
20. ISO Standard 12207 Systems and software engineering – Software life cycle processes
21. ISO 9000:2015 Quality Management Systems (series)- Fundamentals and Vocabulary

# Agile Gets Physical: Slice-Based Integration

**Kathleen A. Iberle**

kiberle@kiberle.com

## Abstract

Are you struggling to integrate the results of your agile software development into a hardware project? Frustrated by waiting weeks to receive test results, or rushing to deliver code only to find the hardware isn't ready to run that code? Do you feel like your hardware partners just don't understand what you're doing?

This paper explores applying agile ideas to a field which isn't entirely agile. We'll show how to use the "integration by slice" method to bring together and test the results of hardware and software development incrementally, planning in a fashion that serves the differing demands of both disciplines.

This paper is adapted from two chapters in my recently published book *When Agile Gets Physical: How to Use Agile Principles to Accelerate Hardware Development*. The book was written for hardware engineers, while this paper takes the perspective of the software tester.

## Biography

**Kathy Iberle** has been working in Lean and Agile product development for over twenty years, using cutting-edge methods to help strengthen development and make mixed hardware/software projects run more smoothly. Much of Kathy's career was spent at Hewlett-Packard, working on a range of products from printers and electronic instruments to medical applications and website apps. She has a wide variety of experience, having worked in roles from developer to quality assurance expert to agile adoption leader. Since 2012, Kathy has been running her own consulting firm to help companies strengthen and accelerate their product development.

Copyright Kathleen A. Iberle 2022

## 1 Introduction

When a product incorporates significant amounts of both software and hardware, integrating the work of the various engineering teams often leads to confusion and conflicts. Code is delivered into integration, only to find that the hardware isn't ready yet, and vice versa. Defect reports arrive long after the code was delivered and the software team moved on to other work. Agile firmware and software teams in particular often feel as if their mechanical or electrical engineering partners are operating in a different world and don't understand their agile work.

This paper presents a solution based on the principles behind agile software development. Applying agile software methods verbatim to mixed hardware/software development generally isn't helpful, but many of the tools of agile can be helpful. The key is understanding why the problems are happening, and choosing the tools which address those particular problems.

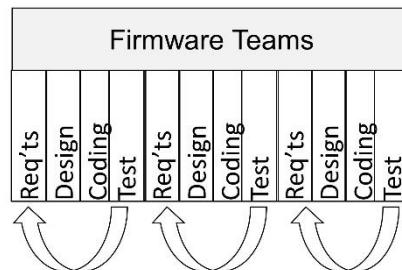
## 2 Why is Hardware/Firmware Integration So Hard?

Hardware and firmware development teams usually have very different product development processes, which leads to differing assumptions about the integration process. When firmware teams are using agile, and the hardware teams are using a phase-gate or "waterfall" style of development, their methods often collide with each other, causing frustration on both sides. Yet the two engineering disciplines are using different methods for very good reasons.

### 2.1 Different Ways to Deal with Uncertainty

Both hardware and software development operate in a realm of significant uncertainty. We can't tell in advance exactly what the customer really wants, or predict the security needs a year in advance of release. We also can't predict exactly how a physical object will work when it is finally built with realistic parts, or precisely what the materials and manufacturing process must be to achieve the desired behavior.

One approach to dealing with uncertainty is to try things on a small scale to see if they work. Agile software development does exactly this. The development is broken into small batches of features, and each small batch is driven to completion in a single sprint or iteration. The feedback loop – discovering and fixing errors or incorrect assumptions – is short and fast. There hasn't been time for the developer to forget the details, there's no need to roll build and test tools back to earlier versions, and the error hasn't been repeated in other related code. Short feedback loops are less expensive.



**Figure 1: Firmware development using agile has short, fast loopbacks.**

"Try it and see" would work in hardware too, except for one thing – it's relatively easy to rework software if your guess is wrong, but it is much more difficult and time-consuming to change hardware. Software, especially software developed with the assumption it may need to be changed, can often be rewritten and retested in days or even hours. Revising hardware, particularly mechanical parts, may take weeks, because physical tooling must be revised or recreated. This is known as a *high cost of change*.

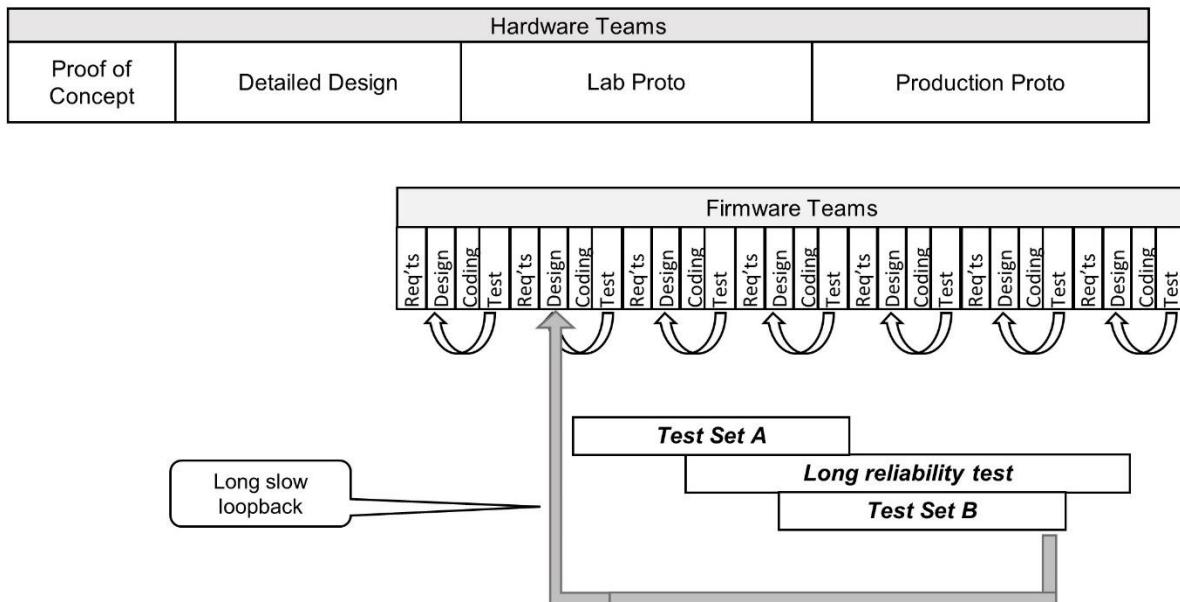
Teams facing a high cost of change want to get it right the first time, so they often use a phase-gate process which looks a lot like waterfall software development. In addition, mechanical features often can't be developed independently of other features, so the "user story" approach isn't entirely feasible. You can't tell how well a handle will work by itself – the rest of the object must be attached. These two factors mean that a hardware team typically develops most features together in one huge batch. The batch progresses through several levels of "realness" from an initial rough mockup through increasingly realistic prototypes to a final manufacturable product. The team also has to concurrently develop the manufacturing process.

These two methods are pretty different, but that typically doesn't cause problems in the early stages of a hardware/firmware project. That's because the engineering disciplines are often working more or less independently of each other at the start. The firmware teams are using software stubs to simulate the actual hardware, or testing with older versions of the product or breadboard emulators<sup>1</sup>. The hardware teams often use the previous product's firmware or a simple throw-away firmware test harness to drive just the behavior they're currently working on. Since the disciplines are largely operating independently, they can use different project management methods without interfering with each other.

## 2.2 Hardware-Style Integration Creates Long, Expensive Loopbacks

The problems show up when the time comes to integrate the firmware with fairly realistic hardware. Here's where we learn that the emulator or the test harness didn't accurately represent the final component. Learning something new isn't necessarily a problem – but **when** you learn it can be problematic. Unfortunately, traditional hardware-focused integration tends to push the time out, creating long, expensive loopbacks instead of short loopbacks.

Traditional hardware-focused test planning minimizes the need for expensive physical prototypes and test beds by organizing the testing into long test suites which test multiple aspects of the system at once. Often tests are checking functionality and collecting statistical data on behavior at the same time. This reduces the need for costly prototype units, but it breaks the connection between when a feature is delivered and when it's tested. This creates long loopbacks, as shown in Figure 2.



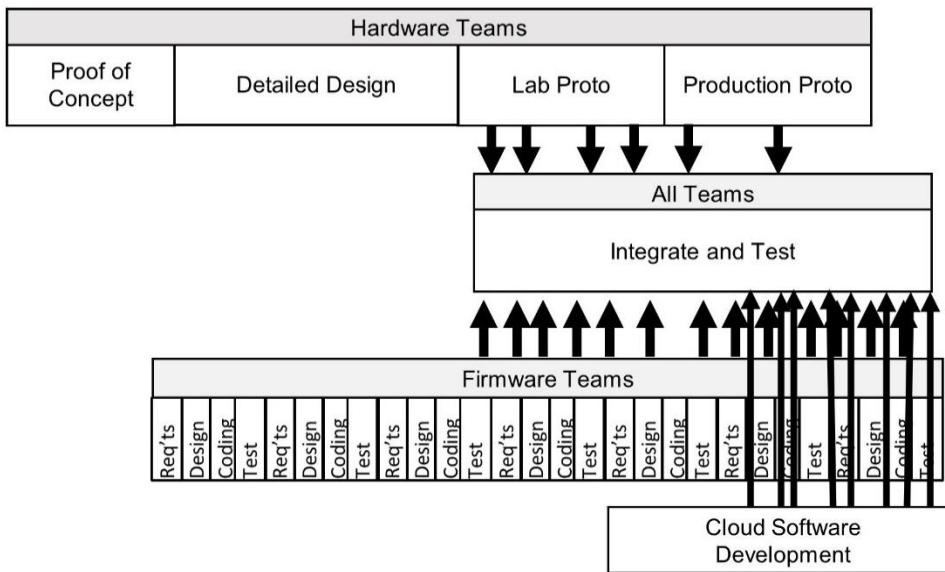
<sup>1</sup> There's a good example of this in chapter 6 of *A Practical Approach to Large-Scale Agile Development* (Gruver et. al. 2013).

**Figure 2: Hardware-style integration causes long, slow loopbacks for firmware teams.**

### 2.3 Large Test Suites Lead to Confusion

Hardware-style integration testing usually reports defects well after the code was delivered, often in batches at the end of each test suite. An agile firmware team may respond with several quick deliveries of new firmware – but the hardware-style integration testing has already started another test suite. Their process isn't designed to handle new revisions in the middle of a test suite. The program doesn't have adequate tools to keep track of what is supposed to be working or not working at a given point, and which revisions of firmware require which sets of hardware.

If the product can be driven or accessed from mobile apps or the cloud, that adds another team or two, which are also firing off revisions of their code. By the middle of integration, I often see a barrage of deliveries - updated hardware subsystems, firmware drops, and drops of cloud-based software services – which aren't coordinated with each other or with the test plan. Relationships are unclear, as shown in Figure 4. It's very easy to lose track of what needs to be done in order to deliver a working product.



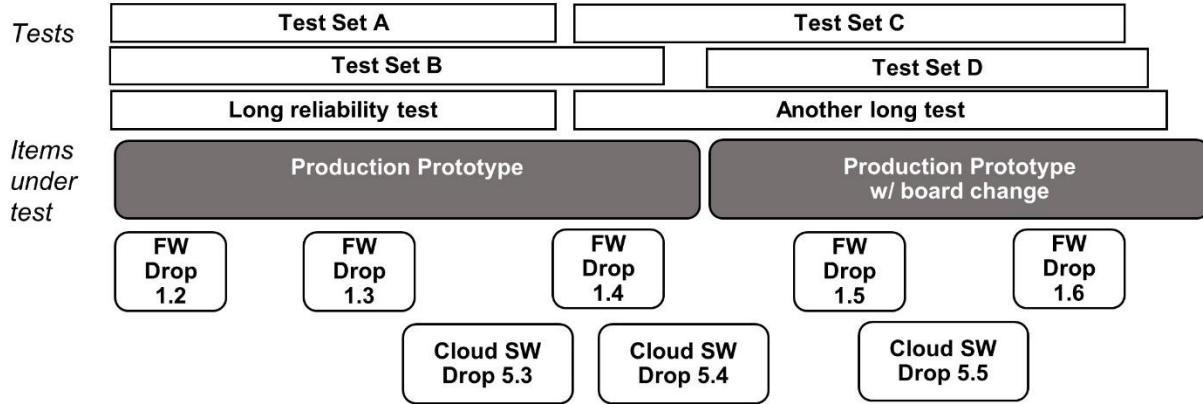
**Figure 3: Unclear relationships between software drops, hardware drops and integration tests.**

This problem is exacerbated by “fixing it in software”. It often makes economic sense to fix a problem with a firmware change rather than reworking the hardware, but that can wreak havoc on firmware project estimates because those fixes are essentially additional features. And it creates yet more need for yet more drops. Chaos can ensue.

Fortunately, applying the principles which underly agile methods results in a much better solution.

## 3 The Solution: Smaller Batches of Integration

The root cause of this chaos is the mismatch between the large batches of work represented by those long integration test suites and the small batches used by firmware and software development teams. There's no way to set up a clear correspondence, as shown in Figure 4.



**Figure 4: A typical integration plan with long test suites.**

The software batches are very small, but we don't want to make them larger. That would create longer loopbacks. The hardware batches are very large, but making them smaller isn't very practical due to the high cost of change and the difficulty of developing a feature independent of the other features.

A third path is to split *the integration testing itself* into smaller batches of tests and focus each of those batches on specific user-visible features or capabilities. This gives us shorter batches of testing with shorter feedback loops, and the focus on user-visible features makes it much easier to see the relationship between development drops and test suites.

### 3.1 Start by Defining Timeboxes to Create a Cadence

I start by splitting the integration schedule into arbitrary, evenly spaced timeboxes of a few weeks' length, as shown in Figure 5. If the firmware team is using agile, the firmware team's sprint or iteration length is often a good choice unless it's shorter than two weeks. A timebox of less than two weeks is rarely optimal for hardware/software integration testing due to the additional overhead.<sup>2</sup> The integration tests have to use real physical objects, so they take longer to set up and tear down than a firmware subsystem test.

Time-boxes	Timebox #1 Weeks 10-12	Timebox #2 Weeks 13-15	Timebox #3 Weeks 16-18	Timebox #4 Weeks 19-21
------------	---------------------------	---------------------------	---------------------------	---------------------------

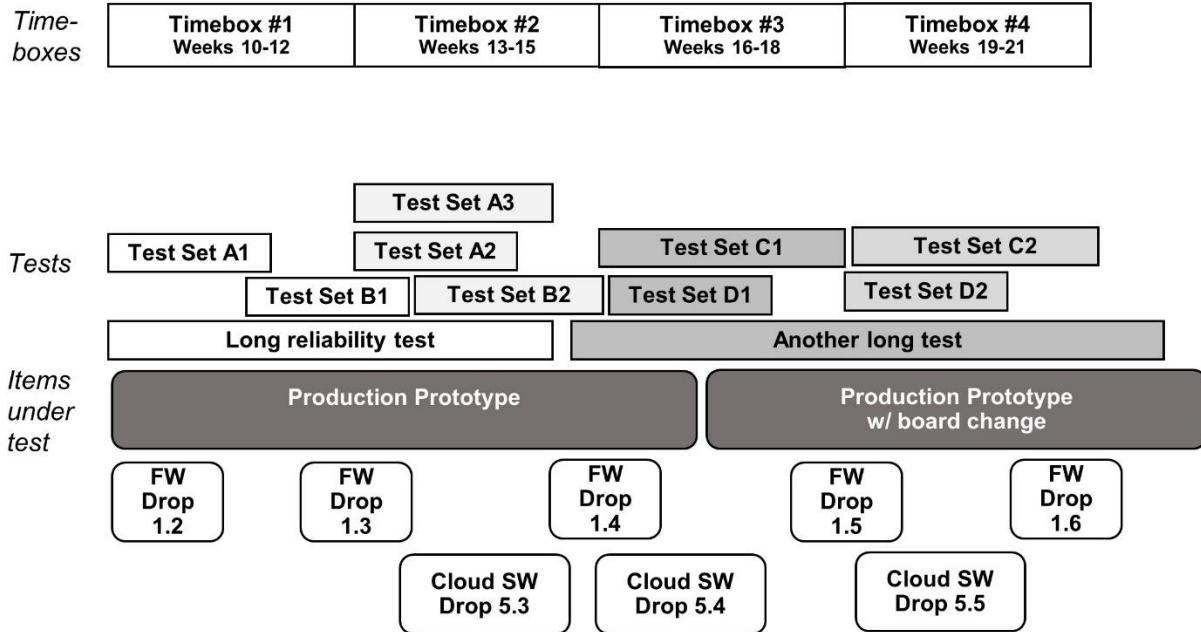
**Figure 5: Integration Timeboxes Create a Cadence for Planning**

Simply creating these timeboxes sometimes improves the project management immediately. The cadence of the timeboxes simplifies schedule discussions by reducing the degrees of freedom, just like sprints. The teams don't talk about which day to deliver a drop, but rather which timebox. That's often a much simpler decision.

### 3.2 Split Up the Test Suites into Smaller Batches

There's still a mismatch between the fast firmware deliveries and the long test suites. The test suites need to be split into smaller suites which will more or less fit into a timebox, as shown in Figure 6. Each of these suites will focus on a specific set of features or capabilities. This won't be perfect – there's usually still a few long test suites, such as those collecting statistical data on repetitive actions.

<sup>2</sup> What is an optimal length? See my online article “Cadence: How Fast is Too Fast?” (Iberle 2021)



**Figure 6: Test Suites are Split Into Smaller Batches.**

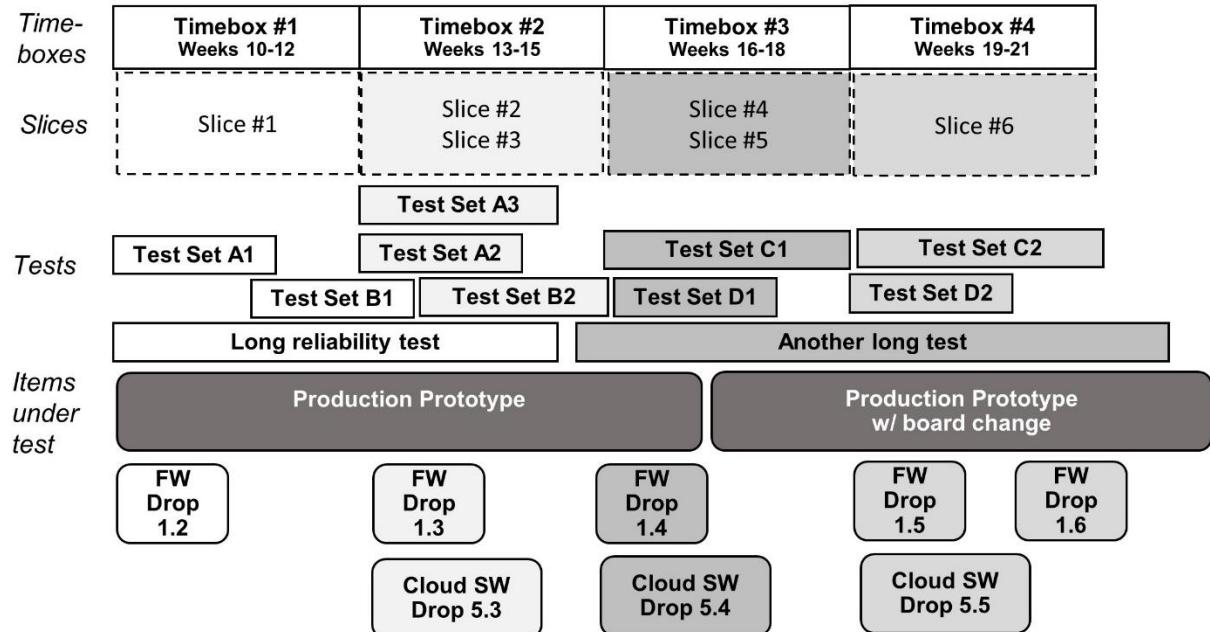
### 3.3 Identify the Development Which Is Needed by the Test Suites

Then, for each timebox, the hardware and software teams together identify the features which must be delivered in order for the planned test suites to pass. Those features are then treated as a group, which we call a *slice*. A slice is a set of features or behavior that will be tested within a single integration timebox. A slice is much larger than a typical user story – it's more the size of an epic or even bigger.

The slices pull the disparate development efforts together, so the firmware, hardware, and test teams are all aiming for the same goal at the same time. The slice definitions provide a solid target for each timebox, as shown in Figure 7. It may seem odd to define a group of features by starting with a test rather than product requirements, but this is quite effective in practice.

Slice-based integration is considerably easier to manage than the more traditional method, because there's a clearer relationship between the test suites and the firmware deliveries. When using a slice-based planning method:

- All development teams know in advance what does and doesn't need to be delivered for any given timebox.
- Test teams can determine whether or not those items have been delivered – which tells the program whether or not the system is ready to run (and pass!) a given test suite.
- Teams get faster feedback, because less time passes between writing software / building the prototype hardware and receiving test results from the integrated system.
- Program management can more easily see what is done, and what still needs work.



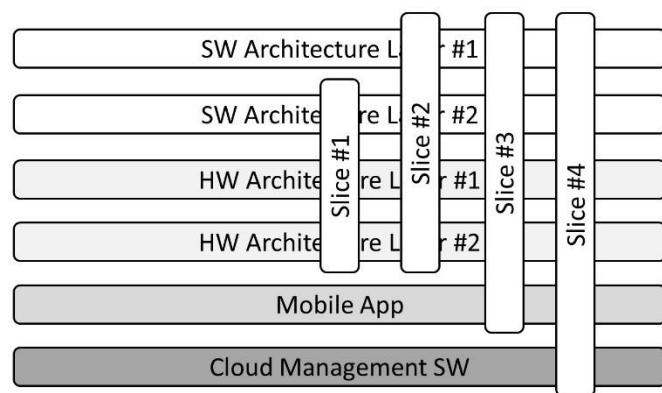
**Figure 7: Integration By Slice Coordinates Hardware Prototypes, Firmware Drops, and Tests.**

### 3.4 What's a Good Slice?

Typically, the functionality in a single slice requires contributions from multiple disciplines and subsystems. That's why it is called a slice - each batch of functionality slices through the architecture just like slicing through a layer cake. Bill Wake used this metaphor in an Internet article "INVEST in Good Stories and SMART Tasks", where he described slicing vertically through the architecture layers to provide the customer with "the essence of the whole cake". (Wake 2003) The same principle works for a physical product plus its firmware.

Figure 8 shows some possible slices for a hypothetical inkjet printer:

- Slice #1: Printer can print a test page in response to a button press.
- Slice #2: Printer can print a page sent from a connected PC.
- Slice #3: Printer can print a page sent from a mobile app.
- Slice #4: Printer can remind user to purchase more ink.



**Figure 8: Slices through the architecture of an inkjet printer.**

The slices in this example describe high-level end-to-end behaviors from the end user's perspective. Your test plans may refer to these as black-box tests, end-to-end tests, or user scenario tests.

The manufacturing process is also a user, so we have slices capturing behavior seen by the manufacturing process:

- Slice #19: Printer can run the production test which verifies alignment of the cartridge.
- Slice #20: Printer can perform post-production tests which verify that most recent firmware has been loaded.

And lastly, the product will need slices which test the non-functional requirements or capabilities of the system: security, usability, reliability and the like.

- Slice #15: Known paths for hacking into network via mobile app connected to printer are blocked.

A good definition for this type of slice enables the team to understand what must be finished before it is sensible to run these tests. If your organization is prone to running tests too early, this can save a lot of time. Writing the definition also is a good opportunity to clarify expectations for the capability. What does “reliable” actually mean? And how will you know if that expectation is met?

Slices are effective only when all teams understand what each slice means and can relate it to their own work and to the integration testing. Teams need a common language for effective coordination. The user’s language is the one most shared across teams, so that’s what we use. “Confirm torque required to produce adequate sealing for the R3448 Pressure Valve” or “Error-handling for temperature controls” may be meaningful to one team, but they don’t tell the other teams what they need to contribute to this slice. When each slice delivers behavior that is meaningful for the end user, it’s also easier to assess progress across the entire program.

If we must refer to the architecture, the slice should still be stated in terms recognizable by a user, such as “Printer functions normally with Rev 4.2 boards” rather than “run regression tests with Rev 4.2 boards”. This also encourages the test team to ask interesting questions regarding what “functions normally” actually means.

### **Attributes of a Good Slice**

An effective slice, however, has more than just a clear scope. A good slice:

- Has a name and definition which clearly communicate its scope
- Enables test teams to identify or describe the tests which will validate the slice
- Enables development teams to easily identify what to deliver so those tests will pass
- Is small enough to be tested within one timebox
- Delivers a usable set of behaviors, or a capability that some user cares about, or can answer a question critical to further development
- Bonus points if the slice allows the team to test riskier aspects of the system early

Splitting the integration work into slices makes it much easier to coordinate the various teams to deliver the right functionality all at once time. And the integration batches are smaller, which creates faster feedback and reduces those long, slow loopbacks.

## **3.5 The Slice-Based Integration Plan**

A slice-based integration plan has four main parts, as shown in Figure 9.

- Timeboxes: The weeks remaining in development, organized into evenly-sized boxes. Typically, these timeboxes are two to four weeks long, and smaller is better. If possible, align with the firmware team’s sprints or iterations so that the firmware team drops new software into integration at the start of each timebox.
- Items Under Test: the deliverables that are in testing during a timebox. These comprise:

- hardware prototypes or prototype variants that will be used for the testing
- firmware drops
- cloud software drops
- Slices: the *features and behaviors* to be delivered in this timebox. The items under test have to come together to deliver these features.
- Tests: The specific tests planned for this timebox which will exercise the delivered features.

Timeboxes	Weeks 10-12 Mar 08-Mar 28	Weeks 13-15 Mar 29-Apr 18	Weeks 16-18 Apr 19-May 09	Weeks 19-21 May 10-May 30
Proto Build:	Prototype 1		Prototype 2	
Hardware Deltas:	<none>	GRS board 4.2	Initial packaging	TBD – board 4.3 release?
Firmware:	Drop 0.2 4/02	Drop 0.3 4/22	Drop 0.4 - TBD	TBD
Mobile SW:	<none>	<none>	Release 1.13 4/20	Release 1.14
Slice Definitions:	- Print test page	- Print from PC - Rev 4.2 boards integrated	- Print from mobile app - Edge-to-edge photo printing	- Mfg alignment test works - Purchase more ink reminder
Tests Planned:	- Basic functionality - UX button response - Life test	- Print, all OS - Board regression - Life test, cont.	- End-to-end mobile printing - In-box durability	- Mfg verification of cartridge alignment - Low on ink - Deplete ink

**Figure 9: Anatomy of a Slice-Based Integration Plan.**

A slice-based integration plan looks similar to an agile roadmap<sup>3</sup>, and is managed similarly with rolling-wave planning. Just as in agile development, the team meets at least once per timebox to review the plan and adjust it based on recent learnings. This plan can be kept in a spreadsheet, or an HTML table, or a physical planning board. The ideal tool is visible to all your team members and relatively easy to change – because it will be changed frequently.

### 3.6 Build a Slice-Based Integration Plan

Moving from a traditional hardware-centric integration plan to a slice-based integration plan can be awkward, since it's rarely practical to re-plan all the integration testing from scratch. Here's one way to create a slice-based integration plan, starting from where you are today. This is best done with all the technical leads from development and testing teams in the room together.

Starting from your existing plan:

1. Define your timeboxes.
2. Place the known plans for hardware prototypes into the “Items In Test” row.
3. Add the planned firmware and cloud software drops, using your usual naming conventions. At this point, some of the drops will have intended contents and others may not yet be planned. That's OK.
4. Draw the planned integration tests on your plan and summarize the functionality each one assumes.
5. Create the first draft of your slices. In the “Slice Definition” row, summarize the planned contents of the firmware and hardware drops – *stated in the form of end-to-end testable behavior*. This

---

<sup>3</sup> There's a good explanation of an agile roadmaps and rolling-wave planning in chapter 6 of Johanna Rothman's *Create Your Successful Agile Project*. (Rothman 2017)

language will allow you to compare the test plan with the delivery plan. This is usually the epic level, not the user story level.

At this point, there's probably a bunch of mismatches between the planned testing and the planned delivery, and a lot of things that are ambiguous or blank.

Start iterating towards a better plan by comparing the slices with the planned integration tests. If the slices are delivered as shown, can the integration tests be run successfully? If the test plan calls for testing something that won't be ready until a later timebox, there are three options:

1. Move the test later.
2. Move the feature development earlier. This may mean reorganizing some firmware drops.
3. Split the test into smaller pieces and put them in different timeboxes, so we can test what's ready as soon as it is ready (but not sooner).

Keep iterating on your plan until you have a schedule that looks feasible given what you know today.

When you have a plan which aligns all the disciplines with a clear set of targets at a manageable level of detail, you're done. The teams now have a shared view of the program's path towards full integration that accounts for the cross-discipline dependencies. Once they have that view, they won't need a detailed Gantt chart showing all these dependencies in great detail. Instead, the technical leaders can use their slice-based integration plan to adapt to changing conditions and manage the complexity by working together. The timeboxes provide a regular cadence for this coordination, and small enough batch sizes to shorten those feedback loops and make progress visible enough to be actionable.

Don't try for perfection. If you have a few tests which span more than one timebox, that's ok. And don't feel you have to plan the entire integration plan all the way through the program. As in agile development, it is ok to have some iterations without defined contents or only roughly defined contents. You'll be updating the plan regularly and frequently.

### **3.7 Alternate Methods**

Sometimes it makes more sense to start from the test suites rather than from the firmware drops. If the test suites already are organized around user-visible behavior, it may be easier to pull out those behavior statements and use them to create the first draft of slices. A single test suite generally tests a good-sized list of behaviors which will have to be split into different slices. The development and test teams will work together to decide which behaviors go in the first slice and which in the next slice, and so forth.

It's also possible to do both. Start with the firmware drop definitions (usually at the epic level) and work towards a set of slices which are smaller than the original test suites and larger than the firmware epics. When the end-to-end behavior isn't clear from the firmware drop definitions, look for that type of behavior in the tests.

It might seem sensible to start defining slices by splitting up the requirements list, but in practice I haven't seen this work well. For one thing, the requirements specification often has no relationship with the order in which features will be developed, whereas the development plans and test plans do. Plus, it's not unusual to find that many requirements aren't stated in the requirements specification at all (although they may be documented in test cases).

### **3.8 Use Rolling Wave Planning to Update the Integration Plan**

Your team leaders will meet at least once per timebox to review the slice-based integration plan and adjust it based on recent learnings. These meetings should be frequent, short, and on a regular schedule. For many teams starting out, once per timebox is not quite often enough. Once a week may be better.

These planning meetings bring together the technical leads and project leaders from all of the disciplines, including the system or integration test groups. Each discipline needs at least one person who can share

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

PNSQC.ORG

Page 10

progress updates and make decisions. The meeting might be run by the overall project leader, or by a chief engineer or architect, or even the integration test lead.

The typical agenda is:

1. Review current work and make adjustments as needed.
2. Review the upcoming timebox and make adjustments as needed.
3. Fill in the next open timebox with a fully detailed plan.

We don't use the traditional Scrum standup questions because those are designed for load-balancing the work across the team members. In most mixed hardware/software programs, it's not practical to shift work across disciplines, so that's not our goal. Instead, our questions are designed to share information efficiently:

1. Has any item-in-test or planned testing changed?
2. Is anything in your way of getting ready for the next timebox?
3. What else do we need to know?

The purpose of the meeting is to adjust the integration plan based on the most current information. It's not a progress report, so the only thing displayed is the integration plan. There's no need to cover things which have already been communicated in another meeting. This meeting is a place to make sure we all understand what to do next, and to identify what might get in our way.

As in any technical meeting, there is a temptation to start solving knotty problems in the meeting. Don't do that. Unless it can be solved in a few minutes – "Does anyone here know where the extra 4.2 boards are?" – the issue should be raised and then taken off-line.

### **3.9 This Is Not a Game of Perfect**

The primary benefit of a slice-based integration plan is better visibility and communication across teams. Each team can see where the program is going, and what they need to do individually and collectively to get there.

A healthy integration plan has these characteristics:

- The timeboxes are roughly equal to the size of firmware sprints / iterations.
- The slice definitions describe recognizable, testable functionality.
- The tests align well with the slices.
- It's obvious what we expect to learn from each test suite.
- The plan is visible, and updated regularly with all disciplines actively participating.
- The participants feel that the meetings are worth their time.

You'll know your slice-based planning method is working when you can change the plan without causing much rework for any of the disciplines. And you'll know it's working when you're spending less time feeling confused or fixing problems caused by miscommunication. People know what they need to deliver when, and what to do if their plans need to change.

If the only benefit from slice-based integration testing is that hardware, software and test teams stop arguing so much and work more effectively together, that can be a big win all by itself. But you can go further.

## 4 Towards a More Agile Approach

### 4.1 Risk-Based Prioritization

Once the slice-based integration plan has provided an explicit correspondence between firmware and hardware deliveries and the tests, and the test suites are smaller, it's easier to move the more critical tests earlier. The "more critical" tests are those which are likely to find either a lot of defects or defects which are particularly time-consuming to fix.

This discussion can lead to another useful discussion: whether some **development** should also be shifted earlier to allow early testing. That might be defect-prone areas, or it might be areas where learning a few things early would provide useful guidance for later development.

You might also see opportunities to shift left – move some integration testing back into subsystem testing. When an integration test finds numerous defects that could have been found without testing on a fully integrated system, some coverage should be moved back into subsystem testing. Again, we will shorten a feedback loop.

### 4.2 Smaller, Earlier Slices

In agile development, we know that smaller batches of features give us feedback sooner, and that's an advantage. The same thing is true in integration testing. .

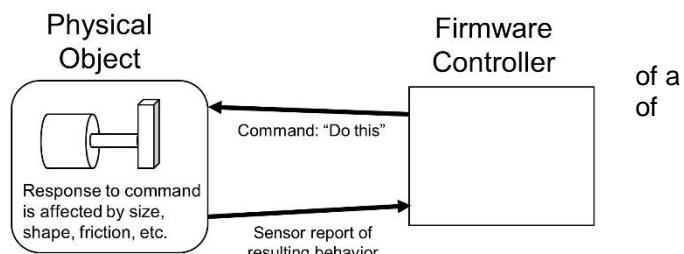
Smaller slices deliver value to the program earlier. An integration slice with its testing delivers value by providing information about the quality and acceptability of those particular features *and* about the progress of the project. When teams know that they have *finished* features to an acceptable level of quality and acceptability, they can use that knowledge to measure progress with certainty.

We can accelerate even more by starting integration by slice earlier in development. Slices nearly always consist of working features because working features deliver the most information about the usefulness and quality of the planned features and about the progress of the project. (Sound familiar?)

However, the earliest slices may not be all that compelling from a user perspective. The earliest possible slices often resemble the "walking skeletons" that authors like Johanna Rothman describe as the earliest outcome of Agile Software Development. Early slices might even include some that aren't user-focused at all, but instead test whether major subsystems are playing nicely together. For instance, a system controlling multiple objects in real-time might have a slice which demonstrates that the communications are in place and happening fast enough.

### 4.3 Emergent Features

Some behavior can only be approximated until the physical system is quite realistic. When behavior depends on precise control physical object, the exact size and properties the parts will affect the behavior, as shown in Figure 6.3.



**Figure 6.3: Emergent Behavior**

This firmware controller must be developed in stages, since the final behavior emerges gradually as successive hardware prototypes become more and more realistic. I call this "emergent behavior". If a

product requires precision control of mechanical or electrical parts or involves fluid dynamics, chances are that it has some emergent behavior. The right firmware settings for this behavior can't be predicted from the design nor determined by early prototypes, but rather must be found by trial and error using the final production prototypes.

Color printing in inkjet printers involves emergent behavior. Printing in color requires precise alignment of multiple ink colors. The ink cartridge, motors, controllers and firmware have to work together to control the movement of the paper and the firing of the cartridge nozzles at the same time. The firmware settings which provide this control for early prototypes likely will not be the settings needed for the final product. An early prototype, or even the final lab prototype, doesn't use the same manufacturing process or sometimes even the same materials as a production printer. These differences make the parts of the overall system move slightly differently, and so adjustments will need to be made. Firmware teams know this and design their code with parameters they can update to quickly dial in the right values once the final hardware is available. Program management sometimes isn't aware this is happening.

When there's emergent behavior, our slices are designed to provide feedback early and shorten those feedback loops, without assuming we are done when we know we are not. For instance:

- Slice #11: Nozzle and roller positioning is accurately controlled by configurable firmware parameters across the full range of the parameters.
- Slice #12: Printer prints in color with correct colors and alignment which appears correct without magnification.
- Slice #13: Printer prints in color with precisely correct color registration using production prototype printers and production ink cartridges.

Slice #11 obviously isn't an end-to-end user-visible behavior (unless you consider the firmware developer to be a user) but it is a very helpful slice. This tells the rest of the team that the firmware team is able to tweak the parameters later on and get the desired results. It's easier to get this nailed down early, so the fine-tuning at the end won't trip over a bug in parameter handling.

Setting up a series of slices like this makes the emergent behavior much more visible to the program team. It becomes obvious why some of the firmware needs to freeze later than the hardware, and where in the schedule firmware teams and test teams have budgeted time for finalizing that emergent behavior.

#### 4.4 Other Opportunities

Many readers will observe that there are other opportunities in a hardware project to take a more agile approach. These opportunities are further discussed in *When Agile Gets Physical*. In short, breaking the work into user stories doesn't really work, but it is possible to break the work into other types of batches which can be completed in timeboxes and deliver immediate value. Namely,

- In early development, we split **the investigative work and research** into batches. Each batch delivers the knowledge to make an important product decision accurately.
- In middle development, we split **routine development work into batches**. These batches don't deliver finished features, but rather deliverables which answer more key questions about the product and feed into the next step of development.
- In late development, we split **the integration testing** into smaller batches. Each batch delivers specific information about the quality of a set of features or attributes of the product,

If you want to read more about why other types of batches can work (and when they don't), see my series of blogs on this topic <https://kiberle.com/2022/04/05/agile-for-hardware/>.

## 5 Conclusion

Slice-based integration improves the traditional hardware/software integration plan by using familiar agile principles.

- Small batches of work
- Batches which deliver user-visible behavior or features
- Cadence (in the form of timeboxes) to simplify planning
- Rolling-wave planning: adjust to reality as you go

Application of these principles in a way that takes into account the different constraints of hardware development and software development provides a method that can substantially simplify the coordination between different engineering disciplines and speed up the program.

Slice-based integration works well because it gives all the teams a common planning method and language which is meaningful to everyone, and it gives the teams opportunity to shorten the loopbacks and save time on the program.

You can learn more about slice-based integration testing and the more advanced “integration train” method in our new book *When Agile Gets Physical: How to Use Agile Principles to Accelerate Hardware Development* by Katherine Radeka and Kathy Iberle. (Radeka 2022)

## References

Gruver, Gary. Young, Mike. Fulghum, Pat. 2013. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*. Addison-Wesley.

Iberle, Kathleen. “Cadence: How Fast is Too Fast?”. <https://kiberle.com/2021/02/14/cadence-how-fast-is-too-fast/> Posted February 14, 2021. Downloaded August 20, 2022.

Iberle, Kathleen. “Agile for Hardware”. <https://kiberle.com/2022/06/10/what-makes-agile-work-2/>. Posted February 21, 2022. Downloaded August 20, 2022.

Iberle, Kathleen. “What Makes Agile Software Development Work?”. <https://kiberle.com/2022/04/05/agile-for-hardware/>. Posted February 10, 2020. Downloaded August 20, 2022.

Rothman, Johanna. 2017. *Create Your Successful Agile Project: Collaborate, Measure, Estimate, Deliver*. Raleigh, North Carolina: Pragmatic Bookshelf.

Rothman, Johanna. 2016. *Agile and Lean Program Management: Scaling Collaboration Across the Organization*. Arlington, Massachusetts: Practical Ink.

Radeka, Katherine and Iberle, Kathy. 2022. *When Agile Gets Physical: How to Use Agile Principles to Accelerate Hardware Development*. Camas, Washington: Chesapeake Research Press.

Wake, Bill. “INVEST in Good Stories, and SMART Tasks”. <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>. Posted August 17, 2003. Downloaded November 15, 2021.

# Service confidence...because code coverage and CRAP scores just don't give you the whole picture

Don Kidd

dkidd@miamioh.edu

## Abstract

At Miami University, we have four to six software development teams responsible for the applications and processes that are written to support the members of the Miami community. Formal testing practices were very new to the organization, but we wanted to get an idea of the quality of our applications. One strategy would be to conduct a static analysis of code to obtain a score, but since we are in higher education, we wanted something more. Thus, we developed the idea of "Service Confidence"—as a group of stakeholders, we discuss and score three factors for our services: Context, Confidence, and Complexity. While the grade that we give our applications is important, we have discovered that the discussion determining the grade is just as important. I will be sharing how these conversations are going, and what we are learning as we go through this scoring exercise of Service Confidence, along with some timing considerations when planning on scheduling these discussions. While this doesn't fully replace the quick analytics obtained from code static analysis, we can show that there is more to Quality than CRAP scores and code coverage.

## Biography

*Don Kidd was a software developer for 16+ years and then made the switch to the quality side about 6 years ago, where he is now tasked to improve quality efforts at Miami. He is currently working with various stakeholders at Miami to understand what "Quality" means to them, so that the IT department can design and implement quality measures to reach those objectives. He also coaches software delivery teams on tools and practices to help build solutions they can be confident in, while making the users of these systems more efficient in the work they are doing.*

Copyright Don Kidd

## 1. Introduction

Quality is a word that is used everywhere. In almost every situation, someone is always inquiring about the quality of something. If you are buying a car, I'm sure that you want a quality one. If you are selling something that is handcrafted, you say that it is "quality crafted." When a company creates software, they want the software that is developed to be a quality solution.

When I began my position as a Quality Analyst, I was the first such position at Miami, and it was really the first time in our organization that we decided that quality was an important attribute of the output from our organization. So, what does quality mean, and how am I going to measure the level of quality we are producing in our organization?

## 2. Problem: What Determines Quality?

If I were to be responsible for quality, I needed to get an understanding on our current quality so that I can help us improve it.

Our IT group writes and maintains all sorts of software. Some would be considered a small application (written normally in PHP), and we have an assortment of other scripts and software written in various languages, such as Perl, Python, PL/SQL, etc. (I think we finally got rid of our last COBOL script a few years ago!). The biggest challenge I had was how to measure the quality of each of these services. To simplify this range of development, we refer to each solution that we generate as a service. We define a "service" as an application or solution component created, deployed, or operated by our department that directly or indirectly fulfills user needs.

No matter where you look, everyone says they make quality widget, or their widget is made with quality parts. But what does that mean? Who is defining quality? According to the Merriam-Webster, Quality is "an inherent feature or a degree of excellence." and the Oxford English Dictionary, says that is "the standard of something as measured against other things of a similar kind; the degree of excellence of something." So that helps us get a basic idea of quality, but what are these "similar kinds"?

One simple way was to start was for us to create the groups of "similar kind" based on the language our "services" are written in.

## 3. Language-Based Groupings

By grouping our services by language, we can run a variety of tools specific to the language that the service was written in.

### 3.1. PHP

- Unit tests and feature tests are run to check on the code to make sure that the tests pass and to see how much of the code is covered with the tests.
- The Change Risk Anti-Patterns (CRAP) Index is calculated based on the cyclomatic complexity and code coverage of a unit of code. Code that is not too complex and has an adequate test coverage will have a low CRAP index. The CRAP index can be lowered by writing tests and by refactoring the code to lower its complexity.
- The code run is analyzed using PHPCS, which is a code sniffer that reviews and detects violation of coding standards.

### 3.2. Shell

- Shell Check is used to find basic bugs in the shell script.

### 3.3. Perl

- Perl::Critic is used to apply coding standards to Perl and check to see which standards are followed.

### 3.4. What we learned

The great thing about tools based on language is that there are already tools built and scripts that can be run to do the analysis on our code to get values that we can use and compare. The downside of this grouping is that we can only compare things that were written in the same language, and we would like to look at services that are developed regardless of the language.

## 4. Quality Grades

The language-based analyses of quality seemed like a good start and one that would work for each specific language, but I was looking for a way to compare services regardless of the language they were written in. While those values generated from language-based analysis will help and are a factor in assessing quality, there must be a way to start comparing all our services.

### 4.1. Testing Quadrants

As a group that is attempting to develop using an Agile mindset, I started doing research on Agile testing and how that might fit into our structure. Crispin and Gregory (2008) put forth the concept of Agile “Testing Quadrants,” which I investigated and tried to apply to the academic setting.

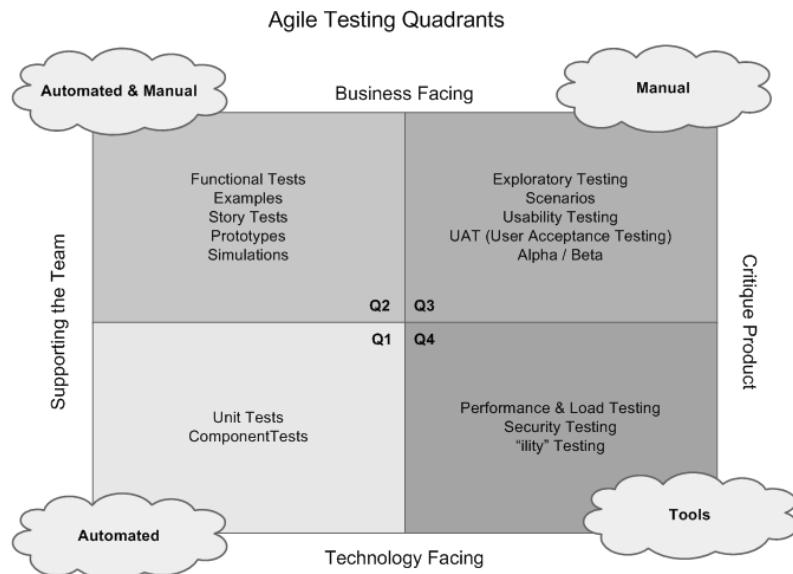


Image Source: <http://agiletester.ca/>

While the quadrants are really guides to help the teams talk about testing, it got me thinking. A lot of our PHP applications teams were beginning to focus on items in Q1 & Q2, specifically, the Unit Tests; they were also starting to get into more Function Tests. Both are great, but if we would test from all 4 quadrants, then wouldn't it be a great piece of software? How could I encourage the development teams to start spending time on in Q3 and Q4 as well? If an application were to be tested in all the Quadrants, it would likely be a high-quality application, but if our application isn't used by many people, is it worth the effort to spend the time to create all the tests?

## 4.2. Testing Maturity Model

Our Vice President of Technology at the time was very focused on Capability Maturity Models (CMM). I was wondering if there were something similar related to testing, and came upon the Testing Maturity Model (TMM). While it was originally developed in 1996 at the Illinois Institute of Technology, it is an idea that is still quite relevant today. The TMM clearly outlines the levels that your testing follows. Each level adds upon the success of prior levels; you can't just jump to Level 4 without first working on Level 2. If an application is in Level 5 of TMM, then some might consider it to be a high-quality piece of software. Like the four testing quadrants, achieving TMM Level 5 may be great—but does everything need to be a Level 5? There needs to be a discussion as to what TMM Level of software is expected.

Since I work at an institution of higher education, we are always talking about grades and how students are always trying to achieve an “A” in a class, but what if a “B” is good enough? A good student could get a “B” if they determined that the cost to get the “A” was too much, or maybe they didn’t need the “A” to maintain a particular GPA. When you combine the ideas from the testing quadrants and convert the concept of TMM into a grading scale, Grading of Services is created.

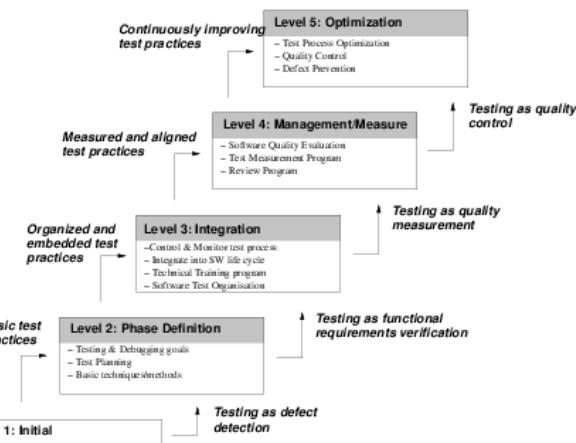


Figure 1: Maturity levels and goals of the Testing Maturity Model TMM.(Moll 2000)

## 4.3. Grading of Services

Since I work in an institution of higher education, I decided to use a grading scale to help us talk about quality. Now, we may always strive for an A, but getting one is not always an achievable goal. There may be other things that are of greater value than getting an A. Maybe it was determined that getting an A was just too hard, so we focused on getting a solid B grade. I've tried to create the chart below as a way for us to discuss the grade of quality that we are striving to get.

Our confidence in our code can be measured in many ways, from a CRAP (Change Risk Anti Pattern) score to Unit Test code coverage. While it is true that we should always try and improve the quality of our code and strive for “perfect, bugless code,” that is not an achievable goal. There are times when the cost to find or prevent bugs outweighs the value of the bug being searched for.

In addition to the confidence level, we also need to gauge the usage of this service. If this Service is going to be used by a lot of users, then we should spend a lot of time on the service to get as good of a grade—and be as confident in our code as we can. But perhaps, if this solution will only be used by a few people, do we need to strive as hard to have high confidence in our solution. Not that it should be terrible and inefficient code, but it shouldn't take inordinate amounts of time and effort to improve it.

I've created this chart to help us visualize the grade of quality in our code. This is by no means a definitive chart or gauge to our level of quality, but two axes of data we can use to help us think about a desired level of quality in our application.

	Small Audience to use Service	Medium Audience to use Service	Large Audience to use Service
High Confidence in Change	A	B	C
Medium Confidence Change	B	B	C
Low Confidence in Change	C	C	D

#### 4.4. What we learned

While this was a good concept overall, we needed more than just letter grades to show our confidence. What went into our confidence grade? We felt like we needed more points of data to help us accurately calculate the confidence. After many discussions with team members and my manager, we came up with the Service Confidence Framework.

### 5. Service Confidence Framework and Process

The Service Confidence Framework describes processes intended to generate conversation and visualizations related to our confidence in sustaining services operated by the department. The framework is intentionally "loose," providing suggested signals to consider when discussing our level of confidence in a service, but not prescribing a fixed set of indicators. The value of this framework is in the conversation and resulting insights.

Our IT department defines a "service" as an application or solution component created, deployed, or operated by our department that directly or indirectly fulfills user needs. This may be an application directly accessed by end users, such as CAS(Central Authentication Service), or it may be APIs that are consumed by other applications. A key characteristic of a service is that we make changes to it, and failures resulting from changes have an impact to users.

We understand that "service" is an overloaded term in our industry. For the purposes of this framework, we suggest a pragmatic approach, with a goal of enabling useful conversation without overburdening the organization. For example, scoring "authentication" as a service is likely to be an effort in frustration, whereas scoring CAS and Shibboleth would be appropriate. Some services may also only make sense in aggregate. When considering Canvas, for example, it makes sense to combine the hosted platform with our data integration scripts.

The Service Confidence Framework describes three factors for a service:

- Context - the impact of the application on the University
- Confidence - our trust in the service reliability and our ability to restore the service in the event of an outage
- Potential impact - combines the context and confidence to represent likely "blast radius" of a disruption

When considered together, these factors provide a useful summary which allows comparison of multiple services, identifies critical services with low confidence, and informs efforts to improve service stability.

There are three primary activities in the Service Confidence Framework process: score, visualize, and act.

#### 5.1. Score

The process of scoring services is not highly prescribed, but it is also not lightweight. The goal is not to simply have a checklist where checking 90% of the boxes gets you 9 out of 10 on the scale. A wide range of signals influence our confidence. Not all apply to every service or have the same weight between

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

PNSQC.ORG

Page 5

services. The framework provides examples of signals to consider when scoring, but the participants should consider any relevant indicators.

The framework requires three scores be generated for each service: context, direct confidence, and complexity. The potential impact value is calculated from these scores. These are then combined by an algorithm to produce the final context and confidence scores. All scoring is done on a scale from 1 (low) to 10 (high).

The framework does not describe who should participate in or the exact procedure for scoring a service. We can offer some general guidelines, however.

- Diverse perspectives are critical, so consider developers, operations, infrastructure, and anyone with responsibility for how the service is developed, deployed, and operated.
- Consider including business stakeholders but beware that they may overestimate the business criticality of services.
- Experience tells us that this scoring conversation is the most valuable part of the process, so ensure an open approach and document the conversation and outcomes.
- Operations of and changes to a service may be different (a service can be very stable except when changes are being made), so consider both while scoring.
- Consider reviewing the scoring outcomes for similar services (if available).

### **5.1.1.Context**

The service context describes the value and availability needs of the service from the user perspective. In some cases, these will be direct (when users directly interact with the service) and in other cases, indirect (the service supports other services). The context of a service can be thought of as factors outside of our control.

Signals to consider for context include:

- Frequency of use
- Number of users per unit of time
- Number of times used per unit of time
- Business function criticality

One of the more interesting signals we should consider is "time to critical." In other words, how long will the business tolerate the service being disrupted? The concept of "time to critical" encapsulates many of the other signals and provides a useful overall measure. However, the "business" must be defined as the institution rather than a single unit.

### **5.1.2.Direct Confidence**

The direct confidence in a service is determined by operational characteristics from development through runtime. Signals to consider for direct confidence include:

- Efficiency of resource utilization
- Automated test coverage
- Resiliency
- Change management
- Documentation (runbooks, operational procedures)
- Observability (monitoring, alerting, logs)
- Positive track record (change success rate, incident rate)

The direct confidence of a service can be thought of as factors we have explicit control over, at least to some extent, and the presence of which is beneficial.

### 5.1.3.Complexity

The complexity of a service represents how hard a service is (to understand its exposure) as well as external factors. Complexity acts as a detractor against direct confidence (more complexity results in lower confidence). Signals to consider for complexity include:

- Code complexity (i.e., CRAP scores)
- Dependencies (number, stability, up to date)
- Influencers (shared services)
- Architecture
- Deployment process (automated, etc.)
- Integrations (diversity, sources, and sinks)
- Contracts with other services

The complexity of a service can be thought of as factors we can influence but not necessarily control.

## 5.2. Visualize

Once scoring is complete, we will record the service scores in a common location so they can be visualized and compared.

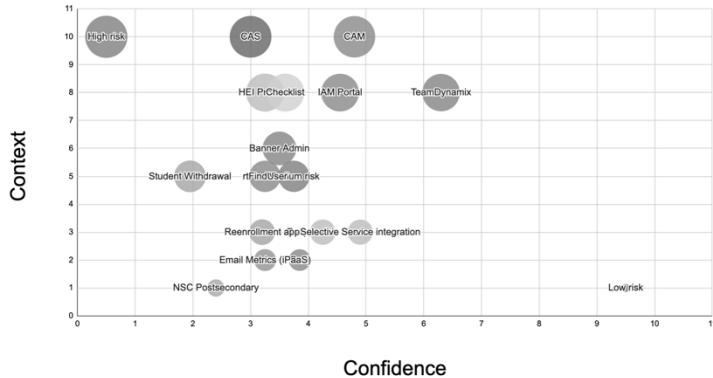
### 5.2.1.Record Scores

The first iteration of the Service Confidence Framework uses a Google Sheet to record scores. A Google Sheet provides a simple, sharable solution while we evolve the process.

Service	Confidence	Context	Potential Impact	Direct Confidence	Complexity
CAS	3	10	97	5	8
Responsible Use	4.25	3	25.75	5	3
CAM	4.8	10	95.2	6	4
Banner Admin	3.5	6	56.5	7	10
Reenrollment app	3.2	3	26.8	4	4
TeamDynamix	6.3	8	73.7	7	2
Grok	3.85	2	16.15	7	9
rtFindUser	3.25	5	46.75	5	7
HEI Process	3.25	8	76.75	5	7
IAM Portal	4.55	8	75.45	7	7
NSC Postsecondary	2.4	1	7.6	4	8
Student Withdrawal	1.95	5	48.05	3	7
Email Metrics (iPaaS)	3.25	2	16.75	5	7
Selective Service integration	4.9	3	25.1	7	6
Checklist	3.6	8	76.4	6	8

### 5.2.2.Plot Services

While the scoring conversation is a valuable way to understand our confidence in an individual service, we gain additional insight by comparing services. The first iteration of the framework is a bubble chart plotting context and confidence on the axis and using potential impact as the bubble size.



### 5.3. Act

With the services scored and visualized, we can gain insight into actions which will reduce risk and improve operational outcomes. We consider two aspects: a "zone" of services in a high-risk situation, and strategies for shifting service confidence.

#### 5.3.1. The "Zone"

Our high-risk "zone" can be considered the upper left quadrant of our bubble chart. That quadrant contains services with high context and low confidence scores, indicative of service at high risk of disruption. Within that zone, we could further refine our view by considering services with a higher potential impact as being of higher priority to address.

We should note that being in the zone, while not desirable, is not in and of itself bad. Some services may simply be of such high context, and our ability to change the direct confidence or complexity so constrained, that it is not practical to move. This should not be seen as a negative, only an indication that we need to take extra precautions with that service.

So, what does it mean to be in the zone? Being in the zone implies a high context, so the business would not tolerate significant disruption, as well as low confidence for changes or operations. In these cases, changes may require extra scrutiny and planning; responsibilities may not "shift left" for operations, or any number of other stipulations to reduce potential disruption.

Since there are consequences for a service in the zone, we should take opportunities to make changes that result in services moving out of the zone when possible. We said previously that we can't really change a service's context, so that leaves direct confidence and complexity as means to move a service. There are any number of ways to improve direct confidence, including improved testing, writing operational documentation, or improving the deployment process. Complexity changes may take more effort, but they also realize significant improvements overall.

#### 5.3.2. Improve

The overall picture of our service confidence scores helps us build a roadmap for improvement. We can assume that increasing our confidence will lead to better outcomes for users and lower operational burden for us. We can use the output of the Service Confidence Framework to identify services we should target for improvement and leverage the scoring conversations to select specific improvement activities.

For example, we may wish to improve our confidence in a service with moderate context. The benefits would be reduced overhead in the change process (moving out of the zone), willingness from the business side for more frequent updates, or any number of things. The conversation around scoring may indicate that the service lacks automated testing and operational documentation, both easily actionable improvements and within the ability of Solution Delivery to carry out.

## 6. What we learned

Utilizing the Service Confidence framework, we are able compare a variety of the services that we support, regardless of the language. We've been able to look at different services that have been created using different languages. We can leverage the Service Confidence scores and discussions to determine where more effort might be needed to improve the confidence of our services.

### 6.1. Conversation

While talking about what the appropriate values for Context, Complexity, and Direct Confidence, often, very good conversations occur. During these conversations, we have learned about documentation that was previously hidden in a department shared drive. When discussing our Direct Confidence of a service and talking through some of the signals, we were able to ask questions to the development team about their service. The conversation brought to light a scenario that the team had not yet thought of. It allowed them to add stories to the project backlog and add features to improve the confidence in their service.

### 6.2. Timing

One thing that really surprised us was that the timing of the Service Confidence discussion can have different results. We were under the assumption that the best time to go over the Service Confidence was towards the end of the update to the service, or when the service was almost complete. In some situations, we had the discussion earlier, and this allowed the team to have a conversation about their design earlier in the process with other stakeholders, which allowed questions to be raised and discussions to be had that would allow the confidence in the service to be improved. But if the conversation were too early, some signals that are discussed might not yet be complete, and the score would be lower at that time. This was very hard for some teams to accept because they felt that the service they were working on should have a higher score than it did.

Given these observations, we are still trying to determine the best time within the development lifecycle to have these discussions.

### 6.3. Overall

While each application is given a confidence score, this is in no way a judgment of the application. Our goal is to map out the services that we provide and help the organization understand the relative strengths and weaknesses that exist. This will eventually help us know where to allocate resources so we can strengthen the services that most need it.

## 7. Conclusion

As I began the position of Quality Analyst at Miami University, I determined that we needed a way to quantify the quality of services that we were developing for use across the university. We were developing services that were very different, yet we needed a way to have all scored in a similar manner. Some of the services that we developed might only be used to help a single office of three to five people. Then there were others that would be used by all faculty and students across the entire University, which might be 23,000 people. With the large variety of both audience and languages, we looked to come up with a way that we could measure the quality of all and compare the confidence we have in each of these services.

By utilizing this Service Confidence Framework, we have been able to compare the services and determine the value of or the necessity of putting more effort into improving the quality of some services, and minimal effort in others. It has been a valuable tool that we will continue to improve upon going forward.

## References

Crispin, Lisa and Janet Gregory. 2008 *Agile Testing: A Practical Guide for Testers and Agile Teams* Addison-Wesley Professional

Jacobs, Jef & Moll, Jan & Stokes, Tom. (2000). *The Process of Test Process Improvement*. 8.

## Acknowledgements

This paper would not have been as well written as it was if it were not for my beautiful wife, Corrine Kidd, who also happens to be my editor. She is the writer in the family, but she has encouraged me and pushed me to finish this paper and made me explain things that just didn't make sense to her. So, with her asking of questions and cleaning up of my sentences this is a much better paper because of her.

I'd also like to thank Dirk Tepe, my manager, who helped form the ideas and thoughts that we have put together to form this Service Confidence Framework into what it is today.

Lastly, I want to thank Miami University, all my co-workers at in the Solution Delivery Group of IT Services, specifically the other members of the 4-Facets, as you have continued to push the use and growth of this framework and are helping us all to see the potential values of using this methodology.

# Software Quality Assurance Methodology for Hybrid Waterfall & Agile Development

**Liu Keping, Eu Felix, Ooi Mei Chen & Peh Wei Wooi**

[keping.liu@intel.com](mailto:keping.liu@intel.com), [felix.eu@intel.com](mailto:felix.eu@intel.com), [mei.chen.ooi@intel.com](mailto:mei.chen.ooi@intel.com),  
[wei.wooi.peh@intel.com](mailto:wei.wooi.peh@intel.com)

## Abstract

Software releases to customers are required to fulfill defined software release criteria to ensure software quality. The software development lifecycle evolves from Waterfall methodology to Agile methodology, which is designed to eliminate various limitations such as scalability and adaptability, while meanwhile enhancing early customer engagement, faster go-to-market, resource optimization, and cost-saving. The incremental turnaround and flexibility of Agile development brings benefits but in parallel brings challenges in project execution. Currently it is not difficult to find that Hybrid Waterfall & Agile methodology are already introduced and frequently used, especially in big platform development.

The software quality assurance methodology for Hybrid Waterfall and Agile Development (HWAM) is used in a situation where multiple software development lifecycles and different quality release acceptance criteria are used in the same program for the same milestone release by different software components. It is intended to assist the team in a large organization to predict issues in ahead, streamline release process workflow, clarify roles and responsibilities, and get team aligned on release criteria. In contrast to traditional software quality assurance approaches which fit for Waterfall methodology, it provides a more agile method to meet the needs of hybrid development methodology and eliminate issues. This paper will identify several challenges that projects often face when adopting the hybrid development methodology and provides workaround solutions based on lessons learnt and best practices in the software industry.

## Biography

*Liu Keping is a Technical Leader in Software Quality Assurance at Intel Corporation based in Shanghai, China. She is a certified CMMI assessor, ISO internal assessor, ASPICE internal assessor, CSQE, and gained 6 Sigma Orange Belt and CPMP certification since 2009. She holds a master's degree in Computer Science and Technology from Central South University in China.*

*Eu Felix is a Software Quality Engineer at Intel Corporation based in Penang, Malaysia. He has held the Lean Six Sigma Green Badge since 2019, certified Software Quality Engineer (CSQE) from ASQ and holds a Degree in Computer Science from University of Bolton in the UK.*

*Ooi Mei Chen is a Software Quality Engineer at Intel Corporation based in Penang, Malaysia. She holds a Degree in Computer Science from University Tunku Abdul Rahman*

*Peh Wei Wooi is a Platform Validation Lead at Intel Corporation based in Penang, Malaysia. He certified as the ISTQB tester and holds a Degree in Information Science from UKM, Malaysia.*

*Copyright Liu Keping, Eu Felix, Ooi Mei Chen, Peh Wei Wooi 2022*

# 1 Introduction

In this introduction, we will discuss what software release and software quality assurance are. We will outline the high-level overview of the software development life cycle, describe the Waterfall and Agile software development lifecycles, introduce the Hybrid Waterfall & Agile Development (HWAD) and Software Quality Assurance Methodology for Hybrid Waterfall and Agile Development (HWAM). HWAD and HWAM will be used in the paper to simplify the description.

In Section 2, we will discuss the potential challenges in HWAD, and how those can be rectified. Section 3 is built upon Section 2 and provides the HWAM solution details for HWAD.

Section 4 describes our experiments and implementation results obtained for HWAM. Section 5 covers the gap analysis and continuous improvement by fine-tuning a better way to improve the efficiency of the HWAM. Finally, in Section 6, we summarize and provide directions for future work and areas to research.

## 1.1 Software Release and Software Quality Assurance

A software release is the distribution of the newest or latest version of software to the end-users publicly or privately. Alpha, Beta, PV, engineering release (ER), and Hotfix (HF) are the common terms used for release milestones, covering major, minor, or specific emergency defect fixing releases, depending on business needs.

Software quality assurance is a systematic software practice used to monitor and control the processes and work products to comply with defined standards and meet software release quality targets.

Software quality assurance includes process qualification and product qualification as shown in Figure-4. It supports the delivery of high-quality products and services by providing the project stakeholders at all levels with objective insight into the processes and associated work product quality throughout the product development lifecycle.

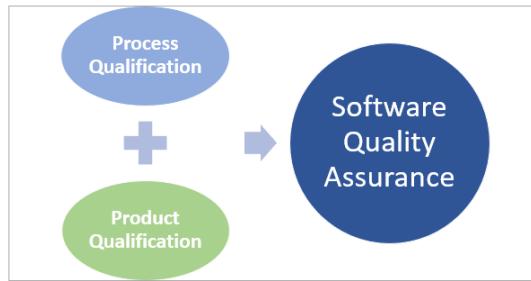


Figure-1 Software Quality Assurance

Software quality assurance is done via evaluation against predefined criteria by an independent organization. Software quality assurance should begin in the early phase of a project to establish plans, processes, standards, and checklists.

## 1.2 Software Development Lifecycle

The software development life cycles (SDLC) and their process models are high-level representations of the software development process. These models define the stages (phases) through which software development moves and the activities performed in each of those stages. Each SDLC model represents one software project, iteration, or increment, from conception until that version of the product is completed and/or released.

Waterfall and Agile are the 2 typical software development lifecycles used in industry.

### 1.2.1 Waterfall Software Development Lifecycle

The Waterfall Model is the first model to define a disciplined approach to software development as shown in Figure-1. It is a breakdown of project activities into linear sequential phases, where each subsequent phase depends on the deliverables of the previous phase. The work products produced in one phase in the waterfall model are typically the inputs into the subsequent phases. The premise of the Waterfall Model is that a project can be planned before it is started and that it will progress in an orderly manner throughout its development. In general, some parts of our industry have interpreted the waterfall model as being a purely sequential lifecycle model, with no feedback loops or iterations, but Winston Royce's [4] original recommendations on the waterfall model are that it includes iteration and feedback loops between life cycle phases.

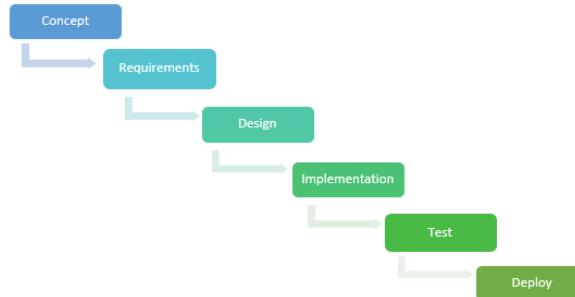


Figure-2 Waterfall Model Software Development Life Cycle

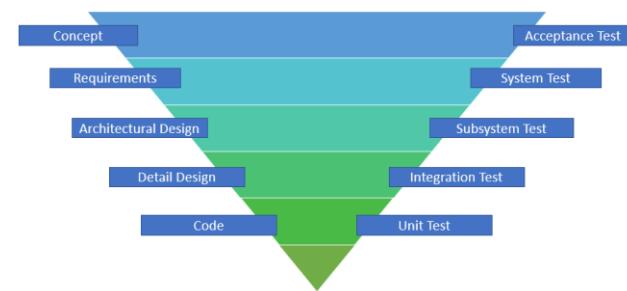


Figure-3 V-Model Waterfall Model Software Development Life Cycle

The V-model is a variation on the Waterfall Model as shown in Figure-2. It highlights the relationship between the testing phases and the products produced in the early life cycle phases. For example, once a sizable number of product requirements are defined, system test planning and design can be started.

### 1.2.2 Agile Software Development Lifecycle

The Agile software development lifecycle is a feature-driven development methodology. It is a software development model where steps or activities are repeated multiple times as shown in Figure-3. This may be done to add increased details to the requirements, design, code, or tests, or it may be done to implement small pieces of new functionality, one after another.

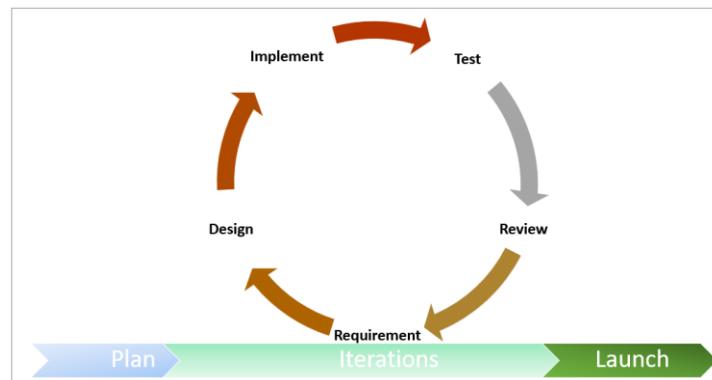


Figure-4 Agile Software Development Life Cycle

Agile is about being responsive quickly to the market/customer's needs and demands and being able to change direction as the situation demands. Agile methodology is a method to manage a project by splitting it up into several SDLC phases. It requires continual collaboration with stakeholders and constant refinement at each stage. Once the work begins, the team runs through a process of planning, executing, and evaluating. Instead of betting everything on a "big bang" launch, Agile delivers work in small increments. Requirements, plans, and results are evaluated continuously.

## 2 Hybrid Waterfall & Agile Development (HWAD)

In many large product development, Waterfall and Agile software development life cycles are used at the same time in different software components within a big platform project, which is called Hybrid Waterfall & Agile Development (HWAD). It is fit for those programs that cannot be satisfied by either Waterfall or Agile development lifecycle alone. For example, a software development that has a silicon dependency on ongoing hardware development plus many feature requests that require customer confirmation or are still in the Proof of Concept (POC) stage.

There are two specific terms used in this paper that requires attention: Platform and software components. “a platform” mentioned in the paper is equal to “the integrated system of a big project”. “software components” mentioned in this paper equals “the software components, services, or middleware running on the platform”.

A platform has multiple software components within it, while a platform could turn into a software component for another platform when it becomes a base for others, as shown in Figure-5.



Figure-5 Hybrid Waterfall and Agile Development (HWAD) Structure

The HWAD consolidates the finest of both methods and provides the opportunity to apply different software development life cycles in the same project. Undeniably HWAD contains the advantages of both Waterfall and Agile development methodology and overcomes many of the limitations of the individual models, however, it also creates new challenges in parallel.

Challenge	Description
CH1: Schedule Alignment	<p>One significant issue encountered in HWAD is the schedule misalignment between different parts. Typically, the schedule between software components and software platform, and the plan between hardware and software.</p> <p>Misalignment on hardware and software development schedule could lead to release delay. On the other hand, not having working software available for system testing until late in the hardware life cycle can lead to hardware defects that are not detected or resolved until late of the life cycle, increasing cost.</p> <p>Misalignment on platform and software component development schedule could lead to release delay as well. Sometimes the misalignment is caused by the software component schedule changed and the platform is not aware of that.</p> <p>See Section 3.1 and 3.4 for solution details.</p>
CH2: Software release acceptance criteria alignment between platform and software components	<p>Another significant issue encountered in HWAD is the release acceptance criteria misalignment between platform and software components.</p> <p>One significant issue that often happens in big platform execution is that: multiple software development lifecycles and different quality release acceptance criteria are used in the same program for the same milestone release by different</p>

	<p>software components. It will cause the problem that platform criteria could not meet at the release readiness review.</p> <p>In HWAD, Waterfall and Agile development methodology are hybrids used by different software components in the same project. In parallel, different software component teams are coming from different organizations. This means different teams could have different quality assurance plans/release criteria working alongside with software quality assurance engineer of that team (tailored to suit their needs). As Waterfall software release criteria are not 100% fit for Agile software release, how to accomplish software release criteria compliance in HWAD? Are we going to use the same criteria to qualify all software components including the one using Agile? Anything could be optimized and what can be reused?</p> <p>See Section 3.2 and 3.3 for solution details.</p>
CH3: Resistance to change and role clarification	<p>Both Waterfall and Agile model are used in HWAD. For the teams that using Waterfall model before, they are asked to transform from the Waterfall model to HWAD model in a short time. People will feel uncomfortable with the sudden change when they are not well trained in the Agile process and are not familiar with the new user roles in the Agile model. They tend to resist the change.</p> <p>Also because of the fast adoption from the Waterfall model to the Agile model without proper training provided, it causes Agile related activities not to run in an efficient way, like the product backlog and sprint backlog prioritization, sprint planning, daily scrum, sprint demonstration and sprint retrospective.</p> <p>See Section 3.4.1 (the key action: Refine Project Management Plan to fit for HWAD) on how to eliminate this in general.</p>
CH4: Big feature evaluation and planning	<p>Feature size is too big to be planned and finished within one sprint so as could not get valid customer or validation feedback for each sprint.</p> <p>See Section3.4.1 (the key action: Refine Project Management Plan to fit for HWAD) on how to eliminate this in general.</p>

Table-1 Challenges in HWAD

### 3 Software Quality Assurance Methodology for Hybrid Waterfall and Agile Development (HWAM)

The HWAM is specially dedicated to HWAD to ease the situation for a program when:

- There are multiple software development life cycles.
- There are different quality release acceptance criteria with different software components.

The HWAM helps to align the project schedule, software release acceptance criteria and leveraging the art of Agile to do software release qualification.

#### 3.1 Align on Schedule

In most projects, software and hardware have their own dedicated implementation schedules. Considering hardware dependency as a key factor when making project software development plans can help to prevent scheduling issues.

##### Best Practices:

- Establish formal communication channels to synchronize hardware and software on schedule, scope, and resource, with which to identify the ideal point at which hardware and software can be integrated.
- Perform risk analysis and mitigation plans to narrow down the impact of hardware.
- Align hardware testing to Agile 'Iterations' as close as possible to get hardware function to be tested in a timely manner.
- Create a dependency matrix to align platform and software component release schedule. Make sure software component release can meet platform release target. Get software components team representative commitment to the release schedule. See Figure -6 for example.

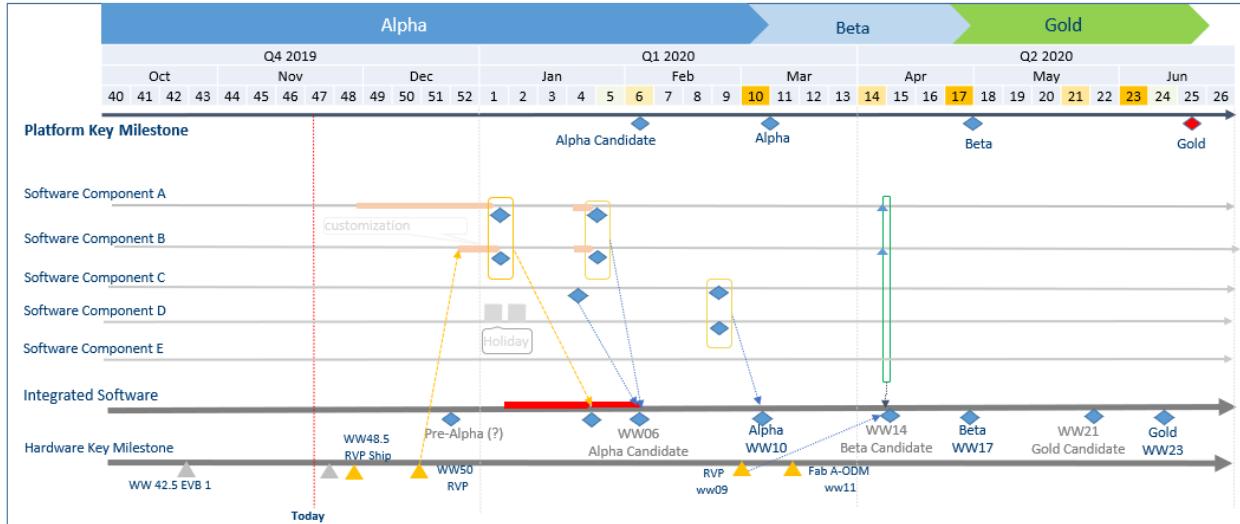


Figure-6 Dependency Matrix

### 3.2 Align on Software Release Acceptance Criteria

A good software quality assurance plan is to ensure process and work product quality assurance is performed at the project level independently and objectively, perform the quality assurance activities according to quality assurance strategy and project schedule to meet defined requirements and goals. Aligning a software quality assurance plan at an earlier stage can help to eliminate the challenges mentioned in section 2.1 and section 2.2.

#### Best Practices:

- Strategically decompose and redefine the software release criteria, so that they can meet the needs of both agile and waterfall methodologies. See section 3.3 for details.
- Involve software component quality representative to review platform software quality assurance plan. Make sure the software component quality assurance plan is aligned with the platform software quality assurance plan without conflict.

### 3.3 HWAD Software Release Acceptance Criteria

Software releases to customers are required to fulfill defined software release criteria to ensure software quality.

Traditional Waterfall methodology is based on the belief that the future is predictable.

Agile is with a light process and fewer documents. Product real-time demonstration and retrospective meetings are used to get customers' earlier engagement, improve customer satisfaction, and achieve high quality targets. Software release compliance qualification done by an independent software quality

team is not as important as Waterfall. The most popular used Scrum is a typical Agile process framework for managing Agile projects. Scrum is based on the idea that people can manage themselves and the future is unpredictable. The best we can do is to make the most intelligent adaptations to it.

Using Agile methodology brings the ability to develop high-value and high-priority software more quickly and increase return on investment. However Agile is more suitable for small or medium programs that have less than 50 people. How to set the software release compliance criteria for the big projects using HWAD?

**Best Practices:**

- Map a certain number of sprints to Waterfall milestones and released it as a major version. Others will be treated as intermediate releases like engineering releases. A major version can be triggered once a big feature is complete or several big features are integrated. See Figure 7:

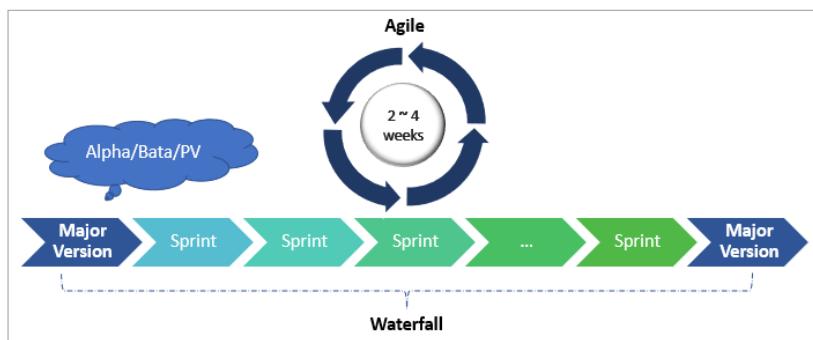


Figure-7 Hybrid Waterfall and Agile Development (HWAD) Software Release Planning

- Create 2 sets of release qualification criteria packages: one for Major Version Release and another for Intermediate Sprint Release.
- For each Major Version Release, apply standard Waterfall software release qualification criteria. Typical software release qualification checkpoints covered in a standard Waterfall development is shown in Figure-8:

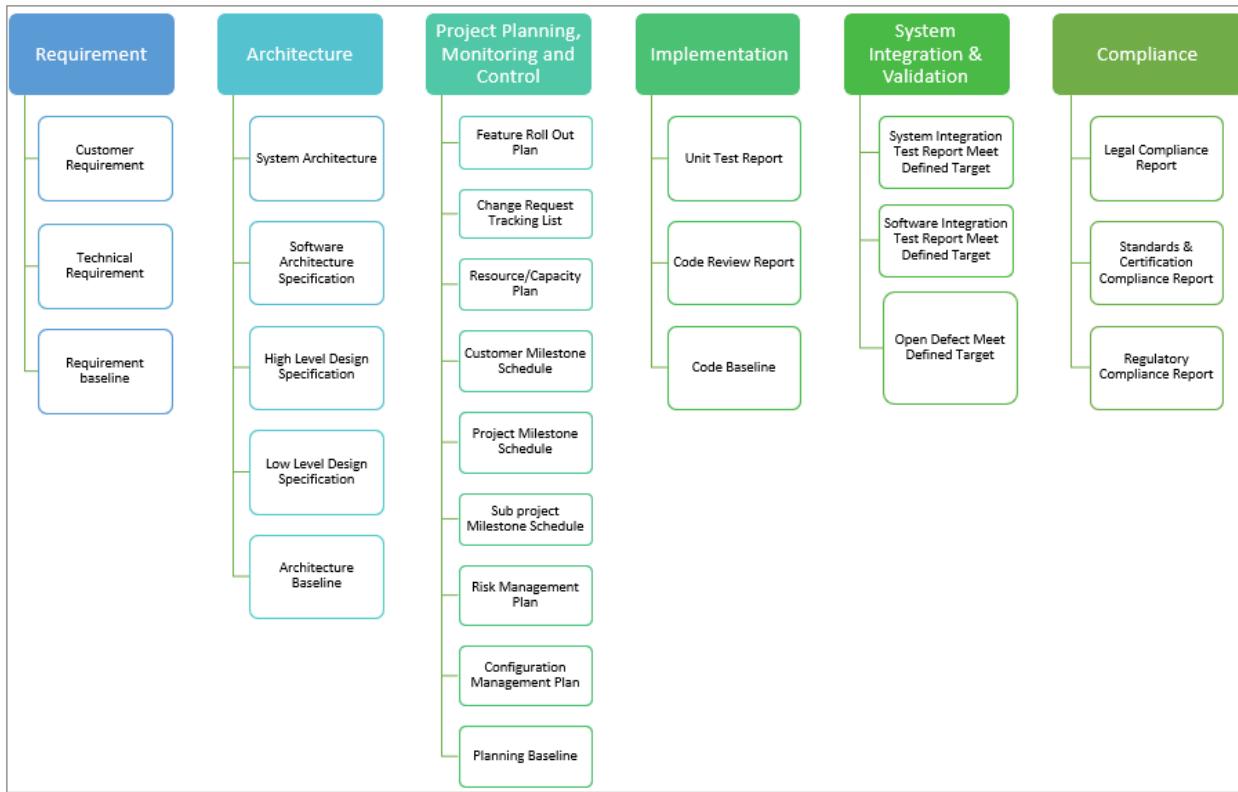


Figure-8 Hybrid Waterfall and Agile Development (HWAD) Major Version Software Release Criteria

- For each Intermediate Sprint Release, strategically decompose and reform software release criteria to meet Agile methodology needs. Typical software release qualification checkpoints are covered in Intermediate Sprint Release as shown in Figure 9.

**Requirement:** Requirements in waterfall projects are often expected to be essentially complete before design commences, whereas, in Agile methodologies, requirement readiness typically increases sprint by sprint. It will always be noncompliant if sticking to Waterfall's 100% requirement complete. Shifting the idea of requirement complete at the entire project requirement base to sprint requirement base will resolve this issue.

**Architecture:** Low-level design specification is required, as Intermediate Sprint Releases focus more on software component release - apply the same rule as Requirement. Others like system architecture, software architecture specification, high-level design specification, and architecture baseline could be skipped as they have already been covered in Major Version Release.

**Project Planning, Monitoring & Control:** The feature roll-out plan should be mapped to the sprint backlog. Resource/Capacity should be mapped to sprint capacity. Others like project milestone schedule, risk management plan, configuration plan, and planning baseline will be skipped and checked in Major Version Release only.

**Implementation:** Same as Waterfall.

**System Integration and Validation:** Checkpoint is same as Waterfall but uses pre-production targets like Alpha or Beta. Project team could determine themselves based on business needs.

**Compliance:** Checkpoint is same as Waterfall but will not be gating – as they have been covered in Major Version Release. The idea that still checking them in Intermediate Sprint Release is trying to mitigate the risk that a big gap found in Major Version Release.

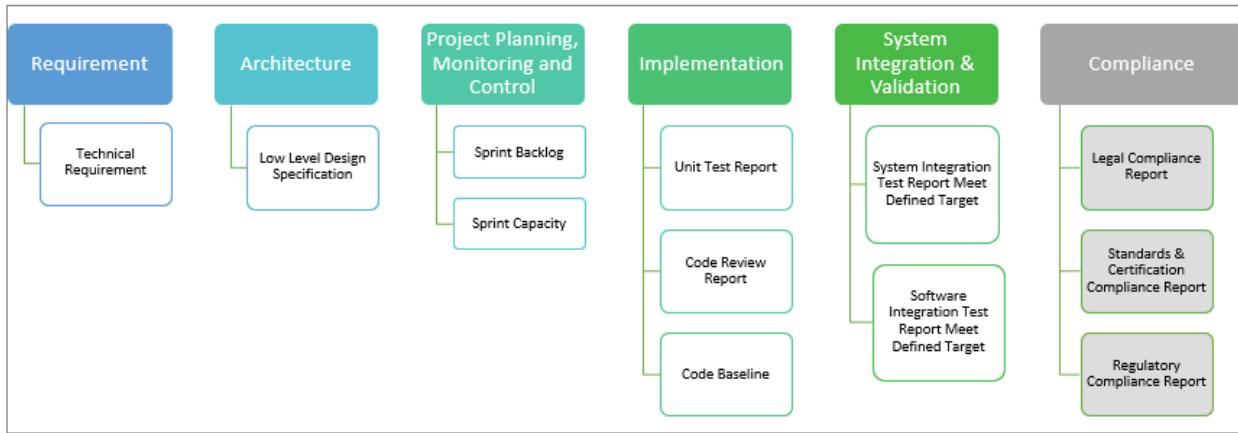


Figure-9 Hybrid Waterfall and Agile Development (HWAD) Intermediate Sprint Release Criteria

### 3.4 HWAM Process Workflow, Roles and Responsibilities

As mentioned in section 1.1, process qualification and product qualification are the 2 key important factors of Software Quality Assurance. The sections above talked about the best practices that can be taken into consideration when applying HWAD.

Here we would like to talk more on how to integrate those things together and form a process, identify clear roles and responsibilities, with which we could achieve a repeatable result. See Figure-10 below for details.

#### 3.4.1 HWAM Process Workflow

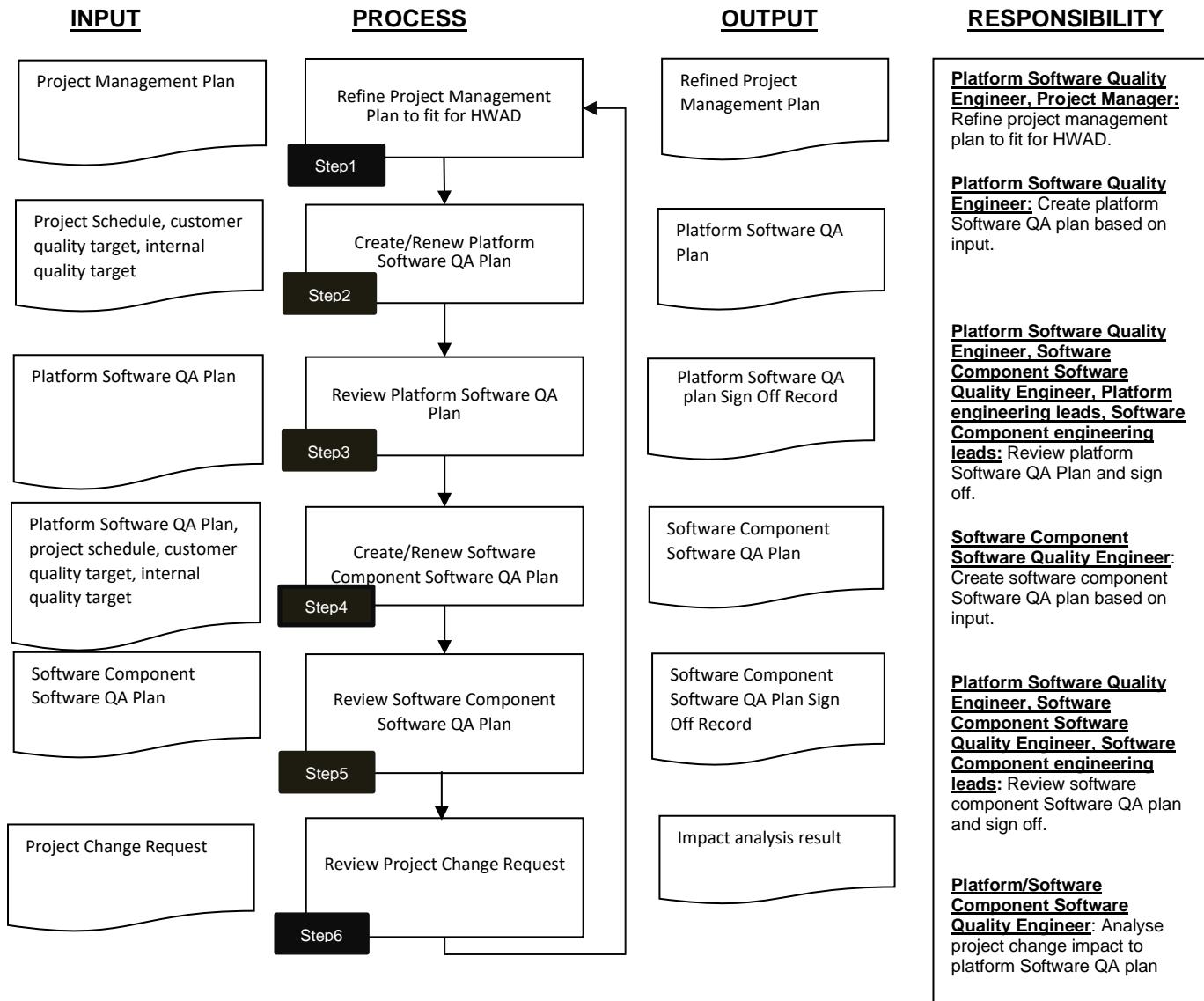


Figure-10 HWAM Process Workflow, Roles and Responsibilities

### 3.4.2 Process Activity Structure

Besides the process flow, it is suggested to describe each process step in detail, which helps on guiding the engineering team on process execution. It is suggested to include the following elements: Purpose, start criteria, input work products, responsible people, detail action list, output work products, exit criteria, and work instructions.

As Step1 is a very important activity within the HWAM process flow, which addresses the major challenges of HWAD, details are shown below as an example:

<b>Activity</b>	<b>Refine Project Management Plan to fit for HWAD</b>
Purpose	The purpose of this activity is to refine the Project Management Plan to fit for HWAD.

Activity	Refine Project Management Plan to fit for HWAD
Start criteria	Project Management Plan in place
Input work products	<ul style="list-style-type: none"> <li>• Project Management Plan</li> </ul>
Responsible	<ul style="list-style-type: none"> <li>• Platform Software Quality Engineer, Project Manager</li> <li>• Add a formal evaluation task in Microsoft Project Planning (MPP) at the beginning of a project to see if the Agile development methodology was fit for the current project. (CH3)</li> <li>• Add Agile development methodology (E.g., Scrum Master, Product Owner, etc.) training to the project training plan. (CH3)</li> <li>• Add big feature evaluation and planning training to the project training plan. (CH4)</li> </ul>
Action list	<ul style="list-style-type: none"> <li>• Create software component and platform dependence matrix to align software release schedule. (CH1)</li> <li>• Create hardware and software dependence matrix to align hardware and software release schedules. (CH1)</li> <li>• Create a formal communication channel to synchronize hardware and software on schedule, scope, and resources. (CH1)</li> <li>• Perform risk analysis and mitigation plans to narrow down the impact of hardware. (CH1)</li> <li>• Add retrospective meetings for each major milestone in MPP (CH2).</li> </ul>
Output work products	Refined Project Management Plan; Refined Microsoft Project Planning
Exit criteria	Refined Project Management Plan; Refined Microsoft Project Planning in place
Work Instruction	Hybrid Waterfall and Agile Execution Checklist

Table-2 Key Action: Refine Project Management Plan to fit for HWAD

## 4 Implementation Results

We applied this methodology and found that the hardware and software delivery schedule trend converged by the 8<sup>th</sup> sprint, shown in Figure 11.

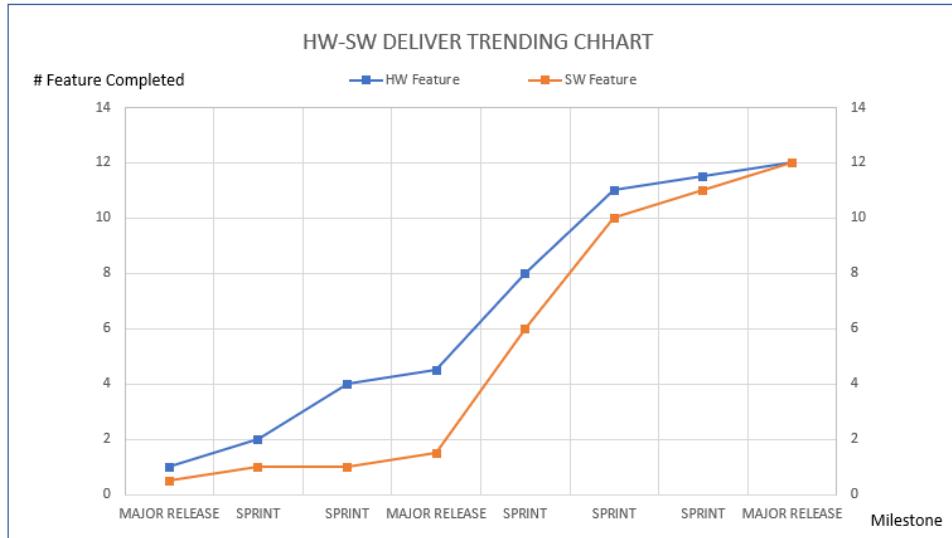


Figure-11 HW-SW Deliver Trending Chart

In parallel, the release cycle decreased, the human resource spent on the entire project reduced accordingly, and customer satisfaction increased. For instance, given a platform with 25 software components program as an example shown in Table 3:

Implementation	Result (Before)	Result (After)	Improvement
Release cycle and Human resource	SW 26 platform releases within 1 year HW 6 releases within 1 year	12 platform releases with both working HW and SW within 1 year	The release cycle and human resource usage is decreased by 62% respectively
Customer evaluation	On average, customer validation feedback was received twice, and feedback is collected late in the delivery lifecycle.	Customer validation feedback was collected $\geq 6$ times throughout the product lifecycle (because of the aligned HW and SW schedule), including early, mid, and end of the delivery period.	Customer feedback received improved $\geq 300\%$

Table-3 Implementation Result Evaluation

## 5 Gap Analysis and Continuous Improvement

Applying the software quality assurance methodology for Hybrid Waterfall and Agile execution proves beneficial as mentioned above. Table 4 shows the areas which could be further improved.

Gap Analysis	Continuous Improvement
Sometimes it is hard to map HW and SW features to get a good schedule alignment.	Use a unified tool to track both HW and SW features in the same location. E.g., Using Jira which has the burndown chart report embedded.

Sometimes HW schedule cannot fit for SW schedule.	Plan additional SW releases to fit for HW release schedule. E.g., the HW B0 release has aligned the SW Alpha release, but the next HW B1 release does not have a corresponding SW release with it.
Process KPIs (key performance indicators, KPI) are not formally defined and measured.	Define formal process KPIs to monitor and control efficiency and quality, identify process gaps, and improve the process. E.g., considering execution velocity, feature completion, customer evaluation, effective communication...etc.

Table-4 Gap Analysis and Continuous Improvement

## 6 Conclusion

In this paper, we analyzed various challenges that often exist in Hybrid Waterfall and Agile Development (HWAD) delivery and proposed recommendations to resolve a variety of challenges, including hardware and software schedule alignment impediments, platform & software component schedule alignment, change resistance and lack of clarity around roles, big feature evaluation and planning...etc.

We introduced the software quality assurance methodology for Hybrid Waterfall and Agile development (HWAM) by aligning on project schedule and software quality assurance plan, leveraging the art of Agile to do software release qualification, defining process workflow with detailed role and responsibility to integrate all best practices together to achieve a repeatable result, and describing how to well define an action within a process.

One of the important factors is to refine the project management plan to fit for HWAD, which is extremely important to ensure program success. Strategically decomposing and reforming software release criteria to meet Agile methodology needs is another key factor to make the program successful.

Hybrid Waterfall and Agile execution has evolved with the evolution of software development life cycles. How to leverage the benefits of both models and how to deal with the complex issues encountered in the execution is a long-term topic. With continuous lessons learned and best practices summarized and shared, we will be confident to predict and prevent issues ahead, reduce risks and meet customer delivery needs.

## References

1. The Certified Software Quality Engineer Handbook, Second Edition, Linda Westfall
2. CMMI, Guidelines for Process Integration and Product Improvement, Mary Beth Chrissies, Mike Konrad, Sandy Shrum
3. Agile Software Development with Scrum, Scrum FAQ by Ken Schwabe
4. Managing The Development of Large Software Systems, Winston W. Royce,  
<http://agileconsortium.pbworks.com/w/fsele/fetch/52184636/waterfall.pdf>

# Test Case Prioritization Using Hybrid Deep Learning Approach

**Loo Willam, Yip Kin Choy**

**Intel Microelectronics Sdn. Bhd.**

willam.loo@intel.com  
kin.choy.yip@intel.com

## Abstract

Exhaustive testing and coverage are not effective methods to perform the validation in manual execution, automation, and continuous integration and development environment. In the current mode of work, we encountered a backlog of execution tasks for up to three weeks instead of the planned one-week duration for completion. Therefore, this research will prioritize test case prioritization which allows the validation engineer and automation framework to execute the validation operatively.

To ensure the validation effectiveness of reducing redundancy of the test cases, a deep learning method and greedy approach to selecting the best combination of test cases that have complete validation coverage will be proposed. The reduction will impact significantly the project cost in terms of resources catered.

These results suggested that using a hybrid approach will expedite the progress by approximately 20%. From the organization's point of view, this proficiency results in cost savings and quality products for the consumer to use.

## Biography

*The primary author is Loo Willam, a validation engineer who works at Intel Penang. His technical expertise is in Validation Methodology, pre-silicon, and post-silicon validation. He did involve validation field for about 6 years from pre-silicon, post-silicon, and software to platform validation. He graduated with a bachelor's degree in Computer Science, majoring in Software Engineering in the year 2014, and currently, he is undertaking a Master of Software Engineering at Universiti Malaysia Pahang (UMP).*

*The co-author is Yip Kin Choy (KC Yip), a principal engineer working at Intel Penang.*

## 1 Introduction

Exhaustive testing and validation on a broad market IoT (Internet of Things) product are not effective from coverage and time to market perspective what-more when this is performed in manual test configuration where development work and continuous integrations are continuously evolved in regular cadence. In the current mode of operation, we encounter a backlog of validation tasks up to as much as three weeks while it was all supposed to be completed in a week. Hence, this paper focuses on research work to develop a smart algorithm to prioritize test cases according to the incoming changes. It allows validation engineers and the automation community to stage the execution more operatively and predictably.

To deliver an effective, lean, and agile approach to validation coverage, a deep learning methodology, recurrent neural network along with the approach named “greedy” approach is being deployed. This is done to reduce the number of test cases and in doing so, selectively pick and choose the best possible combination of test cases that should suffice the incoming requirements. In IoT test qualification, it's critical that the validation encompasses all possible usage scenarios and in that mimicking the customers' end applications for the intended use cases model. In this scenario, running an NVR (network video recorder) application would demand execution of over  $n$  possible combinations of different pipelines and workloads to accommodate the utilization of a digitized surveillance system coupled with secured in-band manageability features. The deployment of deep learning helps to predetermine the selection of test cases based on a set of inputs corresponding to the intended validation coverage. This in turn significantly improves the execution duration, reduces cost in resource allocation, and shortens the overall product qualification cycles.

The outcomes of such a hybrid approach have broken the traditional approach of validation and broken the norm of constant and repetitive execution by introducing smarter solutions in determining the right set of test cases according to the defined requirement. This has expedited the completion by approximately 60% by fully utilizing the software-defined "greedy" approach while delivering top-quality focused validation according to the customer's requirement and use case model.

The hybrid approach comprises deep learning on self-learning on the git log from the repository. We will perform data acquisition and pre-processing of the commit id, author name, and commit date. On the training dataset side, we will put a label on the commit messages for numerous sizes of input. When we are working on the live action, after we obtained the predicted output, we have a query out from the database, and we need to filter out the relevant test cases. To optimize the test coverage, a greedy approach will implement to prioritize the test cases for the full test coverage. The greedy approach is the method that seeks the minimum test cases which will cover several requirements. Hence, with the hybrid approach, we can reach the objective of test case prioritization.

## 2 Literature Review

This literature review will be divided into a few subcategories, which are test case prioritization, test coverage, and deep learning on software validation-related works.

### 2.1 Test Case Prioritization

Yanshan et al [1] mentioned that the behavior pattern of the test will influence the prioritization of test cases in the group of test cases in the pool. Other than that, they realized that the additional effort by applying the greedy method to eliminate the duplication of the test cases, indirectly will augment the

execution process in the manual execution, continuous integration, and content development (CICD), furthermore with software quality gating process.

Besides that, they are using deep neural network models which include a maximum of ten hidden layers and four different types of adversarial attacks, examples are FGSM, JSMA, decision-based Gaussian Noise, and Uniform Noise to generate different sets of the test case pools. In the set of neurons  $N = \{n_1, n_2, \dots, n_p\}$  and the test set  $T = \{t_1, t_2, \dots, t_q\}$ . Given the number of neurons of  $n_i \in N$  and the number of the test set of  $t_i \in T$ , the activation function will fall into the Boolean state and its function will be  $af(n_i, t_i)$ . Here is the state for the activation function, where  $t$  is the limit to determine the states

$$af(n_i, t_i) = \begin{cases} 1, & \text{if } af(n_i, t_i) > t \\ 0, & \text{otherwise} \end{cases}$$

The neurons' attitude for the test  $t_j \in T$  is the array. Following is the expression of the definition, where  $p$  is the Boolean state of the  $p$ th neuron.

$$B(t_i) = [af(n_1, t_i), af(n_2, t_i), \dots, af(n_p, t_i)]$$

From the test set  $T$ , the behavior pattern,  $BP$  can conclude as a mean array:

$$BP(T) = \frac{\sum_{t_i \in T} B(t_i)}{|T|}$$

To calculate the distance between each test data, for example,  $T_1$  and  $T_2$ , it is measured by L1 norm distance:

$$distance = \sum_{i=1}^p |bp_i(T_1) - bp_i(T_2)|$$

From the result outcomes in terms of the distance in their experiments, we can observe that the test set with the closest distance will form a cluster but some of it will fall in the intersection area between clusters. The combination of the Greedy approach will further enhance the test case prioritization from the cluster.

Aizaz et al. [2] discussed that previous researchers had coded one test case prioritization using an automatic history-based approach where it was able to detect as many regression faults as possible and fit it into the execution phase for a specific duration.

In the test suite with the given test cases  $T_S = \{t_1, t_2, \dots, t_n\}$  and  $ExecutionStatus(ES)$  of a test case  $t_i$ :

$$ES_{(i,j)} = \begin{cases} 1 & \text{if } t_i \text{ failed cycle } j \\ 0 & \text{if } t_i \text{ passed cycle } j \\ -1 & \text{if } t_i \text{ not execute at cycle } j \end{cases}$$

The historical test case execution result for previous  $m$  cycles,  $p$  is the calculated test priority for the upcoming  $m + 1$  cycle,  $d$  will be the average execution time during the  $m$  cycle.

$$HistoryTestData = \sum_{j \in 1..m} \frac{d(t_{(i,j)})}{m}$$

In the Deep Order approach, the test prioritization uses history test data execution status and test duration as the baseline. To optimize the priority, Deep Order implements a dual-objective function:

$$g(\max(f), \max(|T|))$$

The main aim is to make the best use of the fault detection ability of the test suite, and the subsequent aim is to fully utilize the number of executed tests in the given timeframe.

To reflect the higher impact of most failures, a particular test case priority value  $p(t_i)$  is gauged as follows:

$$p(t_i) = \sum_{j \in 1..m} w_j \times \max(ES_{(i,j)}, 0)$$

The annotation of this equation is

- $w_j$  is allocated weight in the range of Boolean value to every cycle  $j$ , for instance,  $\sum_j w_j = 1$ .
- $m$  is the number of past cycles,  $p(t_i) \in (0, 1)$
- $\max(ES_{(i,j)}, 0)$  used to eliminate no-run test where  $ES_{(i,j)} = -1$ .

In the deep neural network (DNN), the activation function is using Mish's activation function in the hidden layers of the neural network instead of a normal activation function like ReLU, and this new activation function is intended to overcome its flaws of it.

$$f(x) = x \times \tanh(\text{softplus}(x)) = x \times \tanh(\ln(1 + e^x))$$

In Reinforcement Learning for Test Case Prioritisation paper [3], we are clearly understanding that test prioritization depends on the number of cycles, feature records, and the optimal ranking. Aizaz et al. [2] shared the same thought that the elements above have the closest relationship to the test case prioritization. At the optimum ranking, they derive the ranking function of  $n!$  distinct sequences are given  $n$  test cases,

$$\begin{aligned} & \forall t_1, t_2 \in T, \\ & \text{idx}(s_0, t_1) < \text{idx}(s_0, t_2) \leftrightarrow t_{1.v} > t_{2.v} \text{ or} \\ & (t_{1.v} = t_{2.v} \text{ and } t_{1.e} \leq t_{2.e}) \end{aligned}$$

## 2.2 Test Coverage

There are a few solutions to solve the test coverage issue, the most efficient way is by maximizing the requirement coverage [4] when we are performing our test plan. In the validation world, individual test cases would be able to cover more requirements and if it does cover the critical requirement, that test case will be shortlisted and marked as the golden test case.

$$fit_{req}(TS) = \sum_{i=1}^{|TS|} |\{\text{Req}(tc_i) \mid tc_i \in TS\}|$$

Pairwise testing or t-way testing [5] is the method where we specified at least a pair of test requirements as our exhaustive pair and other associated test requirements are free to match with it and duplicate pairs will be eliminated from the table. With the implementation of the greedy approach and t-way testing, we are ensuring that combinational testing will have complete coverages with the most minimal test cases.

## 2.3 Related Works

In the year 2021 at IEEE International Conference on Software Testing, Verification and Validation Workshop (ICSTW), Eran et. al. [6] proposed the QUADRANT approach. With their designed system architecture as shown below, this approach helps for automated test case generation erstwhile the whole kit and caboodle perfectly at software fault prediction.

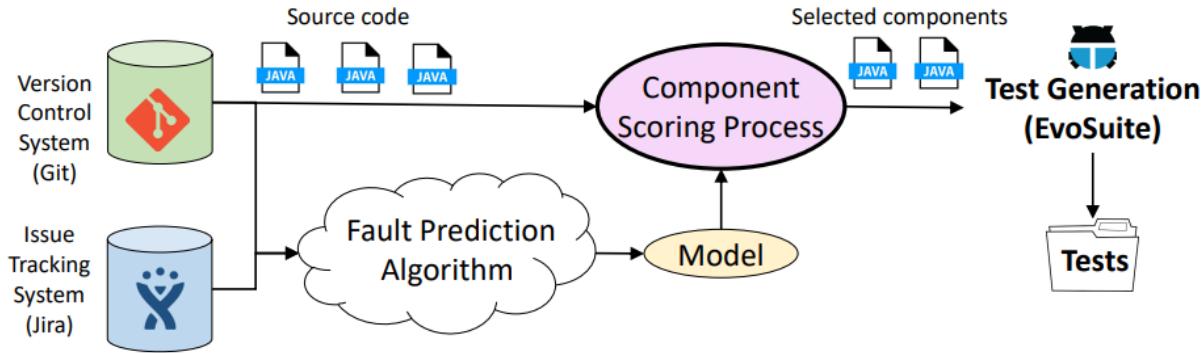


Figure 1: System Architecture with QUADRANT approach

Software test validation does involve manual execution and continuous integration (CI). Test cases will be residing in various repositories including Git, and Tortoise SVN. With incorporation into the bug sighting database, deep learning will take these as input so that they will be fed into the component scoring process. The outcome will be in the form of the test suite.

There are few heuristic ways to contrivance it. The component scoring process will list as follows:

- Scoring with Fault Prediction (FP)
- Scoring with Lines of Codes (LOC)
- Scoring combination in FP and LOC. The formula for this combination is below.

$$FP_\alpha(C) = \alpha \cdot FP(C) + (1 - \alpha) \cdot LOC(C)$$

Indicates that  $\alpha$  is the parameter in  $[0, 1]$ , component scoring function,  $C$  of score  $FP_\alpha(C)$ .

In the research carried out by Kai et. al., [7] they are using a deep neural network (DNN) as the tool for performing the test case prioritization. Generally, in the concept used by YanShen et. al. [1], the key difference between them is the percentage of fault detected.

$$APFD = 1 - \frac{\sum_{i=1}^m w_i}{n \times m} + \frac{1}{2n}$$

Given that the test suite denoted by  $T$  with  $n$  number of test cases,  $m$  errors can be detected in the error set of  $F$ . Assumed that  $w_i$  be the number of test cases required to execute for  $i$ -th of errors found. The range of metrics will be in the range of 0 to 1.

## 2.4 Deep Learning Related Works in Software Test Validation

In the past section, we performed the analysis of fellow researchers with their published papers. Some benefits which can be adopted from it. On the other hand, there are a few points that have room for improvement, just that we are not planning for implementation in this research. There will be the pros and cons discussion as follows: -

With the combination of recurrent neural network (RNN) and deep neural network (DNN) methods, RNN will aid in text classification obtained from the git repository and provides the category that the git commit belongs to. Through this method, DNN plays an important role in test case selection and prioritization.

The greedy approach implemented in DeepOrder [2] helps in test case redundancy. With the redundancy and yet maximize the requirement coverage takes place, for example, 1000 test cases in the pool, with this approach we can reduce and consolidate up to 70% to 85% reduction. Test case prioritization incorporated with the greedy approach will rank the closely related and the highest coverage test cases to be recommended for the automation framework and validation engineers in software quality gating and test case execution.

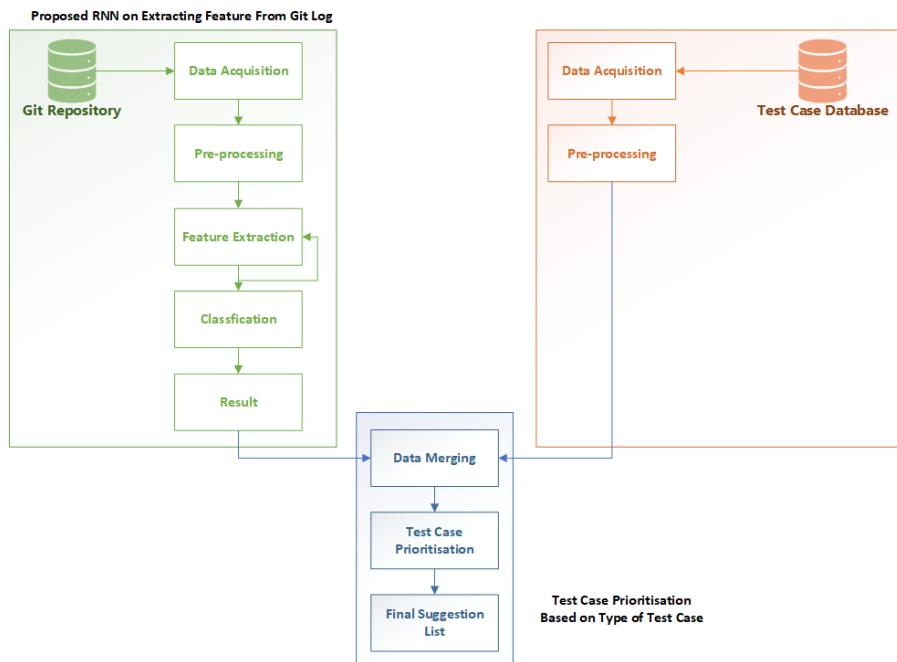
In combinational t-way testing, we know that we are capable to perform exhaustive testing with lesser pair of test cases and selective pair of test cases in the test case pool. Besides that, by combining a few test redundancies methods, we can suggest to the validation engineer or the automation framework to execute their test suite and the time taken will be greatly reduced.

### 3 Proposed Test Case Prioritization Model

This section will explain the structure of the overview of the proposed test case prioritization model.

Figure 2 demonstrates the proposed design and the methodology of this research to solve our issues in our software test validation at Intel Malaysia. Fundamentally, this methodology will divide into three portions, proposed recurrent neural network (RNN) on extracting features from git log, data acquisition and pre-processing, and test case prioritization based on the type of test case.

In addition, this model design is divided into two phases, the training phase, and the testing phase. In the training phase, we will ensure that our design model can suggest a list of the test cases with high similarities to our expected list of test cases by analyzing manually. After several iterations of model training, we will have a certain level of confidentiality to proceed to the testing phase. In the testing phase, we will obtain a live log from the git repository and an updated test case from the database. The outcomes will pass to validation engineers and automation framework via email and triggers execution respectively. Figure 3 will provide the context of the flow for the proposed approach in Software Validation in Intel Malaysia.



*Figure 2: Proposed Approach for Software Test Validation in Intel Malaysia*

```

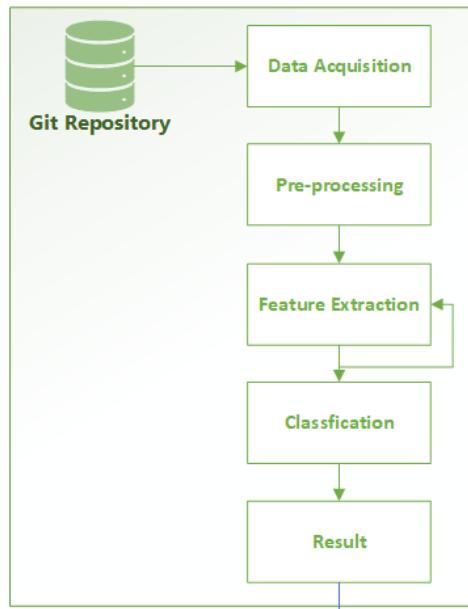
1 BEGIN:
2
3     // Predict git commit category
4     Input : Obtain git log
5     FOR i=0 in range total input:
6         Get commit message
7         Remove stopwords
8         Feed into RNN model
9         Get predicted output
10    ENDFOR
11
12    // Obtain test cases from test case database
13    Query from database
14    Save into variable A
15
16    Select relevant test cases based on predicted output
17
18    FOR j=0 in relevant test case size
19        Search essential requirement
20        Map to test case
21        Search requirement covered in particular test case
22        Save info in variable suggested_test_case
23    ENDFOR
24
25 END

```

*Figure 3: Pseudocode for Proposed Approach*

### 3.1 Proposed RNN on Extracting Feature from Git Log

In this sub-section, we will discuss in deep about the processes involved with the recurrent neural network as shown in Figure 4, the main reason that we planned to implement this neural network due to our input data in text format.



*Figure 4: Proposed RNN Framework*

### 3.1.1 Data Acquisition Phase

Our data is coming from the git repository. The git log consists of the commit hash of the commit, branch header, branch tag information, author of the commit, date, and the commit messages.

The prerequisite is we need to clone a branch from a repository. Inside the git folder, we collected the log and exported it into the text file. An example of the log will present in Figure 5. In a real-case scenario, every commit will trigger the process. Next, we will proceed to the data pre-processing phase.

```
commit fb488a8b8523fa544a8b71c6ab700fe72c0087b7 (HEAD, tag: <branch tag>, <branch_header>)
Author: Authour_name <author_email@intel.com>
Date:   Sat Feb 12 11:21:23 2022 -0700

    fixing GPIO in value

commit fb488a8b8523fa544a8b71c6ab700fe72c0087b7
Author: Authour_name <author_email@intel.com>
Date:   Sat Feb 12 11:21:23 2022 -0700

    UART Tx internal loopback content enabling
```

Figure 5: Sample Log File from Git

### 3.1.2 Data Pre-processing Phase

In this section, after we obtained the log file, it will parse over to this phase. At this phase, we will perform the metadata removal, like commit hash, author name and email, header tag and branch name as well as the date of the commit.

Once this process had been completed, we will acquire the string of the commit message, we need to consume the string into the sentence filtering process where in this case, we just filtered out the punctuations if any. Thus, after filtration is completed, we must serialize every word into an array as an input neuron for the input layer.

### 3.1.3 Feature Extraction Phase and Classification Phase

In this section, we will apply the recurrent neural network (RNN) to extract the features of the sentence array.

The RNN formula is formulated as follows where  $h_t$  represents the current state at period  $t$ ,  $fw$  represents the function with the parameter  $w$ ,  $h_{t-1}$  shows the previous state of the RNN, and  $x_t$  represents the input vector at  $t$  timestamp.

$$h_t = fw(h_{t-1}, x_t)$$

At the end of this phase, we will proceed to the next phase, the classification phase. In this phase, we need to undergo a dense approach to produce the correct classification.

In a conclusion, Figure 6 illustrates the overall structure of the RNN from the input phase to the classification phase with examples.

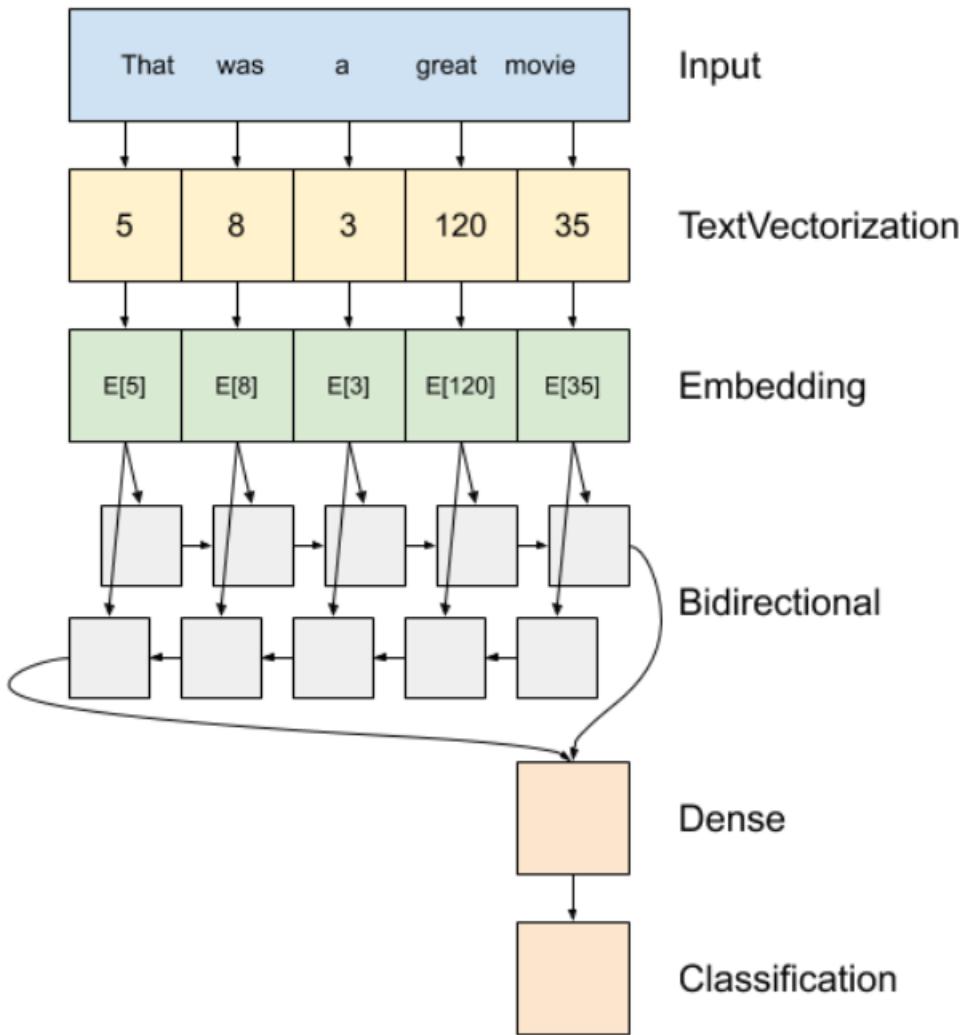


Figure 6: Overall Example Structure of the RNN in detail

### 3.2 Data Merging Phase

After querying all test cases from the database, we performed the data merging. In this phase, we will filter out the test case pool based on the necessary category. Once this process is completed, we recommended proceeding to the next phase. The equation below shows the notation on the method that we used to filter out the test cases pool.

$$FTC_i = TCP_i \cap C_i$$

In this equation,  $FTC$  represents filtered test cases and  $TCP$  represents the test case pool,  $C$  represents the category label we obtained and  $i$  represents the iteration in the *for* loop.

### 3.3 Test Case Prioritization

In this section, we will reduce and prioritize the test cases based on the required coverage. At the end of this section, suggested test cases based on the required coverage will be presented and forwarded to the automation framework and validation engineers kickstart their execution.

For the test case prioritization, we will implement the test case reduction method, which is a Greedy algorithm to remove the redundancy of the test cases and prioritize the test cases by maximizing the coverage of the requirement. From the pseudocode in Figure 3, we are going to drill down the process that we will implement. Equation following is the method that is going to execute.

$$TC_i = \sum_{i=0}^n \left( \sum_{j=0}^p REQ_{(i,j)} \geq 1 \right) \leq \frac{1}{2} n$$

From this equation,  $TC$  shows that the test cases, given that  $TC = \{TC_0, TC_1, \dots, TC_{n-2}, TC_{n-1}\}$ ,  $REQ$  represent the requirement row, given that  $REQ = \{REQ_0, REQ_1, \dots, REQ_{p-2}, REQ_{p-1}\}$ ,  $i$  and  $j$  are the coordinates of the test cases versus to the requirements and  $n$  and  $p$  are the total row and column numbers in the test requirement matrix.

## 4 Result and Discussion

This section will focus on evaluating the performance of the developed algorithm in the software validation process. In the dataset preparation due to the company's private and confidential data in the git log, the test case requirement and the respective test cases definition in the portal, an open-source dataset from Kaggle, which is the customer complaints spreadsheet, and the sample draft of the test requirements versus to the test cases in the CSV format.

### 4.1 Text Analysis Experiment

The customer complaints spreadsheet illustrated in Figure 7, contains 1,324,194 complaints, and here is the sample data snapshot captured from the CSV file. From the screenshot, the product column is the final decision of the complaints that the customer had made.

Date received	Product	Sub-product	Issue	Sub-issue	Consumer	Company	Company	State	ZIP code	Tags	Consumer	Submitted	Date sent	Company	Timely res	Consumer	Complaint ID
#####	Mortgage	Other mort	Loan modification, collection, foreclosure		M&T BANK MI				48382	N/A	Referral	03/17/201	Closed with Yes	No		759217	
#####	Credit reporting	Incorrect i	Account st I have outt	Company	TRANSUNIAL				352XX	Consent pt Web		#####	Closed with Yes	No		2141773	
10/17/201	Consumer	Vehicle loc	Managing the loan or I purchased a new car	CITIZENS F PA					177XX	Older Ame	Consent pt Web	10/20/201	Closed with Yes	No		2163100	
6/8/2014	Credit card	Bankruptcy		AMERICAN ID					83854	Older Ame	N/A	Web	#####	Closed with Yes	Yes		885638

Figure 7: Sample Snapshot from Customer Complaint Spreadsheet

In the list of customer complaints, there are eighteen categories, and the test program will query out the statements of the complaint and remove the stop words and punctuation. After that, it will get padding and populated into the matrix for the LSTM algorithm to consume the matrix. The LSTM algorithms will analyze the customer complaints and it will move to the dense layer which will decide the predicted output.

Once the predicted output is obtained, the result will be parsed to the database for querying out the necessary test cases and tabulated in the form of the list.

## 4.2 Test Case Experiment

When section 6.1 is completed and the test case list had been exported in the text form, it concluded as a requirement test matrix.

In the requirement test matrix (RTM) as shown in Figure 8, a few sets of matrices had been prepared with numerous requirements and test cases listed in there. It does represent the actual test cases that going to execute by the validation engineers in Intel Malaysia. Due to the company's private and confidential information, this research is unable to use the real data for validation, and hence, draft copies of the RTM have been used for demonstration purposes.

```
t0=1:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:1
t1=0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0:0
t2=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t3=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t4=0:0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0
t5=0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0
t6=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t7=0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0:0
t8=0:0:0:0:0:0:0:0:0:0:0:0:1:0:0:0:0
t9=0:0:0:1:0:0:0:0:0:0:0:0:0:0:0:0:0
t10=0:0:0:0:0:1:1:0:0:0:0:0:0:1:0:0:0
t11=0:0:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t12=0:0:0:1:0:0:0:1:0:0:1:0:0:0:0:0:0
t13=0:1:0:0:0:0:0:0:0:0:0:0:0:0:0:1:0
t14=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t15=0:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t16=1:0:0:0:0:0:0:0:1:1:0:0:0:0:0:0:0
t17=0:0:0:0:0:0:0:1:0:0:0:0:1:0:1:0:0
t18=0:0:0:0:1:0:0:0:0:0:0:0:0:0:0:0:0
t19=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
```

*Figure 8: Test Case in The Form of List*

## 4.3 Result Presentation

This section will present the result obtained from the text analysis from section 6.1 and the accuracy with the respective time taken from section 6.2. Therefore, these results will separate into two sub-categories, text analysis, and test case prioritization respectively.

### 4.3.1 Text Analysis Portion

From the outcome in section 6.1, the ideal accuracy for the test data is approximately 88% as the overfitting scale had set to 0.2 meanwhile the ideal loss accuracy for the test data falls at about 32%. Time taken for the training process is about 40 minutes as the epochs are set to 8 and the batch\_size of the LSTM algorithms is set to 128 as the default recommended value needs 7 minutes and 30 seconds for each epoch. The overall train and test data had been plotted in the form of the graph as shown in Figure 9 and Figure 10.

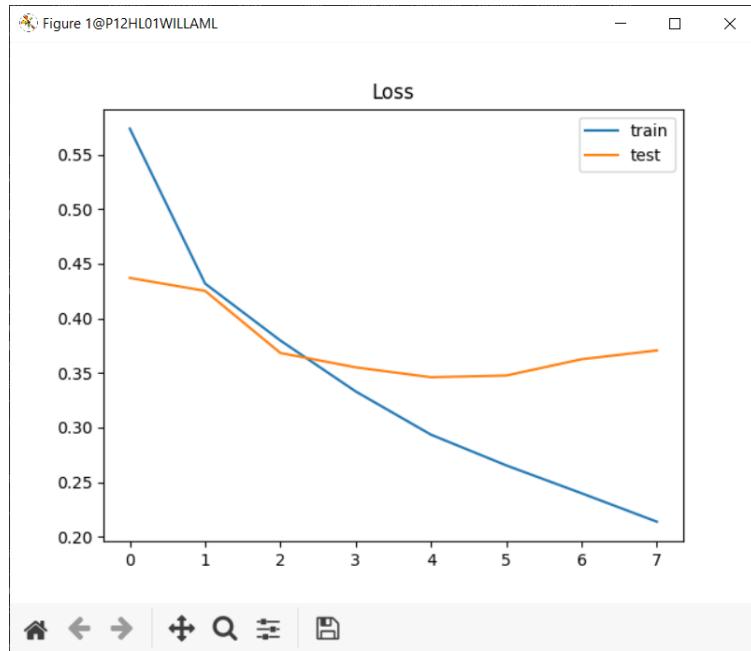


Figure 9: Loss Between Train and Test Data Training

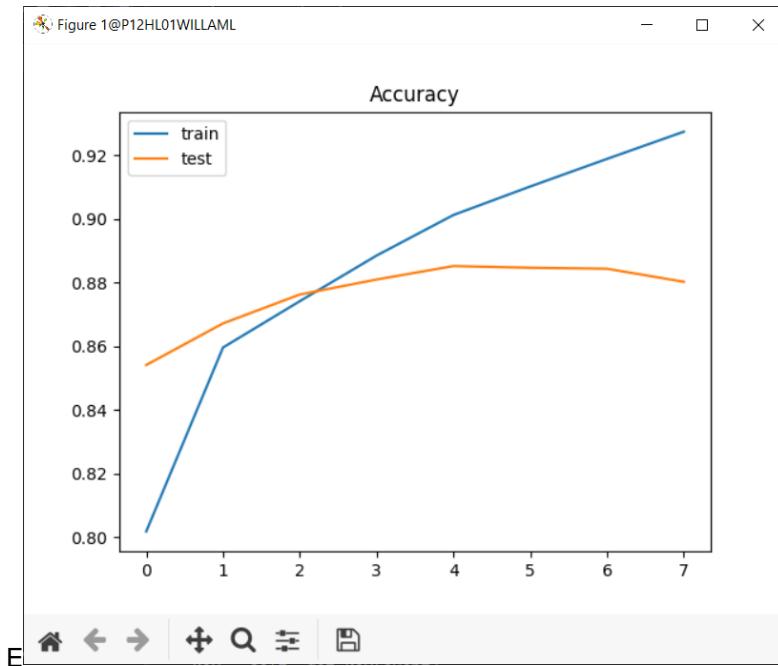


Figure 10: Accuracy Between Train and Test Data Training

### 4.3.2 Test Case Prioritization Portion

From the output in section 4.3, two sets of requirement test matrix (RTM) which comprises the 30 test cases and 51 test cases respectively with the fixed number of the requirements, from  $REQ_0, REQ_1, REQ_2, \dots, REQ_{n-1}$  where  $n = 16$ . Detailed information will tabulate in Table 1.

*Table 1: Detailed Information of the Test Case Reduction*

Requirement Test Matrix (RTM) Number	RTM #1	RTM #2
Original Test Case Number	30	51
Reduced Test Case Number	12	9
The total reduction of test cases in percentage (%)	60.000	82.353
Time Taken (seconds)	0.00081	0.00087

## 4.4 Discussion

This section will perform the discussion and findings that the result collected from section 6.3. The discussion will separate into two sub-sections which encompass the text analysis and the test case prioritization.

### 4.4.1 Text Analysis

In this sub-section, a detailed discussion will be held on the computing efficiency, algorithm, and the result comparison with the previous researchers that had completed the Literature Review section.

In this research, python is used as the programming language compared to C programming which can execute the code by fully utilizing the threads in the compute processing unit (CPU). Besides that, the LSTM algorithm in the recurrent neural network (RNN) will consume more computing power from the multithreading mechanism. Due to the limitation of python programming, each epoch is taking about 5 minutes and 20 seconds with the batch size set to 128. Meanwhile, the default value for the batch size is 64 where it will take about 7 minutes to run per epoch.

From the algorithm perspective, this research uses the Long Short-Term Memory (LSTM), an advanced version of the prevailing Recurrent Neural Network (RNN) with the formula mentioned

$$h_t = fw(h_{t-1}, x_t)$$

In LSTM, it is easier to memorize the past input in the memory. Overall, LSTM is well-suited for cataloging, dealing out, and envisaging time series in the provided time lags with the unforeseen timeframe. In the context of results obtained in Figure 9 and Figure 10, it clearly shows that when the relationship between the loss and the accuracy is reverse proportional to each other. Due to the limitation mentioned in section 6.1, the graph presented was not as ideal as expected.

From the result comparison perspective, the loss and accuracy are highly dependent on the dropout and the recurrent dropout value, which falls between 0 to 1 in the LSTM. The purpose of the dropout and the recurrent dropout is to prevent the overfitting situation from happening. In the current research prototype, the value used is 0.2 respectively for both variables. This value is the most suitable value to overcome this situation. For the training data, we are using the categorical cross-entropy loss function and the activation function wise, SoftMax activation energy was chosen to have a clear graph that consists of the value from value 0 to 1.

#### 4.4.2 Text Analysis

This subsection will converse about the result obtained and compare it with the method used by the researchers deliberated in Literature Review section.

In this research for test case prioritization, by using the formula stated as follows,

$$TC_i = \sum_{i=0}^n \left( \sum_{j=0}^p REQ_{(i,j)} \geq 1 \right) \leq \frac{1}{2}n$$

and the result obtained in sub-section 6.3.1, the test case prioritized is based on the coverage of the requirement for the specific test case. In the greedy approach, the last occurrence of the requirement being covered will be prioritized. Thus, the redundant test cases for the overlapped requirement will be consolidated by the test case which covers the greatest number of requirements. As JPCT performed better than the SPCT [6] carried out by the researchers from Japan and compared the result with the JPCT and the proposed enhanced Greedy algorithm proposed in this research, JPCT can cover 73% of the coverage while the proposed enhanced version of Greedy algorithm covers 100% of the requirements listed with the minimum test cases needed.

## 5 Conclusion

The key objective of this chapter is to epitomize the verdicts of the recurrent neural network (RNN) on the text analysis of the log file in the form of the CSV format and the enriched Greedy algorithm whereby this approach does help the performance of the execution at the end of the day to accomplish the goals and get to the bottom of problems avowed in the problem statements.

Moreover, this research work also provides a detailed analysis of the efficiency of the algorithm that impacted the validation execution progress. The advanced stratagem is aided to ensure the robustness of the set of rules implemented towards the validation coverage and LSTM model training. The objectives and problem statements of this research can be achieved and solved. This project had contented the purposes as follows:

1. To evaluate and determine a deep learning algorithm that is suitable for the software validation process in Intel Malaysia.
2. To develop the determined deep learning algorithm for the software validation process in Intel Malaysia.
3. To evaluate the performance of the developed algorithm in the software validation process.

## 5.1 Research Contribution

### 5.1.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is part of the recurrent neural network (RNN) for the text and character analysis for the result prediction. This technique can learn the knowledge provided from the input and if the previous input is not relevant to it, the algorithm will not memorize it and vice versa.

By denoting to Deep Learning Model article [8], they will be pre-processing their plain input and removing the stop words plus punctuation, data vectorization, building models from the training sets, predicting the test data then populating the accuracy. Aside from that, researchers presented the data or results obtained using long and short sentences as input are very close to the results obtained in this research with the limited number of epochs.

### 5.1.2 Enhanced Greedy Algorithm

Typical Greedy algorithm help in optimizing the coverage but the efficiency wise is not optimum. With the enhanced Greedy algorithm implemented in the previous chapter, it does speed up the execution speed yet had full coverage of the test cases concerning the requirements.

## 5.2 Limitations

This research has some limitations which will be jot down in the following points.

1. LSTM has will drag the performance of the CPU and GPU and it will affect the epoch cycle.
2. Greedy algorithm unable to perform the mix-and-match test cases prioritization while maximizing the requirement coverage.
3. Company's private and confidential git log and test requirement for the internal project is prohibited to use in this research. Hence, an open-source database, named customer complaints had been used in this research.)

## 5.3 Future Works

Several perfections can be suggested for the Validation Test Case Prioritization Using Hybrid Approach for the future to boost the functionalities. The suggested enhancement is shown below:

1. LSTM can train with multiple epochs with better hardware and software requirements.
2. Advanced Greedy algorithm able to incorporate with the pairwise testing which will help in removing the test case redundancy that happened either from Greedy algorithm or pairwise testing.

## 6 References

- [1] Yanshan Chen, Ziyuan Wang, Dong Wang, Yongming Yao, and Zhenyu Chen, Behaviour Pattern-Driven Test Case Selection for Deep Neural Networks, 2019, IEEE International Conference on Artificial Intelligence Testing (AITest) page 89, 90.
- [2] Aizaz Sharif, Dusica Marijan, and Marius Liaaen, DeepOrder: Deep Learning for Test Case Prioritisation in Continuous Integration Testing, 2021, IEEE International Conference on Software Maintenance and Evolution (ICSME) page 525-534.
- [3] Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel Briand, Reinforcement Learning for Test Case Prioritization, 2011, arXiv:2011.01834v [Online Journal]
- [4] Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer, Multi-Objective Black-Box Test Case Selection for System Testing, 2017, GECCO 2017, pg1311-1318.
- [5] Eran Hershkovich, Roni Stern, Rui Abreu, and Amir Elmishali, Prioritized Test Generation Guided by Software Fault Prediction, 2021, IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), page 218-225
- [6] Yuper Lay Myint, Hironori Washizaki, Yoshiaki Fukazawa, Hideyuki Kanuka, and Hiroki Ohbayashi, Test case reduction based on the join condition in pairwise coverage-based database testing, 2018, IEEE International Conference on Software Testing, Verification and Validation Workshops, page 239-243
- [7] Kai Zhang, Yongtai Zhang, Liwei Zhang, Hongyu Gao, Rongjie Yan, and Jun Yan, Neuron Activation Frequency Based Test Case Prioritization, 2020, International Symposium on Theoretical Aspects of Software Engineering (TASE), page 81-88.
- [8] JingJing Cai, Jianping Li, Wei Li, Ji Wang, Deep Learning Model Used in Text Classification, 2018, 978-1-7281-1536-8/18/\$31.00 ©2018 IEEE, page 123 – 126.

# A Method to Select Tests Based on Code Coverage

**Jack Marvin and Trevor Hammock**

[Jack\\_Marvin@mentor.com](mailto:Jack_Marvin@mentor.com) [Trevor\\_Hammock@mentor.com](mailto:Trevor_Hammock@mentor.com)

## Abstract

We all know that to discover software defects, tests need to be run. But how many tests are “good enough”? Many companies do full regression suite runs to validate new software versions. Depending on your situation, this may be fine. If you are part of a start-up, where there is a high rate of code change and potentially fewer tests, it may be preferred. But if you are QA for a mature product with many tests and/or limited resources, you might wonder about the value of running all the tests when little or no changes have occurred, and the turnaround is costly. If you decide to run fewer tests, there are pros and cons. You can save on compute resources, but you may not detect a defect as early, especially if your tests cover another teams’ code and they lack coverage. And if another team is your internal customer and their functionality becomes broken, they will not be pleased.

There are some available tools that can help with determining which tests cover code changes. Jest<sup>1</sup> for JavaScript, pytest<sup>2</sup> for Python. They have their strengths and weaknesses.

There is often the fear ‘what if we miss something’? What about side-effects? Who owns tests upstream and downstream from me in a flow?

The application described in this paper addresses these concerns. From the changes that are detected what should be tested is determined with risk reduced to tolerable levels. The result is a system that you can be confident will provide proper coverage with the fewest number of tests.

## Biography

*Trevor Hammock is a Software QA/Developer at Siemens, with 5 years of testing experience ranging from automation to C++ and python for test suite development.*

*Jack Marvin is a Software QA Engineer at Siemens, with 20 years of experience in automation, exploratory testing, and performance testing.*

## 1 Introduction

This paper is intended for those that run regression test suites regularly and want to run less tests while maintaining coverage, which can benefit any testing interval such as hourly, nightly, or weekly. Given that there is a finite number of resources shared across all teams, and some teams or builds may get higher priority, a reduction in tests run may allow the same level of quality with less resource contention.

The system we will describe to accomplish this task is written in a publicly available coding language (Python) and uses a publicly available application (Gcov<sup>3</sup>) to provide data. The system is named DTS for Dynamic Test Selector. DTS creates a list of tests by determining what code changed and which tests cover those changes.

The act of reducing tests, while providing large run time benefits, imposes some risk of not detecting a defect as early as in a non-DTS system; possibly creating a reduction in confidence by others on the team. These risks may cause enough fear to reduce the adoption rate, especially for teams with a high rate of code change. Fears can be reduced by continually providing a backstop of running as before at a desired interval, possibly increased as confidence is increased.

One example of running tests with low value is when the developer is on vacation. Another is the code change is non-detectable (a change to a comment or a style change). These runs often occur when automation is created to run every build. DTS can easily be inserted into that automation to exclude running those tests.

## 2 DTS Process

DTS uses the idea of mapping a code change to what tests cover that change, which tells us what tests need to be run and which can be deferred to run later. Additional metrics can also be used to calculate which tests should be prioritized over others, greatly increasing efficiency.

The mapping is determined by a code-coverage tool. We use Gcov<sup>3</sup>. There are other tools available, some require a license. Gcov is a component of GCC, the Gnu Compiler Collection, which is the default compiler toolset shipped with the Linux operating system.

### 2.1 Steps:

1. A Gcov<sup>3</sup>-instrumented build is done where the instrumented source code is the code that a team wants to track.
2. The regression test suite is run with that Gcov build.
3. A database is constructed that maps tests to functions.
4. A tool finds the changed functions in the build to be tested and determines the tests that cover those functions. Some optional filtering can be performed to further reduce the test set.
5. The regression suite is run as usual, except for selecting only the tests affected.

### 2.2 Benefits of Running Less Tests

There are many benefits of running fewer tests. The most pronounced being faster approval of code changes, especially if you are competing for resources. In our fast-paced environment with constant pressure to deliver results, achieving testing completion sooner is a plus

Some other benefits include a possible reduction of hardware (computer, network, and disk space), fewer fails from the same defect and less false fails from the testing system or hardware incurring costly analysis time.

### 2.3 Other Benefits of DTS

While a reduced test load is the main priority, DTS provides some other benefits having to do with code and test analysis. The test analysis piece is mostly for how much tests overlap or test case effectiveness.

For instance, DTS can determine how much code a test covers, which may be useful for determining which tests can be removed and not affect quality. There is also an opportunity for test case creation training. An example is whitebox testing where we know X feature covers Y function, so we create a test that covers Y.

DTS can also identify which functions have a large impact. i.e., a function that impacts a large portion of the test suite. This can help determine which functions are more important, providing more data for DTS refinements. If you know a function is covered by many tests, you can choose tests that overlap functions and leave out ones that do not.

You can use DTS as an automated way to map/categorize tests to a given feature, providing another view of which tests are important when a feature changes.

Impact of changes can accurately be quantified. DTS can verify a statement that nothing changed, or the change should only affect one product, being more accurate than human guesses and assumptions.

The tool can aid detection of tests that won't fail or have a high likelihood of not failing. If over time, tests are not selected and did not miss a defect, it may be likely that the test cannot fail. Investigation can determine if the test is poorly constructed or useless.

Early feature evaluation builds can be proven adequate quickly by any team member besides development and QA. For example, a person on the marketing staff can prove runnability prior to build delivery.

Code churn (how often and how much code changed) can be measured to indicate release stability. DTS determines source code differences. It could report those to describe code churn.

### 2.4 Limitations and Risks

The Gcov<sup>3</sup> database is only as good as the build the tests were run with. Future changes and new tests will not be known until the next scheduled run. There is a trade-off since Gcov runs can be 20X the normal run time, meaning you cannot do a Gcov run simultaneously with the code changes. But a short interval between a Gcov build with new code changes and test runs reduces the opportunity for missing a defect.

And proving future changes not affecting current features is valuable. That is, no regressions. Depending on your role and where you are in the release cycle, it may not be a concern. If you are a developer, you may not have many tests created by QA. If you are QA, it might be early in the release cycle where you have not yet had the time to create tests. Another option is if you do have newer tests for new features, you can choose to add a selection from them to your DTS-created test list.

Which tests comprise an optimal set? What is optimal? There are trade-offs of test coverage with run time. Is the runtime of one long running test, with a higher chance of finding a bug, better than running a larger number of quick tests, that are less likely to find a bug? DTS only has one narrow algorithm of optimal test selection and clients could benefit from multiple types of selection algorithms, where the needs of the situation dictate what algorithm is preferable.

Another option if your clients expect a delivery without these failures being missed is to choose to select all the tests that cover a change instead of removing duplicates.

For whom is DTS meant for? If you have a small number of tests that complete quickly enough to meet your expectations, DTS may not be worth the effort. If tests do not complete as soon as desired due to hardware resources being finite, resulting in contention, DTS is probably worthwhile. If certain teams or build types get increased priority, and others compete for resources, DTS is probably useful for all teams.

The usage of DTS may vary depending on if it is being used by a development engineer or a quality assurance engineer. The developer wants a fast turnaround time to not delay development. QA staff may prefer more thorough testing that may take more time to complete. Where you are in the release cycle can also have an effect, you may not have many tests for new features at certain points and may have more coverage at others. Public releases may expect running more tests to minimize risk than internal builds. And as the build stabilizes, you may decide to run fewer tests.

There is a potential for other groups' tests to be the only ones to fail. If the Gcov build is only instrumented for the source code of your concern, a defect in another team's code may be missed if your tests are the only ones to cover it. Similarly, downstream features should also be considered. This concern grows more likely with an increased quantity of shared code. These risks can be mitigated by running non-DTS at your desired interval (weekly) and proper team collaboration.

Code changes to common functions may cause a high percentage of tests to be selected. Additional filtering may be necessary to reduce the number of tests, providing an opportunity to miss a defect. Running as before with an increased interval will catch the missed defect.

Running fewer tests may mean rare (1/10000) race-condition or uninitialized variable kinds of defects may not be detected as early.

DTS is designed for use with Gcov, which generates coverage for gcc compiled code (C-based languages). Products can be comprised of other languages, which Gcov will not cover. Other tools may need to be developed or discovered. Running these tools would be costly because it would increase the turnaround time of getting code coverage but would reduce the likelihood of uncovered tests.

Functional coverage won't have the accuracy of line coverage. Besides reduced granularity, code outside of any function is a blind spot. However, line coverage being more volatile, may prove more difficult

### 3 What We Want to Test

Our teams run hundreds of thousands of tests every day. We want to be sure we discover defects as soon as possible so they can be resolved easily and before visible to customers, which can be internal or external. Defects that customers find can result in revenue loss due to the software causing them more work and can result in a lack of confidence.

Developers usually run a smaller number of tests than QA and want a quick turnaround so they can be confident and move on to the next change. QA can be more patient and expect the additional tests to take more time.

We also have multiple binaries for each build. One for each supported operating system (OS). We decided that is adequate to usually run on one OS for nightly builds and all OSs for public releases.

We have limited hardware resources. All tests cannot be run for all builds. We end up forcibly reducing our test runs by alternating run modes or other variants. Now we have DTS, which tells us which tests to run without reducing coverage. This has worked well for teams with many developers simultaneously making many changes to teams with a few developers making much less changes.

Table 1 shows team ABC data from incorporating DTS in November 2021 for nightly builds, not for public releases. DTS is not used for Tue and Thu to prove accuracy. Reduction will be increased as confidence becomes sufficient to use DTS for all builds.

<b>Year.Month</b>	<b>No. of Jobs (K)</b>
2021.01	555
2021.02	478
2021.03	556
2021.04	547
2021.05	581
2021.06	580
2021.07	593
2021.08	502
2021.09	479
2021.1	555
2021.11 <sup>1</sup>	294
2021.12	268
2022.01	302
2022.02	126
Average Pre-DTS	542.6
Average Post-DTS	247.5
<b>Average % Reduction</b>	<b>45.61</b>

1: DTS Usage Started

**Table 1** Number of tests by year and month

## 4 Some Adoption Variants to Consider

Given that you may miss a defect when running fewer tests, which may cause a lack of trust by developers and others on the team, you may wish to consider different strategies to adopt DTS. Some variables to consider include how much testing the developer does, their expertise (how often do they make defects), and how often they make changes.

The risk tolerance of the team ends up driving the adoption, including for which builds to apply to. Some teams are happy to run much fewer tests with the chance of missing a defect. Others are more careful and fearful and want to be sure to run all tests for every build.

These are some ideas that we came up with. Other teams may discover alternatives that they decide fit their development process, and as DTS becomes more common to both Dev and QA, other options may be discovered. We have also seen minor enhancement requests as usage increases by QA and Dev. Dev requests are usually for more analysis of code changes while QA's are more for analysis of tests. These requests being minor indicates the initial success and strength of DTS.

## 4.1 QA Usage

1. Switch 1-N (where N is all) regression test runs for DTS until accuracy confidence is high.  
Example: You run a full regression suite five times a week. You could start slow and change one of those to be a list of tests determined by DTS or you could switch to DTS for all five.
2. Run DTS in parallel to show DTS selected tests have equivalent coverage. Or to avoid duplication, compare the generated test list to determine if it selected the failing tests.
3. Use DTS for nightly builds and not for public releases until confidence is high.

## 4.2 Developer Usage

1. Provide developers with a risk vs accuracy report to convince them of the reward. Strong data would be to show that no defects have been missed in QA testing for those developers.
2. Sell the idea to key developers that others will follow.

## 5 Conclusion

Although there is a risk of missing a defect when using DTS, the benefits for us far outweigh that risk. Our reduced testing load by ~50% without missing a defect proves we can be confident. The other benefits of increased productivity and determining test value are also a huge plus.

As our teams begin adoption, they have been satisfied and pleased with the large time savings, which gives room for other work, including implementing other changes sooner. And since no defect has been missed, we have been able to move forward with using DTS for more builds and across more teams.

As we make improvements with test selection (which tests provide the most coverage, keeping in mind fail history) and resource consumption (CPU time and memory) we hope to ultimately use it for all builds.

## References

1. Jest <https://jestjs.io/> (accessed June 29, 2020)
2. Pytest <https://docs.pytest.org/en/7.1.x/> (accessed June 29, 2020)
3. Gcov <https://gcc.gnu.org/onlinedocs/gcc/Gcov.html> (accessed June 24, 2020)

# Artificial Intelligence is the New Astrology of Software Quality

**Jack McDowell**

**Ying Ki Kwong**

jackmcdowell@gmail.com

## Abstract

Artificial Intelligence / Machine Learning are catchall terms, which often involve neural nets which are “trained” using training data and other types of input to then respond to future input. A well-trained AI can “accurately” respond to stimuli in ways that meet business needs, cutting down on human intervention or complex rule-based coding. However, just as modern science is based on fundamental first principles, software applications are supposed to be driven by clearly defined business rules and attendant business logic. Yet, AI frequently amounts to statistical correlation between “black box” models and training data with unknown scientific, legal, moral, or ethical assumptions. Therefore, concerns about biased and insufficient training data are only part of the problem.

When considering implementing AI or AI assisted technologies, management must understand the business risks of such a move, despite the seemingly unlimited enthusiasm and optimism of the technologists. The authors believe these considerations are especially relevant in the quality assurance of mission critical applications in large enterprises and government organizations.

## Biography

*Jack McDowell is the Statewide QA Program Manager with the State of Oregon. The Statewide QA Program provides Quality Assurance services for Oregon’s Major IT Projects, and Quality Assurance consultation to Oregon State Agencies. Before this, he was a web developer and the chief editor of a community newspaper in Arlington, Virginia. Originally from Buenos Aires, Argentina where he lived before attending college in the US. He holds a master’s degree in political science from the University of Oregon and a certification in ITIL.*

*Ying Ki Kwong is the E-Government Program Manager with the State of Oregon. Prior he was the IT Investment Oversight Coordinator in the same office and was Project Office Manager of the Medicaid Management Information System Project in the Oregon Department of Human Services. In the private sector, Dr. Kwong was CEO of a Hong Kong-based internet B2B portal and a program manager in the Video & Networking Division of Tektronix. In these roles, he has managed software-based systems/applications, products, and business process improvements. He received the doctorate from the School of Applied & Engineering Physics at Cornell University and was adjunct faculty in the School of Business Administration at Portland State University. He holds the PMP certification since 2003.*

Copyright Jack McDowell July 1, 2022

## 1 Introduction

Artificial Intelligence (AI) seems to be everywhere these days. The popular media often talk about AI-based tests, data processing, and even decision making in complex situations. AI can perform tasks as simple as turning on a light through voice recognition to trigger “If This Then That (IFTTT)” logic, or as complex as diagnosing medical conditions through IBM’s Watson Health (O’Leary 2022). The fact that an AI-based computing system can defeat the world champion of the board game Go is a sort of testament that AI advances in recent years have rivaled certain aspects of human intelligence (Mozur 2017).

Yet, everyone who has interacted with a voice recognition AI may have experienced a moment when an instruction, such as “turn on living room lights”, resulted in a nonsensical response. Likewise, when interacting with AI driven chatbots, frustration frequently ensues after ending up in a redirect loop. For the most part, these non-critical situations are innocuous and simply cause the user some frustrations or inconvenience. However, what happens when the business needs involve social benefits determinations (Carney 2021), shortening prison time of inmates (Rieland 2018), activating safety features of heavy machinery, or whether a self-driving car should collide with an elderly person, a child, or a tree?

AI and machine learning are often associated with “learning” neural networks trained over time to respond to different inputs. In an ideal world, a well-trained AI can “accurately” respond to stimuli in ways that meet business needs, cutting down on human intervention or complex rule-based coding. However, just as modern science is based on first principles, modern software applications are supposed to be driven by clearly defined business rules and attendant business logic that lend themselves to formal verification & validation. Yet, AI frequently amounts to statistical correlation between models and training data – however carefully done and well intentioned. At best, this statistical correlation may entail the non-existence of sound *a priori* first principles – scientific, legal, moral, ethical, or otherwise. At worst, statistical correlation may be based on biased, inaccurate, or even false premises that are embodied in models and training data that reflect real-word biases that are unscientific, illegal, immoral, or unethical.

This paper will explore how AI differs from traditional modeling and development, how AI works in practice, and the potential for inherent inaccuracy of AI Algorithms. We will discuss management considerations in addressing typical issues that may arise when applying AI techniques. These considerations include re-training of models to remove biases, better understanding of the limits of probabilistic “black box” algorithms, and potential decisions to not use AI.

## 2 Artificial Intelligence Software: How does it differ from Functional Software?

At its most basic level, software programs take inputs, do something with that data and produce an output. Figure 1.1 illustrates what this most basic workflow looks like. An example could be something as simple as switching a relay at a given time, where input = time, do something = compare input to value, and output = activate relay.



Figure 1.1: Basic Program Flow

Inputs can be user input data, data stores, big data, etc., and outputs can be returning data to a user, saving to a data store, etc.

The “Do Something” is traditionally composed of a set of operations or functions that are generated to meet a certain need. These functions can be as simple or as complex as needed to fulfill a certain task. As depicted in Figure 1.2, a simple program to determine whether x is greater than y could look as follows in C++:

```
#include <iostream>
using namespace std;
int main() {
    int x = 5;
    int y = 2;
    if (x > y) {
        cout << "x is greater than y." << endl;
    } else {
}
```

Figure 1.2: Program

Which would produce the output as shown in Figure 1.3:

```
x is greater than y.
Program ended with exit code: 0
```

Figure 1.3: Output

The code above is based on the “greater than” operator `>`. Its underlying logic is *a priori* explicit, with no *a posteriori* (statistical) variability for acceptable input.

Whether we are thinking of procedural, functional, or object-oriented programming, specifications are translated into a finite set of operations to be conducted. These set of operations typically derive from a set of functional specifications and can number from a few to thousands of functions depending on a software’s complexity. However, no matter the level of complexity, when software is developed, a person or group of people convert specifications into code.

AI programs invert this development paradigm. Instead of translating requirements into functions, AI development entails feeding training data, comprised of Inputs AND Outputs into a neural network, so that the AI can learn from the data and generate metaphorical functions.

When creating AI generated algorithms, instead of writing lines of code with explicit logic like that above, a developer may “train” a neural network using a training data set. Roughly speaking, a neural network can be thought of as layers of interconnected nodes of processing elements (neurons) as depicted in Figure 1.4, where the connection strength between interconnected nodes are adjustable parameters of the model (NeuralDesigner 2022).

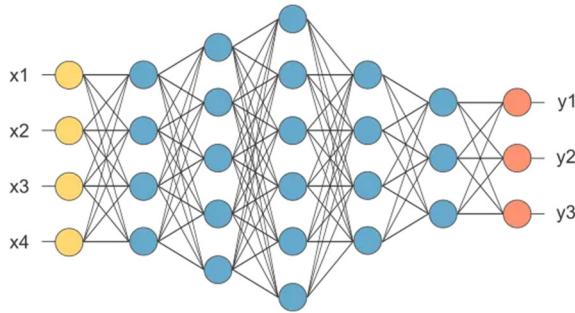


Figure 1.4: Neural Network Nodes

The process of training is to use real-world data (training data) to optimize the values of these adjustable neural network parameters. In a typical training process, a formal loss index (a measure of the different types of model errors) is minimized by iteratively adjusting neural network parameters. Iterations in the direction of lower loss index (the training direction) is done by applying an optimization algorithm (e.g., gradient descent, the Newton method, or other techniques used to numerically calculate local minimum of functions of many variables). This process of optimizing neural network parameters is as depicted in Figure 1.5 (Quesada 2022):

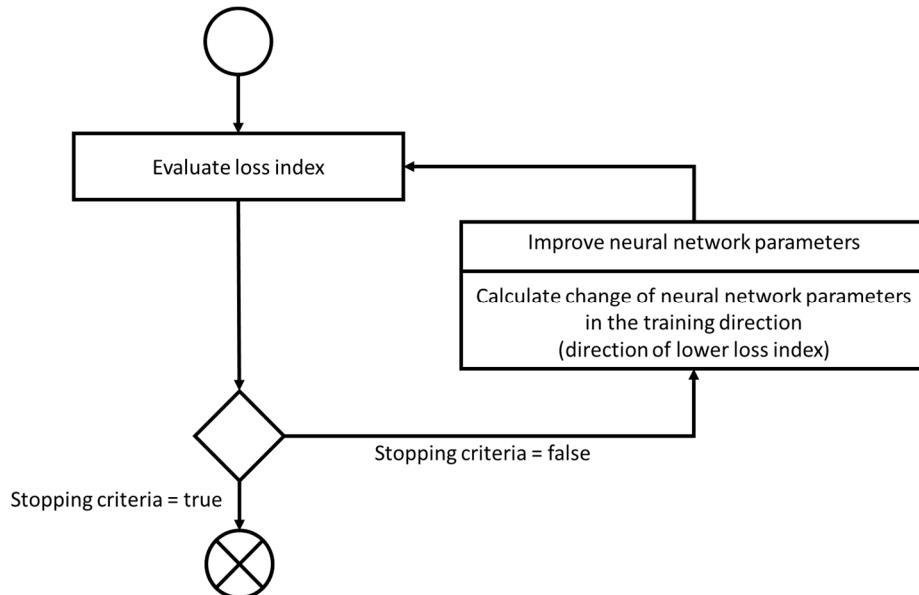


Figure 1.5: Generic Optimization Procedure for Neural Network Parameters that quantify the connection strength between neuron nodes

Generally speaking, optimization algorithms used to train neural networks that have greater computational speeds tend to require more memory and higher floating point calculation precision.

Note that the training of a neural network amounts to calculating neural network parameters that “best fit” the training data. Thus, trained neural networks have characteristics that are inherently quite different from computer codes that are based on explicit business logic. Specifically, trained neural networks contain inherent *a posteriori* (statistical) uncertainty, and they may exhibit acceptable / predictable variability only when new data to be analyzed by the trained neural network are sufficiently “like” the training data used to train the neural network in the first place.

As a model with many adjustable parameters, trained neural networks are complex emergent systems with underlying systematics that may be difficult to discern. Even with careful analysis, it is difficult to

formally demonstrate or verify that a trained neural network processes information consistent with physical laws, acceptable engineering principles or practices, or in accordance with generally acceptable legal, social, or community standards. This, in a way, is the fundamental challenge of processing information using “black box” business logic that may not be explicitly traceable to specific mathematical relations, rules, decision trees, and other pre-vetted criteria.

## 2.1 Comparing AI in Software Systems to Traditional Development

The process of designing and building software begins with a business need or opportunity, which is then translated into specifications, which in turn result in a software product to fulfill those needs. The final product can then be tested to validate that it meets business needs.

Developing traditional applications and AI applications begins with identifying business needs, but the approach begins to differ after identifying business needs.

Typically, business needs are translated into functional and non-functional specifications, which in turn form models that perform logical or mathematical operations on data. In a way, every software process can be deconstructed to its model and specification (see Figure 2.1). In turn, software can be tested to ensure that models are acting as expected and, thus, specific requirements can be verified.

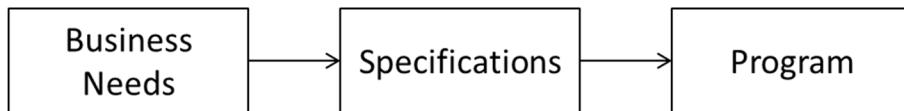


Figure 2.1. Traditional Software Development

AI software changes this paradigm by optimizing algorithms with adjustable parameters against training data based on business needs. In neural networks, this optimization process is called training; in which the element parameter (connection strengths) between processing elements (neural network nodes) are adjusted to minimize some measure of discrepancy between algorithm output and data (see Figure 2.2). As more data are collected from the real world, training is repeated to improve the quality of the algorithm. Because this process may be computationally intensive, the size of the training data sets, or frequency of re-training may be constrained in practice.

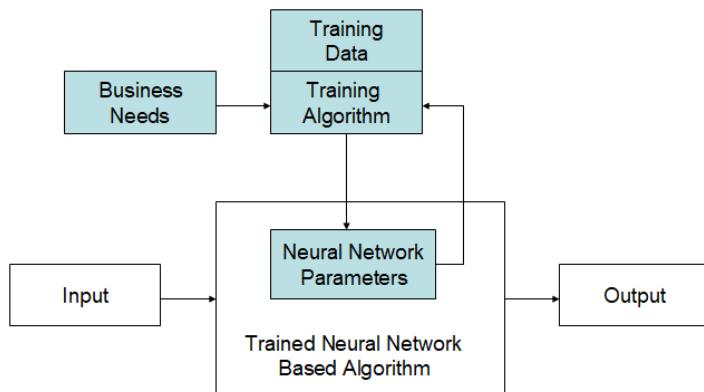


Figure 2.2: AI Training Process

### 3 Artificial Intelligence in Practice

#### 3.1 AI algorithms in practice

AI and machine learning are all around us, from chatbots, to voice and phrase recognition (Amazon's Echo, Apple's Siri, Google), to cars and even SpaceX rockets and capsules (Patel 2020). Although these examples seem disparate, these applications all have the fundamental basics in common, they take an input (text, voice, telemetry data) and produce an output (answer a question, play a song, calculate a rocket's trajectory).

We may think that of the above examples, calculating a rocket's trajectory is the most complex for us to calculate. After all, if the reader were asked to play a certain song one could likely complete the task with ease; yet being given reams of telemetry data and being asked to compute a parabolic trajectory would be beyond the scope of most readers. And yet, the miscommunications we have with our smart assistants would render their use as flight computers useless. What explains the difference between these acceptable failures of AI, successes of AI in advanced computations, and when are the potential failures of AI too risky to outweigh their usefulness?

The following sections will discuss considerations regarding ensuring the quality of AI models, where AI can work well, and certain situations where the risk of utilizing AI may outweigh the benefits.

#### 3.2 Ensuring the quality of AI models

Certain challenges in using AI driven models can be distilled by verifying and validating an application. Typically, verification and validation occur against specifications and business requirements, respectively. To conduct verification on an application, we would use manual and automated tests to ensure that specifications are met, followed by debugging of code to rectify issues. Likewise, to validate that software meets business needs, we would conduct user acceptance testing to evaluate the software, and if necessary, produce new requirements and modify the application to meet those needs.

Similarly, an AI model would be put to the test to determine if an algorithm meets business needs. However, as Gao, et al. (Gao 2019) discuss, there are unique challenges in ensuring the quality assurance of AI software. Key challenges include:

- limited data training and validation. AI algorithms are only validated with limited input data under ad-hoc contexts.
- data-driven learning features, static or dynamic, that negatively affect software outcomes, results, and actions.
- inconsistent system outputs, responses, or actions due to uncertainty inherent in statistical models.

Much of the literature on AI verification has been focused on biases introduced by incomplete or biased training data (Lee 2022). For example, research has shown that AI facial recognition algorithms trained with predominantly white faces tend to be better at identifying white faces than people of color (Najibi 2020 and Hardesty 2022). These algorithmic fails have led to contentious public discussions, such as when the IRS decided to require, and then provide alternatives, to the use of ID.me facial recognition to access tax records (Butler 2022).

Issues related to poor or biased training data can be solved by improving data sets or adding weights to datasets. However, issues related to uncertainty inherent in statistical models are harder to resolve. Whenever we think of a traditional software tool, it is possible to explore code and debug software, but when an AI is computing data, it is impossible to pinpoint what is causing an unexpected outcome or bias.

### 3.3 Where AI can work well

Scientific problems which can be falsified are perhaps the best ones suited to AI. This is because the AI can be given working parameters where out of bounds results are clear, and problems are repeatable experimentally. As mentioned previously, Machine Learning models are now able to compute complex rocket trajectories or drive cars autonomously. These algorithms are generally successful provided that the inputs are predictable (to match training data). When algorithms encounter data that does not fit with their existing training model, they make probabilistic assumptions to provide an answer and may not respond as expected, as in the case of autonomous vehicles confusing an overturned semi-trailer with a clear right of way (Stumpf 2020). These types of issues may be improved upon with better training data or lower thresholds for identification.

An important question to consider when deploying these systems depends on accountability for the decisions that the system made. Automations are part of many vehicles' safety features, such as airbags and automatic headlights. These features are rigorously tested to ensure that at a specific threshold, features are activated as expected. With AI automation, testing can still be conducted to ensure that safety features respond as expected in the event they are needed. Many of the so-called failures that we have seen regarding AI automation are not due to the AI itself, but due to lack of AI training and testing.

### 3.4 Where AI does not work well but can be improved

Most of the criticisms that we hear about AI failures are related to poor data sets or poor models. For example, facial recognition software has been found to poorly discern people of color, resulting in incorrect identifications (Najibi 2020). This has resulted in allegations that AI is biased against people of color, due to the disproportionate impacts AI has had on certain groups (Hardesty 2022). However, while the models themselves may be biased due to biased training data, the concept of utilizing AI for tasks such as facial recognition is not always inherently biased. With additional training data and better / more comprehensive data sets, these biases can be mitigated and removed.

Because machines can treat similarly situated people and objects differently, research is starting to reveal some troubling examples in which the reality of algorithmic decision-making falls short of our expectations. Given this, some algorithms run the risk of replicating and even amplifying human biases, particularly those affecting protected groups. For example, automated risk assessments used by U.S. judges to determine bail and sentencing limits can generate incorrect conclusions, resulting in large cumulative effects on certain groups, like longer prison sentences or higher bails imposed on people of color (Lee 2022).

There is also the question of risk that needs to be addressed. For example, if AI does not recognize someone's face to unlock their phone on the first attempt, it may not be a big issue if one can try again or dial an emergency call via a fallback mechanism. However, if the same facial recognition fails to provide access to a time sensitive or mission critical service or system, the risk would be exponentially higher. "Surfacing and responding to algorithmic bias upfront can potentially avert harmful impacts to users and heavy liabilities against the operators and creators of algorithms, including computer programmers, government, and industry leaders" (Lee 2022).

## 4 Inherently inaccurate AI Algorithms

So far, we have discussed where algorithms can work well, and where algorithms can work well if biases and training data is comprehensive. However, area where AI may face the insurmountable limitations is when algorithms cannot be improved with additional training data due to limits of knowledge (unknown information, unknown unknowns), and where probabilistic algorithms are legally or morally problematic. The problem with AI Algorithms, is that the AI will still provide a result based on its training, and identifying an issue, and then uncovering why the result may be incorrect, can be an impossible task.

## 4.1 What is an inaccurate model?

Artificial neural networks are often empirical models that rely on observation rather than theory. For that reason, they only need to be consistent most of the time with empirical observations. Since empirical models are models that are not generated from scientific theories, but are generated from observations, they may not be supported by theories at all in a scientific sense.

We say that AI models are the new astrology because, much like astrology, AI models look at swaths of data and make predictions. In astrology, models are created based on correlation, not theory based causal mechanisms. According to astrology when a planet transits a house in someone's birth chart, the effects of that planet are felt by the person. For example, "Venus transiting through the 4th house brings us new comforts that make navigating difficult times smoother" (Janssens 2022). When these predictions are made, there is no causal mechanism or theoretical framework that explains why the planet Venus has brought comforts to someone navigating difficult times; rather, Astrologists claim to have observed many cases where people have found comfort as Venus was transiting their 4<sup>th</sup> house. The falsifiability of an astrological model, in this case the effects of Venus in the 4<sup>th</sup> house becomes increasingly complicated due to the additional variables (planets and houses) that may impact a person's transits in a moment in time. Therefore, while we can observe that many people with Venus transiting their 4<sup>th</sup> house may find comfort in difficult times (as may the general population), the model is not falsified by people who do not find comfort, since we can easily explain that away through additional variables, such as perhaps a Mars transit.

Scientific theories, on the other hand, are often derivable from first principles such as physical laws or accepted principles. With this said, some scientific theories are phenomenological in that they describe the empirical relationships of phenomena in a way consistent and do not contradict accepted physical laws or principles. When a model contradicts accepted physical laws and principles, we can say with good certainty that the model is not good or at least outside the regime of its validity. This is an important distinction: a model that failed to make good predictions under certain circumstances may make very good predictions within its regime of validity.

As discussed previously when AI works well, models that deal with physical laws and engineering principles are the simplest problems for AI to solve, when comprehensive training data and robust neural networks can interpret predictable data. The task of evaluating models becomes increasingly complex as we transit into the social sphere and ask AI to work with applicable laws or statutes of a specific geography, or jurisdiction, social, religious, or cultural values, and protocol and etiquette. While it is straightforward to identify a model that contradicts the laws of physics, it is much more complex to identify a model that produces outputs that contradict certain cultural values.

## 4.2 Limits of knowledge

Issues arise with AI when it clashes with limits of knowledge, such as when the training data does not exist or when an AI is asked to solve a new problem. For example, an AI can become an expert level GO player by being taught the game's rules and being trained with thousands of completed games. However, if an AI is tasked with the very basic task of walking, absent knowledge about walking, the results tend to be somewhat comical (TechInsider 2017). When Deep Mind taught itself to walk, it is important to remember that according to the algorithm, it was successful in completing its task of locomotion. To the observer, Deep Mind's solution appears comical because we know, from our a priori knowledge of what it is to walk, that one does not walk by running around with their arms flailing. Similarly, IBM tried to use Watson to assist with cancer screenings by training it with imaging data and found that the accuracy of Watson's predictions was no better than chance (O'Leary 2022).

## 4.3 Legal and moral issues with AI modeling

Inherent to probabilistic models, is the legal question as to whether an AI algorithm's conclusion is sufficient to pass legal muster, and whether its findings would hold up in court. For example, states such as Pennsylvania and Oregon used AI algorithms to assist in the determination of child placement in foster

care (Burke 2022). The degree to which these determinations can be considered reliable, and these determinations can be defended is problematic in multiple ways:

1. How can we trust that the subject of an AI algorithm's decision fits within the confidence interval and probability of the model, instead of being an outlier?
2. How can we unpack the algorithm's determination, given the black box nature of the neural network?
3. Can we trust, as a society, an AI algorithm to determine such unique decisions as determining whether a child should be removed from a household and placed in foster care?

These are not merely academic musings. The U.S. Federal Government has recently published guidance related to the use of potentially biased algorithms (Jillson 2022), as well as properly trained algorithms generated from training data that may have been inappropriately obtained (Kaye 2021).

## 5 Conclusion

This paper considered the limitations of AI in software applications. The authors believe these considerations are especially relevant in the quality assurance of mission critical AI-based applications that support large enterprises and government organizations.

We contrasted traditional software design vs. AI-based software design as two different development paradigms:

- Traditional software design is based on algorithms whose underlying logic is *a priori* pre-determined. As such, the resultant logic can be unpacked and analyzed formally to support software verification & validation.
- AI-based software design is based on algorithm whose underlying "logic" is derived from *a posteriori* (statistical) training procedures that can be repeated in time when more training data become available. As such, the resultant logic cannot be easily unpacked and analyzed formally to support verification & validation beyond statistical correlations.

As a researcher in the Advanced Research in Cyber Systems group at Los Alamos National Labs explained, "The artificial intelligence research community doesn't necessarily have a complete understanding of what neural networks are doing; they give us good results, but we don't know how or why" (Njegomir 2022). We believe this situation has important management implications.

When considering implementing AI or AI assisted technologies, management must understand the business risks of such a move, to provide checks & balance on the seemingly unlimited enthusiasm and optimism of the technologists. As discussed in this paper, typical issues that arise with AI may include but are not limited to:

- Models may not be good or are poorly trained.
- Models may be well trained but poorly understood.
- Inherent biases are embodied in the training data.
- Lack of accountability with accuracy of models and when models should / should not work.
- Lack of formal understanding about the models beyond *a posteriori* statistical correlation.

Although astrology may provide entertainment through plausible correlations, we would not use astrology to determine social benefits or activate safety features of heavy machinery. However, we should not base business logic on plausible correlations. When we use AI in real world applications, we have the obligation to understand the regime of validity of our models and avoid underlying logic based on plausible correlations.

The authors are especially concerned about the use of AI in situations that may affect the rights, dignity, safety, and physical / mental health of individuals or minority groups. For applications in these spaces, we believe developers have the responsibilities to assure fairness and ethical development of their AI-based

software. For example, if an AI-based facial recognition software is believed to be highly accurate among the general population of a country, the developers should be aware that this is a statistical statement, and this sort of statistical statement may be consistent with the same software being highly inaccurate within an ethnic minority of the same country or possibly highly inaccurate among the general population of another country (Hardesty 2022).

We would like to close this paper with the following quotes from the Brookings Institution and the United Nations:

It is important for algorithm operators and developers to always be asking themselves: Will we leave some groups of people worse off as a result of the algorithm's design or its unintended consequences? – *Brookings Institution* (Lee 2022)

The world needs rules for artificial intelligence to benefit humanity. The Recommendation on the ethics of AI is a major answer. It sets the first global normative framework while giving States the responsibility to apply it at their level. UNESCO will support its 193 Member States in its implementation and ask them to report regularly on their progress and practices. – *UNESCO* (Azoulay 2022)

## Acknowledgement

The authors would like to acknowledge helpful discussions with Karen J. Johnson, Nicholas Betsacon, Brooke Janssens and John Cvetko during the planning of this paper.

## References

- Azoulay, Audrey, and Gabriela Ramos. "UNESCO Member States Adopt the First Ever Global Agreement on the Ethics of Artificial Intelligence." UNESCO, February 1, 2022. <https://en.unesco.org/news/unesco-member-states-adopt-first-ever-global-agreement-ethics-artificial-intelligence>.
- Burke, Sally Ho and Garance. "Oregon Dropping AI Tool Used in Child Abuse Cases." AP NEWS. Associated Press, June 2, 2022. <https://apnews.com/article/politics-technology-pennsylvania-child-abuse-1ea160dc5c2c203fdab456e3c2d97930>.
- Butler, Peter. "How to Register on the IRS Website without Taking a Video Selfie." CNET. Accessed September 30, 2022. <https://www.cnet.com/personal-finance/taxes/irs-to-back-off-third-party-facial-recognition-what-happens-to-id-me/>.
- Carney AO, Terry, Artificial Intelligence in Welfare: Striking the Vulnerability Balance? (March 15, 2021). Carney, T., 'Artificial Intelligence in Welfare: Striking the vulnerability balance? (2020) 46(2) Monash University Law Review Advance 1-29 DOI <https://doi.org/10.26180/13370369.v1>, Available at SSRN: <https://ssrn.com/abstract=3805329> or <http://dx.doi.org/10.2139/ssrn.3805329>
- Gao, Jerry, Chuanqi Tao, Dou Jie, and Shengqiang Lu. "Invited Paper: What Is Ai Software Testing? and Why." 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), 2019. <https://doi.org/10.1109/sose.2019.00015>.
- Hardesty, Larry. "Study Finds Gender and Skin-Type Bias in Commercial Artificial-Intelligence Systems." MIT News | Massachusetts Institute of Technology. Accessed September 30, 2022. <http://news.mit.edu/2018/study-finds-gender-skin-type-bias-artificial-intelligence-systems-0212>.

Janssens, Brooke. "Venus Retrograde." Star Oracles, February 1, 2022. <https://staroracles.com/venus-retrograde/>.

Jillson, Elisa, Helen Clark, and Lesley Fair. "Aiming for Truth, Fairness, and Equity in Your Company's Use of Ai." Federal Trade Commission, May 24, 2022. <https://www.ftc.gov/business-guidance/blog/2021/04/aiming-truth-fairness-equity-your-companys-use-ai>.

Kaye, Kate. "Why the FTC Is Forcing Tech Firms to Kill Their Algorithms along with Ill-Gotten Data." Digiday, November 17, 2021. <https://digiday.com/media/why-the-ftc-is-forcing-tech-firms-to-kill-their-algorithms-along-with-ill-gotten-data/>.

Lee, Nicol Turner, Paul Resnick, and Genie Barton. "Algorithmic Bias Detection and Mitigation: Best Practices and Policies to Reduce Consumer Harms." Brookings. Brookings, March 9, 2022. <https://www.brookings.edu/research/algorithmic-bias-detection-and-mitigation-best-practices-and-policies-to-reduce-consumer-harms/>.

Mozur, Paul. "Google's AlphaGo Defeats Chinese Go Master in Win for A.I." The New York Times. The New York Times, May 23, 2017. <https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html>.

Najibi, Alex. "Racial Discrimination in Face Recognition Technology." Science in the News, October 26, 2020. <https://sitn.hms.harvard.edu/flash/2020/racial-discrimination-in-face-recognition-technology>.

NeuralDesigner. "3. Neural Network." Neural networks tutorial: Neural network | Neural Designer. Accessed September 30, 2022. <https://www.neuraldesigner.com/learning/tutorials/neural-network>.

Njegomir, Nick. "New Method for Comparing Neural Networks Exposes How Artificial Intelligence Works." Discover Los Alamos National Laboratory. Accessed September 30, 2022. <https://discover.lanl.gov/news/0913-neural-networks>.

O'Leary, Lizzie. "How IBM's Watson Went from the Future of Health Care to Sold off for Parts." Slate Magazine. Slate, January 31, 2022. <https://slate.com/technology/2022/01/ibm-watson-health-failure-artificial-intelligence.html>.

Patel, Neel V. "Are We Making Spacecraft Too Autonomous?" MIT Technology Review. MIT Technology Review, July 3, 2020. <https://www.technologyreview.com/2020/07/03/1004788/spacecraft-spacefight-autonomous-software-ai/>.

Quesada, Alberto. 5 algorithms to train a neural network. Accessed September 30, 2022. [https://www.neuraldesigner.com/blog/5\\_algorithms\\_to\\_train\\_a\\_neural\\_network](https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network).

Rieland, Randy. "Artificial Intelligence Is Now Used to Predict Crime. but Is It Biased?" Smithsonian.com. Smithsonian Institution, March 5, 2018. <https://www.smithsonianmag.com/innovation/artificial-intelligence-is-now-used-predict-crime-is-it-biased-180968337/>.

Stumpf, Rob. "Autopilot Blamed for Tesla's Crash into Overturned Truck." The Drive, June 16, 2020. <https://www.thedrive.com/news/33789/autopilot-blamed-for-teslas-crash-into-overturned-truck>.

TechInsider. *Google's DeepMind AI Just Taught Itself To Walk.* YouTube. YouTube, 2017. <https://www.youtube.com/watch?v=gn4nRCC9TwQ>.

# Role of Emerging Technology in Securing Rural Healthcare

**Sneha Mirajkar  
Vittalkumar Mirajkar**

SMirajka@cisco.com  
Vittalkumar\_Mirajkar@mcafee.com

## Abstract

“With great power comes great responsibility”. As the rural healthcare moves towards digital transformation, adopting new and emerging technologies fast pacing the need to cater patient needs and increase the reach to wider population.

In steps, the greatest need for data security or “Patient Data Security” in rural healthcare. Especially, when the need for digitization of rural healthcare is a necessity owing to the pandemic in the recent years. Community and rural hospitals face many challenges in effective communication. Patients might have limited access to healthcare portals or apps. Providers may find it hard to stay in alignment with individual patient needs. Streamlining clinical collaboration is critical to maintaining quality of care while demand increases. Deploying secure messaging platforms enables easy provider-to-provider, provider-to-nurse and system-to-patient communication over voice, video or text.

The primary focus of this paper is to present an overview of the security gaps in the process of digitization of rural healthcare and to highlight the role of emerging technologies in securing the rural healthcare system ensuring Patient Data Security while providing seamless transformation. As the healthcare cybersecurity landscape grows and new threats emerge, healthcare IT professionals must balance IT security policy, processes, and compliance with the need to maintain seamless data flow and timely access to patient information.

## Biography

*Sneha Mirajkar is a Senior Software Engineer at Cisco, with 14+ years of experience in software testing and extensive hands-on in test automation using Python, AWS, web-services. She has expertise in AWS applications.*

*Vittalkumar Mirajkar is a Senior Engineering Manager at McAfee, with 16+ years of testing experience ranging from device driver testing, application testing and server testing. He specializes in testing security products. His area of interest is performance testing, soak testing, data analysis and exploratory testing.*

## 1. Introduction

With more sophisticated cyber-threats against healthcare organizations on the rise, having a comprehensive solution in place is critical. The right measures can protect sensitive patient data and prevent security breaches from interfering with patient care. How far would you go to protect your patients' sensitive data? One thing is for sure — you'll need to stay ahead of the cybercriminals. The topic of healthcare data security has recently come up in the media on multiple occasions, mostly in connection with breaches and leaks.

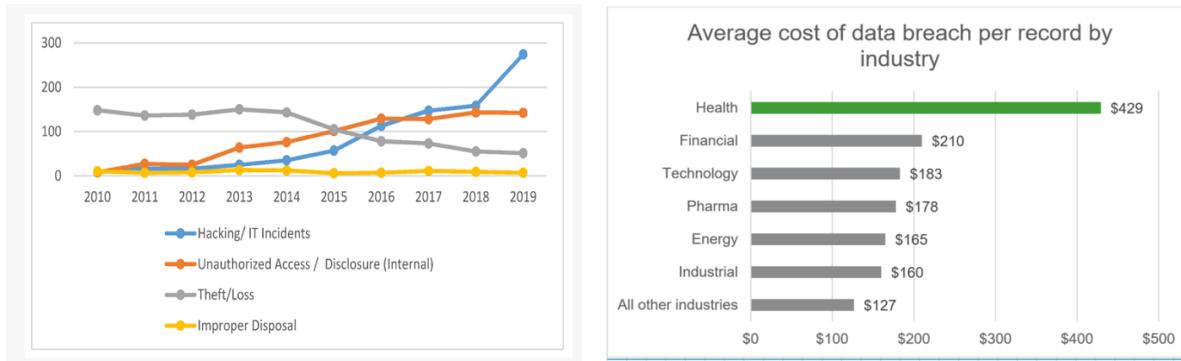
Digital methods as mentioned below; are important to improve the quality, safety, effectiveness, and delivery of healthcare services in rural communities. Connecting rural patients and providers in remote locations to specialists in urban areas. Implementing, maintaining, updating, and optimizing the same can be an ongoing challenge for rural facilities and providers with limited resources and expertise.

- Electronic health records (EHR) for patients, instead of paper records.
- Secure digital networks to send and deliver up-to-date records whenever and wherever the patient or clinician may need them.
- Electronic transmittal of medical test results.
- Confidential and secure patient health portals for patients to access their personal health information online.
- Mobile devices and tablets to update patient records in real time and document at the point of care.
- Telemonitoring applications that allow patients to transmit vitals or diagnostic information to providers remotely.

## 2. Impacts of Security gaps

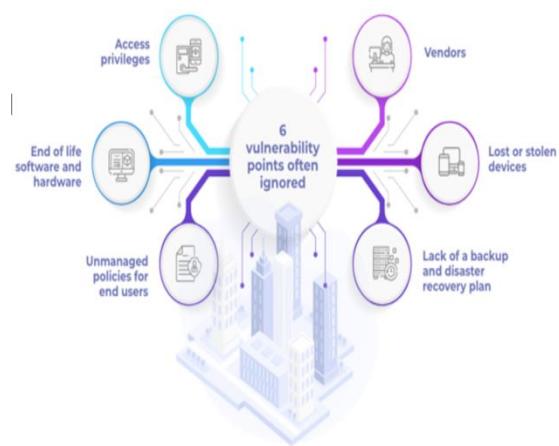
Healthcare data breaches hit all time high in 2021, impacting 45M people. While we continue to rave about how Artificial Intelligence (AI)/(Electronic health records (EHRs) can improvise rural healthcare. Healthcare remains costliest industry for data breaches. While electronic health records (EHRs) offer incredible advantages for quickly accessing patient data, eliminating fragmentation, and improving information sharing, they also present security challenges. As statics show. a shocking 41,335,889 patient records were leaked during data breaches in recent times. this raises grave concerns about the Patient data security.

Patient data breaches occur due to both internal and external attacks, such as hacking, theft/loss, unauthentic internal disclosure, and the improper disposal of unnecessary but sensitive data. However, hacking/IT incidents are most used by attackers. Furthermore, the Email and Network servers are the main locations from where confidential health data is breached. healthcare data breaches are far more expensive than the average cost of data breaches.



### 3. Analyzing of Security gaps in rural healthcare

While rapid increase in digitization of rural healthcare is welcome transformation, many a times, the lack of policies and access control leads to staggering amount of Patient Data compromise and breaches at multiple levels that include identity theft and financial threat. Threat management is a major challenge to all healthcare organizations especially in rural communities. Most common vulnerability often ignored are:



1. **Access privileges:** Unmanaged firewall configurations and remote access, lack of access policies for employees and vendors.
2. **End of life software and hardware:** Vendors stop providing patches and using out-of-date and high-risk technology such as java applets.
3. **Unmanaged policies for end users:** Lack of website control and application download control, sharing credentials, etc.
4. **Vendors:** Hospitals work with vendors (insurance companies, etc.) without assessing the accompanying security risks.
5. **Lack of a backup and disaster recovery plan:** The quickest way to recover from an attack is to rebuild all the breached systems. A good backup and recovery plan can help.

Healthcare security is always at the mercy of the multiple cyber threats listing a few recent incidents.

- 34% of the respondents were affected by ransomware attacks in the last year.
- 5% of those affected had their data encrypted by the cybercriminals.
- 34% of those whose files had been held hostage via encryption paid the ransom, with the total losses amounting to an average of US\$1.27 million.
- Only 24% of the participants do not expect to be hit by a ransomware attack within the next year.

Recent healthcare security issues highlight the use of emerging technology is creating these.

**Ransomware scams:** Ransomware remains one of the biggest challenges healthcare faces.

**Hacktivism:** With many hacker groups and Anonymous roaming the digital land, patient data security is always at risk.

**User error:** Sending unencrypted medical data over email or in messages is a surefire way to lose it in a cyberattack. Even better: sending one's access credentials or storing them in a simple Google doc.

**Use of legacy technology:** As it turns out, using older technology can't guarantee the security of your patients' data either. Outdated software can be a major source of security issues in Healthcare.



## **Adoption of cloud technology and mobile apps:**

The issue with cloud and mobile tech mainly has to do with on-the-fly encryption of the data that's constantly traveling between servers and devices.

#### **4. Role of Emerging Technology in Securing Patient Data**

Digital Transformation in rural healthcare is all about adopting new age technologies to deliver efficient patient care backed by robust and secure systems that enable instant and secure access to patients 'data across platforms. Building a strong line of defense based on the emerging technologies to counter cyberattacks and security vulnerabilities is extremely important. As a result, it is crucial that organizations implement healthcare data security solutions that will protect important assets while also satisfying healthcare compliance mandates. Various technologies are in use for protecting the security and privacy of healthcare data. Most widely used technologies are:

## 4.1 Authentication

Authentication is the act of establishing or confirming claims made by or about the subject are true and authentic. It serves a vital function within any organization: securing access to corporate networks, protecting the identities of users, and ensuring that a user is who he claims to be. Most cryptographic protocols include some form of endpoint authentication specifically to prevent man-in-the-middle (MITM) attacks. For instance, Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide security for communications over networks such as the Internet.

TLS and SSL encrypt the segments of network connections at the Transport Layer end-to-end. Several versions of the protocols are in widespread use in applications like web browsing, electronic mail, Internet faxing, instant messaging and voiceover-IP (VoIP). One can use SSL or TLS to authenticate the server using a mutually trusted certification authority. Additionally, Bull Eye algorithm can be used for monitoring all sensitive information in 360° . This algorithm has been used to make sure data security and manage relations between original data and replicated data. It also allows only authorized person to read or write critical data. Paper <sup>13</sup> proposes a novel and a simple authentication model using one time pad algorithm. It provides removing the communication of passwords between the servers. In a healthcare system, both healthcare information offered by providers and identities of consumers should be verified at the entry of every access.

## 4.2 Encryption

Data encryption is an efficient means of preventing unauthorized access of sensitive data. Its solutions protect and maintain ownership of data throughout its lifecycle — from the data center to the endpoint (including mobile devices used by physicians, clinicians and administrators) and into the cloud. Encryption is useful to avoid exposure to breaches such as packet sniffing and theft of storage devices. Rural Healthcare organizations must ensure that encryption scheme is efficient, easy to use by both patients and healthcare professionals and easily extensible to include new electronic health records. Furthermore, the number of keys owned by each party should be minimized. Although various encryption algorithms have been developed and deployed relatively well (RSA, Rijndael, AES and RC6<sup>14, 16, 17</sup>, DES, 3DES, RC4<sup>15</sup>, IDEA, Blowfish ...), the proper selection of suitable encryption algorithms to enforce secure storage remains a difficult problem.

## 4.3 Data Masking

Masking replaces sensitive data elements with an unidentifiable value, but is not truly an encryption technique so the original value cannot be returned from the masked value. It uses a strategy of de-identifying the data sets or masking personal identifiers such as name, social security number and suppressing or generalizing quasi - identifiers like date-of-birth and zip-codes.

Thus, data masking is one of the most popular approaches to live data anonymization. k-anonymity first proposed by Swaney and Samrati<sup>18, 19</sup> protects against identity disclosure but failed to protect against attribute disclosure. Truta et al.<sup>20</sup> have presented p-sensitive anonymity that protects against both identity and attribute disclosure.

Other anonymization methods fall into the classes of adding noise to the data, swapping cells within columns and replacing groups of k records with k copies of a single representative. These methods have a common problem of difficulty in anonymizing high dimensional data sets<sup>21, 22</sup>. A significant benefit of this technique is that the cost of securing a big data deployment is reduced. As secure data is migrated from a secure source into the platform, masking reduces the need for applying additional security controls on that data while it resides in the platform.

## 4.4 Access Control

Once authenticated, the users can enter an information system, but their access will still be governed by an access control policy which is typically based on the privilege and right of each practitioner authorized by patient or a trusted third party. It is then, a powerful and flexible mechanism to grant permissions for users. It provides sophisticated authorization controls to ensure that users can perform only the activities for which they have permissions, such as data access, job submission, cluster administration, etc. Several solutions have been proposed to address the security and access control concerns. Role-Based Access Control (RBAC)<sup>23</sup> and Attribute-Based Access Control (ABAC)<sup>25, 26</sup> are the most popular models for EHR. RBAC and ABAC have shown some limitations when they are used alone in medical system. Paper<sup>24</sup> also proposes a cloud-oriented storage efficient dynamic access control scheme cipher text based on the CP-ABE and symmetric encryption algorithm (such as AES). To satisfy requirements of fine-grained access control yet security and privacy preserving, we suggest adopting technologies conjunction of other security techniques, e.g., encryption, and access control method.

Additionally, data protection laws are more than ever crucial in healthcare organizations to manage and safeguard personal information and address their risks and legal responsibilities in relation to processing personal data, to address the growing thicket of applicable data protection legislation. Different countries have different policies and laws for data privacy. Data protection regulations and laws in some of the countries along with salient features are listed in the Table below.

**Data protection laws in some of the countries:**

Country	Law	Salient Features
U.S.A	HIPAA Act Patient Safety and Quality Improvement Act (PSQIA) HITECH Act	Requires the establishment of national standards for electronic health care transactions. Gives the right to privacy to individuals from age 12 through 18. Signed disclosure from the affected before giving out any information on provided health care to anyone, including parents. Patient Safety Work Product must not be disclosed <sup>27</sup> . Individual violating the confidentiality provisions is subject to a civil penalty. Protect security and privacy of electronic health information.
EU	Data Protection Directive	Protect people's fundamental rights and freedoms and in particular their right to privacy with respect to the processing of personal data. <sup>29</sup>
Canada	Personal Information Protection and Electronic Documents Act ('PIPEDA')	Individual is given the right to know the reasons for collection or use of personal information, so that organizations are required to protect this information in a reasonable and secure way. <sup>28</sup>
UK	Data Protection Act (DPA)	Provides a way for individuals to control information about themselves. Personal data shall not be transferred to a country or territory outside the European Economic Area unless that country or territory ensures an adequate level of protection for the rights and freedoms of data subjects.
Russia	Russian Federal Law on Personal Data	Requires data operators to take "all the necessary organizational and technical measures required for protecting personal data against unlawful or accidental access".
India	IT Act and IT (Amendment) Act	Implement reasonable security practices for sensitive personal data or information. Provides for compensation to person affected by wrongful loss or wrongful gain. Provides for imprisonment and/or fine for a person who causes wrongful loss or wrongful gain by disclosing personal information of another person while providing services under the terms of lawful contract.
Angola	Data Protection Law (Law no. 22/11 of 17 June)	With respect to sensitive data processing, collection and processing is only allowed where there is a legal provision allowing such processing and prior authorization from the APD is obtained.

## 5. Conclusion

Limitless opportunities are offered to drive health research, knowledge discovery, clinical care, and personal health management. However, there are several obstacles and challenges that impede its true potential in the rural healthcare field, including technical challenges, privacy and security issues and skilled talent. Big data security and privacy are considered as the huge Privacy and security issues. In addition of the methods currently used to ensure patient's privacy in rural healthcare, there are more various techniques include Hiding a needle in a haystack<sup>30</sup>, Attribute based encryption Access control, Homomorphic encryption, Storage path encryption and so on. However, the problem is always imposed. In this context, as our future direction, perspectives will focus more on achieving effective solutions to the scalability problem of privacy and security in the area of rural healthcare. Also multiple designs and frameworks can be implemented to solve the problem of reconciling security and privacy models by simulating diverse approaches using exploiting the MapReduce framework. To, ultimately, support decision making and planning strategies, for a robust rural healthcare infrastructure.

## References

1. <https://www.hipaajournal.com/category/healthcare-data-security/>
2. <https://www.fiercehealthcare.com/health-tech/healthcare-data-breaches-hit-all-time-high-2021-impacting-45m-people>
3. <https://www.mdpi.com/2227-9032/8/2/133/htm>
4. <https://www.futurismtechnologies.com/solutions/futurism-data-protect-services/>
5. <https://www.cdwg.com/content/cdwg/en/industries/healthcare-technology/rural-healthcare.html>
6. "State Of Enterprise IoT Security: A Spotlight On Healthcare", Forrester Consulting, September 2019.
7. "Largest Healthcare Data Breaches of 2021", HIPPA Journal, December 2021.
8. "Ransomware attack on Georgia health system endangers info of 1.4M patients", Healthcare IT News, August 2021.
9. "Takeover of a Spacelabs Xprezzon Patient Monitor Demo", Armis.
10. "URGENT/11: II Zero Day Vulnerabilities Impacting VxWorks, the Most Widely Used Real-Time Operating System (RTOS)", Armis, August 2019
11. "TLStorm: Three critical vulnerabilities discovered in APC Smart-UPS devices can allow attackers to remotely manipulate the power of millions of enterprise devices", Armis, 2022.
12. <https://demigos.com/blog-post/data-security-in-healthcare-importance-challenges-solutions/>
13. C. Yang, W. Lin, and M. Liu, "A Novel Triple Encryption Scheme for Hadoop-Based Cloud Data Security," in Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on, 2013, pp. 437-442.
14. Federal Information Processing Standards Publication 197, "Specification for the Advanced Encryption Standards (AES)", 2001.
15. S. Fluhrer, I. Mantin, and A. Shamir, "Weakness in the Key scheduling algorithm of RC4", 8th Annual International Workshop on Selected Areas in Cryptography, Springer-Verlag London, UK, 2001.
16. J. Shafer, S. Rixner, and A. L. Cox. The Hadoop Distributed File system: Balancing Portability and Performance. Proc. of 2010 IEEE Int. Symposium on Performance Analysis of Systems & Software (ISPASS), March 2010, White Plain, NY, pp. 122-133.
17. N. Somu, A. Gangaa, and V. S. Sriram, "Authentication Service in Hadoop Using one Time Pad," Indian Journal of Science and Technology, vol. 7, pp. 56-62, 2014.
18. L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," in International journal on uncertainty, fuzziness and knowledge based systems, vol. 10, 2002, pp. 571 – 588.
19. P. Samarati, "Protecting respondents identities in microdata release," in IEEE transactions on knowledge and data engineering, vol. 13, 2001, pp. 1010 – 1027.

20. T. M. Truta and B. Vinay, "Privacy protection: p-sensitive k-anonymity property," in Proceedings of 22nd International Conference on Data Engineering Workshops, 2006, p. 94.
21. N. Spruill, "The confidentiality and analytic usefulness of masked business microdata," in Proceedings on survey research methods, 1983, pp. 602–607.
22. S. Chawala, C. Dwork, F. M. Sheny, A. Smith, and H. Wee, "Towards privacy in public databases," in Proceedings on second theory of cryptography conference, 2005.
23. Science Applications International Corporation (SAIC). Role-Based Access Control (RBAC) Role Engineering Process Version 3.0. 11 May 2004.
24. H. Zhou and Q. Wen, "Data Security Accessing for HDFS Based on Attribute-Group in Cloud Computing," in International Conference on Logistics Engineering, Management and Computer Science (LEMCS 2014), 2014.
25. A. Mohan, D. M. Blough, An Attribute-Based Authorization Policy Framework with Dynamic Conflict Resolution, Proceedings of the 9th Symposium on Identity and Trust on the Internet, 2010.
26. M. Hagner. Security infrastructure and national patent summary. In Tromso Telemedicine and eHealth Conference, 2007. 2
27. Data Protection Laws of the World. 2017 DLA Piper. [Online]. Available: <http://www.dlapiperdataprotection.com> 2
28. Challenges of privacy protection in Big Data Analytics –Meiko Jensen- 2013 IEEE International Congress on Big Data. 2013.
29. Privacy and Big Data – Terence Craig & Mary E.Ludloff
30. Jung K, Park S, Park S. Hiding a needle in a haystack: privacy preserving Apriori algorithm in MapReduce framework PSBD'14, Shanghai; 2014. p. 11–17.

# How AI Can Help Accelerate Testing

**Chris Navrides**

chris.navrides@gmail.com

## Abstract

AI can beat humans at chess and Go, drive a car more safely than humans, identify cat videos better than any human; soon AI will help test code better than humans too. What is really going on at the forefront of building AI systems for test automation? This paper will walk through where AI is being applied across the software testing field, as well as practical applications being used today to test mobile apps and web pages.

## Biography

Chris Navrides is the CEO and founder of Dev Tools AI which is building developer and tester centric testing tools. Prior to Dev Tools, he was the VP of Engineering at test.ai, which built the first AI based mobile & web testing solution. Before that, Chris worked at Google on automation testing for Google Play and mobile ads, and Dropbox on their mobile platform team. Chris received his Bachelors and Masters from Colorado School of Mines.

# 1 Introduction

Every day there are new applications and products with Artificial Intelligence that are being launched or talked about. There are now computers beating humans in Go and chess, driving safety, and other tasks that we thought were too complex for machines. We are also seeing computers starting to assist humans in creative applications, like Github's CoPilot for writing code. Google Engineers are even discussing bots they say are sentient.

Artificial Intelligence (AI) in the testing space is not new at this point. Over the last 5+ years AI has been a common talking point in many testing tools companies. There are many applications of AI within the testing space, such as API, microservices and data generation, to name a few. But this paper will focus mostly on applications within the UI testing space.

# 2 Background

## 2.1 What is AI/ML

Put simply, Artificial Intelligence (AI) and Machine Learning (ML) is where a computer program or system is not programmed for a specific set of rules about a given problem, but instead is programmed to optimize its output for a given set of inputs. It then is trained over a set of data to learn those rules.

For example, if you wanted to build a system to determine if a given object is a dog or a car. In traditional programming, you would build a set of rules like "if it has wheels and doors, it is a car, else it is a dog". With machine learning you would structure the data and let it learn those rules by training. You would give it a bunch of cars and dogs, and let it guess what it thinks it is. If it is right, it gets a positive reward, and that behavior is reinforced. Eventually it builds its own set of rules that allow it to guess if a given object is a dog or a car without the programmer needing to write.

There are many types of AI and ML out there. The example above shows the power of predictive AI. This is applicable today in many apps that you use frequently from product suggestions from Amazon & Netflix, to search suggestions from Google. There are also generative AI systems that are used to create objects. Examples of these are systems like MidJourney that can make art images from a text prompt. Finally, there is reinforcement learning. These are the systems that are used to do complex actions such as driving and playing video games.

## 2.2 Predictive AI

This is the most common type of AI/ML systems people are used to. These systems are used across many industries and professions. They rely on having large amounts of data that is labeled correctly. This type of model in the testing space can be used to determine if a given element is a shopping cart or a search icon. It is also used for things like Optical Character recognition to look at an image and read the characters within it.

## 2.3 Generative AI

Generative AI is a new and growing field. It is the process of using an AI/ML model to make new data. An example of a generative model of this type is OpenAI's GPT-3[1] that can generate fake news articles.

There are two ways that these models are trained. The first way is where they use existing training samples, say articles and text blocks, and the first part is given to the model. It is then scored based upon how well it generates the second part. The second way models like this can be trained is to have models compete against each other. You have one model generate fake data, and then another model tries to detect if the data is real or fake. The generative model is rewarded when it “tricks” the detector model, and the detector model is rewarded when it picks real data. This type of training is called Generative Adversarial Networks (GANs) [2]. GANs can be used in the testing space to generate test data, such as API requests. It can also be used to generate test suites and test cases [3]. It is a powerful way to quickly create test data and coverage.

## 2.4 Reinforcement Learning

Reinforcement learning is a process where you build a model that will learn abstract concepts. Where the above examples are scored based upon “correctness” to a known solution, reinforcement learning can be applied to unknown domains. The simplest example of this you can imagine is a robot exploring a maze. There are many paths that it can take, and it needs to find the optimal one. However, it can't see the full maze, so it must explore and try to reach the end with the quickest amount of time. It will eventually reach the end, but it then has to optimize the path. That path can include large optimizations from the original path that it found, or the potential for the original path to be the best.

This type of system is applied in many complex systems that you see today like self-driving cars or playing video games [4] or even testing the Windows operating system [5]. In the example of self-driving cars, the system itself is given a general reward (ex: get from point a to point b) and then it optimizes that reward for shortest distance, and least obstacles, like humans, hit. This can be applied in testing domain for finding the shortest path within a given application to get to a particular screen. That system can then learn all the paths to say a settings page and use that instead of an end user writing a hard coded script for the exact steps.

## 2.5 UI Testing Fundamentals

Within User Interface (UI) testing there are two core components that are at the heart of it: locating objects on the page and asserting against the system. All frameworks, for both mobile and web, and all other types of UI testing (ex: Roku, Xbox, etc.) rely on these two core concepts.

### 2.5.1 Locators

Locators, or selectors, are used to find an individual or a set of elements on a given page. This is done by building a set of filters against all of the elements on the page, to narrow it down until there is only the desired set of elements. For example, if you were on a login page and wanted to find the username text box, you would build a filter that looks for a text box that has name username. If you were to just look for a text box, then you would also get the password box which you don't want.

### 2.5.2 Assertions

Assertions can be done on any part of a given application; against the page or the backend or even the properties of the application such as the log file. The nature of these assertions can be derived from the needs of the business.

Most assertions are done to make sure a given object is visible or not. However, there are lots of other types of assertions such as making sure a given object is the right size, or in the right place on the screen. Assertions can even be used to make sure it is exactly correct down to the individual pixel.

Additional assertions are often useful as they can tell a more holistic view of the application under test. These can include verifying network logs were properly formed and sent or that there were no errors within the console log.

## 3 Problem

When writing UI automation, the test is written for the state of the app as it was in that particular moment in time. All subsequent changes to the app or page after that moment, could result in potentially broken or flaky tests. Changes can occur for a variety of reasons:

- Refactorings, such as changing the class names to be more appropriate
- Changing of web frameworks, i.e.; asp.net app being re-written to use react
- Random experiments that can alter the UI or backend to change the user experience
- Backend data changes that can result in the UI test assumptions being challenged

All of these changes cause instability in the underlying UI tests that must then be fixed and maintained. These issues are usually grouped into one of two types of errors: The page model/DOM changes, or changes to the flow (or test steps) the test needs to take in order to complete the desired scenario.

### 3.1 DOM Changes

Document Object Model (DOM) changes occur when some value within the page is modified. These can be minor, such as a CSS class changing, or major where entire elements disappear. When these happen in a given test case that is looking for them, it results in a failure; even when visually the object is still there.

### 3.2 Flow Changes

Flow changes occur when a different set of steps appear in the application being tested. This is a common practice for things like seasonal promotions or alerting users. Many free apps will randomly show ads to a user that can take over the entire screen. International companies where payments and security requirements of specific regions have additional demands. If a test is meant to execute 100% of the time, in all these scenarios, this presents a difficult challenge.

## 4 Using AI/ML for UI Testing

### 4.1 Element Selection

One potential solution to building locators within a UI testing framework is to use visual AI to find the element on the screen instead of using the DOM to find the element. This has several advantages in that

it allows the test case to execute like an end user, and not have false negatives when an underlying change occurs. It also would be able to be executed across different platforms, such as iOS/Android and web as most applications tend to use the same iconography for all use cases.

An approach to detecting a specific element, on a given screenshot, would be to look at the screenshot and visually locate all elements on that screen. That would then enable an iterative approach to look over each element individually and determine what it is (ex: menu icon, logo, search button, etc.).

## 4.2 Object Detection

To detect elements on a given screen, we need to build a model. There are several model types that are valid such as R-CNN [8], You Only Look Once (YOLO) [9] and Single Shot Detectors [10]. These can be taken from reference examples or trained specifically for a given task.

What this looks like in practice is a set of bounding boxes for a given screenshot (fig.1) that the model detects. The model outputs boxes of what it sees are given elements (fig. 2).

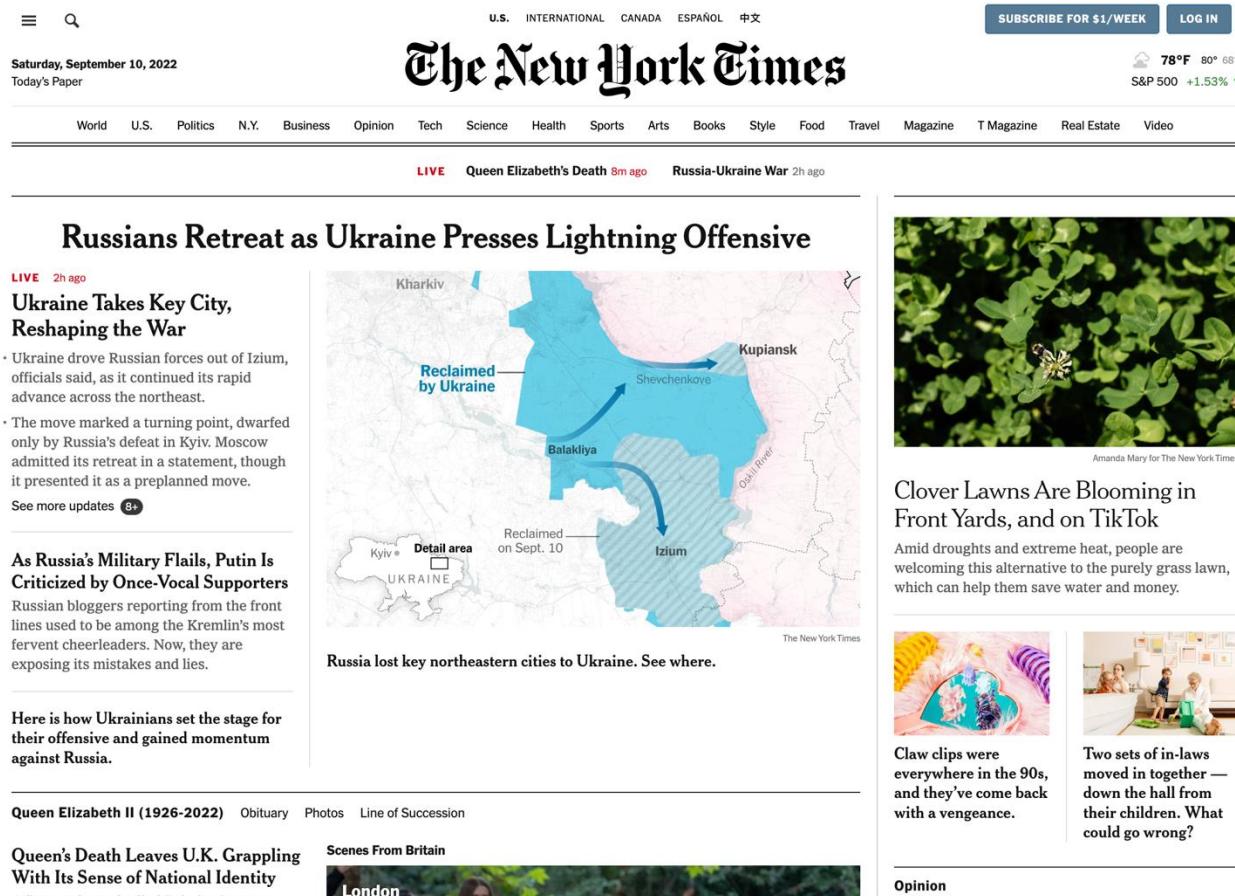


Fig 1. A reference image of the website <https://nytimes.com>



Fig 2. Element bounding boxes over the screenshot

### 4.3 Image Classification

The individual elements that are taken from the object detection network can then be passed to an image classification system. This system can be trained with the application specific data to better classify a given element.

This approach is agnostic to the underlying rendering technology, such as React / ASP.NET/ Kotlin / Swift/ etc. It looks purely at a given image, and what elements are there, seeing them as an end user. It gives the advantage too of not needing to understand the underlying page DOM properties.

### 4.4 Navigation

To assist in flow changes, tests can use reinforcement learning within a given application. This will give them the ability to robustly execute a given test, without needing to build the additional logic to check for possible new screens/pop-ups.

#### 4.4.1 App Graph

To start with this approach, an understanding of the application is first needed. This can be accomplished by learning from existing tests, or by doing a crawl. The purpose of this is to understand which buttons on a given screen result in going to a new a screen.

You can determine which screen is which by using the URL or page name for each screen. Each element can be determined via it's XPath or element name. When you combine these, you will get a graph where nodes represent a screen and edges are elements.

#### 4.4.2 Q-Learn

Upon building the app graph you can apply a reinforcement learning approach. A popular reinforcement learning technique is Q-Learn. Q-Learn is an algorithm that seeks to find the best action to take given the current state [11].

This state-action graph can then be applied where for a given element you wish to interact with, you know which state/node it should be in. For each action in a given test script, a system can then look at what the current state is, and the set of actions to get to a state where the element exists. It then changes the nature of a test script to only list what actions it should execute that are pertinent to the core logic being tested by that test case.

## 5 Additional Uses of AI/ML in the Testing Space

Within the testing domain, AI & ML is being applied with great effect. When used effectively it can help reduce the amount of time spent on testing, while improving test coverage and product quality. These are just a few examples of testing within the space.

### 5.1 Test Selection

Running a large test suite can take a long time and cost a large amount of money. This is a compounding problem as a team grows. There will be more people adding code, which means more tests will be written, and those tests will be run more frequently. Eventually running every test, for every change, becomes incredibly expensive.

This was the challenge at Google. In 2014 a team was formed to see if machine learning could be applied to analyze a given change set, and only run a subset of tests [7]. The results they found were that they could reduce the number of tests executed by 25% while still obtaining high test coverage numbers.

### 5.2 Mutation Testing

Mutation Testing is the software testing approach where values within the object under test are randomly mutated or changed (ex: Flag turned from True → False). The test suite is run to determine if that mutation is detected. It is useful as an approach to evaluate a test framework and see the systems recovery mechanisms.

The problem with this is that there are too many permutations to try to make it effective. Machine learning is being applied to learn which mutations to optimize for and reduce the number of mutations to run [6]. This will bring the cost and time to run this down which will make it a more valuable tool for everyone.

## 6 Summary

AI & ML is being used for testing today. It has real applications at companies both big and small. There are a variety of different methods that can be used, and various areas that it can be applied to.

## References

- [1] <https://towardsdatascience.com/creating-fake-news-with-openais-language-models-368e01a698a3>
- [2] <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>
- [3] <https://arxiv.org/abs/2104.11069>
- [4] <https://www.technologyreview.com/2022/05/27/1052826/ai-reinforcement-learning-self-driving-cars-autonomous-vehicles-wayve-waabi-cruise/>
- [5] <https://arxiv.org/pdf/2007.08220.pdf>
- [6] [https://link.springer.com/content/pdf/10.1007/978-3-642-34691-0\\_15.pdf](https://link.springer.com/content/pdf/10.1007/978-3-642-34691-0_15.pdf)
- [7] <https://testing.googleblog.com/2018/09/>
- [8] <https://medium.com/coinmonks/review-r-cnn-object-detection-b476aba290d1>
- [9] <https://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012029/pdf>
- [10] <https://iwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-020-01826-x>
- [11] <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>

# Driving Quality Improvement Through Root Cause Analysis

**Amol Patil**

apatil@mimecast.com

## Abstract

This paper discusses the approaches used at a software company building solutions for healthcare payers . Achieving engineering process improvements and gaining confidence in quality led to the adoption of RCA as a primary driver for engineering best practices. This paper covers what worked effectively to introduce a structured RCA program and also discusses the challenges encountered. Key success factors and an overall methodology are highlighted that include;

- Scaling improvements for large, distributed engineering teams
- Customer satisfaction and its relationship with defect removals that can be used to trigger process improvements
- Effectiveness of improvements evaluated using metrics

## Biography

*Amol Patil is Director of Quality Engineering & Services at Mimecast. He is responsible for ensuring world-wide quality standards for Mimecast products that use A.I. and Machine Learning models for threat detection and analysis of emails and attachments. Prior to Mimecast, he held Quality Engineering leadership positions at Healthedge and PTC. He is a Software Engineering leader with experience in leading and scaling distributed engineering teams to deliver high quality customer focused solutions. At PTC, he was responsible for building geographically distributed Engineering teams with strong ownership and morale. At Healthedge, he has achieved high automation levels and shorter lead times using CI/CD, improved productivity and ultimately improved customer confidence and building corporate value. Amol has experience in many markets; Cyber Security, Healthcare, IoT, PLM, Mobile and cloud platforms and has been delivering new products and platform integration's using any cloud-based DevOps infrastructure leveraging Docker, Terraform and Kubernetes. Amol has a M.S. degree in Industrial Engineering from University of Cincinnati.*

## 1.1 Opportunity for increasing the confidence in Quality

Software development teams should have a way of understanding what is most important to the customers and how they are doing with delivering great customer experiences. Teams unfortunately get that feedback through post-release defects. Teams then consume more than half their software development effort by applying it towards defect repair and testing. The CoPQ (Cost of Poor Quality) has already kicked in and a vicious cycle of engineering rework and rising customer care costs puts the relationship between customers, account managers and the Engineering organization into stress over the perception of quality.

Engineering adhered to all the practices that needed to be followed, like;

1. Understood the business needs
2. Brainstormed the solution
3. Implemented the solution
4. Examined the results through different levels of testing

All of this was achieved through an efficiently running software development engine that included

- Continuous integration
- Automated build and build verification
- TDD: Test Driven Development
- Automated regression testing

Engineering teams are now thinking about where it went awry.

This presents an opportunity for taking a step back and retrospect on why this happens and how the confidence in quality can be improved going forward.

## 1.2 Understanding the leading factors contributing to quality perceptions

In a typical enterprise software deliverable, there are major releases and minor releases.

The content and the date for receiving the major release are determined in advance and made available in the product roadmap. New features and business needs are addressed in the major release. Then, the minor releases are used to make improvements to the system that has been in production on the supported tech platform. In the minor releases, no new functions are added to the major release, and only improvements are made through bug fixing and software maintenance through software refactoring.

Now, when the real platform that houses the software in production has a large number of issues, the problems are visible for everyone to see it. Things start to get unsustainable when stakeholders cannot put a finger on why the system isn't functioning as expected. Doing a root cause analysis will provide the evidence and point in the direction of weak points. Engineering leaders can then drive outcomes based on that analysis.

I have always felt that, if you can solve a problem by asking the right questions the improvement path kind of makes itself visible. Let's start applying this questioning strategy beginning with engineering perceptions

1. Understood the business needs
  - What is the business routine that helps the customer generate revenue and how is it changing with this major release
  - Which systems are integrated upstream, midstream, and downstream that facilitate the customers business
2. Brainstormed the solution
  - Were all the components that constitute the system definition engineered to work together
  - Does the system scale
3. Implemented the solution
  - Was the software construction done consistently with best practices that consider functional and non-functional requirements
  - Was the system built with safeguards and default behavior
4. Examined the results through different levels of testing
  - Do the tests provide sufficient code coverage and scenario coverage
  - How effective are the tests to catch system level problems

A retrospective analysis of known defects can be applied to generate answers to all the above questions. Those answers should lead to gaps being identified and measures to improve defect avoidance. This type of root cause analysis and actionable improvements are being presented in this paper.

## 2.1 RCA Methodology

Following the principle that 'A defect in the software is a defect in the process', the leadership team decided to adopt the Orthogonal Defect Classification approach to understand and solve the underlying software problems and quality perceptions. The ODC technique was adopted as a methodology to characterize software defects and translate into process defects.

## 2.2 Orthogonal Defect Classification (ODC) technique

When successfully initiated, the ODC technique can be used for categorizing defects and reducing the cost of analysis based on predefined attributes. The main purpose of ODC is to extract semantic information from software defects to take actions against their re-occurrence to improve the process. In this sense, ODC can be used as a technique to realize the specific goals

Statistical KPI's like defect flow, defect density, defect remediation rates and test coverage are measured against test beds containing artificially created data and against a hardware footprint that is imperfectly sized. These types of KPI's do not set a relation to the system where defects originate and tend to fall short in identifying the root causes. A simple defect classification scheme like the one highlighted below provided distributions of defects against semantic data in each category.

Defect Origin → Defect Trigger → Defect → Defect Type → Fix Category

In this study of 200 defects reported and fixed in a span of 6 months in the 2020/2021 timeframe, the focus was on manually capturing and analyzing of defect data. The defect categories in this scheme are explained in the tables below

### **2.2.1 DEFECT TRIGGER:**

No.	Defect Trigger Name	Description	Example
1	3 <sup>rd</sup> Party Integration	Defect symptoms manifest when interaction happens with any external Integration	<i>Repricers, groupers, Claims editing systems</i>
2	Component Interoperability	Defect symptoms manifest when interaction happens with other components of the HE product portfolio	Platform, Care product, Custom code
3	Specific Data Condition	Data sources input had something unique or unexpected. System works as designed for majority of the input, but a certain small % of data fails functional coverage due to the distinctive nature of the data	-- <i>Payment cycle not correct</i> -- <i>Member links not updated after changes</i> -- <i>Validation policies not triggered</i> -- <i>Showing incorrect information in the UI interface</i> -- <i>Logic retrieving incorrect claim processing details</i>
4	Data Consistency	The business transaction failed to change affected data only in allowed ways	-- <i>Data processed differently by Payer Engine and the webservices</i> -- <i>Data replication / streaming. Mismatch / Missing records between OLTP and DW</i> -- <i>Mismatch in Source EDI and the Payer engine</i>
5	Selective Transaction	A few handful transactions out of many failed to complete or completed but not as expected. High material impact	-- <i>Items in the work basket. 1 claim, 3 members, 5 accounts etc.</i> -- <i>Items that put the HIC out of compliance</i> -- <i>penalties, interests</i>
6	Batch Transaction	Event driven / scheduled operational items that are part of regular business routines failed to complete. High % of the volume routed from auto-processing to manual processing.	-- <i>Adjudication sending large # of items to workbasket, payment batches not completing, member enrollment, Selective bootstrapping.</i> -- <i>Transient data clean up scripts, custom server scripts</i> -- <i>Manual re-adjudication/repair became necessary</i>
7	Software Upgrade	Codebase upgraded and end user tried to repeat a transaction that was working as expected in the previous version	-- <i>Job performance is not on par with pre-upgrade performance and is currently impacting regular business</i>

			-- <i>Claims Search is taking over 2 minutes after upgrade</i> -- <i>Mapping for a field has changed</i> -- <i>Data Integrity between OLTP and DW</i>
8	Configuration	Changes in configuration happened before the symptoms manifested	-- <i>Updating configurable parameters of the product to support new business functions</i>
9	Design	Logic Design and Data Design	-- <i>Index added to improve the performance, but caused contention during heavy inserts/updates/deletes of the attachments</i>
10	Volume	The business routines work as designed with expected results up to a certain threshold. Beyond the threshold, transactions and computing fails	Search throws OOM error.

## 2.2.2 DEFECT ORIGIN:

No.	Defect Origin Name	Description	Example
1	Customer Prod	Production environment	-- PROD
2	Customer Non-Prod	Non-Production environments but in control of the customer	-- QA, Dev, SIT, Stress, Test, Pre-Prod
3	Internal Non-Prod	Environments in control of the Software Engineering group	<i>Example:</i> -- Engineering or Agile Services environments -- Perf CI environments

### 2.2.3 DEFECT TYPE:

No.	Defect Type Name	Description	Example
1	System Integration	System integration with various 'external' components not native to the product under implementation	-- <i>Repricers, Fraud Detection</i> -- <i>Upstream loaders, Downstream replicated datastore</i> -- <i>Custom code</i>
2	Regression	Business routines under use stop working as expected after a change in functional configuration/customization/logic	-- <i>Payment cycle not correct</i> -- <i>Member links not updated after changes</i> -- <i>Validation policies not triggered</i> -- <i>Showing incorrect information in the client</i> -- <i>Logic retrieving incorrect claim processing details</i>
3	Performance Improvement	Functionality is not broken, but the code was refactored to show performance gains	-- <i>'Out of Memory' problems</i> -- <i>Services responding slowly</i> -- <i>UI results responding slowly</i> -- <i>Concurrent calls to a single record (member/account/claim)</i> -- <i>Database locks and transaction locks</i> -- <i>Calling many more rows than required</i>
4	Negative Use Case	End users transaction was not conforming to the expected incoming data sources leading to un-expected outcomes	<i>User passed 1010-10-10 in the date range that expects date range to be within 1800-01-01 to 3000-01-01</i>
5	Func. Requirements not documented, but expected to work	Major functionality works, but specific scenario(s) do not work as expected	<i>These are scenarios or conditions that end up causing major business impact. Expected case not completely understood or missed</i>
6	Data Validation	Needs adjustment to the software logic that properly validates the data and values before used in computational logic or database storage	
7	Instructions Not Clear	End user followed the documentation, but for areas where there was ambiguity or lack of clarity, the end user expected the functionality to be supported as per the business use case	

## 2.2.4 FIX CATEGORY:

No.	Fix Category Name	Description	Example
1	User Interface	Fix addressed through UI changes	<i>Grid View changes</i>
2	Build/Package/Merge	Solution existed but was not included due to procedural issues	<i>Rebuilt the package to include newer or missing libraries or missing LOC of known solutions not merged forward</i>
3	Functional Coverage	New functional logic introduced to handle missing or wrong functionality	<i>LOC that introduced logic to process and compute newer types that are known at time of requirements gathering</i>
4	Logic Coverage	Addressed an inadequate (efficiency) or wrong (correctness) algorithmic realization	<i>LOC that introduced new -- Service methods -- getAction() -- setAction()</i>
5	Defensive Coverage	Address poorly defined code boundaries and data validation for unexpected data resources	<i>LOC that introduced handling of -- Amounts set to 0 -- null or !null values -- handling of terminated / missing / canceled / inactive / invalid states of transactions -- Service date mismatches -- Exception handling: NPE, NumberFormatException</i>
6	Checking	Affects program logic that would properly validate data and values before they are stored or used in computation.	<i>-- Matching approved conditions for Authorizations, Agreement details, Service provisions -- Initialization of control blocks or data structures</i>
7	Runtime Resource Handling	Addressed the code to handle proper management of shared and real-time resources	<i>Free resources at runtime -- Network connections -- database connections -- file streams -- occupied memory -- Timeouts, socket timeouts -- Serialization/multi-threading</i>
8	Database Query	Queries adjusted/introduced/enhanced to handle the case highlighted in the defect	
9	Database Design	Schema adjustments made	
10	Data Migration	Migration scripts modified/introduced to upgrade to a newer software version	
11	Documentation	Addressed the technical documentation for missing instructions/information	

### 3.1 Data Analysis

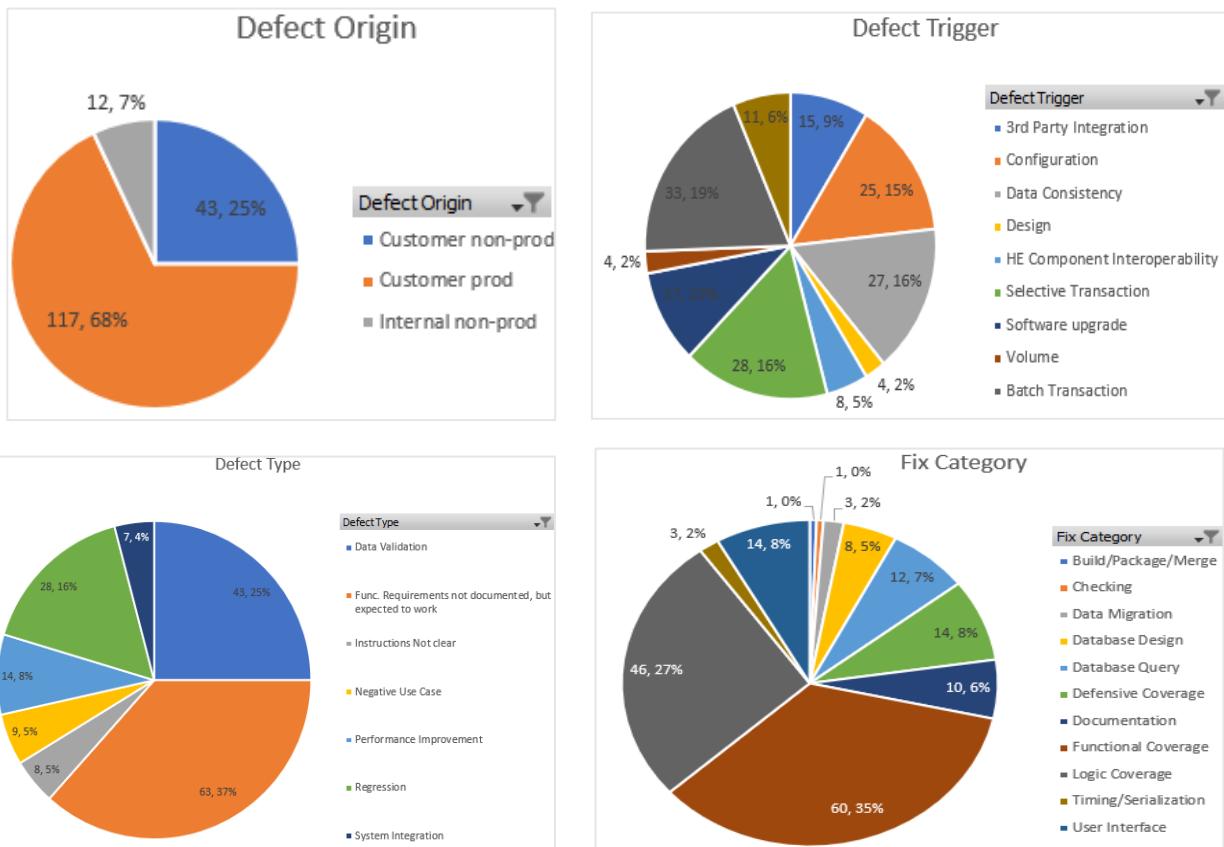
The defect set was chosen for a product that is subject to improvement.

The data was prepared by analysis completed by a group of engineers that undertook training in the RCA program pilot implementation. The definitions were approved through a consensus method. The defect classifications for each category were recorded and discussions were held to clarify whether the process is proceeding according to the modeled process.

- The Defect Origin and Defect Trigger was obtained from the incident report management tool (*Salesforce*), activity information, and the person who reported the defect
- The Defect Type and Fix category was identified by analyzing the traces and explanations given for the correction of defects in the defect management tool (*Jira*) and the source code management tool (*Bitbucket*)

By profiling the defects using the classification scheme and analyzing the incidents, problematic areas and underlying process gaps bubbled up to the top as the likely causes of software defects.

The figures below show the distribution of each category



**3.1.1:** The figures below show the examples of distributions of Defect Types with respect to Defect Triggers

→ Defect Type ↓ Defect Trigger	Data Validation	Func. Requirements not documented, but expected to work	Instructions Not clear	No default behavior for negative use case	Performance Improvement	Regression	System Integration	Grand Total
3rd Party Integration	4	4	1	1		3	6	<b>19</b>
Batch Transaction	13	12			4	9		<b>38</b>
Configuration	3	14	4		3	4		<b>28</b>
Data Consistency	18	3	1	2		6	1	<b>31</b>
Design		2			1	1		<b>4</b>
Component Interoperability	1	3	1	1	2	3		<b>11</b>
Selective Transaction	7	15	1	3	4	6		<b>36</b>
Software upgrade	4	7		1	1	6	1	<b>20</b>
Specific Data Condition		11				1		<b>12</b>
Volume		1		1	2			<b>4</b>
<b>Grand Total</b>	<b>50</b>	<b>72</b>	<b>8</b>	<b>9</b>	<b>17</b>	<b>39</b>	<b>8</b>	<b>203</b>

**3.1.2:** The figures below show the examples of distributions of Defect Types with respect to Fix category

→ Defect Type ↓ Fix Category	Data Validation	Func. Requirements not documented, but expected to work	Instructions Not clear	No default behavior for negative use case	Performance Improvement	Regression	System Integration	Grand Total
Build/Package/Merge						2		<b>2</b>
Checking		1				1		<b>2</b>
Data Migration	2	1				1		<b>4</b>
Database Design	2	1			3	2		<b>8</b>
Database Query	1	5			2	4		<b>12</b>
Defensive Coverage	18			1				<b>19</b>
Documentation	1	1	8			2		<b>12</b>
Functional Coverage	9	37		1	7	8	5	<b>67</b>
Logic Coverage	16	20		4	2	12	2	<b>56</b>
Timing/Serialization	1				1	1		<b>3</b>
User Interface		6		3		6	1	<b>16</b>
Runtime Resource Handling					2			<b>2</b>
<b>Grand Total</b>	<b>50</b>	<b>72</b>	<b>8</b>	<b>9</b>	<b>17</b>	<b>39</b>	<b>8</b>	<b>203</b>

**3.1.3:** The figures below show the examples of distributions of Defect Triggers with respect to Fix category

→ Defect Trigger ↓ Fix Category	3rd Party Integration	Batch Transaction	Configuration	Data Consistency	Design	HE Component Interoperability	Selective Transaction	Software upgrade	Specific Data Condition	Volume	Grand Total
Build/Package/Merge			1					1			2
Checking		1				1					2
Data Migration		1					1	2			4
Database Design		1			2		2	3			8
Database Query		6		1			2	3			12
Defensive Coverage		7		12							19
Documentation	2	1	4	1		1	1	1	1		12
Functional Coverage	7	12	10	6		3	12	6	8	3	67
Logic Coverage	9	6	8	9	1	5	13	3	2		56
Timing/Serialization			2	1							3
User Interface	1	3	3	1	1	1	3	1	1	1	16
Runtime Resource Handling							2				2
<b>Grand Total</b>	<b>19</b>	<b>38</b>	<b>28</b>	<b>31</b>	<b>4</b>	<b>11</b>	<b>36</b>	<b>20</b>	<b>12</b>	<b>4</b>	<b>203</b>

## 3.2 Root Cause Disposition

The educated evaluation from the data analysis will help reach conclusions on the root cause or the best fit for the root cause. Some examples;

- ↓ Defect Type of 'Requirements not documented, but expected to work' could not be discovered in the requirements intake process and therefore reached the codebase as conditions that could not be satisfactorily handled. This is a gap in the requirements gathering and requirements breakdown stage. Source – 3.1.2
- ↓ The Fix Category of 'Logic coverage' not handling 'Data validation' type of defects indicates that Design, and Code reviews were found inefficient. This engineering gap needs to be addressed. Source – 3.1.2
- ↓ Regression type of defects were mostly triggered when running batch transactions. This could be an area of improvement for inspection and testing as code reviews and unit tests were not sufficient. Source – 3.1.1
- ↓ Defects triggered by 'Specific Data Conditions' needed to be adjusted for coverage – functional and logic. Story Grooming practices were not considering all possible data conditions. Source – 3.1.3

During the discussion of the defect classification findings with engineers, the following evaluations and comments were showing up in the heatmap as frequent causes with respect to the ODC attributes:

- Requirement reviews can be more effective
- Design and Code reviews can be more effective
- Grooming can be more regular and effective
- The effectiveness of system level tests can be increased
- Defects in database queries can be reduced
- The number of changes in the design due to misunderstandings in requirement stage can be reduced
- The effectiveness of unit tests can be increased to discover code-related defects in logic and validation

Since the process outcomes were the measure of success for implementing process improvements, a root cause disposition for each defect was created. Accountability and ownership were assigned to each function for new process enactments and measuring their success and effectiveness. The table below shows the final root cause dispositions that were put in use by the Engineering department.

### 3.2.1 ROOT CAUSE DISPOSITION:

No.	Root Cause Disposition	Behavior Change Responsibility	Example
1	Engineering Practice Gaps	Tech Leads will bring about an actionable change & measure effect of implemented actions	-- extend training offers and attendance on architecture and improve systems design skills -- enhance or tighten the DoD -- introduce or update the checklist for application domain to be used in backlog refinement
2	Lack of Business knowledge	PO's will bring about an actionable change and use it to measure outcomes PM's take the use cases back to customer and plug the gaps for future PFRs	-- extend training offers and attendance on application domain -- Continuous and iterative improvement in requirements intake -- Update story template to provide NFR
3	Performance not considered	Performance refactoring team	-- Non-Functional requirements section to be added to the requirements
4	Instructions not clear	Tech Writers to bring about an actionable change to remove ambiguity and introduce clarity in technical write ups	-- Use Defect Trigger categories to better document the functional usage of the features
5	Architecture Gap	Architecture review Board should take up actionable changes to bring tech stack changes that will last the next 10 years	-- Defects resulting from obsoleted and deprecated libraries -- data governance and data model governance policies for engineers

From all the root cause dispositions available for selection, each defect was tagged with the root cause

### 3.2.1: The figures below show the examples of distributions of RCA with respect to Defect Type

Row Labels	Instruct-ions Not clear	Negative Use Case	Regress-ion	Perfor-mance Improve-ment	Func. Requirements not documented, but expected to work	Data Validation	System Integration	Grand Total
Engineering Practice Gaps	1	8	36	7	28	39	7	126
Lack of Business knowledge	7		1	3	44	10	1	66
Performance not considered		1		7		1		9
Instructions not clear			2					2
<b>Grand Total</b>	<b>8</b>	<b>9</b>	<b>39</b>	<b>17</b>	<b>72</b>	<b>50</b>	<b>8</b>	<b>203</b>

### 3.2.2: The figures below show the distributions of RCA with respect to Fix category

Row Labels	Build / Pack-age / Merge	Data Migra-tion	DB Design	DB Query	Docu-menta-tion	Timing / Serial-ization	Logic Cover-age	Func. Cover-age	UI	Def. Cover-age	Check -ing	Runtime Resource Handling	Total
Engineering Practice Gaps	2	3	7	9	2	2	39	30	12	19	1		126
Lack of Business knowledge		1		2	8		16	34	4		1		66
Performanc e not considered			1	1		1	1	3				2	9
Instructions not clear					2								2
<b>Grand Total</b>	<b>2</b>	<b>4</b>	<b>8</b>	<b>12</b>	<b>12</b>	<b>3</b>	<b>56</b>	<b>67</b>	<b>16</b>	<b>19</b>	<b>2</b>	<b>2</b>	<b>203</b>

This RCA was assigned using a narrative flow that can lead to simplified way of reaching the RC conclusion

- Defect Trigger in Defect Origin led to Defect Type that was addressed in this Fix category leading to the RC Disposition

In an agile environment, it is difficult to find dedicated time and resources to perform detailed fish bone diagram or 5 Why's method. So, in practice an RCA can be done for each defect in a spreadsheet using the defect characteristics. It does not take much time when institutionalized within the process.

Once RCA is assigned, the next step is to determine the concrete improvements that will lead to a change in process.

## 4. Measures for Defect Avoidance

The results of the analysis through this simple defect classification scheme will be used to build new engineering improvements that can help answer and measure the questions asked previously in Section 1.2. Agile teams need to constantly decide what practices to keep and which practices to discontinue. The RCA distributions can help to come up with a disciplined and structured approach to improvement schemes.

Engineering Practice gaps and Lack of Business knowledge are the top 2 categories covering 75% of the defect population. (Source - 3.2.1 and 3.2.2). These 2 RCA areas can be used to come up with defect avoidance practices and reduce defect driven rework

Each grouping will provide a correlation that can be used. For example: Defects are requiring mostly major functional coverage changes to address the issue (Source - 3.2.2) and large portion of the defects are stemming 'lack of business knowledge' (Source - 3.2.2). The defects of Fix category "Functional Coverage" and "Logic Coverage" dominate by far all the other fix categories. 68% or more than 2/3<sup>rd</sup> of the defects were found in Customers production environment (Source - 3.1.1), leading to believe that the defects are insidious in nature until the customers business routines are run by customers business users. This analysis shows that important areas to look for improvements are reviews, grooming, domain and system knowledge and test strategies.

To address the causes, that make defects detected only by the customers end users, a quality improvement plan was implemented and tracked. Here are some high-level examples of goals set for each department:

- Product Management Team: Requirements gaps
  - Use checklist for systemic understanding of data entities in use and business needs
  - Completeness of requirements should be ensured
  - Confirmation of the final requirements and acceptance criteria by the customer should be improved
  - The experience of software engineers should be extended through training
- Software Engineering: Engineering Gaps
  - Backlog refinement process to be more systemic and detailed using checklists
  - Coding practices. Examples:
    - Adding null checks for any method which returns a string
    - Null check for optional attributes
    - Transient record creation
    - Duplicate values
    - Checking for stale objects
  - Code coverage and static analysis reports to be incorporated into the Definition of Done
    - Green pipeline criteria introduced before approving pull requests
  - Default Behavior should be introduced when coding new functional coverages
    - Rather than throwing assert exceptions, the code should execute default behavior
  - Strengthen Definition of Done to include missing technical aspects like data validation
  - Adherence to review and best practices to be enforced and measured
    - Fun and friendly audits to check adherence

## 5. Conclusion

Implementing ODC way of Root Cause Analysis is very cost-effective and applied using a simple principle of “*Take what you already know and apply it to what you think you know to produce quality software*”. The focus is on the data already collected (software defects). Defect profiling will also help the engineers to build their standards for design, architecture, policies to be focused on prevention.

The defect classification scheme can be implemented in stages by starting with a simple scheme and then moving on to in-process analysis. Fields can be tailored to your own organization. It can be tooled quickly to do in-process defect profiling. It can become a part of the Definition of Done to make sure the analysis is always complete. Fields added to data management tools like Jira and Salesforce that are completed in real time, will make the data collection virtually free.

Many things can impact the confidence in quality. To motivate the teams to make the needed investments in quality driven practices and defect avoidance mechanisms, the situations need to be evaluated first. The RCA method using defect profiling will help the organization understand the motivations for continuously building and adopting effective practice changes that will lead to greater confidence in quality.

## References

<http://www.chillarege.com/odc>

Challenges of software process and product quality improvement: catalyzing defect root-cause investigation by process enactment data analysis by Mehmet Soylemez and Ayca Tarhan, 2016

A Case Study in Root Cause Defect Analysis by Marek Leszak, Dewayne E. Perry, and Dieter Stoll

# The Versus Framework: A File Comparison Standard for Software Testing

**Omar Mohamed Ragi  
Ahmed Sayed Ali  
Reem Al-Adawi**

omar\_ragi@mentor.com  
ahmed\_alii@mentor.com  
reem\_eladawi@mentor.com

## Abstract

In any software automated testing environment, file comparison is an essential step to determine if the outputs of the system under test match the expected results. Improving this testing step significantly enhances the testing flow and makes the whole process robust and reliable. For this reason, we created Versus, a file comparison framework and standard designed to facilitate the files comparison task by offering QA engineers a state-of-the-art internal tool to perform the comparison using the optimal environment and settings, eliminating the need to write and maintain complex scripts.

The main concepts of the Versus framework are maintainability, and ease of use. The framework is designed to only accept one input, a human-readable, easy to create and maintain, [YAML](#)-formatted configuration file that contains definitions of all the comparison checks. Versus parses this YAML file, executes the comparison checks, and generates an easy-to-analyze comparison summary report. This report, designed with the goal of reducing test failure analysis time, directs engineers to the fastest method for reviewing detected differences.

By standardizing file comparison, one of the most important steps in any automated testing flow, the Versus framework is a great tool for the automated testing environment. It makes fast, accurate file comparison accessible to all QA engineers, regardless of experience, saving time while ensuring best practices are used for file comparison in every automated test scenario.

This paper presents the concepts of the Versus framework, and how these concepts can and should be adopted in any QA testing automation process aiming for high efficiency and productivity.

## Biography

**Omar Mohammed Ragi** is a SW QA manager for Siemens Digital Industries Software. He has a B.Sc. degree in Computer Science from Arab Academy for Science, Technology & Maritime Transport, Cairo, Egypt.

**Ahmed Sayed Ali** is a Senior QA Engineer for Siemens Digital Industries Software. He has a B.Sc. degree in Electronics and Communications from Ain Shams University, Cairo, Egypt.

**Reem El-adawi** is Director of Quality for Siemens Digital Industries Software. She holds a B.Sc., M.Sc., and Ph.D. from Ain Shams University, Electronics and Communication department, Egypt.

# 1 Introduction

Any testing automation system has two main steps:

- running the tool under test after providing the correct inputs that define the scenario to be tested
- comparing the outputs of the tool under test after the run is complete to determine if they match the expected results (the baseline or “golden” results).

The Versus framework simplifies and standardizes the comparison step, making the file comparison easy, and accessible for any QA, regardless of their experience level, while offering a robust process based on best practices. The Versus framework is a valuable component of ensuring the best tools, practices, and optimization are maintained in testing flows. In this paper, we present the main components of the Versus standard, and explain how they are synergized to achieve the described goals.

## 2 Principles of the Versus standard

Versus is both a framework and a file comparison standard that can be used to compare files efficiently and easily, making it the perfect tool for testing automation comparison step. There are five principles in the Versus comparison standard—these principles are the pillars on which Versus is created and designed (Fig. 1). These principles serve and underlie the ultimate goal of making fast, accurate file comparison accessible to all QA engineers without the need for prior experience or scripting skills.

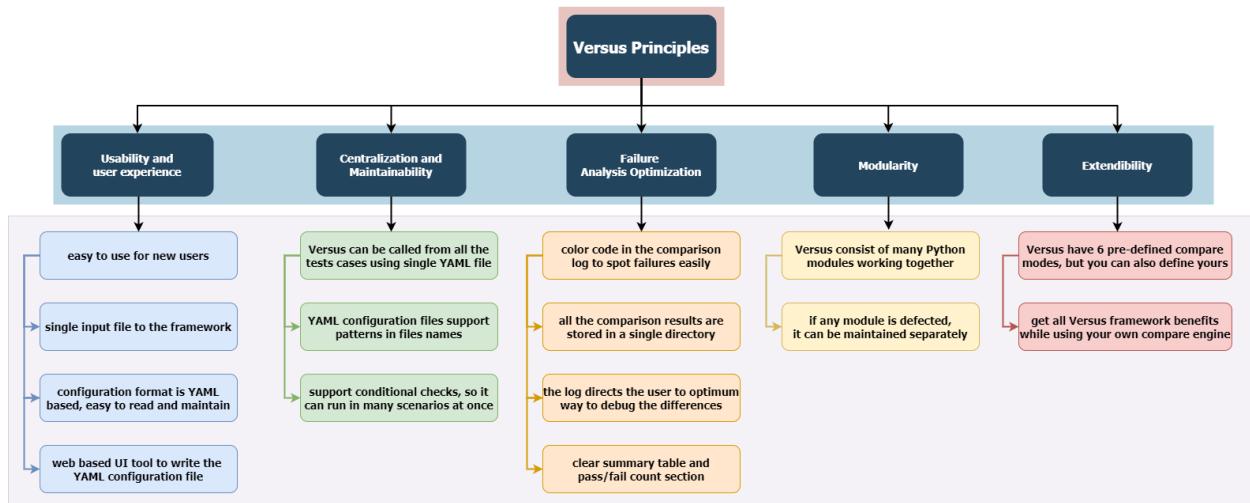


Fig. 1. The five principles of the Versus framework.

In the following sections, we define these principles and show how they are incorporated into the design of Versus.

### 2.1 Usability and user experience

The first principle of the Versus framework is usability. To enhance ease of use for new or inexperienced QA engineers, we focused on making the getting-started experience with the Versus framework easy and fast. By default, the Versus framework only accepts one input file, which reduces potential sources of complexity in the file comparison process.

The Versus input is a [YAML](#) format file, which is not only human-readable, but can also be parsed by many modules and libraries in almost any scripting language. Entry names and reserved words used in this YAML format input file are chosen carefully to be intuitive and easy to remember, to limit the need to consult the documentation whenever users need to write a new Versus configuration file.

To make the Versus framework easier to configure, we built a web-based UI application that enables users to choose the type of comparison, inputs, and settings for each check. The application then automatically generates the YAML configuration file, ready to be used with the Versus framework (additional details are provided in section 8).

This web-based UI application is also deliberately designed to minimize any need to consult the documentation of the Versus framework. It starts by asking users the important questions first, then guiding them through multiple questions until it has all the information needed to create the YAML configuration file. This approach eliminates the need for users to have prior knowledge about the syntax of the input configuration file, or the available entries it contains (additional details provided in section 7).

## 2.2 Centralization and maintainability

Centralization is a very powerful concept that is essential for a successful and maintainable testing automation process. Centralizing key components of the testing automation scripts and tools reduces the time and cost needed to develop them, allowing many team members to use a single source of code or tool that would otherwise have been re-invented many times in potentially different levels of quality. Centralization also supports and simplifies system maintenance—if any part of the testing tools needs modification or fixing, it can be done on a single location, instantly providing the resulting corrections or enhancements to all the test cases (scenarios) in the testing automation regression suite using the centralized testing tool or script.

The Versus framework is designed with centralization in mind to maximize its maintainability. If many test cases use the same comparison check, they can all use the same YAML configuration file, so changing anything in this single YAML configuration file instantly updates all of the testing suites using that check.

The Versus configuration file also supports defining file names with patterns, and even adds arrays of file names in both the entries of the output file and the baseline data. These features mean that even if the test cases in an automated testing suite are comparing different types of files, or files with different names, all the test cases can share the same compare checks, and whatever output in each test matches the file name pattern you entered will be compared just as if the full name of the file was written.

The Versus configuration file also supports conditional compare checks (i.e., those that depend on certain conditions to trigger the comparison). If the conditions are not present, the comparison will be ignored, or be performed with relaxed tolerances. This feature makes it easier to support a centralized configuration file.

## 2.3 Failure analysis optimization

Another goal of the Versus framework is to minimize the time required to analyze the failure of an automated test case, which includes spotting the source of the failure in the shortest time possible. The Versus framework achieves this goal with an optimized output log containing multiple shortcuts and aids to save analysis time.

If Versus is run from a terminal, important data in the log is color-coded so users can quickly distinguish passing compare checks from failing checks without the need to read every line in the log (refer to section 6).

Versus generates a “transcripts” directory while executing the comparison checks that contains all the comparison results, as well as the intermediate files used in the comparison execution. This directory is well organized with a specific hierarchy based on the compare checks parameters (refer to section 7).

Log contents are minimized. At the end of a log, Versus prints two summary reports: a table that contains all the compare checks and their final status, and a final count of the checks, including how many passed, and how many failed. This gives users a quick glimpse of the general status of the automated test case before they begin analyzing the failures and going deep into the comparison failure log (refer to section 6). However, the log doesn’t contain the results of the comparison in the summary report, but provides a

reference to its location in the transcripts directory. (refer to section 6) The log also provides the optimal way to debug a comparison check failure—e.g., if the check type is an image comparison, the log provides a link to an image that contains a merged version of the two compared images, with the differences highlighted (refer to section 6).

Following these guidelines cuts down the automated test failure analysis time by a considerable amount and saves a lot of time for the QA engineers in one of the most important roles of their jobs, which is maintaining their automated testing suite.

## 2.4 Modularity

Versus code is built in a modular style, meaning each comparison mode, each extractor, and even the part that generates the final log and the summary report, are all built into separate python code sources. Modularity ensures that if any defect is introduced into one module, the other modules and pieces of code are not affected, while also making it easier to find and fix errors.

Modularity also supports some “plug-ins” that can be used to extract and filter the output files (or metric values) to be compared to the baseline version, so users can create their own python code to manipulate the output data, making it even more modular.

## 2.5 Extendability

Versus allows users to include a custom comparison mode that uses a different comparison engine, rather than the engines supported by Versus by default (those described in section 5). This feature lets users use their preferred engine within the Versus framework environment, providing them all the benefits of Versus while allowing them the flexibility to use their own comparator (refer to section 5.7).

# 3 Versus framework components

In this section, we describe the main framework components that define the experience of using Versus in testing automation (Fig 2).

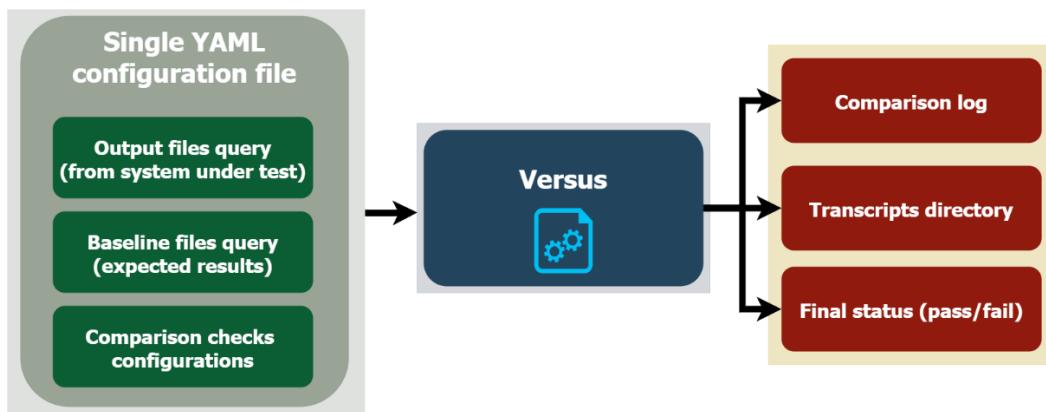


Fig. 2. Versus framework components.

### 3.1 Comparison script inclusion

Each test in the automated testing suite is expected to call Versus framework from its main location (source), using a central configuration file that contains all the needed comparison checks. This recommended approach achieves the Versus principle of centralization.

Fig. 3 shows a typical Versus call inside the main compare script of each automated test case.

```

1  #!/bin/sh
2
3  #calling Versus.
4  /user/pvtools/bin/versus --yaml /central/location/versus.yaml
5  status=$?
6
7  #exiting with proper status.
8  exit $status

```

Fig. 3. The recommended method for calling the Versus framework.

### 3.2 Configuration file

The configuration file is the only input Versus needs. It contains all the data needed to extract the files and metrics to be compared, and all the comparison checks to be executed on the outputs against the expected results (baseline version).

The configuration file must be written in YAML format (while Versus accepts JSON, it is not recommended as it is not as human readable). The typical Versus configuration file contains three sections in total (Fig. 3):

- **scripts**: this section is dedicated to any script or shell command users want to run inside the automated test before running the extraction and comparison checks. It is typically used to run some code to filter unwanted data from the output files of the tool under test, or to generate an altered version of these output files that are more suitable for comparison.
- **extract\_metrics**: this section is dedicated to defining the extractors, which are a set of python functions to be run with the parameters passed to them by the users. Its goal is to extract the needed values (metrics) from the output files to be compared to the baseline version.
- **compare\_metrics**: this section is dedicated to defining the comparison checks that Versus will execute on the output files, the extracted metrics, or the filtered/altered versions of the output files.

```

1   scripts:
2   |
3   |   - /path/to/script1.sh
4   |   - /path/to/script2.sh
5   |   - ...
6
7   extract_metrics:
8   |
9   |   - metric1:
10  |   |   <parameters>
11  |   - metric2:
12  |   |   <parameters>
13  |   - ...
14
15   compare_metrics:
16   |
17   |   - check1:
18   |   |   <parameters>
19   |   - check2:
20   |   |   <parameters>
21   |   - check3:
22   |   |   <parameters>
23   |   - ...

```

Fig. 4. Versus configuration file sections.

More details on the configuration file sections are provided in section 4.

### 3.3 Log and summary report

Versus is always verbose about what it is doing at any moment. The log it generates shows the current step and the results of this step, as shown in Fig. 5. After the comparison is done, Versus generates a summary report table that summarizes the results of all of the compare checks and gives users final counts of passing and failed checks. More details about the log and summary report are provided in section 6.

```

Reading versus.yaml

Check: CSV_COMP_normal
| /user/peteoss/aoi/bin/numdiff -z @ -S --separators=' \t\n\r, ::;"' ./my_out1.csv ./baseline/my_out1.csv
| FAILED, check this file: ./transcripts/numdiff_transcripts/CSV_COMP_normal.transcript

Check: CSV_COMP_fast
| /user/peteoss/aoi/bin/numdiff -z @ -S --separators=' \t\n\r, ::;"' ./my_out2.csv ./baseline/my_out2.csv
| FAILED, check this file: ./transcripts/numdiff_transcripts/CSV_COMP_fast.transcript

Compare Report:
+-----+-----+-----+-----+
| check | baseline value [Baseline] | metric value | check type | status | comments |
+-----+-----+-----+-----+
| CSV_COMP_normal | ./baseline/my_out1.csv | my_out1.csv | NUMDIFF | FAIL | ./transcripts/numdiff_transcripts/CSV_COMP_normal.transcript |
+-----+-----+-----+-----+
| CSV_COMP_fast | ./baseline/my_out2.csv | my_out2.csv | NUMDIFF | FAIL | ./transcripts/numdiff_transcripts/CSV_COMP_fast.transcript |
+-----+-----+-----+-----+

Summary Report:
=====
Total:2
PASS :0
FAIL :2
=====
```

Fig. 5. Versus log and summary.

### 3.4 Transcripts directory

The transcripts directory is where Versus keeps all intermediate files it generated, any filtered files, and the results of the comparisons it made. It is organized in a specific hierarchy to be intuitive and self-explanatory. If users know the check names they chose in the configuration file, they can easily navigate the transcripts directory to get the files they want for failure analysis (Fig. 6). More details about the structure of the “transcripts” directory are provided in section 7.

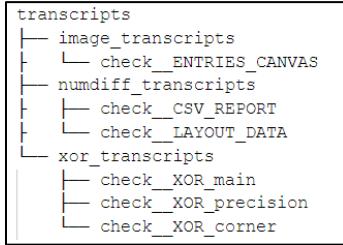


Fig. 6. Transcripts directory hierarchy.

## 4 Versus configuration file

### 4.1 Introduction

The configuration file is the Versus framework’s entry point, where users describe the comparison checks for all automated tests. It is designed to meet the following criteria:

- Easy to read, understand and maintain
- Supports generic comparison checks, so that one configuration file can be used by the whole regression suite
- Modular and re-usable (check descriptions can be re-used in other configuration files)

In the following sections, the main sections of Versus configuration file will explained in details.

### 4.2 Versus YAML configuration file sections

The Versus configuration file contains three main sections:

- Scripts
- Metrics extraction
- Metrics comparison

#### 4.2.1 Scripts

The *scripts* section is the first to be executed by Versus. Users can use this section to define any script/command that must be run before the comparison checks (Fig. 7). These commands/scripts are typically used to prepare the output files for comparison (filtration, aggregation, ... etc.).

```

1   scripts:
2     - /path/to/filter.py output.txt > outputfilt
3     - grep 'error' transcript*log* > errors.log
4     - ls core.* > core_files.list
5     - ...
6     - ...

```

Fig. 7. *scripts* section of configuration file.

#### 4.2.2 Metrics extraction

The *extract\_metrics* section is used to call pre-defined metric extractor functions for extracting and storing the metrics to be compared in a standard format that can be easily compared (Fig. 8).

```

1   extract_metrics:
2     - <extraction_step_1>
3       functions: [extractor_func_1]
4       params: { name1: value1 }
5     - <extraction_step_2>
6       functions: [extractor_func_1,extractor_func_2,...]
7       params: { name1: value1, name2: value2 }
8     - ...

```

Fig. 8. *extract\_metrics* section syntax.

#### 4.2.3 Comparison metrics

The *compare\_metrics* section is used to define the comparison checks that use one of the pre-defined comparison modes (Fig. 9).

```

1   compare_metrics:
2     - <check_name_1>
3       mode: <check_type>
4       metric: <output file(s)/folder(s)>
5       baseline: <baseline file(s)>
6       <check_parameters>
7     - <check_name_2>
8       mode: <check_type>
9       metric: <output file(s)/folder(s)>
10      baseline: <baseline file(s)>
11      <check_parameters>
12     - <check_name_3>
13       mode: <check_type>
14       metric: <output file(s)/folder(s)>
15       baseline: <baseline file(s)>
16       <check_parameters>
17     ...

```

Fig. 9. *compare\_metrics* section syntax.

## 5 Versus comparison modes

Versus has six pre-defined comparison modes that cover most of the common output file types, as well as a seventh custom mode that can be used to compare any special file type by incorporating another comparison engine within the Versus framework, gaining all of the benefits of the framework without re-inventing the wheel.

### 5.1 diff mode

The *diff* mode is the simplest comparison mode in Versus—it compares two files line by line using the standard Linux diff command [1]. The comparison check passes only if the output file and baseline are identical (no tolerance support in this mode). Fig. 10 shows some examples of the *diff* mode usage.

```

1  compare_metrics:
2    #simplest scenario
3    - check_one:
4      mode: diff
5      metric: ./file.txt
6      baseline: ./baseline/file.txt
7    #comparing array of files.
8    - check_two:
9      mode: diff
10     metric: ['./outputs/file1.txt','./outputs/file2.txt,']
11     baseline: ['./baseline/file1.txt','./baseline/file2.txt,']
12   #having wildcards in the baseline (multiple files comparison per check).
13   - check_three:
14     mode: diff
15     metric: ./outputs
16     baseline: ./baseline/*.{txt,csv}

```

Fig. 10. *diff* mode syntax and examples.

### 5.2 scalar mode

The *scalar* mode is used to compare two singular numerical values, usually stored in files that are generated by the *extract\_metrics* functions. This mode supports numerical tolerance, and ignores positive/negative differences. The comparison check passes only if the two metrics match, or if the error is within the defined tolerance. Fig. 11 shows the syntax of the *scalar* mode and its available parameters.

```

1  compare_metrics:
2    - check_name:
3      mode: scalar
4      metric: <output file(s)/folder(s)>
5      baseline: <baseline file(s)> | <number>
6      a: <absolute_tolerance>
7      r: <relative_tolerance>
8      ignore_negative: <'0'||'1'>
9      ignore_positive: <'0'||'1'>

```

Fig. 11, *scalar* mode syntax.

Fig. 12 shows examples of the *scalar* mode.

```

1 compare_metrics:
2     #simplest scenario
3     - check_one:
4         mode: scalar
5         metric: ./runtime.txt
6         baseline: ./baseline/runtime.txt
7     #having absolute tolerance.
8     - check_two:
9         mode: scalar
10        metric: ./memory.txt
11        baseline: ./baseline/memory.txt
12        a: '1300'
13    #having relative tolerance.
14    - check_three:
15        mode: scalar
16        metric: ./result_count.txt
17        baseline: '15334'      #hardcoded baseline
18        r: '0.04'            #this is 4% tolerance
19    #having both (higher tolerance wins)
20    - check_four:
21        mode: scalar
22        metric: ./LVHEAP.txt
23        baseline: ./baseline/LVHEAP.txt
24        a: '2235'
25        r: '0.10'
26    #ignore_negative
27    - check_five:
28        mode: scalar
29        metric: [ ./output , . ]      #metric is array of directories, including the current directory (.)
30        baseline: ./baseline/*.txt   #baseline is a pattern
31        a: '32'
32        r: '0.10'
33        ignore_negative: '1'
34    #ignore_positive
35    - check_five:
36        mode: scalar
37        metric: ./output      #metric is array of directories, including the current directory (.)
38        baseline: ./baseline/*.txt      #baseline is a pattern
39        r: '0.03'
40        ignore_positive: '1'

```

Fig. 12. *scalar* mode examples.

### 5.3 operator mode

The *operator* mode is also used to compare two singular numerical values, usually stored in files that are generated by the *extract\_metrics* functions. This mode supports all python comparison operators ( $==$ ,  $\neq$ ,  $\leq$ ,  $\geq$ ,  $<$ ,  $>$ ). The comparison check passes only if the python operator check is true. Fig. 13 shows the syntax of the *operator* mode and its available parameters.

```

1 compare_metrics:
2     - check_name:
3         mode: operator
4         metric: <output file(s)/folder(s)>
5         baseline: <baseline file(s)> | <number>
6         check: <one of the standard python numerical checks>

```

Figure 13: *operator* mode syntax.

```

1 compare_metrics:
2     #first scenario
3     - check_one:
4         mode: operator
5         metric: ./memory.txt
6         baseline: ./baseline/memory.txt
7         check: '<='
8     #second scenario
9     - check_two:
10        mode: operator
11        metric: ./results_count.txt
12        baseline: '0'
13        check: '=='           #baseline is hardcoded to '0' here.
14                                #making sure that the results count is always equal to zero.

```

Fig. 14. *operator* mode examples.

## 5.4 numdiff mode

The *numdiff* mode is used to compare text files that contain numbers, using the *numdiff* tool [3]. It supports the comparison of these files with tolerance for the numbers within these files. It is usually used for the CSV file comparison, as it has many options for facilitating the CSV file comparison. Fig. 15 shows the parameters and the syntax of the *numdiff* mode.

```

1  compare_metrics:
2      - check_name:
3          mode: numdiff
4          metric: <output file(s)/folder(s)>
5          baseline: <baseline file(s)>
6          a: <absolute_tolerance>
7          r: <relative_tolerance>
8          separators: <separators_string>
9          columns: [<array_of_column_header_names>]
10         csv_separator: <csv_separators_string>
11         tolerance_if:
12             VARs:
13                 - [<array_of_key=value_variables>]
14                 - a: <absolute_tolerance>
15                 - r: <relative_tolerance>
16             VCOs:
17                 - [<array_of_VCOs>]
18                 - a: <absolute_tolerance>
19                 - r: <relative_tolerance>

```

Fig. 15. *numdiff* mode syntax.

Fig. 16 shows examples of the *numdiff* mode.

```

1  compare_metrics:
2      #having absolute tolerance.
3      - check_two:
4          mode: numdiff
5          metric: ./f2.csv
6          baseline: ./baseline/f2.csv
7          a: '13'
8      #having relative tolerance.
9      - check_three:
10         mode: numdiff
11         metric: ./f3.csv
12         baseline: ./baseline/f3.csv
13         r: '0.15' # this is 15% tolerance
14     #having both (higher tolerance wins)
15     - check_four:
16         mode: numdiff
17         metric: ./f4.csv
18         baseline: ./baseline/f4.csv
19         a: '22'
20         r: '0.10'
21     #second complex example (useful if you want higher tolerance if certain environment variable is set.)
22     - check_five:
23         mode: numdiff
24         metric: [ ./output , . ]
25         baseline: ./baseline/*.csv
26         a: '32'
27         r: '0.10'
28         separators: ' ;,\n'
29         tolerance_if:
30             VARs:
31                 - [USE_HIGH_TOL=yes,RELAX_TOL=1,MT_FLEX_RUN=1]
32                 - a: '40'
33                 - r: '0.15'

```

Fig. 16. *numdiff* mode examples.

## 5.5 image mode

The *image* mode is used to compare images. It supports comparison with tolerance, so if the images are close to each other within the defined tolerance, the comparison passes. The Frog-Logic engine is used in this mode [2]. Fig. 17 shows the parameters syntax of the *image* mode.

```

1  compare_metrics:
2    - check_name:
3      mode: image
4      metric: <output file(s)/folder(s)>
5      baseline: <baseline file(s)>
6      correlation-or-tolerance: <number> #you use either "correlation" or "tolerance", not both.

```

Fig. 17. *image* mode syntax.

Fig. 18 shows some examples of the *image* mode.

```

1  compare_metrics:
2    #image with no tolerance
3    - check_one:
4      metric: ./test.png
5      baseline: baseline/test.png
6      mode: image
7    #image with correlation (allowed similarity range)
8    - check_two:
9      metric: ./out1.png
10     baseline: baseline/out1_diff_small.png
11     mode: image
12     correlation: '99'
13   #image with tolerance (allowed difference range)
14   - check_three:
15     metric: ./out1.png
16     baseline: baseline/out1_diff_big.png
17     mode: image
18     tolerance: '90'

```

Fig. 18. *image* mode examples.

## 5.6 xor mode

The *xor* mode is used to compare two physical layout files of an integrated circuit (IC) design mask that are stored in different file formats [4]. It is common practice in Siemens to compare these types of files, as one of our primary software solutions is focused on IC design intellectual property (IP). This mode supports geometrical tolerance and conditional tolerance based on variables or current CPU architecture and operating system that might give slightly different output.

Fig. 19 shows the main syntax of the *xor* mode.

```

1  compare_metrics:
2    - check_name:
3      mode: xor
4      metric: <output file(s)/folder(s)>
5      baseline: <baseline file(s)>
6      layers: [<array_of_layers>]
7      bins: [<array_of_bins>]
8      exclude_bins: [<array_of_bins>]
9      exclude_bins_if:
10        VARs:
11          - [<array_of_key=value_variables>]
12          - bins: [<array_of_bins>]
13        VCos:
14          - [<array_of_VCos>]
15          - bins: [<array_of_bins>]

```

Fig. 19. *xor* mode syntax.

Fig. 20 shows some examples of the xor mode.

```

1 compare_metrics:
2     #xor-ing certain layers and bins.
3     - check_two:
4         mode: xor
5         metric: ./output/layout2.oas
6         baseline: ./baseline/layout2.oas
7         layers: [1,2,9]
8         bins: [1dbu,2dbu]
9         #excluding bins.
10    - check_three:
11        mode: xor
12        metric: ./output/layout3.oas
13        baseline: ./baseline/layout3.oas
14        layers: [1,2,9]
15        bins: [1,2,5,10]
16        exclude_bins: [1,2]
17        #excluding bins if (more bins will be excluded if certain environment variable is set)
18    - check_five:
19        mode: xor
20        metric: ./output/layout4.oas
21        baseline: ./baseline/layout4.oas
22        layers: [1,2,9]
23        bins: [1,2,5,10]
24        exclude_bins: [1]
25        exclude_bins_if:
26            VARS:
27                - [IGNORE_MORE_BINS=yes,RELAX_XOR=1]
28                - bins: [1,2,5]

```

Fig. 20. xor mode examples.

## 5.7 Custom mode

The Versus framework can be extended to accommodate any custom comparison engine and integrate it within the framework, enabling users to realize the framework's benefits while using the comparison engine that meets their functional needs. To implement this process, users employ the Versus environment variable `QA_VERSUS_CUSTOM_MODE`, as shown in Fig. 21. This variable must point to the custom comparison engine (or engines) so the framework can recognize and use them in the comparison checks.

```

1 #!/bin/sh
2
3 #define custom mode engine
4 export QA_VERSUS_CUSTOM_MODE="/path/to/custom_mode_engine/compare_database.sh"
5
6 #run Versus
7 /path/to/versus --yaml versus.yaml
8 status=$?
9
10 #exit with proper status
11 exit $status

```

Fig. 21. Custom mode usage.

Once the `QA_VERSUS_CUSTOM_MODE` variable is defined, users can use the custom mode in the YAML configuration file just as they do any of the pre-defined modes, getting the benefits of the framework right out of the box. In the example shown in Fig. 22, a DRC Results Database comparison engine (which compares Siemens EDA format databases used in the Calibre nmDRC tool) is used with the Versus custom mode.

```

1 compare metrics:
2   - MY_MODE:
3     mode: compare_database
4     metric: ./waived.rdb
5     baseline: baseline/waived2.rdb

```

Fig. 22. Custom mode configuration file example.

## 6 Versus output log and summary report

The output log of the Versus framework is designed to give users all the information they need about the comparisons without being distracted by lengthy or unnecessary data that would add time to the test failure analysis without adding any significant value.

Fig. 23 shows all the relevant sections in a typical Versus output log.

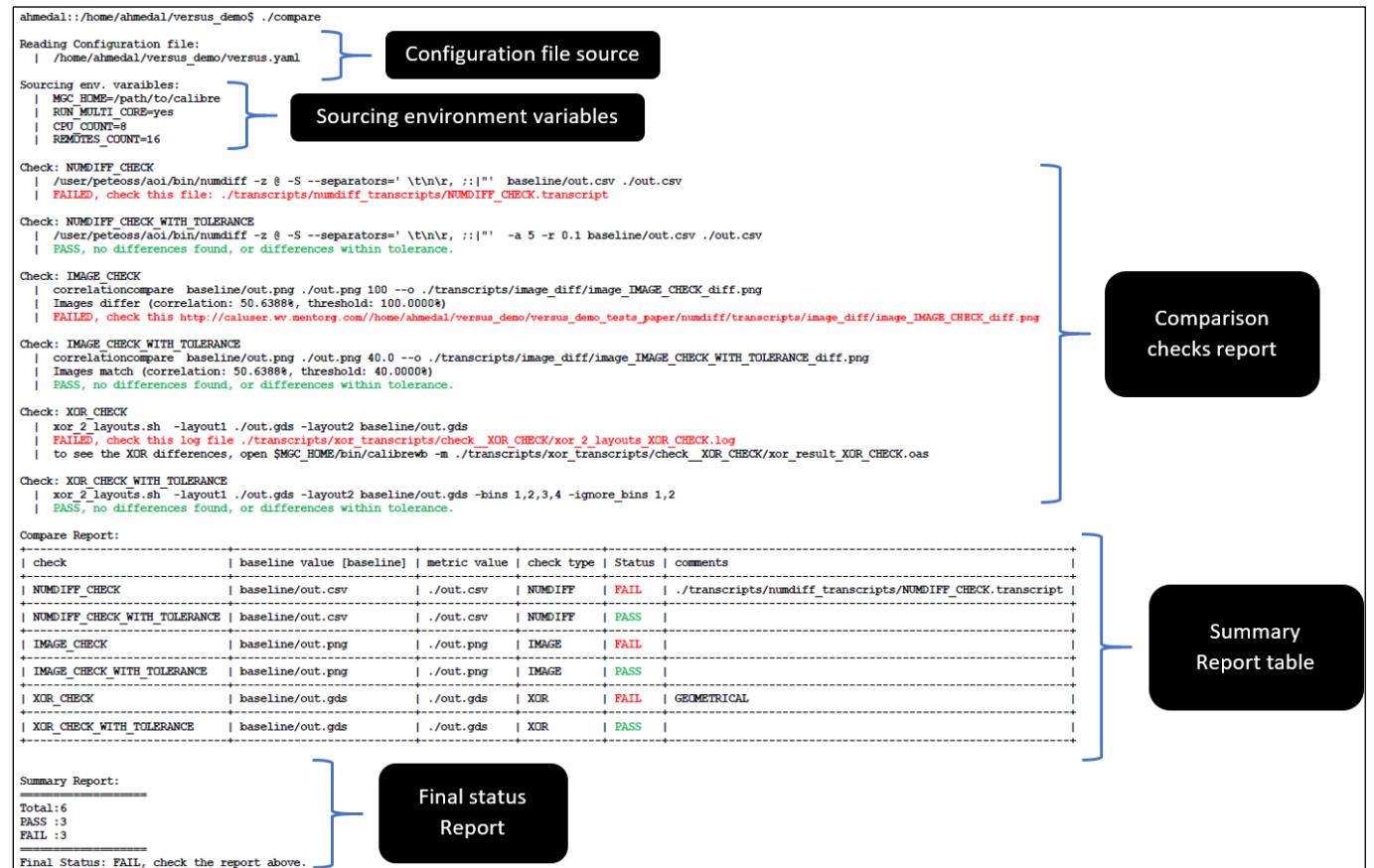


Fig. 23. Typical Versus output log, with each relevant section highlighted.

The following sections provide a closer look at each of the output log sections, providing more detail about the information presented, and the options designed to shorten failure analysis time.

### 6.1 Configuration file source and environment variables

Fig. 24 shows file sources and environmental variables used in the configuration file:

```

Reading Configuration file:
| /home/ahmedal/versus_demo/versus.yaml

Sourcing env. variables:
| MGC_HOME=/path/to/calibre
| RUN_MULTI_CORE=yes
| CPU_COUNT=8
| REMOTES_COUNT=16

```

Fig. 24. Configuration source and environment variables.

## 6.2 Comparison checks report

Fig. 25 shows the Versus comparison checks report. Each comparison check has its own mini check log:

- This log states the internal command used to make the comparison, and all the options and arguments used with this command, including the output and baseline files. It also states whether this check failed or passed, using color-coded lines to make reading the log easier and faster.
- In case of a check failure, the log directs users to the best option for checking the difference in the shortest time.

```

Check: NUMDIFF_CHECK
| /user/peteos/aoi/bin/numdiff -z @ -S --separators=' \t\n\r, ::|"' baseline/out.csv ./out.csv
| FAILED, check this file: ./transcripts/numdiff_transcripts/NUMDIFF_CHECK.transcript

Check: NUMDIFF_CHECK_WITH_TOLERANCE
| /user/peteos/aoi/bin/numdiff -z @ -S --separators=' \t\n\r, ::|"' -a 5 -r 0.1 baseline/out.csv ./out.csv
| PASS, no differences found, or differences within tolerance.

Check: IMAGE_CHECK
| correlationcompare baseline/out.png ./out.png 100 --o ./transcripts/image_diff/image_IMAGE_CHECK_diff.png
| Images differ (correlation: 50.6388%, threshold: 100.0000)
| FAILED, check this http://caluser.wv.mentorg.com//home/ahmedal/versus_demo/versus_demo_tests_paper/numdiff/transcripts/image_diff/image_IMAGE_CHECK_diff.png

Check: IMAGE_CHECK_WITH_TOLERANCE
| correlationcompare baseline/out.png ./out.png 40.0 --o ./transcripts/image_diff/image_IMAGE_CHECK_WITH_TOLERANCE_diff.png
| Images match (correlation: 50.6388%, threshold: 40.0000)
| PASS, no differences found, or differences within tolerance.

Check: XOR_CHECK
| xor_2_layouts.sh -layout1 ./out.gds -layout2 baseline/out.gds
| FAILED, check this log file ./transcripts/xor_transcripts/check_XOR_CHECK/xor_2_layouts_XOR_CHECK.log
| to see the XOR differences, open $MGC_HOME/bin/calibrewb -m ./transcripts/xor_transcripts/check_XOR_CHECK/xor_result_XOR_CHECK.oas

Check: XOR_CHECK_WITH_TOLERANCE
| xor_2_layouts.sh -layout1 ./out.gds -layout2 baseline/out.gds -bins 1,2,3,4 -ignore_bins 1,2
| PASS, no differences found, or differences within tolerance.

```

Fig. 25. Comparison checks report showing the check log for each comparison check.

## 6.3 Summary table report

The summary table report summarizes all the executed checks with their results in an organized table (Fig. 26). This report also includes the output and baseline file names, as well as comments (if possible) to direct users to the shortest path for checking the differences.

Compare Report:					
check	baseline value [baseline]	metric value	check type	Status	comments
NUMDIFF_CHECK	baseline/out.csv	./out.csv	NUMDIFF	FAIL	./transcripts/numdiff_transcripts/NUMDIFF_CHECK.transcript
NUMDIFF_CHECK_WITH_TOLERANCE	baseline/out.csv	./out.csv	NUMDIFF	PASS	
IMAGE_CHECK	baseline/out.png	./out.png	IMAGE	FAIL	
IMAGE_CHECK_WITH_TOLERANCE	baseline/out.png	./out.png	IMAGE	PASS	
XOR_CHECK	baseline/out.gds	./out.gds	XOR	FAIL	GEOMETRICAL
XOR_CHECK_WITH_TOLERANCE	baseline/out.gds	./out.gds	XOR	PASS	

Fig. 26: Summary table report.

## 6.4 Final pass/fail count report

The final pass/fail count is the last report that appears in the Versus output log (Fig. 27). It provides users with an overall view of how many checks failed and how many passed, so users can quickly determine how many checks passed without having to read the details of every single comparison check.

```

Summary Report:
=====
Total :6
PASS :3
FAIL :3
=====
Final Status: FAIL, check the report above.

```

Fig. 27. Final pass/fail counts report.

## 7 Versus configuration generator UI

Writing YAML code that contains multiple options can be challenging and time-consuming, so a web-based graphical user interface (GUI) tool was developed to create and edit Versus YAML configuration files (Fig. 28). This tool makes it easy to write Versus configuration files, eliminating the need to memorize any of the configuration syntax for all of the pre-defined modes while also reminding users of all of the available options the Versus framework offers, simplifying and speeding up configuration file creation for both new and experienced users.

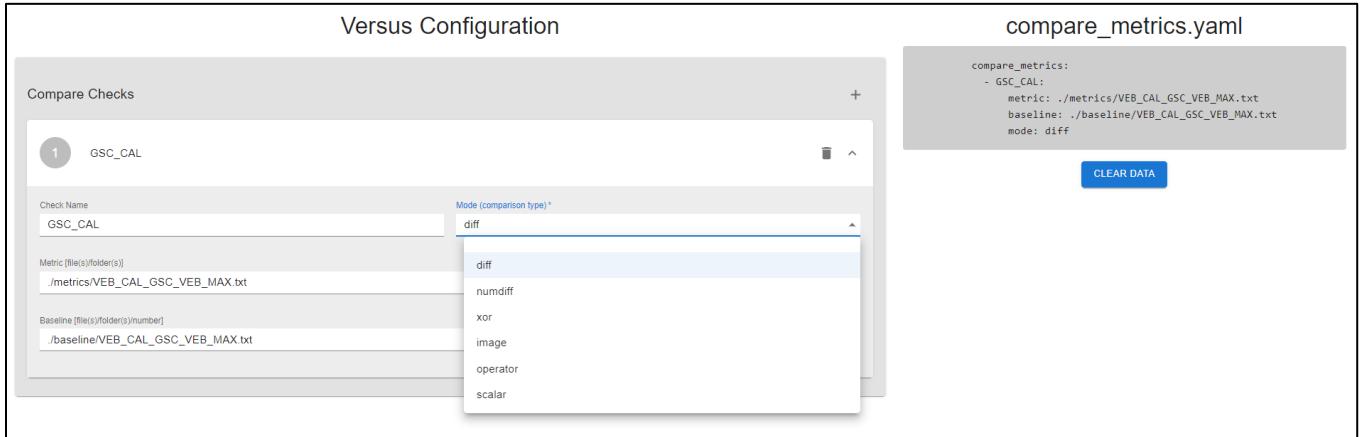


Fig. 28. Versus UI web-based tool for generating/editing configuration files.

## 8 Conclusion

The Versus framework is primarily designed for flexibility, maintainability, and control. It defines a standard for files comparison within automated regression suites, enabling QA engineers to use the best state-of-art tools for files comparison, while eliminating the need to write their own code every time a new comparison type is needed.

## Acknowledgements

The authors would like to extend their gratitude to Siemens Calibre QA teams who assisted in testing the framework in-house, and provided insightful and helpful ideas and feedback to help the framework reach its maximum potential. We would also like to thank Shelly Stalnaker for editorial assistance in the preparation of this manuscript.

## References

- M. Kerrick, "The Linux Programming Interface," Oct. 2010. <https://man7.org/linux/man-pages/man1/diff.1.html>
- S. Vrkacevic, "Using command line tools for image comparisons," froglogic, Oct 10, 2017. <https://www.froglogic.com/blog/command-line-tools-for-image-comparisons/>
- I. Primi, "Numdiff," Feb. 25, 2017. <https://www.nongnu.org/numdiff/>
- "Open Artwork System Interchange Standard," Layout Editor. <https://www.layouteditor.org/layout/file-formats/oasis>
- "GDSII," Layout Editor. <https://www.layouteditor.org/layout/file-formats/gdsii>

# Building a Smart High Quality Software Pipeline

**Richard Robinson**

Richard.Robinson@bongolearn.com

## Abstract

How can we ensure quality with quick releases full of great features and changes in today's modern software world?

As software development, integration, and delivery processes continue to speed up with increased demand and growth throughout the industry there is a growing need to evolve our quality assurance approaches to ensure solid and quality code arrives in the hands of every user along with a constant flow of new features and changes. Quick releases, stringent SLAs for uptime and service, massive and quick scale up/down needs, accessibility, localization, and a huge array of user devices are some of the challenges that can cause significant issues internally and for our customers without the right changes.

We are constantly evolving our own approach and have learned some good lessons along the way on how to build in quality at each phase with automation and integration of tools and processes throughout our CI/CD pipeline to significantly decrease our hotfixes and interruptions in production while increasing our delivery of features to our customer base. We have gone from significant down time each release in each supported region worldwide to zero downtime releases with better automated and manual verification tests along the way improving efficiencies of our engineers internally and our customers in production.

We're still learning, but in this paper, I share some of the lessons from our journey that could help others with practical strategies and approaches to enable quality software in a modern CI/CD pipeline world.

## Biography

*Richard Robinson currently leads the DevOps, QA, Infrastructure, and IT engineering groups supporting all operations for BongoLearn, Inc. We at Bongo create video training, learning, and assessment workflows and technologies embedded in many of the prominent Learning Management Systems in the EdTech industry. I have a background in QA, automation, and managing QA and systems engineering teams for large and small companies over the last 22 years. I am passionate about quality and ensuring customers and companies can effectively use technology to solve their problems without needing to worry about issues and problems that plague our world and get in the way of real work.*

Copyright Richard Robinson 2022

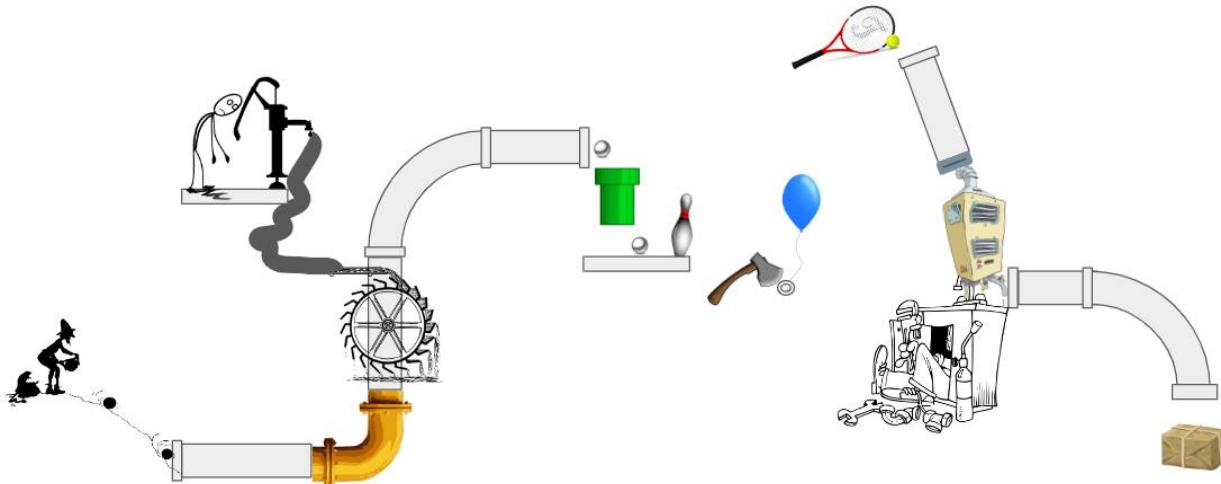
## 1 Introduction

Much of the modern software engineering world is transforming from larger releases to smaller, more frequent releases. So how do we support integration and release pipelines that are tailored to these types of releases? We might start with just a series of different phases of the software development lifecycle loosely connected like a Rube Goldberg machine sometimes getting stuck and needing a nudge to go to the next step. Then with the right planning and perseverance this can evolve into a robust, dependable, and fully automated smart pipeline enabling continuous integration of new/modified code and quality delivery of that to production to delight end users.

The answer I think lies in a lot of the contributing aspects of software development partnered with operations and modern DevOps principles applied in a quality fashion. It is also constantly evolving and will look different for different businesses and domains. Critical software that could kill someone might use different approaches or levels than less critical software, but I believe there is a huge overlap in the principles, approaches, and tools that can be utilized. This is similar to differences between 99.999% uptime and 99.9% uptime for different software for different purposes seeking "...to balance the risk of unavailability with the goals of rapid innovation and efficient service operations, so that users' overall happiness—with features, service, and performance—is optimized." The risk tolerance of failures compared to engineering cost is continually considered and can even be measured like Google's SRE (Site Reliability Engineering) Error Budgets.

So much depends on what kind of pipeline we are building and all the contributing factors to what goes into that pipeline. From the right tools for good development environments for individual software engineers to deployment tools and processes ensuring solid delivery and execution of code in a variety of real-world environments. In the middle we find things like a good code review process and culture, code analysis tools, maintainable automated test suites, Infrastructure as Code (IaC), and other positively contributing factors.

I will cover some of these areas and what we have found most helpful for evolving from a rusty old pipe, or sections of pipe, loosely held together to get software into the hands of our end users to a pipe that while not finished (are CI/CD pipelines ever finished?) has dramatically improved our ability to deliver quality software applications to our customers.



## 2 Enabling Zero Downtime Releases

### Four-hour downtime maintenance windows to zero downtime releases

A few years ago, it was typical for our customers to experience three to four hours of downtime showing a construction page or an unhelpful error for each maintenance window. With six production environments deployed throughout the world that was becoming pretty noticeable and painful to our end customers, partners, and the engineering team manually deploying all of those environments (4 hours x 6 environments = 24 hours for each release). Those three to four hours were often set from midnight to 3 or 4 am to minimize impact on end users which further negatively impacted the engineering team involved in each release. Increasing the release frequency with such a situation was clearly untenable and human errors were common.

To move from this situation took a few key initial first steps:

- First, we needed to understand all the pieces and each step involved in the deployment.
- Second, we created a checklist to ensure that the manual and brittle process could be performed without missing key steps to keep supporting the business by releasing new features and fixes until automation could be built.
- Third, we needed to investigate and employ techniques and tools so each step could be performed without bringing the system down.
- Fourth (although really started after the first step), we needed representative internal environments to test the whole process and automate against.

The first and second steps go nicely together as we documented which steps were being done and which needed to happen in what order reviewing and discussing as we went. This laid out almost a blueprint of sorts for all the following steps and the evolution as we prioritized which steps to optimize and automate later. The checklists we then used to execute the deployment of a release didn't actually need every detail and specific piece, but rather reminders of the key steps, correct sequencing, and overall flow. The pipeline started to take shape and we even saw the satisfaction as we turned the checklist items that were automated a different color while retaining them in the checklist. Atul Gawande in his book "The Checklist Manifesto: How to Get Things Right" reinforces the point that in today's modern world of complexity whether in surgery (he personally is a surgeon), building modern skyscrapers, flying an airplane, or building and deploying software the volume and complexity our human brains are dealing with needs a level of assistance to consistently get it right. "...the volume and complexity of what we know has exceeded our individual ability to deliver its benefits correctly, safely, or reliably."

The third step we took was to investigate all of the approaches for delivering software without any impact to end users or need for a customer visible maintenance window. This took time to shift to use approaches like blue/green or black/red approaches. Basically, most of the approaches deal with delivering a set of software alongside the current running software and flipping it over to run with the newer version either all at once or incrementally as multiple nodes within a cluster of nodes are updated and users start hitting the new code. This can often be partnered with feature flags so the new code can be in place with the same behavior as the old code until a flag is triggered to activate the new code. These and various other approaches were key to shifting the mindset so there was no impact to an end user until they just clicked on the next page/button/menu or the next API call and started using new code. As is probably the case with most software projects we had to take each part and incrementally make changes to accommodate that and adopt new software practices to maintain backward compatibility and ensure each new release could also be deployed without impact to end users. Significant testing and verification were performed on the application as a whole and on the specific areas of change to ensure changes could be rolled out at any time. Also, I'll admit that to begin with until we were more confident in our processes, we still deployed in off hours in case something was missed to minimize potential impact and now we regularly release any time and even at usage peaks in the middle of the day.

The fourth and either final or ongoing step after identifying what steps need to be taken is to ensure that there are representative internal environments that mimic production so well that the whole process can be created and verified outside of production. This will likely require several iterations to work out the differences between internal environments and actual production environments and ongoing work to keep the environments 'in sync'. The industry typically has the concept of staging environments where new software can be staged before deploying to production or switching between production and staging and back again as blue goes to green and then back again to blue. In whatever way it is implemented the important principle I think is to make sure there are internal environments that are close enough to production that steps, checklists, and automation used against those environments will perform the same as in production. This should be considered all the way back to development environments for engineers as much as possible.

A key item to capture and include through all these steps is adequate testing and verification steps whether performed manually to begin with or in an automated fashion. These are key to ensuring quality as this process evolves and in the end result. Wherever an organization is in this evolution I believe implementing even some of these steps will start to yield the desired results and the return on investment of these activities can fuel the engineering necessary to continue the evolutionary process. While it would be nice to pause everything and create the whole process and then make it live I believe most organizations are in the position where incremental progress evolving something that already exists is what reasonably can be done. Of course, if a brand-new process is being constructed for a brand-new application, I think these same steps can still be used enabling an orderly design and creation. Like Test Driven Development (TDD) principles where tests are created first which all fail until the code is written that enables the tests to pass these steps should provide a nice framework for even a green field CI/CD pipeline for zero downtime deployments.

### **3 Building Confidence in Test Suites**

#### **How to build confidence in your manual and automated regression suites**

The quality of the testing, validation, and verification capability of a software development organization is key to building a pipeline you will be able to use and rely on and especially as was just mentioned with zero downtime deployments at any time of day.

Some initial pieces to start with:

- Internal environments and a pure CI (Continuous Integration) target where each check-in of code is put into an environment with other code and tests are run.
- Some of the most basic tests just to build out the process and build that "new muscle". Even a smoke test run against a build of some new code put into even the most basic integration environment can be a great foundation to start with.
- Manual test suites alongside any sort of automated unit tests, functional tests, regression suites, or other more specialized tests so there is always a clear view of what is being covered and what could be the next priority to build up a solid automated suite.
- Using test suites regularly for internal environments and then injected into the correct places in the manual, partially automated, or fully automated deployment pipeline.

As mentioned earlier, using a checklist approach of specific items to cover in the correct sequence and configuration helps ensure quality in the short-term and a map of what needs to be created to take humans out of the execution phase. As we take humans, and our human engineering hours, out of the execution phase and more into the design and architecting of test approaches, deployment considerations, etc. quality improves, and the overall engineering process can accelerate without compromising quality. As the checklists and results are reviewed the checklist evolves to include the items that are missed in previous iterations and there is a clearer map of the items that can and should be

automated to put the pieces of the pipeline together. One powerful interim result here is that even without all the pieces automated together benefits are still realized as each piece itself is solidified and improved.

Risk-based testing approaches can be used to ensure the evolution of the pipeline is providing Return on Investment (ROI) as we go, and we are hitting on the most critical items to the business or the current user base. Whenever tests are considered for a given application or feature it can be helpful to go through a list of potential dimensions or considerations to determine which are more, or less, significant, or unique to that particular application, use case, or domain. Considering all the various dimensions up front enables a team to consciously and more holistically prioritize or deprioritize and not just prioritize what is currently in the list or top of mind and possibly miss some area that might be even more significant.

Of course, areas of functionality for the product should also be considered and where changes are being made in a particular release that might impact one area more than another area and warrant additional testing. This is especially important when that testing in the short-term requires manual test effort. Knowing whether to do a light touch of an area or a deeper regression test is important to budgeting QA time and enable the appropriate build out of additional automated tests and complex testing scenarios/configs.

Some potential areas to consider:

High-level Area or Consideration	Details/Description
Accessibility	Does this system properly support accessibility with low vision and screenreader support? Also closed captioning capabilities
Performance	Performance tests to ensure that the system's response times and throughput meet the user expectations and meet specified performance criteria or goals.
Scale/Sizing	Horizontal and Vertical scaling considerations
Stress	Pushing the boundaries of performance limits or even just limits of specific fields/types (e.g., numeric limits for integer fields or overflowing string sizes especially when storing in DB tables with columns of certain types)
Security & Privacy	Security tests to determine how secure the system is and if specific security requirements have been met. Could include GDPR, FERPA, HIPAA, data residency restrictions, etc.
Upgrade and Migration	Data preserved after upgrades and properly migrated to any new formats
Stability/Reliability	Tests performed to run the product in a customer-like environment over a period of time to verify that the system remains stable and there are no significant memory/handle/thread leaks or degradation to the system over time.
Usability	Is the system usable without intensive training or use of workarounds? Is it suitable for the target user community? Ease of use and standard UI behavior is good to consider here as well including use of phones, tablets, laptops, etc.
Supportability and Maintainability	Are there logs, debug levels, design docs, troubleshooting guides, API specs, and other things in place so that the product can be supported
Etc.	

I have found that reviewing a list of these types of considerations and discussing with others on the team enables reasonable tradeoffs to be made and appropriate plans to automate, adjust the pipeline, and otherwise improve the process without compromising quality or generating unacceptable risk.

In order to keep momentum during the evolution of an automated test suite and not lose ground it is important to include as part of the standard done criteria the automated deployment capability as well as automated unit, integration, end to end, and other tests. There are a lot of forces that can work against that (time pressure, additional engineering cost, etc.). Practical tradeoffs/compromises sometimes needed to keep overall solid strategy moving forward balancing business and market needs and high-quality engineering needs. However, even if some short-term tradeoffs need to be made the overall strategy can be protected as current automated verification capability is protected as a key priority along with properly working features.

For example: A new feature X is needed to be delivered as quickly as possible and impacts current functionality/behavior. At a minimum the current automated suite needs to be preserved by making it compatible with changes for the new feature even if full testing around new functionality needs to be manually finished first and then a follow-on sprint after release remaining automated tests fleshed out to ensure ongoing automated coverage. Designing a test suite for maintainability and to expect changes is of course a significant advantage for the current and continuing evolving application and pipeline.

One strategy that can assist with these short-term needs and incremental evolution is to build in processes to allow the pipeline to support 'augmentation' as well as 'fully automated' pieces. We have found it helpful to have automated scripts which sometimes only partially automate complex steps or even have pauses to enable a human to interject a manual step into the process so the overall process can flow with the nudges to the Rube Goldberg machine necessary in the short-term knowing that those will be eliminated over time with fully automated and robust processes in the future. There are some good approaches in the industry to even have scripts or automated pieces that are just placeholders for a manual step until the placeholder can be 'fleshed out' with actual automation that performs the task. Of course, with all of these processes good software engineering principles should be adhered to so code is re-used properly, properly checked in, managed, and reviewed by others on the team and regularly tested as a part of the overall solution.

## 4 Components of a Solid Pipeline

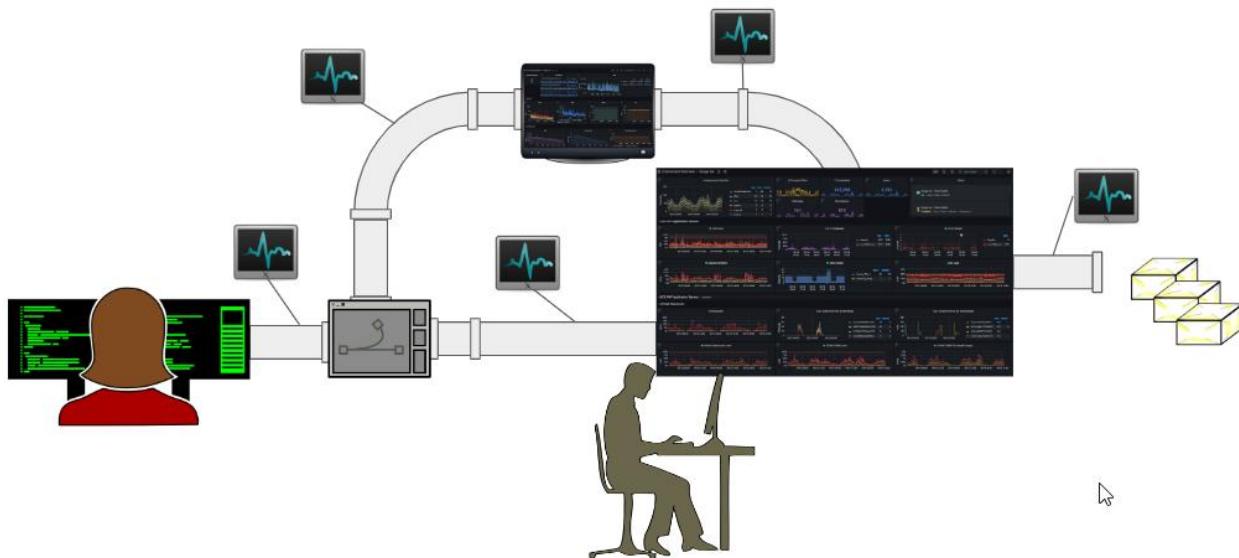
### **Important layers and aspects of a solid CI/CD pipeline**

Some of the important layers that need to be formed around a CI/CD pipeline to enable the pipe to be solid now and in the future are:

- Flexibility - need to be able to fit in a variety of tools and processes. Even manual processes that might need to be interjected in.
- Code scanning - static code analysis for quality, security scans, licensing, etc.
- Test execution – unit tests, functional tests, integration tests, more end-to-end tests, and even specialty tests like perf/load/scale or accessibility tests can be added in
- Monitoring of behavior as software goes through the pipeline and in production. If there is no visibility into the pipe until it comes out the other end, then that just adds risk and surprises into a system that needs to be robust and deterministic.
- Tools like ELK (ElasticSearch, Logstash, and Kibana), Sentry, or others to enable thorough monitoring while things are in internal environments, traveling through the pipeline, and in production

- Incorporating automated functional tests to explore load, scale, and performance as well as to enhance monitoring with synthetic usage (good for regular heartbeat monitoring, exact/expected results in production alongside real-world usage)

As instrumentation is added into a pipeline and into production environments then the overall process becomes much more solid and reliable, and everyone benefits. It will also be easier to spot the bottlenecks or where issues are most prevalent for proper prioritization of next steps.



## 5 Summary/Conclusion

In summary I would like to review a few of the key principles that can be applied regardless of business, domain, or toolset to help with your building or enhancing of a smart, high quality, software pipeline.

- Checklists - ensure immediate quality and 'fill in the gaps (w/ automation, tools, expertise, whatever) and provide a map of what is needed to work on next
- Showing ROI and pipeline improvements along the way
- Risk-based testing approaches and how to build them into a pipeline
- Various aspects of a pipeline that can be overlooked as we might focus too much on just putting code into it and it plopping out the other side into production.

The success of an engineering organization and a business reliant on good software applications rises or falls to a significant degree with how solid the quality software pipeline is and how it is used. If great ideas and awesome features are created without the necessary quality processes to ensure they work, are released in a timely and reliable fashion, and continue to work well for the end users then users will go elsewhere and either immediately or over time every other aspect of the business will suffer. I hope that some ideas presented here will help you and your business ensure quality with quick releases full of great features and changes in today's modern software world.

## References

Book:

Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy. 2016 *Google Site Reliability Engineering*. O'Reilly Media, Incorporated.

Atul Gawande. 2011. *The Checklist Manifesto*. New York, NY: Metropolitan Books

# Active Feedback Loops in Software Testing

**Robert Sabourin**

robsab@gmail.com

**Chris Blain**

cblain@gmail.com

## Abstract

Rob Sabourin and Chris Blain share many decades of experience implementing software engineering concepts in a wide variety of business, technical, organizational, and cultural contexts.

While testing we may miss opportunities to act. Can testing become a call to action? Can testers hold a leading role driving decisions and getting things done?

Testing can be focused on taking action.

Feedback loops can be established between testers and programmers, test leads, other testers, subject matter experts, scrum masters, product owners, customers, and users.

Feedback loops apply to traditional or “agile” methods and are particularly valuable in regulated projects. These loops can be more useful and motivating to teams versus traditional phase gates that teams often dread. If the feedback loop is seen as a way to improve and increase the quality of testing (and the entire project as a result), teams will be more engaged.

Feedback from the testers can guide programmers to improve code quality and adjust programming methods especially when a cluster of related bugs is identified. Feedback from programmers to testers can lead to revising the scope and depth of testing and better understanding technical risk. Feedback from testers to product managers can help the team understand what can really get “done-done” in a certain time period.

Feedback from product owners and customers can help testers understand business context factors which influence testing and test reporting.

Feedback from testers to product owners and customers can build confidence in product quality influencing go/no go deployment decisions.

Feedback to testers from users can offer a much-needed dose of realism to the scenarios and data exercised in a session. As well as the real usage of various features.

Testers can suggest improved tools and test design techniques.

All stakeholders can guide testers in how to make their findings more relevant.

## Biography

*Robert Sabourin has more than forty years of management experience, leading teams of software development professionals. A well-respected member of the software engineering community, Robert has managed, trained, mentored, and coached thousands of top professionals in the field. He frequently speaks at conferences and writes on software engineering, SQA, testing, management, and internationalization. The author of "I am a Bug!" the popular software testing children's book, Robert is an adjunct professor of Software Engineering at McGill University. Robert is the principal consultant (&president/janitor) of AmiBug.Com, Inc.*

*Chris Blain has more than twenty years of experience working in software development on projects ranging from embedded systems to SaaS applications. He is a former board member of the Pacific Northwest Software Quality Conference and a semi-regular conference speaker. His main interests are testing, distributed systems, programming languages, and debugging.*

© 2022 Robert Sabourin and Chris Blain

# 1 Introduction

This paper will describe several different approaches used by the authors' to implement active feedback related to testing activities in software engineering projects. Approaches have been successfully applied to a wide variety of process models including blends of agile, traditional and hybrid life cycles.

In this paper Robert Sabourin and Chris Blain will share several aspects of Active Feedback through examples taken from real software development projects.

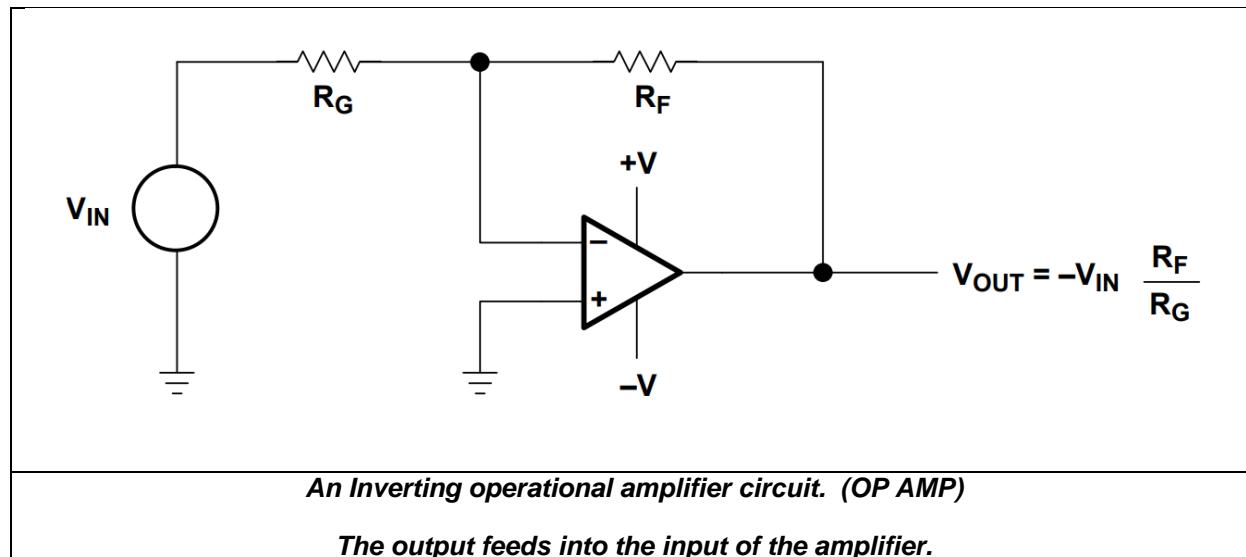
The lessons learned may be interesting to organizations and development professionals seeking to identify methods to improve their software engineering process and practice as a natural extension of the technical work being accomplished related to product requirements, designs, process and software quality assurance. In addition, ideas for improved training engagement with technical teams are contained as well.

## 2 Feedback Loops from Analogy to Reality

### 2.1 Feedback in Electronics

Feedback is a concept used in engineering to implement adaptive systems. The behavior of the system is designed to adapt based on system inputs combined with the system outputs. A classic example is feedback being used to implement amplifier circuits.

It is possible to design feedback circuits which perform many different operations on the input signals, including addition, difference, filtering, exponential and logarithmic calculations and combinations of these and many other operations.



### 2.2 Feedback in General Systems Thinking

Feedback is a concept used in general systems thinking when modeling a process, product, or production approach.

A feedback loop is a relationship between actions identified in general systems thinking. General systems thinking focuses on the way that a system's constituent parts interrelate.

An example might be to the following cycle of four activities:

1. Select an action
2. Perform the action
3. Wait for a response
4. Observe the response

As the system continues to operate the action chosen depends on the observed response. This is a feedback loop. Observation of teams has shown it is common for teams to only perform a subset of these activities. Some might even stop after the second step.

Two types of feedback are observed with general systems thinking: reinforcement and balancing.

Reinforcement is feedback that leads to an increase, or encouragement, in some element of the system.

Balancing is feedback that leads to maintain equilibrium, or diminishment, in some element of the system.

### **2.3 Feedback in Organizational Behavior**

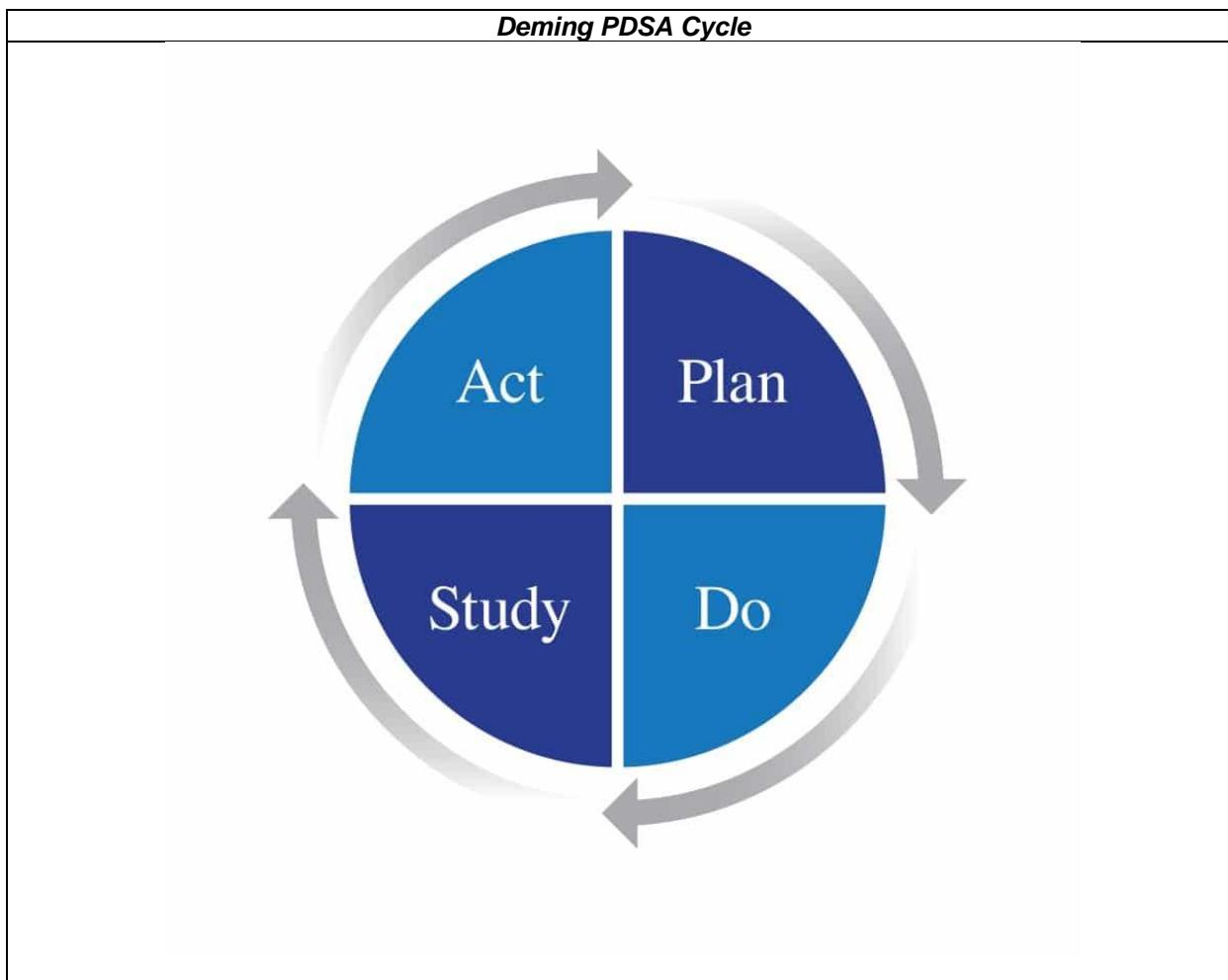
Feedback is a concept used in managing organizational behavior to provide important guidance to organizations, teams, or individuals.

Ken Blanchard describes feedback as the “Breakfast of Champions”. In his many books on the topic of organizational behavior Blanchard considers feedback as a critical success factor. When directing someone whose task is to accomplish a well-defined goal, Blanchard suggests that timely feedback should be provided to reinforce or redirect behavior. Delivering feedback is a leadership skill found in successful organizations fostering high performing teams. Feedback is part of the communication found in successful collaborative teams.

### **2.4 Feedback in Process Improvement**

Edward Deming spent his career helping organizations implement quality process control systems.

Deming used a four-step feedback loop model which he called a “Shewhart” (named for his mentor), “Plan Do Check Act” or a “Plan Do Study Act” cycle.



Using PDSA practitioners review the results of process change to determine if it is effective. Process changes are identified to encourage improvements and to discourage degradation.

- **Plan**, prepare for change
- **Do**, execute the plan
- **Study**, study the results
- **Act**, adapt based on your findings

There are many practices which the authors have integrated into software engineering lifecycles based on the Deming Cycle feedback loop. All these practices provide the team with frequent feedback which guides quality process improvement. The team identifies behaviors to instantiate, to encourage or increase, behaviors to discourage or decrease or eliminate.

- Incremental Development
- Product Testing
- Peer Reviews
- Team Retrospectives
- Root Cause Analysis
- User Experience Surveys
- Analytics

## 2.5 Importance of Soft Skills in Feedback Loops

It seems quite easy to describe feedback from a system engineering perspective. Feedback can sound objective and clinical. The authors experience indicates that accepting or delivering feedback involves tact, diplomacy, and delicate manipulations of soft people-oriented skills. As I wrote in the book “I am a Bug!” – testing is indeed all about people and the occasional bug.

A common feedback related risk experienced by testers is in bug reports being perceived as critical of the programmer.

The authors have seen many cases in which feedback is provided to practitioners based on systematic and formal inspection models, only to find that the recipient has experienced an emotional response. The authors thought they did a great job but were shocked to discover dozens of bugs, risk of other criticisms in their good works.

Blanchard recommends practicing situational leadership when providing task related feedback to team members. The leadership style should be selected based on factors such as the level of professional experience and takes experience of the team member into account. Someone new at a task with significant experience may benefit from a more coaching style of feedback whereas a new team member working on a business-critical task may benefit from more directive feedback.

Although out of the scope of this article it is important in any feedback loops implemented as part of a software engineering process consider the soft skills and emotional responses. For bugs you might suggest that bug reports describe the behavior of the software rather than the behavior of the programmer.

The authors have found the following references to offer excellent discussions about how to develop and use soft skills in effective use of feedback loops.

- DeMarco, Tom, and Timothy R. Lister. 1987. *Peopleware: productive projects and teams*. New York, NY: Dorset House Pub. Co.
- Kaner, Cem and Rebecca Fiedler. 2015. *Bug Advocacy: A BBST Workbook*. U.S.A.: Context Driven Press.
- Gilb, Tom, Dorothy Graham, and Susannah Finzi. 1993. *Software Inspection*. Wokingham, England: Addison-Wesley.
- Blanchard, K. A. J. 2022. *One Minute Manager* (New Thorsons Classics edition). Harper.
- Seashore, Charles N., Seashore, Edith W., and Weinberg, Gerald M., 2013. *The Art of Giving and Receiving Feedback*, Smashwords, U.S.A.

## 3 Task Analysis of Ticket Driven Software Development

### 3.1 Critical Incident Method

The authors have used formal task analysis methods to study how software testing is implemented at different organizations, teams. The authors have used task analysis in software engineering consulting and in software engineering practices.

The critical incident method of task analysis involves studying engineering practice by interviewing practitioners about their formative experiences. Delegates are interviewed to share experiences of typical, successful, and failed practices.

Since February of 2020 Robert Sabourin performed task analysis about software testing practices at many organizations implementing variations of “agile” development with Scrum-like or Scrum-inspired development frameworks.

The organizations are in many different business domains including Enterprise Management Systems, Transactional E-commerce Systems, Medical Information Management, Financial Services, and Insurance Industries.

The common thread between these organizations was that software engineering activities were managed and tracked through ticket-oriented workflow management systems.

## **3.2 Ticket Oriented Testing**

### **3.2.1 Workflow Models**

In the ticket-oriented workflow systems team members invariably received work assignments as tickets which were somehow assigned to them by other team members.

Team members would receive all their work instructions through the ticket management system. If clarification was required a chain of questions were attached to the ticket. In more difficult situations the questions were exiled to enterprise chat applications or email. The problem with this is that it is too easy for decisions to be lost.

Many tools from various vendors were observed to be used in the task analysis.

There seem to be two different approaches to completing testing work.

### **3.2.2 Testing as a Ticket Subtask**

Programming, testing, data, documentation, and other task types are identified and attached to a parent ticket. The parent ticket represents a requirement or field reported bug requiring resolution. A testing task would be assigned to a testing team member.

### **3.2.3 Testing as a Ticket State**

A ticket representing a requirement is set up and has many possible states. When some type of work activities is completed the state of the ticket is changed. For example, a ticket could have five states:

1. Ready for programming
2. Programming in progress
3. Ready for testing
4. Testing in progress
5. Done

When the state becomes "Ready for testing" the ticket is assigned to a member of the testing team. Ironically, in the experience of the authors, the "Ready for testing" state is usually the latest time to start testing activities related to the requirement. All of the benefits of focusing and learning the skills of incremental development are lost as a result.

### **3.2.4 Task Analysis Results**

Task analysis led to several common work dynamics which the authors now use to characterize risks of using ticket-oriented workflows for managing testing activities as part of a software development lifecycle model.

Based on a survey of over 100 task analysis assessments, the following points have been frequently observed by the authors.

1. Testers have little knowledge of the relationship between their assigned ticket and the rest of development project. Often the testing work is just a stream of tickets to close rather than a prioritized list that focuses the team on learning or project risks.

2. Testers first see the ticket when it is assigned to them. They are not involved in preparing or refining the ticket.
3. Testers focus on closing ticket metrics including time and frequency. Such measures have been observed in outsourced testing projects managing work assignments via tickets. These metrics have been observed in several financial service organizations, insurance companies and contact management services primarily in distributed teams.
4. Reopening a closed ticket is looked down upon.
5. Long chains of closed form questions are used to clarify work assignment.
6. Scope and depth of testing is limited to the specifics of ticket. This tends to severely limit testing beyond the happy path. It also conditions testers away from more creative testing ideas.
7. Lack of consideration of actual technical software engineering changes implemented.
8. Lack of consideration of usage of the system to solve end user problems.
9. Lack of consideration of relevant context factors.
10. No evidence of knowledge being collected or managed other than long clarifying question chains.
11. No evidence of technical collaboration between team members is observed.
12. Test findings are used to close tickets. Test findings are not used to improve the software development process model.
13. Reduced understanding of the larger architectural frame of the project. Everything is seen through a keyhole so the whole is a mystery.

There is little evidence of the use of feedback loops in ticket driven software engineering workflows. The only feedback loop observed were “agile” retrospective rituals mentioned by some delegates. Maybe some metrics around team performance are captured (cycle time).

## 4 Feedback Loops with Product Owners

### 4.1 Product Owner Role

On a software engineering team, the product owner role is that of a customer advocate. Product owners are responsible for understanding and communicating software requirements to the other team members.

Product owners are responsible for prioritizing requirements based on business needs and in collaboration with the development team.

Product owners are responsible for establishing what quality means to the team and ensure that this relates to business goals and customer expectations.

Product owners have budget responsibility (most often by controlling for project duration) for the software development project.

A feedback loop between a tester and the product owner can both guide testing and drive decision making. This is one of the most active feedback loops as feedback from testers in the best case can help drive ready for release decisions if the product owner is tuned to this information. In the worst-case testers are frustrated as product owners release despite the testers concerns. It is important for the product owner to make clear the business drivers that often drive these otherwise puzzling decisions.

### 4.2 Context Factors

Context factors are variables which influence how decisions are made in a project. Testers can scope testing activities based on knowledge of context factors. The scope of testing describes which elements of a testable object should be exercised and which elements of a testable object can be excluded.

Active context listening is a feedback loop between the product ownership role and a testing role.

Here are some example steps in the active context listening feedback loop.

1. Product owner makes a prioritization decision
2. Tester identifies, and isolates the context factors driving the prioritization decision
  - a. Business
  - b. Technical
  - c. Organizational
  - d. Cultural
  - e. Quality
3. Tester categories identified context factors
  - a. Are there any new context factors?
  - b. Are the context factors already known?
  - c. Did the impact of a known context factor change?
4. Tester updates their scoping model based on context factors
5. Any in process testing is adapted based on changes to test scoping model.

The tester by monitoring prioritization decisions of the product owner role can adapt, support, or redirect test scoping.

Note that in this context listening approach all team members including the product owner have visibility into how testers define the scope of a testing activity.

### **4.3 Product Requirement Feedback loops**

Testers have opportunities to influence the product requirements development teams are called upon to implement. Requirement management approaches vary dramatically across different software development lifecycle models, technologies, and domains. Some lifecycle models have dedicated rituals enabling teams to refine requirement definitions. Some lifecycle models rely on formal reviews or inspections to improve or adapt the statement of product requirements.

The basic feedback loop between testing roles and product owner roles follows an iterative pattern.

1. Product owner updates requirement statement or product description
2. Team members review requirement statement
3. Team comes to consensus on requirement critique
  - a. Relate requirement to other aspects of product
  - b. Disambiguate statement of requirement
  - c. Identify missing attributes of requirement
  - d. Identify unnecessary elements of requirement
  - e. Improve requirement acceptance criteria
  - f. Estimate work required to implement requirement

## **5 Feedback Loops with Programmers**

Some testers frequently interact with programmers. Feedback loops can be established between programmers and testers enabling faster deliberately focused testing and minimizing development rework.

### **5.1 Design and Planning**

During planning, programmers establish a technical direction to solve the problem of implementing a product requirement, bug fix or change request.

Although often the programming approach is governed by technical intricacies indicated by tools, frameworks, platforms, operational concerns, and software architecture, there is still an opportunity for a direct feedback loop between testers and programmers related to design.

Consider the following design approach used by a customer of Robert Sabourin during the teams Scrum planning session.

1. Programmer identifies two or more design alternatives
2. Programmer models solution indicating scope and nature of technical changes
3. Programmers identify benefits associated with each alternative
4. Testers identify potential risks associated with each alternative
5. Action take
  - a. Suitable alternative selected
  - b. New alternative added
  - c. Existing alternative adapted

Even though testers may not be skilled in software design and architecture (though the exceptions are less rare than you might think), they are able to participate in an affinity analysis of the design alternatives considering the benefits and consequences of each alternative. Essentially understanding what can go wrong or how resilient a particular approach is can help the team chose the best course of action.

Testers can be key at this stage to see issues in things like state models, complex business rules, operational issues and the like that can be very valuable. Testers can be involved in the evaluation of proof-of-concept spikes to evaluate design alternatives which is another feedback input that can be very valuable.

The design decision feedback loop allows testers to provide critical input establishing the technical approach taken by the team.

## 5.2 Understanding Code Changes

While consulting in the desktop security domain Robert Sabourin established programmer tester feedback loops related to understanding code changes in order to focus regression testing. The regression testing was expected to expose unintended side effects of code changes.

The feedback loop was implemented with the following steps.

1. Tester reviewed code changes with programmer
2. Each changed module was assigned a risk factor based on the nature of the technical change
  - a. No change in program logic
  - b. Some change in program logic
  - c. Significant change in program logic
  - d. Redesign program logic
3. Tester identified usage scenarios related to changed code
4. Action
  - a. Tests revealed bugs requiring attention
  - b. Tests did not reveal bugs requiring attention

In this case testers understood the relationship between source code module and workflow of the software under development.

When bugs were revealed, programmers completed rework and instantiated another iteration of the feedback loop.

## 5.3 Paired Testing in the Development Environment

While working in the medical software industry it was a common practice for testers and programmers to participate in a type of exploratory testing in the development environment before the code was committed to the main code line. The testing took place on the developer's desktop using the day-by-day development environment including a private data set completely in the control of the programmer

The charter for this exploration was twofold. The tester had a chance to become familiar with new behaviors being implemented in the system. The developer had a chance to exercise the software while it was still a work in process thus making it relatively straight forward to tweak the systems behavior without the need of an additional build cycle,

Here are the basic steps involved in the paired exploratory testing feedback loop.

1. Programmer implements requirement to team definition of done, thus completing unit testing and any technical reviews or other established checklist items
2. Tester identifies usage scenarios related to newly implemented requirements
3. Tester identified variables impacting or impacted by the identified usage scenarios
4. Tester uses test design techniques to select values for variables of interest
5. Programmer sets up development environments with break points and logging of code being exercised.
6. Tester and programmer walk through usage scenario with prescribed values for selected variables observing systems behaviour and ensuring code is executed as intended.
7. Action
  - a. Bugs are identified, and attended to on the spot
  - b. Tests do not reveal bugs
  - c. Tester learns about the newly implemented system behaviour
  - d. Tester identifies technical risks based on code changes
  - e. Tester identifies strategies to assess correctness
  - f. Tester identifies mechanisms such as hooks to facilitate testing and result interpretation
  - g. Tester identified testing ideas based on the technical changes to the source code

## **5.4 Bug Reporting**

Bug reports are a potentially influential deliverable of a testing activity.

Many software testing texts, articles and courses have been dedicated to guiding testers in excellent bug reporting. Cem Kaner's book "Bug Advocacy: A BBST Workbook" offers respected guidance to writing an effective bug report.

Bug reporting is a part of two important feedback loops in software engineering,

### **5.4.1 Bug Improvement**

The tester actively elicits feedback from peers, programmers and other stakeholders about the form and content of the bug report. Based on feedback the tester improves the statement of the bug including information which helps teams make better decisions about bugs, helps programmers isolate and correct the bug efficiently and helps support organizations help customers work around or otherwise deal with the potential impact of the bug.

Improvements in bug reporting usually result in a combination of improved individual skills and standard practices and guidelines for future defect reporting and management.

### **5.4.2 Bug Triage**

#### **5.4.2.1 Feedback in Triage**

Invariably software development teams need to decide what to do about a bug. Do we fix it now? Should we fix it later? Should we leave it in the product? Should we publish a way to work around the problem?

When working in the medical knowledge base field, Robert Sabourin established some important tester, programmer feedback loops relating to bug triage. A helpful heuristic had to do with the effectiveness of testers. During projects the team established that the most effective testers were testers who found and reported bugs that ended up being fixed. Effective testers were able to persuasively advocate for their

bugs. Effective testers learned which types of bugs to look for and did not invest a lot of effort in searching for bugs that would not get corrected.

The basic feedback loop was described as:

1. Tester identifies a bug
2. Tester reports the bug
3. Bug is triaged
4. If bug is corrected
  - a. Tester learns factors making this bug important
  - b. Encourage testing which would expose bugs with similar factors
5. If bug is not corrected
  - a. Tester learns factors making this bug unimportant
  - b. Discourage testing which would expose bugs with similar factors

Note that any bug identified was always reported, what changed was the type of bug testers were looking for. For example, if grammar errors were not considered important then testers would not explicitly look for grammar errors, however if the tester chanced upon a grammar error it would be reported.

#### **5.4.2.2 Becoming an effective Bug Advocate**

An effective tester is an informed tester sensitive to context and skilled in rhetoric.

Feedback loops are critical to help testers understand the impact of context on both the severity and priority of bugs.

Robert Sabourin, when working in the medical software domain, observed many bugs that changed both priority and severity based on project context. A clinician could be distracted by a subtle cosmetic error. The same clinician would never trust software which conspicuously crashed. Counterintuitively, the conspicuous crash could have a lower severity than the cosmetic error. The conference presentation, “The Elevator Parable” [33] shows several examples of priority and severity being sensitive to context.

The effective tester learns from stakeholder feedback about the relative importance, and risks, associated with bugs. The effective tester judiciously chooses which bugs to advocate.

Feedback loops are also critical in helping the testers understand the value and limits of persistent argumentation about bugs. As Ross Collard indicates in “The Tester’s Guide to People & Organization Issues the Tester’s Survival Guide” [31], the successful tester is a great salesperson. The skills of persuasion needed to convince stakeholders that a bug should be fixed are familiar to expert sales professionals. Many sales skills are about relationships and values. Testers should look at books such as [32] Johnson, Spencer, and Larry Wilson. 2002. *The One Minute \$ales Person* to pick up some tips about sales skills that matter.

## **5.5 Troubleshooting and Debugging**

### **5.5.1 Troubleshooting**

A lot of training focuses testers on the act of identifying and reporting about bugs in software. Rich terminology and vocabulary are established to emphasize the difference between errors, failures, issues, incidents, and defects.

The authors of this paper come from an engineering background. The authors consider it critical for testers to provide insights into the cause of a problem, not just the externally observed behavior of the system. Testers often have deep knowledge of the system data for example that can be invaluable in understanding many types of problems.

The authors define troubleshooting as a term used to describe isolating the cause of a specific problem.

This is an example of a tester programmer feedback loop involving trouble shooting.

1. Programmer implements a requirement
2. Tester identifies a requirement related unexpected behavior of the system under test
3. Tester formulates a hypothesis of which factors drive the unexpected behavior
4. Tester runs a trial experiment to confirm or refute the hypotheses
  - a. If not confirmed, consider another factor
  - b. If confirmed report bug

### **5.5.2 Debugging**

The authors define debugging as is a series of technical activities directed at finding ways to resolve a problem.

In the text, Systematic Software Testing, SST, author Rick Craig indicates that testing and debugging should be isolated from each other. In SST, testers are expected to identify defects but are not expected to isolate their cause nor to find a solution. In SST, testing is not debugging. In SST, testing is not troubleshooting.

This is an example of a tester programmer feedback loop involving debugging.

1. Tester reports a bug
2. Programmer formulates a hypothesis of which factors influence the bug
3. Programmer runs a trial experiment to confirm or refute the hypotheses
  - a. If not confirmed, consider another factor
  - b. If confirmed, then correct the bug

### **5.5.3 Active participation by Testers**

A tester can work with a programmer in the development environment to help identify which factors cause a problem and to verify that a proposed solution yields the correct results.

Testers can pair with programmers and other team members to go beyond just identifying a problem, they can collaborate to help resolve the problem as well. You do not need to be a programmer to learn bug isolation and debugging techniques with a modern development environment.

## **6 Feedback Loops Driving Process Improvements**

### **6.1 Bug Clusters**

Many aspects of software testing can influence process improvement. Testing findings can help identify quality concerns about a product. Test findings can also help identify potential process improvements for all aspects of software development.

When testers identify multiple bugs which have a common cause then a bug cluster has been identified. By eliminating the cause multiple bugs can be corrected and future related bugs can be avoided.

In order to understand the cause of bugs testers should review with programmers the actual technical work done to correct the defect. Clusters are defined by common cause, not necessarily by common effect.

### **6.2 Team Retrospectives**

Team retrospectives are examples of feedback loops directed at the work dynamic of the entire team.

Many “agile” teams hold a retrospective meeting immediately following each development iteration or sprint. The retrospective meeting reviews the work done in the sprint.

The goal of the retrospective meeting is to review work done by the team in the previous sprint and look for opportunities to self-organize encouraging effective practices and changing ineffective practices. Changes are to be implemented in the following sprint.

During the retrospective teams review the technical work done. The team discusses the following points:

- What worked well?
- What did not work well?

The team notes practices to continue using and reinforce.

For points that did not work well the team discusses alternatives and decides upon changes redirecting their method of operation for the subsequent sprints.

The retrospective meeting is used to identify improvements in programming and testing tasks. The team discusses the following points:

- Any wasted time?
- Any missing tasks?

The team uses knowledge gained to improve planning in future sprints.

If a task was a waste of time then the team discusses whether it could be avoided or implemented differently in the future. Missing tasks could be added to checklists maintained to help planning future sprints.

The team reviews collaboration within the team and with people outside of the team. Excellent collaboration practices are to be encouraged. Weak collaboration practices lead to discussions of how to improve and redirect such collaboration in the future.

## 7 Feedback Loops with the User Community

When working under extreme time pressure the authors have had to make some important tradeoffs in testing. Deciding which testing activities deserve more attention, or more effort, can help teams manage product risks.

Robert Sabourin had recent experience in a medical analysis company to identify testing focus under strict time pressure working to a fixed deliver date release. In this project Robert set up a feedback loop between the user community and the testing team. Note that even though the system under test had hundreds of menus, controls and dialogues, there were only 30 usage scenarios identified.

The intent was to learn what users did and thus to focus testing on ensuring the users could do their job with the system under test.

Several short task analysis interviews took place to identify who were the users and subsequently what did they do.

The idea was to model what the users did with the system instead of the traditional test model of what the system does for the user. Respectfully rephrasing John Kennedy’s inaugural address “Ask not what the software does for the user but ask what the user does with the software”.

The basic elements of the user tester feedback loop were:

1. Identify different types of users
2. For each user type identify the tasks they are trying to accomplish
3. Prioritize usage tasks
  - a. Pareto analysis
  - b. Business importance
  - c. Technical risk
4. For each high priority task
  - a. Identify factors which influence outcome of task
    - i. Conditions
    - ii. Variables
    - iii. Factors
    - iv. Environment
    - v. Coresident software
  - b. Identify factors which are influenced by outcome of task
  - c. Select values for factors
  - d. Walk through scenario from start to end
    - i. Pair with member of user community
  - e. Identify concerns
    - i. unexpected behaviors
    - ii. inconsistent behaviors
    - iii. missing behaviors

## 8 Feedback Loops in Session Based Exploratory Testing

### 8.1 Exploratory Testing

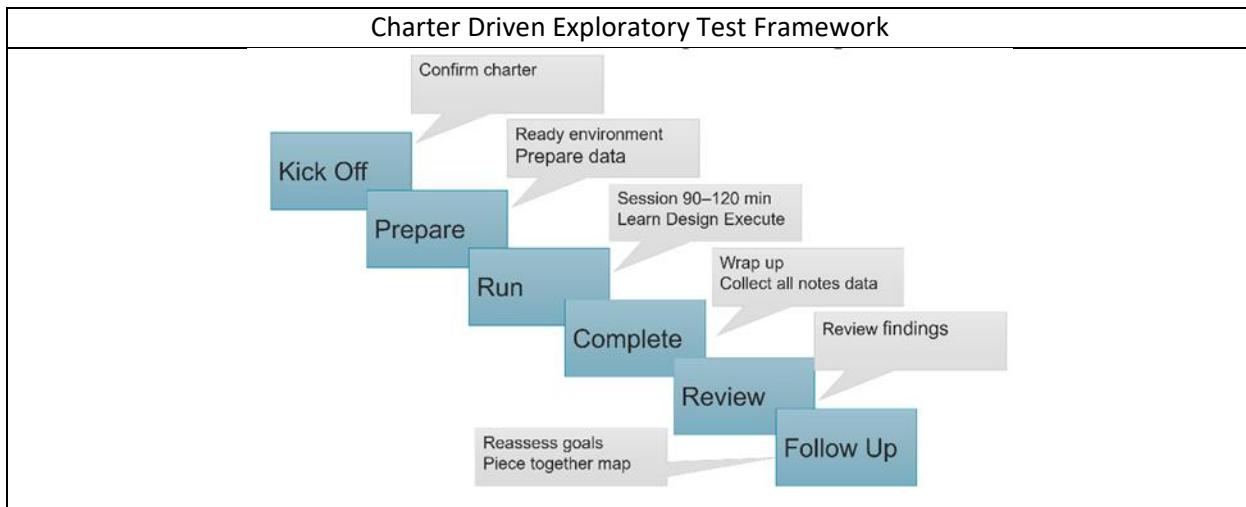
To begin with the authors could suggest that all testing is exploratory and thus exploratory testing is just another word for testing.

Cem Kaner posted the following description of exploratory testing on his blog entitled: "On the craft and community of software testing".

"Exploratory software testing is a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the value of her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project."

## 8.2 Charter Driven Exploratory Test Feedback Loop

An example exploratory testing framework implements the following steps in a feedback loop:



The tester will be exploring. A collaborator will be commissioning the exploration. The collaborator can be a programmer as is more common in an “agile” team. The collaborator can be a test-lead as is more common in a traditional or structured software development life cycle such as Waterfall, Rational Unified Process or V-Models.

Step	Description
Kick Off	The tester reviews the charter with the person commissioning the testing. The tester makes sure that the scope and depth of the charter are discussed and agreed to. The duration of the upcoming session is established. Testers should review which variables, factors, conditions, or data sources may be relevant.
Preparation	The tester gathers whatever resource, data, tools, or equipment are required for the upcoming session of testing.
Run	This is a time boxed session. The tester is expected to focus 100% of their attention of the session of testing. The tester will design and execute tests trying to learn about the charter. The tester will keep a record of decisions made, of tests attempted and of observations. The

	tester will use many diverse tools and technologies to complete this step.
Completion	The tester gathers their findings. The tester reports any bugs in bug tracking tools as required by the project team's workflow. The tester relinquishes the environment. Note that the tester must be able to reset the environment to a predictable state so very often at the completion step the testing will create virtual images of the test environment and data.
Review	The tester reviews their findings with the person who commissioned the testing. In "agile" teams this is often the developer. In the review meeting all findings are reviewed and decisions are made on how to act on the finding in essence the review step is culling test findings and turning them into action. Test results are fed back to the person who commissioned the testing and to any stakeholder would benefit from knowledge of the findings. This is a call for action. The review meeting usually takes place on the same day as the session of testing.  The key decision made is – should we do another session on the same charter or should we move onto something else.
Follow Up	The team acts based on the finding of the tester. The tester acts based on the feedback from the person who commissioned the work and any other stakeholder involved. This list will vary from charter to charter.

### 8.3 Charters

A test charter is a mission statement for testing. It is a goal. What does the tester what to learn about?

Note that charters derive from test ideas. One test idea can map to many charters. Multiple test ideas can map to one charter or there can also be a one-to-one mapping between test ideas and test charters.

On an "agile" team a charter statement can be the "name" or "title" of a testing task.

## 8.4 Session Based

Session based testing is implemented one time-boxed session at a time. Sessions are typically 90 minutes to 2 hours long. Sessions can be shorter or longer, but it is important to agree on the session duration before starting to test.

During a testing session the tester is uninterrupted. The tester focuses on designing and executing tests related to the charter. Testers keep track of their work and record their findings.

At the end of a session the tester reviews their findings with the charter commissioner who is typically a programmer or a test lead depending on the lifecycle model being used.

When reviewing the findings, a decision is made as to whether additional sessions should be implemented to further fulfill the test charter.

## 8.5 Test Findings

There are many ways that exploratory testers can express their findings. A charter can be represented by a task in a workflow management system, for example a Jira ticket. Each session associated with that charter would need to be a sibling object. In a session object the tester would include their findings, session notes, screen shots, screen videos with audio commentary, spreadsheets, data records, virtual images of system under test, pointers to bug descriptions and commentary from developers, product owners, teammates, and other interested project stakeholders. Collecting and recording findings should be a natural part of the testing workflow.

Session notes are like medical notes on a patient chart or a professional engineer logbook. This is a record of the testing done describing decisions made and trials attempted. The session notes do not need to include analysis or assessment, generally session notes focus on recording facts. Some team cultures allow for a section of analysis and recommendations from the tester.

Many testers choose mind mapping tools such as XMind or FreeMind to capture visual representation of test findings. Mind maps can include images, text, links to other objects and relationships between objects.

When testing in a regulated environment consistent session note can be used to demonstrate compliance to regulatory standards.

## 8.6 Feedback Loops to the Programmer

The programmer acts on findings from the session review. This may lead to code modification, bug fixes or further experimentation with the technical solution being implemented.

## 8.7 Feedback Loops to the Tester

The tester learns how to focus their work. Were the findings relevant? Was the scope inclusive of factors of importance? Were some factors superfluous were other factors on target? Were the bugs identified relevant to the project?

The tester adjusts the scope and focus of testing,

The tester adjusts the variables under study.

The tester learns how test findings are used and interpreted. This feedback is important to adjust the information gathered and how it is both recorded and reported. The tester streamlines their notes to focus on actionable information without distraction.

## 8.8 Feedback Loops to and from Other Stakeholders

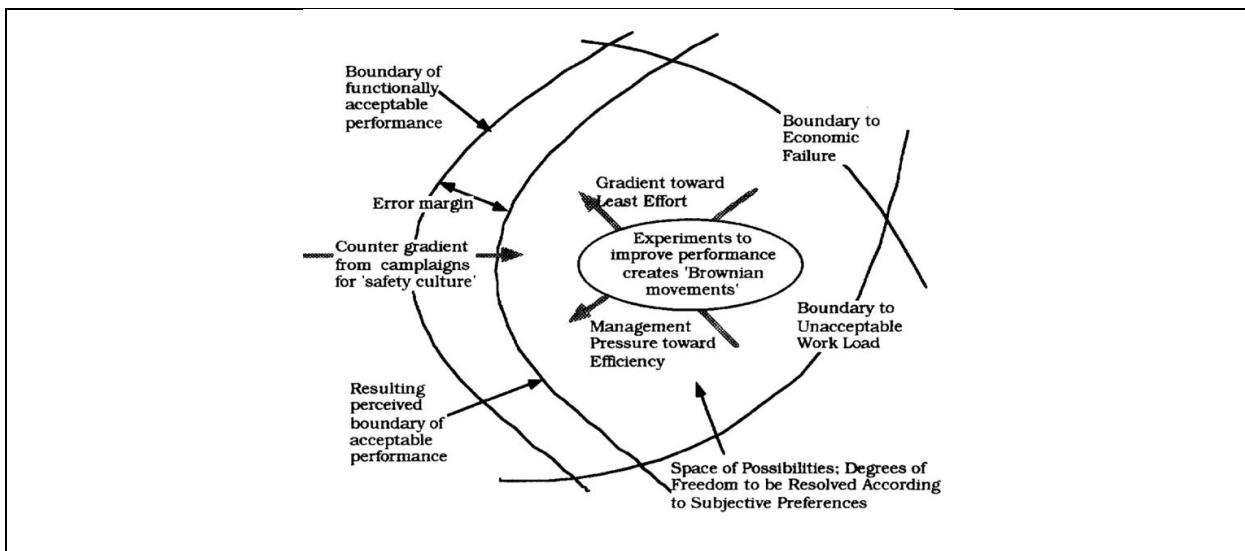
Exploratory test findings can be reviewed in feedback loop with many other project stakeholders. Here are some examples from recent projects.

- The “customer” provides feedback about whether the product delivers the value required.
- An “end user” provides feedback about whether they can complete tasks and workflow with the software under test.
- A “domain expert” provides feedback on strategies to assess correctness and functionality omissions.

## 9 Feedback to Management and the Business

Testers can also provide feedback directly to the business. As mentioned earlier the feedback loop between the product owner and the testers is an active and critical feedback loop for projects. A less talked about feedback loop is direct to management and “the business”. We often talk about “what the business wants”, high level roadmaps over several years (whether you think these are valid, most businesses have them), and other context drivers that make this particular feedback loop so interesting.

One example is a company that needs to demonstrate a particular feature set or capability to a key customer or investors. This is common for companies in the startup phase or ones trying to break into a new market. This demo becomes everything for the team and has a very different focus than normal product development. The team needs to be careful to pay attention to non-functional requirements so they don't regret creating a system they can't operate effectively, but these concerns are diminished in favor of the creation of an effective demo. In our experience management is often very involved in the demos and lean on the testers as much as the developers in understanding if the demo is ready. It isn't uncommon for the testers to be very involved in the creation of the demo script. It's an art to have a demo that is effective but doesn't imply more than the team can deliver.



Jens Rasmussen in “Risk Management in a Dynamic Society” has a fantastic summary of what drives systems, which of course includes software. This is another “triangle” in software development that everyone feels implicitly and seems to be gaining more explicit attention with the increase in systems thinking and complexity that is seeping into our industry. It's easy to see where the concerns that dominate what the business is thinking about, and the concerns of the team intersect. This balance is well

informed by testers who tend to be closer to the business and customers than the rest of the team. As all the forces in play are always pulling at each other, testers are often key to finding balance.

- What issues have workarounds that the customer can live with (helping to ship on time)?
- Are there re-orderings of workflows that can make performance acceptable?
- What bugs just must be fixed? (Harking back to the bug advocacy discussion and concerns about support costs)

Management feels like a constant risk management problem that needs to balance the resources that are available (budget, time, vendors, technology, team skillset, etc.), c-level and investor expectations, the energy of the team, customer expectations and all the rest. Testers are often in the intersection of these concerns and play an important part in getting the best outcome.

## 10 Concluding Remarks

Feedback loops can help amplify the effectiveness of testing and the overall software development process.

Testing related feedback loops are demonstrated to:

- improve product requirements early, before implementation begins
- influence software design decisions
- improve personal and team software process
- Increase engagement in active processes instead of milestones and phase gates that loom over the team.
- to dynamically adapt the focus, scope and depth of testing based on active feedback
- to influence process, change in self-organized teams to improve product quality
- find and fix the bugs that matter sooner
- provide vehicles to collaborate with stakeholders, programmers, and members of the user community

## 11 Acknowledgments

The authors wish to thank their peers in the software engineering community.

Robert Sabourin would like to thank the thousands of students who have participated in his software engineering and software testing courses over the past several decades.

## References

- [1] Myers, et al. *The Art of Software Testing*. John Wiley & Sons, 2012.
- [2] Kaner, Cem, and James Bach. *Lessons Learned in Software Testing*. Wiley, 2001
- [3] Pólya, George. *How to Solve It: A New Aspect of Mathematical Method*. Doubleday, 1957.
- [4] Sommerville, Ian. *Engineering Software Products*. Pearson Education, Inc., 2020.
- [5] Copeland, Lee. *A Practitioner's Guide to Software Test Design*. Artech House, 2008.
- [6] Sabourin, R. *Charting the Course Coming Up with Great Test Ideas Just in Time*. AmiBug, 2020.
- [7] Dijkstra, Edsger, "Programming methodologies, their objectives and their nature." 1969
- [8] Hersey, Paul, and Kenneth H Blanchard. 1969. *Management of Organizational Behavior: Utilizing Human Resources*. Englewood Cliffs, N.J: Prentice-Hall.
- [9] Seashore, Charles N., Seashore, Edith W., and Weinberg, Gerald M., 2013. *The Art of Giving and Receiving Feedback*, Smashwords, U.S.A.
- [10] Blanchard, Kenneth H, Patricia Zigarmi, and Drea Zigarmi. 1985. *Leadership and the One Minute Manager: Increasing Effectiveness through Situational Leadership*. 1st ed. New York: Morrow.
- [11] Blanchard, K. A. J. 2022. *One Minute Manager* (New Thorsons Classics edition). Harper.
- [12] Shewhart, Walter A. 1939. *Statistical Method from the Viewpoint of Quality Control*. Edited by W. Edwards Deming. Washington: Graduate School, the Dept. of Agriculture.
- [13] Deming, W. Edwards, and Joyce Nilsson Orsini. 2013. *The Essential Deming: Leadership Principles from the Father of Total Quality Management*. New York: McGraw-Hill.
- [14] Kiran, D. R. 2016. *Total Quality Management: Key Concepts and Case Studies*. U.S.A.: Elsevier Ltd.
- [15] Jonassen, David H, Martin Tessmer, and Wallace H Hannum. 1999. *Task Analysis Methods for Instructional Design*. Mahwah, N.J.: L. Erlbaum Associates.
- [16] Kaner, Cem and Rebecca Fiedler. 2015. *Bug Advocacy: A BBST Workbook*. U.S.A.: Context Driven Press.
- [17] Geyer Studio, Copyright Claimant. Inaugural address, by John Fitzgerald Kennedy, President of the United States, 1961 to 1963., ca. 1966. Photograph. <https://www.loc.gov/item/2015649386/>.
- [18] Craig, Rick D, and Stefan P Jaskiel. 2002. *Systematic Software Testing*. Artech House Computing Library. Boston: Artech House.
- [19] Weinberg, Gerald M. 2001. *An Introduction to General Systems Thinking* Silver anniversary ed. New York: Dorset House.
- [20] Cohn, Mike. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley Signature Series. Boston: Addison-Wesley.
- [21] Derby, Esther, and Diana Larsen. 2006. *Agile Retrospectives: Making Good Teams Great*. The Pragmatic Programmers. Raleigh, NC: Pragmatic Bookshelf.
- [22] Schwaber, Ken, and Mike Beedle. 2002. *Agile Software Development with Scrum*. Series in Agile Software Development. Upper Saddle River, NJ: Prentice Hall.

- [23] Schwaber, Ken. 2004. Agile Project Management with Scrum. Redmond, Wash.: Microsoft Press,
- [24] Freedman, Daniel P, and Gerald M Weinberg. 1990. Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products. 3rd ed. Little, Brown Computer Systems Series. New York, NY: Dorset House Pub.
- [25] Gilb, Tom, Dorothy Graham, and Susannah Finzi. 1993. Software Inspection. Wokingham, England: Addison-Wesley.
- [26] Wiegers, Karl Eugene. 2002. Peer Reviews in Software: A Practical Guide. The Addison-Wesley Information Technology Series. Boston, MA: Addison-Wesley.
- [27] Fagan Michael, "A History of Software Inspections" in Broy, M, Ernst Denert, and Sd & m AG. 2002. Software Pioneers: Contributions to Software Engineering. Berlin: Springer.
- [28] Kaner, Sam; Lind, Lenny; Toldi, Catherine; Fisk, Sarah; Berger, Duane; Doyle, Michael. 2007. Facilitator's Guide to Participatory Decision-Making. Hoboken, NJ: Jossey-Bass.
- [29] DeMarco, Tom, and Timothy R. Lister. 1987. Peopleware: productive projects and teams. New York, NY: Dorset House Pub. Co.
- [30] Kaner, Cem, 2006, "Defining Exploratory Testing", <https://kaner.com/?p=46>
- [31] Collard, Ross. The Tester's Guide to People & Organization Issues the Tester's Survival Guide. New York, Collard & Company, 2008.
- [32] Johnson, Spencer, and Larry Wilson. 2002. The One Minute \$ales Person. New York: W. Morrow.
- [33] Sabourin, Robert. 2003. "Establishing Bug Priority And Severity: The Elevator Parable". Stickyminds. <https://www.stickyminds.com/presentation/establishing-bug-priority-and-severity-elevator-parable>.
- [34] Rasmussen, Jens. 1997. "Risk Management in a Dynamic Society: A Modelling Problem." Safety Science Vol. 27, No. 2/3. pp. 183-213.

# Building a Modern Quality Program from the Ground Up

**Jeff Sing**

[jeff.sing@iterable.com](mailto:jeff.sing@iterable.com)

## Abstract

As startups mature, one of the biggest trends is the decision to build a Quality Program from the ground up. However, these smaller, agile, and more DevOps-centric companies aren't looking to hire a horde of testing specialists who act as a safety net—waiting for code to be tossed over the proverbial wall by developers to perform validation before a release is completed. Instead, engineering leaders are looking for a modern Quality Program that is significantly smaller and more specialized. They want Quality Leaders that rely on technology, push a concept of "Quality Culture," and are Quality Program Managers to ensure that engineering organizations successfully deliver value to their customers.

## Biography

*Jeff Sing is a Quality Leader who has been in the testing industry for over 15 years. During this time, he has built test automation frameworks, tested strategies, and executed quality initiatives for fields such as medical devices, infrastructure security, web identification, marketing tech, and experimentation and progressive delivery.*

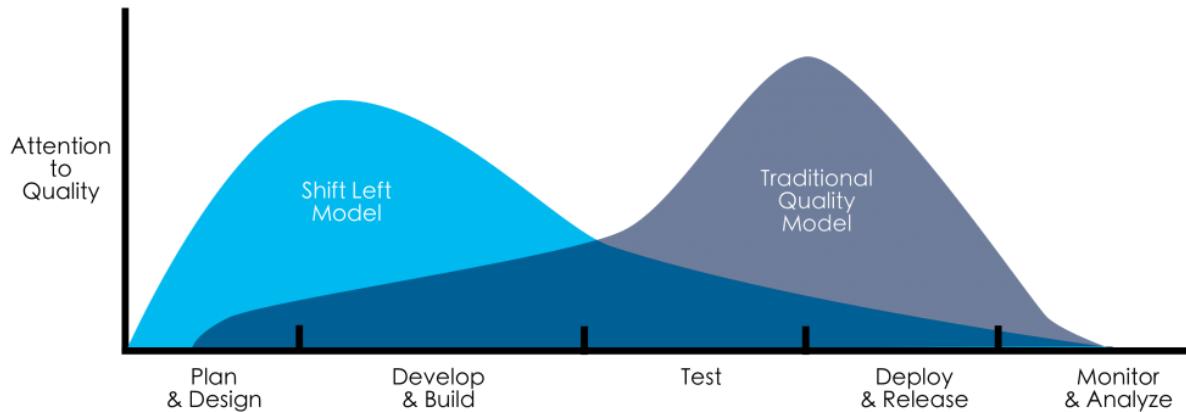
*Jeff is currently a Sr. Engineering Manager at Iterable where he has both built and leads the Quality Engineering and Engineering Operations teams. Prior to joining Iterable, Jeff built the Quality Engineering team at Optimizely where he was both the Software QE Manager and Chief of Staff to the CTO.*

*Jeff currently resides in Redwood City, California along with his wife and three children. Jeff studied Computer Science at UCSD, and to this day is still a passionate Chargers fan.*

## 1. Introduction

One of the more interesting trends I have noticed perusing job boards during the last five years was the proliferation of companies looking for new QA Managers or QA Leads who would either build their "Quality Engineering" program or significantly revamp their current ones. When talking to these engineering leaders at these companies, they were clear that they were looking for an individual to be some amalgamation of an Automation Architect, Testing Coach, and a Quality Agile Leader that could "improve quality." But the catch was none could clearly describe how they expected quality to improve (Better test coverage? New automation framework? Quality Best Practices?), nor did they really understand how these quality initiatives would actually impact quality improvement (eg: Does more automation equate to fewer bugs? Does this mean you will have happier customers? What if the trade off is slower deployment speed, how does this affect customer satisfaction now?).

This struggle can be attributed to what software quality programs used to be like, where companies staffed teams of testing specialists who act as a safety net, waiting for code to be tossed over the proverbial wall by developers to perform validation before a release can happen [1]. These software quality engineers were responsible for testing the application and finding defects.



*Shift Left is about doing things earlier in the development cycle. Source: van der Cruijsen 2017.*

However, modern agile stresses velocity paired with modern DevOps practices in Shift Left (*an approach to software testing in which testing is performed earlier in the development cycle*) to attain validation. This shift requires developers to bake in testing inherently into the cycle of software releases as they are built, rather than in the end stage pre-deploy. This motion of testing earlier and often and automatically on each code build, paired with maturity in deployment frameworks that allow you to quickly deploy bug fixes to production, made engineering companies re-evaluate if they needed testers at all to perform validation in the latter stages of the development cycle. *"The increasing adoption of agile and DevOps is minimizing the importance of QA for many teams because these ideologies focus on speed, and quality can become secondary. (Mason, 2019)[2]"*.

In 2015, Yahoo eliminated QA by moving to CI/CD (Continuous Integration/Continuous Delivery) and having software engineers be responsible for code quality which forced them to develop automated testing tools and execute tests earlier in the pipeline[3]. Many software companies followed this model, cutting a lot of their testing team, or relegating their testing team to be pure automation groups [4]. Quality (requirements, test scenarios, and creation of test cases to be automated) was owned by all developers and no longer a QA team.

However, even with this change, engineering organizations were still plagued with quality issues. What they found was the correlation between having a lot of automation tests didn't always translate to answering the question "Are my customers really enjoying my product and is it of high quality?" [5]

This question, in fact, has become more critical in how successful engineering organizations function. In the World Quality Report in 2011-22, the top quote in the executive summary was "*Most CIOs now value testing more than ever before, and the onward march towards digitization is ensuring that customer experience and quality are of utmost importance.*" [6]

To address how to build a Modern Quality Program that champions the customer experience and enables engineering to deliver with quality, the Modern Quality Program requires a leader that is able to combine both the technical leadership needed to architect the right tooling, and a Quality Program Manager that understands how to mature the Quality processes in that company's Software Development Life Cycle(SDLC). Intertwined with the above, the leader also needs to be able to build the entire operation to ensure that the tooling and process iteratively and pragmatically scales for the engineering organization at hand, and measure to ensure the program is not just solving the right problems, but solving them efficiently. Lastly, this engineering leader needs management savvy to be a solid people leader in order to scale and grow the quality organization to support the company as it expands.

## **2. The Four Key Roles when Building a Modern Quality Program**

The Modern Quality Program is built on the balance of enacting the right quality program and process (Quality Program Manager) versus executing the right automation tooling (Automation Technical Architect). To determine if either programs are fit-for-purpose requires the Quality Operations role. Lastly, none of this is sustainable as a one-person organization if the company scales and grows. The fourth role (Engineering Leader) is essential to properly grow the correct team to carry on forward. Below are the definitions of each key role.

**Quality Program Manager:** Implements the Quality SDLC Program, driving the engineering organization towards its Quality North Star. By building and enforcing quality guard rails, the Quality Program Manager remediates the friction of failed orchestrations or confusions during the design and build phase, and, instead, allows engineers to focus more on the implementation of their features.

**Technical Architect:** Visionary of the technical roadmap of tools, frameworks, and QAOPS (the integration of testing and QA with releases during CI/CD) needed for engineering to ship with high velocity. The Technical Architect creates the partnership with engineering to ensure there is a full spectrum of sustainable automation across the testing pyramid that guarantees high feature coverage.

**Quality Operations:** Identifies what KPI/Metrics represent Quality, and reviews if the engineering organization achieves its goals. Ensures that the tools and processes implemented are actually efficient and effective.

**Engineering Leader:** Builds and grows the team to execute all of the above. Ensures that said team works well together, and that each individual member is supported in reaching their career goals. The Engineering Leader balances between validating that the output of your team's current outcome is good while setting up the team for great outcomes in the future. [7]

## **3. The Quality Program Manager Role**

The Quality Program Manager creates and coordinates the various quality programs and processes across the engineering organization. To accomplish this, the leader needs to establish Program Vision, Quality Coaching, and Customer Advocacy.

### 3.1 Program Vision

Vision can be described as your quality North Star.

Questions I like to pose:

- *What is the charter of the organization that you are trying to build?* I always tie mine to the company's vision statement. At Iterable, my Quality Charter is: To ensure our engineering team is enabled to deliver joyful customer experiences for every organization in the world.
- *What is the stack rank of things that you need to accomplish?* What are the critical areas that I see damaging the ability for us to fulfill our promise of delivery of joyful experiences? This is important because understanding what your customer truly cares about allows you to prioritize. If your customers rely on your product's up-time versus utilizing the application's user interface, then my highest priority for quality will revolve around ensuring our product is available
- *How are you going to go about achieving this vision?* You've already established your roadmap, now you need to define your program milestones. Defining your program milestones and reporting on the velocity of how you are doing to your stakeholders keeps you on track.

Starting a new Quality Program is a lot like being thrown into the middle of the ocean and told to swim to shore. On top of this you will be crashed upon with waves of requests from different stakeholders looking for help. Having a clearly defined vision allows you to steer in one direction by determining what your quality program needs at this time versus what doesn't fit at this juncture.

### 3.2 Quality Coaching

One of the most critical roles of the Quality Program Manager is Quality Coaching. Coaching is defined by the International Coaching Community [8] as an individual who provides guidance to a client on their goals and helps them reach their full potential. In the same vein, Quality Coaching is helping your engineering organization and leadership team achieve their quality goals. I divide Quality Coaching into two categories: Quality Leadership and Thought Leadership.

#### 3.2.1 Quality Leadership

Quality Leadership revolves around how to establish the right process and programs that will yield better outcomes. This involves understanding the current engineering SDLC and identifying where Quality Artifacts and Programs should fit in (eg: Who should be building a Test Plan if we have no QA engineers at this moment? What does the Test Plan look like? How does it affect the release process?) and working with engineering leadership and developers to officially be part of the SDLC (eg: Training teams to write a Test Plan, Auditing that all releases have a Test Plan).

### 3.2.2 Thought Leadership

Thought Leadership determines the ROI of each quality process being implemented. Depending on the maturity of the engineering organization, it may not always make sense to implement every standard QA process. On the other hand, as an organization matures and scales, some older practices should be sunsetted. An example of this would be an organization that didn't have automated tests and required all its engineers to manually validate features prior to a deployment. Once a suitable framework that gave good feature coverage was implemented, the act of manually testing probably would not result in a good ROI.

### 3.3 Customer Advocate

A lot of testing and validation is built around answering the question "Did I build the product right?" But engineering often relies on Product Management to answer that question. This essentially becomes a single point of failure, since engineers are relying purely on Product as a gatekeeper. One of the essential functions of a Quality Program Manager is to act as a Customer Advocate throughout the entire SDLC.

Some examples:

- Partnering with the Product Manager to review the Acceptance Criteria and Workflow Scenarios (which eventually will become Test Cases) through with your GTM (Go to Market) organization. Getting feedback from Solution Engineers and Account Executives over certain features and how customers are most likely to use them gives you insight on what to test against.
- Inviting Technical Solutions Engineers or Customer Service Managers into release Bug Bashes to give you a customer's perspective of how your new features will fare. This often catches usability bugs that would become major issues later on.
- Meeting with the employees that support, sell, and manage your customers to understand their biggest pain points or complaints with the product. This helps you determine where to harden your testing for new features.

## 4. The Technical Architect Role

The Technical Architect Role revolves around partnering with engineering to drive best automation testing practices, and partner/build/lead the charge in creating/maintaining testing frameworks across the application and platform it's built on. To accomplish this, the leader needs to establish Technical Leadership and Automation Mentorship.

### 4.1 Technical Leadership

Technical Leadership encompasses not just the testing pyramid but the infrastructure the tests are built on as well as the environments in which it runs. A technical leader needs to give direction on the following:

- How good is my unit testing and what should our code coverage be? (Is this important to us or not?)

- Do my developers utilize integration testing? Do they need help in developing this? (Who should maintain this framework?)
- Who is currently writing end to end tests? (Who maintains these? Do developers and QE both write them, or is this the role of your QE team?)
- What environment are we testing in? (Do I have useful test data?)
- Is my current CI/CD pipeline running my tests? (Are they flakey and causing issues? If so, who should be fixing these?)
- How do we test in production thoughtfully? (Who is managing my Feature Flags?)
- Should I build or buy tools for visual testing, AI/ML testing, crowd testing? (What is the integration cost really to stand these up?)

## 4.2 Automation Mentorship

More likely than not, if you're building your Quality Program you don't have a team, or, if you do, it's very small and unsustainable for building all the automation testing. Even if you had a large Quality team, engineers should be participating in writing automation tests. Automation mentorship revolves around building the programs to help uplevel engineers in writing good automation. Here are some examples of programs that really help mentor engineers in building good automation:

- **Testing Guild or Testing Community of Practice:** Weekly or Bi-Weekly meeting of engineers working on testing best practices or tooling. Engineers that attend bring these lessons with them back to their individual teams.
- **Quality Champions:** Identifying individual engineers who want to engage quality initiatives and find ways to encourage or empower them. At a previous company we had a developer who built a proof-of-concept Flakey Test Detector. His solution would have decreased the time to build in our pipeline since we had less failures. We found time for him to take a few sprints to productionize his tool and roll out to the rest of engineering. The end result was that we saw a 30% improvement in failures and 60% faster deployment speeds.

## 5. The Quality Operations Role

Quality Operations involves identifying the indicators that we want to monitor and measure to determine our system quality and customer satisfaction. This allows us to utilize data driven hypotheses on what adjustments we should make to our quality programs or technical initiatives to achieve better outcomes. Quality Operations runs in a cycle where we constantly review our KPI and plan out our next initiatives.

### 5.1 KPI and Metrics

I like to use a mix of KPI and metrics to measure quality. As with all metrics, it's important to view them collectively to understand what is happening with the system as opposed to using them individually. I also want to introduce a caveat, the KPI and Metrics I highlight on the bottom are different than your traditional QA Metrics (eg: escaped bugs, test coverage, test reliability, time to test, time to fix, etc) because depending on your program you are building, these might be prioritized or targeted differently. Some companies will absolutely care and set high SLA, while others may have other more pressing challenges and not prioritizing these at all. Therefore I like to pick metrics that tell a more visual story of overall

engineering health and effectiveness which rises above simple quality metrics. I break these up into two categories: KPIs that are benchmarkable across multiple engineering companies, and KPIs that link quality with business success.

### 5.1.1 KPI and Metrics that are Benchmarkable

The following metrics like Change Failure Rate and Mean Time To Restore allow me to benchmark against other companies to see how far we are faring [9].

	<b>Elite</b>	<b>High</b>	<b>Medium</b>	<b>Low</b>
Deployment frequency	On-demand (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time to change	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service	Less than one hour	Less than one day	Less than one day	Between one week and one month
Change failure rate	0-15%	0-15%	0-15%	46-60%

*Key Metrics of the Dora Benchmark*

- **Change Failure Rate:** For the primary application, what percentage of changes to production resulted in degraded services that subsequently required remediation like a hotfix or rollback? I use this benchmark to determine how effective our system is in prevention of major issues per deployment.
- **Mean Time to Restore (MTTR):** Time to restore service that causes failures. The longer the outage the more unhappy our customers are which is an important factor. MTTR gives insights into code quality, reliability, and stability. Proper quality processes can help improve on these all and can reduce MTTR.
- **Lead Time for Changes:** How long does it take to go from code-committed to code running in production. As our testing framework is often part of this calculation, it's important to see if we are negatively impacting this time or not. A low lead time can affect incident response time and will lead to a drop in developer productivity.

### 5.1.2 KPI and Metrics that link Quality with Business Success

The following metrics are some that I have found measure quality in the view of business success. Every company I have implemented these at have had different formats and values, so they aren't benchmarkable like the last section.

- **Normalized Quality Ratio:** The number of incoming customer reported bugs versus the number of outgoing pull requests. This allows me to get context in what our bug counts mean. I could understand if we delivered a lot of features and had a lot of incoming bugs, but I would be

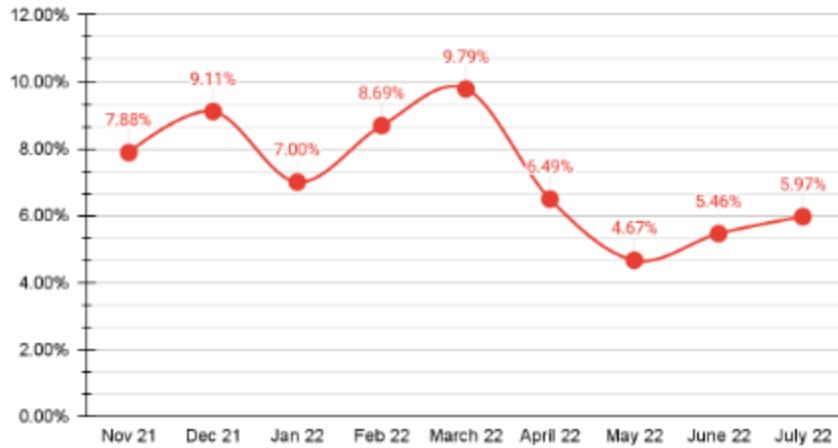
concerned if we had a lot of incoming bugs and shipped no code. Generally if we had the latter, this is symptomatic of severe technical debt and is a great indicator that the costs to just Keep the Lights On (KTLO) is a risk factor that needs to be addressed.

- **Time to Close High Severity/Mid Severity Bugs:** Depending on if we had an SLA, this data is important to collect. The business contract, either internal to customer success or external to our customers, needs to be upheld. This data allows us to help drive prioritization over future work.
- **Customer Net Promoter Score (NPS):** Depending on how your Customer Support team collects this information, I like to see how happy my customers are using our product. No matter how many bugs you close, if your customers are miserable, then you aren't winning. The caveat here is NPS can be influenced by a lot of factors and not feature quality, so the signal here isn't always as important.

## 5.2 The Quality Operation Cycle

1. **Data Collection:** Identify what Key Performance Indicators (see section 5.1) you want to monitor to determine the quality of your engineering output.. Identify how to collect this data and begin collecting.
2. **Visibility:** How do you want to share your findings, and how do you tell the story of what the data is showing you? I used a combination of monthly Quality Reports and a Quarterly Service Delivery Review to go highlight our findings.
3. **Implementation:** Drive discussion amongst stakeholders around findings. Partner with stakeholders to create engineering initiatives to implement action items in addressing issues discovered.
4. **Governance:** Build into place a system to hold the stakeholders accountable. These could be dashboards to show progress, or error budgets that trigger to force a code freeze.
5. **Reflection and Analytics:** Understand if the initiatives actually made a difference. If not, how should we iterate to perfection? Were we looking at the wrong data? What learnings should we take as we make changes? All of these should be applied as we start back at step one. One example seen at Iterable was we benchmarked our PR Normalized Bug Ratio over a course of a quarter and noticed over 30% of incoming bugs were related to one area of our code base. We saw that the new code was breaking existing functionality, creating regression issues. The Quality Engineering team created an initiative to increase test coverage in that area of code by 275% (8 tests specs to 22). The Quality Engineering team also required a launch checklist and combined bug bashes before teams could release. After a quarter we saw a dramatic decrease in defects from new code in that one area and an improvement in regression. Looking at our PR Normalized Bug Ratio, we see that we are still shipping features at the same rate, but the incoming bugs have improved, which ties together our initiative with real time results.

PR Normalized Bug Ratio (Nov 21 - July 22)



*Tracking to see if our new testing initiatives improved our metrics. PR Normalized Bug Ratio is defined in 5.1.2 and the lower the value the better quality exhibited.*

## 6. The Engineering Leader Role

While to be successful in the Engineering Leader role is not much different than being a successful engineering manager in general, there are a few caveats that I have found to be essential to consider:

- **Thoughtful Team Building:** Modern Quality Programs tend to be very lean agile teams. This means that while you may need an engineer that has Mobile testing experience, hiring a specialized engineer may not give you the most bang for your buck. Instead, figure out what can be trained and what skill set is the minimally viable for success.
- **Security Champion:** Partner closely with the security leadership to embed a lot of the security improvements into your roadmap. As the world improves connectivity, modern “bad actors” have become more savvy in penetrating applications. This creates a huge business requirement for greater security and resilience. This requires earlier testing and QA which requires closer partnerships. [6]
- **Soft Power:** Modern Quality Programs rely heavily on other organizations to accomplish compliance and engineering resources. You will need to be comfortable utilizing soft power to accomplish your goals
- **Swiss Army Knife Job Roles:** While building a Modern Quality Program, there will be times when the required solution is outside your job description, but still heavily impacts quality. To be successful, you may have to build the solutions in order to enable your own success (be comfortable unblocking yourself). It isn’t uncommon that while in this role you may have to become a Jira Admin, Technical Program Manager, Product Manager for certain internal tools, Release Manager, or a Developer Experience Champion for initiatives that help enable your engineers.
- **People Manager:** A lot of the above roles I mentioned end up leaning very heavily into the Individual Contributor work sphere. It’s essential that to grow a successful Quality Program, your program has great staff to execute your quality vision. Things to remember:
  - **Empower them to execute in their role:** Remember from above, you aren’t building a program of testers. You are building a program for quality. Ensure that your engineers are

spending the right ratio of time driving the quality program, coding automation, and measuring the success of their own actions.

- **Delegate:** As your program grows, don't be afraid to delegate responsibilities to your engineers as you spend more time trying to scale your program. You won't be successful if you keep trying to do everything, eventually you will hit your breaking point.
- **Mature and Grow your engineers:** Most likely going to be hiring and leading a team. Your responsibility as a people manager means that you need to grow your individual directs while setting them up for success. This might include ensuring they are getting the right technical training, mentorship on career growth, understanding their impact, or just ensuring they are motivated and satisfied in their role. Ultimately the success of your program depends more so on the people who will execute on your behalf and less on your pure execution alone.

## 7. Conclusion

As companies scale and grow, their engineering organizations need to be able to justify whether or not their outputs are achieving effective business outcomes. Building a Modern Quality Program is meant to help engineering define the correct outcome, put in place the right governance and guide rails to help developers efficiently and effectively deliver, and successfully scale the whole infrastructure as the engineering team matures and grows.

The greatest challenge isn't necessarily implementing a certain process or building the automation framework, it's determining if that process or testing will actually improve the business outcomes and growth against the tradeoffs of implementation. Using the correct data to justify the right decisions is the crux of deciding on which tasks the program should execute.

## References

1. Heusser, M. (Retrieved on 2022, June 9) The future of software testing: How to adapt and remain relevant. Retrieved from:  
<https://techbeacon.com/app-dev-testing/future-software-testing-how-adapt-remain-relevant>
2. Mason, R (2019, May 18) Is Agile Killing QA. Retrieved from:  
<https://www.forbes.com/sites/forbestechcouncil/2019/03/18/is-agile-killing-qa/?sh=40f846d734d6>
3. Perry, T.S. (2015, Dec 11) Yahoo's Engineers Move to Coding Without a Net > What happens when you eliminate tests and QA? Fewer errors and faster development, says Yahoo's tech leaders. Retrieved from:  
<https://spectrum.ieee.org/yahoos-engineers-move-to-coding-without-a-net>
4. HelpSystems Armenia (2019, Oct 10) Is QA Dying? [Blog Post] Retried from:  
<https://medium.com/helpsystems-armenia-writes/is-qa-dying-730b0a166c4d>
5. Ghahrai, A (2020, March 11) Problems with Test Automation and Modern QA [Blog Post] Retrieved from: <https://devqa.io/problems-test-automation-modern-qa/>
6. World Quality Report, 2021-22, 13th Edition. Capgemini. Retrieved from:  
<https://www.sogeti.com/globalassets/reports/wqr-21-22/world-quality-report-2021-22.pdf>
7. Zhuo, J. "What is Management?" The Making of a Manager, 2019.

8. What is Coaching (Retrieved on 2022, June 9) Retrieved from:  
<https://internationalcoachingcommunity.com/what-is-coaching/>
9. Accelerate State of Devops 2021. Google Cloud. Retrieved from:  
<https://services.google.com/fh/files/misc/state-of-devops-2021.pdf>

# Leading successful teams

**Vadiraj R Thayur**

Vadiraj.Thayur@gmail.com

## Abstract

In today's agile world, managing teams is quite a challenge. The pandemic has added another dimension to the problem: Remote work. A blend of management experience and skills are necessary to successfully manage teams in the current scenario.

This paper is an attempt to help current and future managers and leaders gain some of these necessities. The paper talks about the various aspects to be considered to build and manage successful teams along with real-world instances to support them. This approach is based on the various experiences of the author in his management journey as well as some learnings from observing other teams and management styles. In this paper, we discuss key aspects of successful teams: Hiring, Build Trust, Always-in-sync, Ownership, Freedom, Career Path, Leadership, Celebrate and Separation.

Team management is never straight forward, approach that works for one team might not work for another. However, the basic framework remains valid across teams and this paper aims at providing the framework for current and upcoming managers and leaders.

## Biography

*Vadiraj has more than 18 years' experience in Software Quality with a master's in quality management from BITS Pilani. He is working in Trellix as QA Director, driving Software Quality for the Enterprise SaaS solution. He carries more than 8 years' experience in managing teams. He presented papers at PNSQC in 2012 and 2014 and a poster paper in 2015. He was also involved with PNSQC as a paper reviewer since then for 3 editions of the conference.*

*Disclaimer: The views and opinions expressed in this paper are the personal views and opinions of the author and do not represent the views and opinions of his employer.*

Copyright Vadiraj Thayur 2022

## 1 Introduction

*"A team is defined as a group of people who perform interdependent tasks to work toward accomplishing a common mission or specific objective."*

Each team is unique, with individuals from diverse backgrounds, skills, strengths and weaknesses. So, what does it take for a team to be a high performing successful one? The onus of sharing the vision, motivating the team members, bringing in a culture, resolving conflicts and creating an environment for them to thrive and excel lies with the leader.

Managing teams is an art as well as science. Leaders must apply their knowledge and experience to develop the skill of managing teams. The approach each leader adopts depends on their style, beliefs, values and intentions. Leaders should ensure the team marches towards the common goal and at the same time, they need to ensure the individuality of each member is preserved and utilized in a positive way to achieve the goal. Success of teams depends heavily on the members as well as the leadership.

Leaders face many challenges in successfully managing teams:

- Hiring the right people and bringing them up to speed and ensuring productivity plays a key role.
- Diversity brings in a lot of advantages to a team. However, diverse backgrounds and beliefs can pose challenges.
- Keeping employees motivated constantly is one of the most sensitive and difficult tasks. Another important factor is to build and maintain trust with employees.
- Along with the organization's goals, a leader has to focus on employee development and balancing both requires a certain skill.
- Organizational changes bring in uncertainty and might also impact the focus and motivation of employees. Handling this is important for a leader.
- Technology changes are constant and it brings up issues of skillset mismatch. It also results in attrition as people might look at opportunities to work on newer technologies.
- Changes in trends in the job market impact employee retention which is one of the toughest challenges to face as a leader.

## 2 The Approach

While there are a lot of challenges in managing teams and ensuring success, there are best practices or recommendations that help address them. Each team and each team member are unique, and their challenges are also unique. Hence, the same approach might work well in some cases whereas it might need changes in other cases.

In this section, a common set of guidelines or recommendations are described which match many scenarios but minor tweaks might be needed based on the dynamics of the organization. This paper is based on the experience of the author in managing teams as well as some of the best practices learnt from other leaders and research and analysis.

### 2.1 Hiring

*"Hiring the best is your most important task"* – Steve Jobs, Former CEO of Apple (Jobs, n.d.)

Building a team starts at hiring. Hiring the right resource with right attitude and skillsets is crucial. Even one wrong hire can impact the entire team. So, this is an important process and the first steppingstone to success.

Some important aspects to be considered in hiring are:

### 2.1.1 Attitude

*"Ability is what you're capable of doing. Motivation determines what you do. Attitude determines how well you do it."* – Lou Holtz (Holtz, n.d.)

This quote about attitude speaks volumes about its importance in successful people. The same applies to hiring as well. Looking for people with the right attitude goes a long way in ensuring success of the team. Technical skills, communication skills, experience, problem solving skills etc. are quite important while looking for a right fit, but the attitude of the person is even more important. An intelligent, skilled and experienced person might not benefit the team if attitude is not right. On the other hand, if the person has a great attitude and there is a slight shortfall in the other hiring criteria, it would be fine to go ahead with the hire. With some learning and hard work, the person can bridge the gap soon.

The responsibility of a leader is to look for people with the right attitude.

In my experience: Conducting interviews for more than 10 years, I have seen instances where technically strong candidates with a wrong attitude failed to add value to the group and on the other hand, new hires who had decent technical knowledge but had the right attitude proved to be an asset to the team.

### 2.1.2 Inclusive hiring process

Diversity and Inclusion are two important aspects while building teams. Diversity ensures different experiences, skills, knowledge and thought processes are brought to the table. Diversity maximizes productivity, creativity and loyalty within teams.

To hire the right talent, the leader should ensure:

- Hiring process needs to be inclusive and inculcate diversity.
- Starting with the job advertisement, the content should not offend any group of people and at the same time should attract diverse talent.
- The interview panel should also be diverse. There should be an adequate representation of all diverse groups within the organization. This also plays a major role in attracting diverse talent.
- The interview and evaluation process should be totally unbiased and should not favor any group(s).
- The compensation and benefits should not have any discrimination as well.

### 2.1.3 Employee referrals

It is a proven fact that one of the most effective ways to hire is through employee referrals. Employees know the pulse of the organization, many times they know the requirements of the hiring group and more importantly might have their friends and ex-colleagues with related skill sets. The other advantage of referrals is that it helps save costs incurred with engaging hiring agencies.

Whenever there are job openings created within the team, the first people to be informed should be the members of the team itself. They know best about what kind of people are required in the team, both in terms of skills and attitude. And people within the team are also mindful of the fact that the people they refer would mostly end up working with them in the future. They naturally look for the best candidates. In this way, the understanding within the team would also enhance and help the team work as one unit. So, the leader should encourage referrals.

In my experience: Most of the successful hires have been through employee referrals, many times from people working within the same team. There have been instances where we have found the right candidate within few days of opening the requisition, all due to excellent employee referrals.

Statistics: Only 25% of workers recruited through job boards stay for more than two years, whereas 45 percent of employees obtained through employee referrals stay for more than four years. (Apollo Technical, n.d.)

## 2.2 Build Trust

*"A team is not a group of people who work together. A team is a group of people who trust each other."* - Simon Sinek (Sinek, n.d.)

Bonding within any team is built on trust. Transparency, trust and mutual respect are like three interlinked pillars. Even if one of these links is broken, the team would be destined to fail.

It is important for the leader as well as team members to spend significant time establishing trust. The leader leads the way here, by propagating the idea of trust and taking measures to ensure the team starts trusting the leader as well as themselves. Along with that, the leader should walk the talk: conduct in a way such that the team begins to believe in the leader.

Some aspects to consider in order to build an atmosphere of trust are:

### 2.2.1 Friendly atmosphere

An important aspect in building an atmosphere of trust is to promote a friendly atmosphere without any fear or bureaucracy. When a friendly atmosphere exists, team members can communicate freely, express ideas and collaborate which eventually leads to the establishment of trust.

- Leaders should build a friendly environment within the team through informal meet ups like eating together, celebrations general discussions and so on. In the current scenario where most people are working online, some of these could be done online although not as effectively as in person.
- Leaders should get involved in the informal team activities as well.
- Leaders should be approachable at all times and not be an occasional visitor. The sense of approachability also brings about a friendly vibe.
- Leaders should also structure one-on-one discussions such that topics not related to work also get discussed, maybe a common area of interest or some issue outside of work. These discussions if managed properly build a great rapport and also develop trust.

In my experience: In the time when we used to work from office, I used to regularly have lunch and tea with the team. I participated in the small celebrations and also ensure team outings and lunches happen. I also shared a workstation with teammates instead of sitting in my cube which was separate from the team area. In the online work model, I made sure to organize informal meetups and also find innovative ideas to celebrate. All these really helped me maintain a friendly relationship with the team and also ensure a friendly atmosphere in the group. This helped in the long run.

### 2.2.2 Involvement

Team leaders plays an important role in driving trust through involvement. The leader should be involved in key sessions which could be driven by someone else in the team but the leader's presence in the session gives a reassuring feeling to the employees and gives them more courage to move ahead. This also builds in a trust factor with the leader. Common examples could be, when there is urgent work being done post office hours or an unavoidable weekend activity. The presence of the leader creates a sense of involvement with the team.

The other aspect of involvement is about involving the team in decision making. The leader should take steps to involve the team in discussions and decision-making process wherever applicable. In some scenarios, the leader could involve senior members of the team where involving all members might not be feasible. This creates a sense of involvement in the team and they feel valued. This also enhances the trust factor within the team and with the leadership.

### 2.2.3 Transparency

This is one of the most important aspects of building trust. A leader should be transparent wherever feasible and also encourage the team to practice transparency. Whenever there is information to be shared like management goals, company directives or any other information applicable to the entire team, the leader should communicate in an open manner and also encourage people to raise concerns or questions. When there are decisions made which impact the team, the leader should explain the rationale behind the move in an open team setting, if that is something to be shared. All these actions create a level of trust with the leadership. The leader should also ensure that the team members know what each one is working on and also encourage ideas to do things in a better way.

Another area where the leader should exhibit transparency is in the performance feedback or one-on-one sessions. The leader should give candid feedback and also seek suggestions and feedback from the employee. The goals should also be set clearly and reviewed from time to time. The leader should not collate all feedback towards the end of the appraisal cycle, rather give them as and when it is needed. This instills transparency and trust.

## 2.3 Always in sync

*"No one can whistle a symphony. It takes a whole Orchestra to play it."* – H E Luccock (Luccock, n.d.)

It is important for the entire team to be in complete sync with what is going on with the deliverables, the company and the team. Only then will the team be able to work well together and deliver success. It is not enough if the team is in sync, its leader should also be in sync:

- Attend the standup meetings regularly to be updated about the day-to-day work that is in progress. The leader should be careful not to get into a micro-management mode and that would not be scalable as well. However, emphasis could be on providing guidance wherever needed and helping in resolving impediments. The main job of a leader should be to facilitate the team to work better without distractions.
- Attend any other team discussions as much as possible to stay up to date.
- Regular one-on-one discussion with team members which should cover the regular work as well as the employee development aspects.
- The leader should connect informally with the team, join them in informal chat sessions, lunch etc. This persistent connection creates a level of comfort and that enables team members to discuss openly with the leader.
- The leader should be well versed with the work done by each team member. The daily sync up and other meetings should give vital inputs. The one-on-one sessions also help understand the work done. The leader should be able to assess the performance of each employee even before the yearly performance review discussion with the employee.
- The leader should also ensure that the team spends time on team building and fun activities. That keeps them energized and engaged.

All these measures not only improve the bonding between team members but also ensure a good rapport with the leader. This goes a long way in the success of the team.

In my experience: As a leader I try to attend standup meetings as much as possible. In addition, I have regular sync up meetings with the team trying to always stay in sync with the happenings. Regular one-on-one meeting also helps me stay in sync with what each team member is working on, their concerns, their feedback etc. As a result, I end up reminding many team members about achievements they missed mentioning in the performance review discussions. This has a positive influence in my relationship with them and also elevates the trust between us.

## 2.4 Ownership

*"Responsibility equals accountability equals ownership. A sense of ownership is the most powerful weapon a team or organization can have"* – Pat Summitt, Former Tennessee Women's Basketball Coach (Summitt, n.d.)

An efficient leader delegates work to team members. A smart leader delegates ownership to team members.

If team members own a part of the product or deliverable, it drives a sense of responsibility and accountability and eventually keeps them motivated. This approach makes each team member think of their area as an owner, devise plans and strategies, drive the execution, showcase the results and constantly try to improve. This will not only result in team excellence but also enable the leader to concentrate on other strategic aspects to take the team to greater heights.

Some key aspects to consider here:

### 2.4.1 No micromanagement

One of the most important guidelines for a leader is to avoid micromanagement. It is tempting to get into the finer details of each work happening within the team and dictate terms. That gives a deep sense of control and also makes the leader feel valued. But that is like slow poison, for the leader as well as the team.

Micromanagement makes the team feel less valued. It brings in a feeling that the leader does not trust the employees and their work. It makes the team members restrict themselves to whatever the leader asks them to do and in the longer run, it leads to frustration, stagnation and eventually attrition.

Micromanagement also has a negative impact on the leader. It consumes a lot of time and can also drain mentally. This means that the leader would not have much time to think beyond the day-to-day work which can hinder the growth of the team as well as the leader. It also diminishes the trust factor.

Leaders should drive the ownership model in the team culture, encouraging each team member to own and drive initiatives.

This approach releases a lot of time for the leader which can be used in strategic thinking and decision making for the team and project. That also enables leaders to scale up and start moving towards higher levels in their own career. It also builds in a deep sense of passion and commitment within the employees eventually leading to consistently successful teams.

In my experience: Each member in my team owns one or more components depending upon their experience and capability. Whenever a new member joins the team, the natural progression path set for them is to start owning at least one area or component within 3-4 months. Regular sync up with the owners helps me stay up to date and also ensure things are moving as per plan. This approach has helped all team members grow in their career and also helped me grow.

### 2.4.2 Ensure visibility

Another important aspect of driving the ownership model is to ensure visibility in two ways:

- Along with delegating ownership of components or projects to team members, it is very important to give them the visibility of the impact of their work on the organization's goals. They should understand how their work is directly or indirectly contributing to the organization's goals. This gives them an idea of the importance of the work they own and also drive commitment and responsibility.
- The other aspect of visibility is to provide opportunities for team members to showcase their work in larger forums like demos to executives or demos at organization level. Wherever possible, they

should represent the work they own such as sending status reports or presenting in meetings. This will not only give them the experience of bigger forums but will also give others visibility of the work and ownership that the team members exhibit. This will increase their credibility at the organization level and eventually contributes to their growth.

### **2.4.3 Build next level leadership**

*“Leaders don’t create followers, they create more leaders.”* – Tom Peters, author, speaker, management expert (Peters, n.d.)

One key aspect of leadership is to build successors. A leader should always have the next level of leaders groomed and ready to take responsibility in absence of the leader. When a leader is on vacation, or otherwise occupied, the next level leaders should be capable of handling the situation and continuing the work. This can only happen if the next level leaders are identified, groomed and given ownership with the freedom to execute and take decisions.

Some key things to do as a leader:

- Identify people in the team who are passionate, responsible and capable of managing additional responsibility.
- Groom them by providing them opportunities to drive small projects or own components.
- Make them visible outside the team, showcase them as owners or SMEs.
- Keep in constant touch with the next level leaders, have frequent discussions, discuss strategy, consult them on decisions etc.
- Encourage them to find their way of driving things, do not restrict them.

Last but not least, a leader should be selfless. The leader should always think beyond oneself, about their successor even in the event of the leader leaving the organization. A successful leader always prepares the next level of leadership in such a way that when the leader leaves the organization, the next level can continue the work with minimal or no disruption.

## **2.5 Freedom**

*“The ultimate freedom for creative groups is the freedom to experiment with new ideas.”* – Daniel H Pink (Pink, n.d.)

The sense of ownership among team members eventually requires freedom for them to experiment and drive in their own way. It could be freedom in technology, process and even some individual aspects. The leader can show the way and the employee should think and devise the plan to achieve it, manage the execution and own the result. Leaders should always be available for any kind of guidance but should not interfere in the way unless something is terribly wrong.

This approach has a certain level of risk, there might be small failures or delays as people learn to drive things in their own way. The leader needs to support the employees so that the employees can think and experiment without fear of failure.

Some key aspects to consider here:

### **2.5.1 No Fear of failure**

Fear of failure constrains creativity. They hesitate to try something new and always try to stick to the same old methods to get the work done. Imbibing freedom to think and experiment brings out creative ways and many times generates amazing results. There may be some cases of failure, but the leader can step in at the time and guide.

The most important thing is that in failure scenarios, the blame game should never happen. When there is an issue, the primary focus should be on how to fix the issue rather than trying to find the person at fault.

Even after the issue is fixed, there should be an analysis on cause of the issue with the intention of preventing that from happening in the future. The moment a culture of blame game creeps in, people will get defensive and fear to take up responsibilities and that hinders the success of the team.

*In my experience:* I have seen leaders on both sides. I have worked with leaders who never pinpointed at anyone in case of a failure but brought together the whole team to work out of the issue. Those leaders went on to create successful teams and rose to higher positions in their career.

On the other hand, I have also come across leaders who always tried to achieve perfection by driving in the fear that a particular person or group would be responsible in case of issues. And in such situations, they even went to the extent of placing the blame on people and acting against them. Such teams have always diminished, and such leaders faced more and more of such issues in future as well.

## 2.5.2 Employee Leaves

Although this topic might seem trivial, it can have serious implications. Employees earn their leaves and possess the right to avail them.

As a leader, employees should be given the freedom and responsibility to decide on their leaves. Eventually, if any employee is prevented from taking leave, they would definitely not have the concentration and involvement at work and would eventually lose the motivation. Instead of this, allowing them to take leaves allows them to address their priority and come back more committed and motivated to complete the work on hand.

There might be occasional delays that occur due to leaves taken but that should be managed by the leader. The long-term benefits of providing freedom in this aspect is far more rewarding than the short-term drawbacks.

## 2.5.3 Flexibility

This is another important aspect, especially after the COVID pandemic. The pandemic introduced the unknown concept of complete work from home. This brought up many challenges and one of them was on work-life balance. People had to manage their personal work and their professional work such that neither suffered. At times, personal work had to be addressed in between office work sometimes leading to conflicts.

Flexibility is the key in such situations and even in the work from office situation as well. Again, this is linked to ownership and freedom. Leaders should drive in the concept of owning the work and delivering it as per schedule. But how they pace their work, what timings they follow etc. should be left to the discretion of the individuals. Each individual might have their own work patterns and leaders should allow them the freedom to follow. However, the leader should find ways to ensure a minimal overlap such that all team members could attend the important discussions.

*In my experience:* I have practiced the above approach towards leaves and flexibility in my teams and it has provided amazing results. In some rare occasions, there were delays but that was something I handled as a leader, sometimes seeking additional time and sometimes seeking help. But this approach has inculcated so much commitment and motivation in the employees that in times of crisis, they have worked extra time and extra days to meet the deliverables. And the most important thing is that even after all the extra time, they ended up being even more motivated and engaged.

## 2.6 Career Path

Organization goals, strategy, teamwork, leadership etc. is quite important to make teams successful but employees' career path and growth is equally important. Employees who have a clear career path remain engaged for a longer time. Along with organizational goals, achieving career goals of employees is equally important.

Some key aspects to consider here:

Excerpt from PNSQC Proceedings

Copies may not be made or distributed for commercial use

### **2.6.1 Career path of employees**

A leader should balance organizational goals and employees' career path. The career path of an employee should be in sync with the organization's strategy or roadmap and at the same time should enable the employee to gain and improve skills to grow in the direction of their choice.

Clarify the employee's long-term goals or plans. The leader should discuss with the employee on how to move ahead. The analysis of where the employee stands at present is important. Based on that, the leader can help the employee devise a career growth plan to reach their intended goal.

Another thing that the leader should do is to identify a suitable mentor for each employee to take them along in their career path.

### **2.6.2 Higher education**

Another important aspect of career growth is higher education or certifications. Leader should encourage employees to explore new certifications or higher degrees that they could pursue to grow in their career. Organizations might have policies to provide financial assistance to employees to pursue their education. Leaders should be aware of this and encourage employees to make good use of the facility. Mentors would also add a lot of value in this aspect. They can guide their mentees on what new certification or higher education to pursue.

Higher education helps employees grow in their career, gain confidence and contribute positively towards the growth of their team and organization. Such employees are also more likely to be motivated and loyal towards their organization. All this in turn results in higher employee engagement.

### **2.6.3 Alternate career opportunities**

However good a leader is, a situation might arise where they might have to make compromises to achieve something good. Leaders might do everything to keep their employees happy, engaged and motivated, but situations might arise when some employees or their aspirations might not fit in the team. Situations such as:

- Employees might desire a role which is not available in the team
- Employees might want to switch to a different career path
- Employee's skills might not be needed in the team or suitable work might not be there

In such situations, trying to keep the employee within the team impacts the employee's career and aspirations and over a period of time the employee loses motivation and quits. In such circumstances, the leader's efforts should find a suitable role elsewhere in the organization that might suit the employee's desire and should also recommend for the role. It is better to lose an employee within the organization than to completely lose the employee from the organization.

This might be a compromise or loss in terms of the leader's team but, it is a win for the organization and the employee.

*In my experience:* I encountered one situation where I could not offer a role that one of my team members desired because the role did not exist in the team. Since the employee had trust in me, the employee approached me with an open position in another team and the role offered was the role the employee desired. I helped the employee prepare for the role and also recommended the employee to the hiring manager. That helped the employee advance in his career and at the same time the organization benefited from the change. Infact, my trust level with the employee moved up.

## **2.7 Leadership**

*"Leadership is the ability to guide others without force into a direction or decision that leaves them still feeling empowered and accomplished." – Lisa Cash Hanson (Hanson, n.d.)*

A leader is as important as the team itself. Leadership has a major say in how the team performs. Although a good leader cannot make the team succeed without the team's commitment, a good leader can facilitate, inspire and guide the team to that level of commitment. Hence the famous quote: *People leave managers, not companies*".

### **2.7.1 Facilitation**

Leadership is not about controlling, it is about facilitating. One of the main responsibilities of a leader is to ensure that the team can do their work seamlessly.

- The leader needs to handle the dependencies and roadblocks that the team might face and work to get them resolved. Sometimes, these could be people or company related as well.
- The leader should work on getting more clarity on the scope of work and first see if that can be taken up later. Similarly, scope creep should also be handled.
- Leader should handle the communication and set realistic expectations with external groups.

Organizing skills, negotiating skills and communication skills of the leader come in handy while facilitating for the team.

In my experience: When I became a manager, one of my old managers called me and congratulated on my promotion. The first thing he said was that a leader should always be part of the team, never should the team feel they are under their leader. The leader should always work for the team, the team should never feel that they work for their leader. This advice has helped me throughout my managerial journey.

### **2.7.2 Do not dictate**

One key point to remember for a leader is that leadership is not about authority, it is about being humble, empathetic and tactful. Leaders should never think of the team members as subordinates, rather the leader should be one among the team. That is when the team members feel comfortable to work with the leader and share their candid feedback, thoughts and concerns.

Rather than communicating decisions to the team, the leader should involve the team or senior team members in decision making whenever possible. This gives the leader a broader idea pool to consider and the team also feels included and valued. This enhances the commitment of the team and inspire them as well. This would eventually result in success of the team and deliverables.

In this aspect, a leader's patience, listening ability, inclusive nature and humility are put to test.

### **2.7.3 Managing conflicts**

Conflicts occur everywhere, even in well-oiled teams. Infact, a certain degree of conflict is healthy for any team as long as it is constructive and within control. Conflicts bring out new ideas and, in some cases, avoid mistakes. Healthy conflicts within a team indicate the freedom of thought in the team.

- The leader should be well aware of conflicts in the team and intervene before it gets bad.
- If the conflict is between team members, the leader could mediate as a neutral person, avoid bias and maintain confidentiality.
- If any team member has a conflict with the leader, the leader should first try to understand the concern, listen to the employee with an unbiased mind and if needed seek help of other leaders to mediate.
- In any case, the leader must ensure to the employees that raising concerns is rightful and they should not hold back any concerns. This will happen only if the leader has established adequate trust with the team and also established credibility.

Here, the leader's patience, morals, conflict resolution techniques and unbiased decision making are key.

#### 2.7.4 Reward and Recognition

This is important for any team. Recognizing and rewarding deserving people goes a long way in motivating them. Motivated and self-inspired team members go out of their way to deliver the desired outcome. Such people also come up with new ideas.

Team leaders play an important role in recognizing and rewarding team members:

- Needs to identify the deserving candidate. Not rewarding a deserving candidate can be disastrous but rewarding an undeserving candidate can send the wrong signals.
- Needs to reward at the right time. Delay might demotivate people.
- The leader should be in sync with the team's activities.
- Some rewards which should be kept confidential, like a spot bonus or a stock grant. In such cases, the leader should ensure the confidentiality is maintained
- Sometimes, normal reward process might not be possible due to budgetary constraints or maybe the contribution might not be big enough to qualify for a normal reward. The leader should come up with creative ideas to reward and recognize in such scenarios like granting a special leave, face time with executives, taking out for lunch and so on.

The leader's creativity and transparency are key traits needed here.

*In my experience:* Our company has a peer recognition program where each employee gets some reward points to give to peers for good work done. Those points expire each quarter. So, I started a recognition meeting towards the end of each quarter where each team member lists out the names of peers who helped them or overachieved. And if they wish, they could also reward them with some points. This received a positive response and team members feel good about this exercise.

### 2.8 Celebrate

*"Every win is a celebration not because it's a win but because of all the hard work that went behind it."*  
(Pinterest, n.d.)

All work and no fun would make anyone dull and sad. It is the same with teams. However, engaged or motivated a team is, lack of fun at workplace eventually begins a downward trend. Fun and celebration at workplace are important and as a leader, the emphasis should be to continue that even in critical situations. It is easy to shrug off the topic saying that everyone is too busy and there is no time for fun. But that can easily become a habit and excuse to defer fun activities. As a leader, one should make sure that does not happen. Leaders should not only worry about the health of the project but also about the well-being of the employees. And fun and relaxation is the step in the right direction.

#### 2.8.1 Fun in Day-to-Day activities

It is true that parties and celebrations cannot happen every day and it should not happen as well. Those lose their value if they are done every day. But there are other ways to bring life into the day-to-day schedule like

- Go together as a team for lunch. Make sure nothing work related is discussed there.
- Take coffee breaks as a team whenever possible. Again, keep it informal
- Meetings can begin or end with a light element to relax the group
- As a leader, make sure you are informal in interactions with the team. Make them feel comfortable and that help relax them.
- Find innovative ways to give relaxation time, like a short walk, or a short exercise break, for example.

One thing to keep in mind is that such relaxation should not be seen as a waste of time. Infact, a little bit of relaxation can bring about new ideas and new energy into people.

## 2.8.2 Celebrating occasions

Occasions are not to be missed. Be it a birthday or work anniversary, it deserves a small celebration at least.

- Celebrate on the same day as much as possible.
- Maintain consistency so that there is no feeling of differentiation.
- Celebrate larger occasions with an outing or lunch/dinner.

In the post-pandemic world where many people work remotely, such celebrations could be done online, like a video call. If the company policy allows and it is safe, team outings or lunches could be organized occasionally.

These celebrations not only add fun into the stressful work-life but also bring people together. In such occasions, the bonding between team members improves thus contributing to team's success.

*In my experience:* In the two years of pandemic, we figured out innovative ways to celebrate. For Diwali which is a major festival in India, we shipped dry fruit boxes to each employee in the group. And for major work anniversaries like 5, 10 or 15 years, the leader sent sweets or chocolates to those employees with a Thank You note. Although this was work to organize and execute, the end result was pleasant. Everyone on the team liked the gesture.

## 2.9 Separation

Even the best of leaders have to face attrition at some point in time. However, the important thing is to positively utilize the situation. Although a member leaving the team is a negative development for the team, there are a few positives that can come out of these circumstances as well. As a leader, staying calm and composed in these situations is quite important.

### 2.9.1 Seek feedback

As a leader, the first thing to do when you learn of a resignation is to talk to the person. Try to find the reason behind the move and try to find what change might help retain the employee. In most situations, retaining an employee is more profitable than hiring a replacement.

The trust factor plays a huge role here. If the employee has trust in the leader, some of these conversations would have happened during a one on one itself. This conversation would not happen after resignation in most circumstances.

Anyways, such a conversation can help retain the employee, especially when the employee is looking for a different role or has an issue at work. In cases where compensation is the reason, this conversation could help gather the information. Then, depending on the ask and feasibility, an offer could be made.

### 2.9.2 Try to find another role

In situations where an employee might be looking for a different role or different kind of work, the leader can think whether that alignment could happen within the team itself. If the leader and employee have a good trust relationship and regular one-on-one meetings, such requests would have come up there itself.

In some situations, the role or type of work desired by the employee might not be feasible within the team. Then, the leader should look for alignment elsewhere within the organization. Although the team might lose a resource, the organization benefits.

### 2.9.3 Offer help

When all efforts to retain the employee fail, the leader can offer help to the employee with the next job. It could be like providing a recommendation, or providing a reference through contacts, offering guidance

for the new role and so on. Although this might not benefit the organization in any way, it enhances the trust factor which might come in handy in future situations when the employee might look for a change.

#### **2.9.4 Blessing in disguise**

In few situations, the resignation of an employee might come in as a blessing in disguise. There might be employees who are not performing well or they might not be engaged with the team. Despite many efforts if the behavior does not change, it might not be in the best interest of the team. At some point in the future, such employees might go through a forceful exit. But, if such employees decide to quit on their own before that, it benefits the team. One classic example of how a negative development could sometimes prove beneficial.

*In my experience:* One employee in my team decided to quit to explore a new role. Despite many efforts to retain, the employee decided to go ahead with the decision. But the employee had a strong trust relationship and was leaving on a positive note. Few months later, the person reached out to me and expressed interest to come back into the company and the team in particular since the new job was not as per expectations. This turned out to be a win-win situation for both.

### **3 Exceptions**

The approach described above covers majority of situations that occur in teams. However, outliers will always be there. Leaders have to consider those outliers and devise innovative approaches to tackle them. Following is a list of exceptions that could occur but however this is not a final list. It could differ according to the scenario.

- Micromanagement in crisis: Though it is always advisable to give a free hand to team members to own the product/component and drive independently, there might be situations where leaders have to intervene and ensure that the deliverables are met. It could be an urgent customer escalation or a critical time bound activity. In such cases, the leader would have to be more involved and track the minute details to ensure the expectation is met. One or more team members could be given responsibility to drive it but the leader should also follow up closely and micro-manage to some level.
- Abuse of Freedom: Though it is recommended to allow freedom to team members to drive their owned components, there could be rare situations where team members could misuse the freedom and negatively impact the deliverable. If the leader is in constant sync on all the work in the team, intervention is needed in such situations and corrective action needs to be initiated by the leader.
- Retention only as needed: Though leaders should try to retain employees, this might not apply to some cases in attrition scenarios. There might be members who might have performance issues or might not be a good fit in the team. Despite repeated feedback, if the behavior continues then retention of such employees might not be desirable.
- Reward judiciously: Just like not rewarding is dangerous, over-rewarding or wrong rewards might be even more dangerous. So, the leader should be vigilant to ensure that the deserving people get rewarded appropriately at the right time. Any reward or recommendation that the leader feels might be inappropriate should be brought to a halt.
- Transparency / confidentiality: Although transparency in the team and transparency from the leader is important, there are situations where confidentiality has to be maintained. Like performance and compensation related topics, some company confidential topics like strategy organizational changes, important decisions and so on. In such situations, the leader should be careful not to violate confidentiality and share information only on a need-to-know basis.
- Hiring through referrals: Though hiring through referrals is mostly beneficial, it could at times lead to groupism, especially if the referral is from an existing team member. In such situations, gauging the attitude of the candidate, the intentions of the referring team member and team dynamics becomes important. If the leader feels that the team could be negatively impacted due to the referral, it should be rejected or not considered for hiring.

- Pandemic changes: All the concepts discussed consider normal work environment, however in some situations the effect of the pandemic is also considered. As the pandemic evolves, there might be changes to the approaches and that should be taken up on a case-by-case basis. And some of these might be company specific as well.
- Company policy: Although majority of the topics discussed here apply to all companies and industries, there might be exceptions in certain industries which have to be considered on a case-by-case basis. For instance, the work environment, freedom and transparency in a military defense organization might be more rigid and the approaches have to be altered accordingly.

## 4 Conclusion

Managing people is an art as well as a science and it requires patience, sensitivity and creativity. Challenges differ from team to team and from organization to organization. Accordingly, leaders should adapt and innovate new ways to tackle the situation and ensure success. This paper is an effort to highlight key aspects to consider while managing teams. This is just a framework that leaders can use, improvise and achieve success within their groups.

## References

1. (n.d.). Retrieved from Apollo Technical: <https://www.apollotechnical.com/employee-referral-statistics>
2. (n.d.). Retrieved from Pinterest: <https://www.pinterest.com/pin/110197522102594157/>
3. Hanson, L. C. (n.d.). Retrieved from Ask Ideas: <https://www.askideas.com/leadership-is-the-ability-to-guide-others-without-force-into-a-direction-or-decision-that-leaves-them-still-feeling-empowered-and-accomplished-lisa-cash-hanson/>
4. Holtz, L. (n.d.). Retrieved from Brainy quote: [https://www.brainyquote.com/quotes/lou\\_holtz\\_450789](https://www.brainyquote.com/quotes/lou_holtz_450789)
5. Jobs, S. (n.d.). Retrieved from Quotesgram: <https://quotesgram.com/quotes-about-hiring-people/>
6. Luccock, H. E. (n.d.). Retrieved from CCSENet: <https://www.ccsenet.org/journal/index.php/ass/article/view/9394>
7. Peters, T. (n.d.). Retrieved from Get Light House: <https://getlighthouse.com/blog/people-leave-managers-not-companies/>
8. Pink, D. H. (n.d.). Retrieved from Quotefancy: <https://quotefancy.com/quote/1536890>
9. Sinek, S. (n.d.). Retrieved from Quotefancy: <https://quotefancy.com/quote/1415610>
10. Summitt, P. (n.d.). Retrieved from Quote Master: <https://www.quotemaster.org/employee+ownership>