Evolution and Future of Software Testing

Mesut Durukal

durukalmesut@gmail.com

1 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.

2 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.

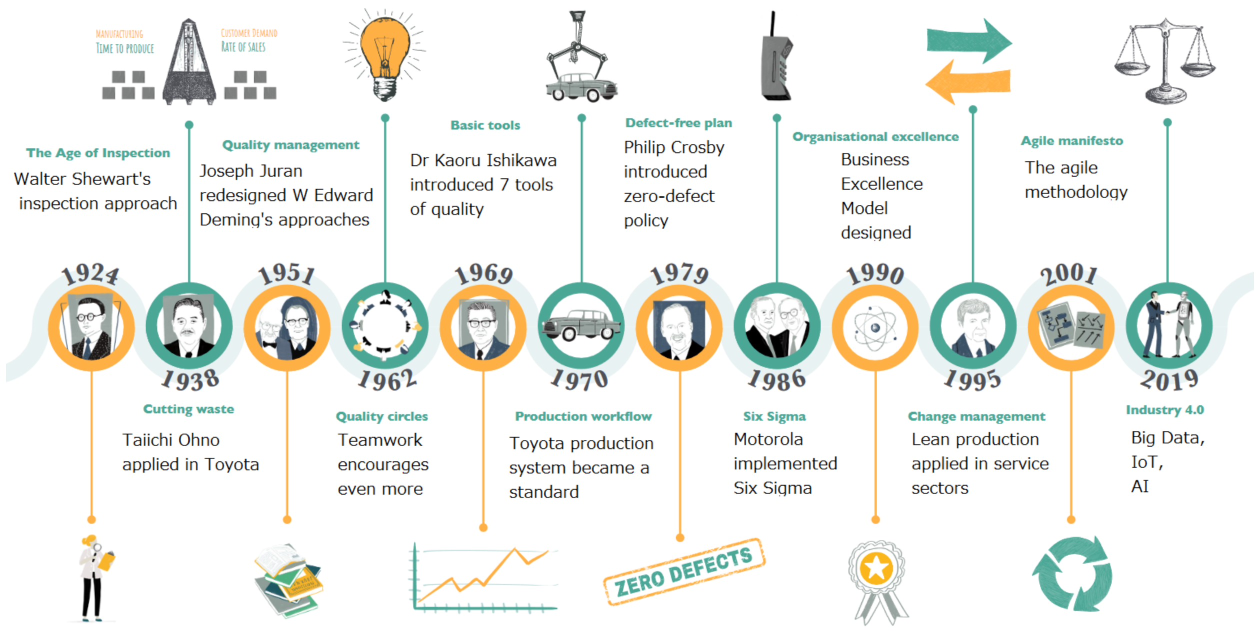
3 Introduction

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.

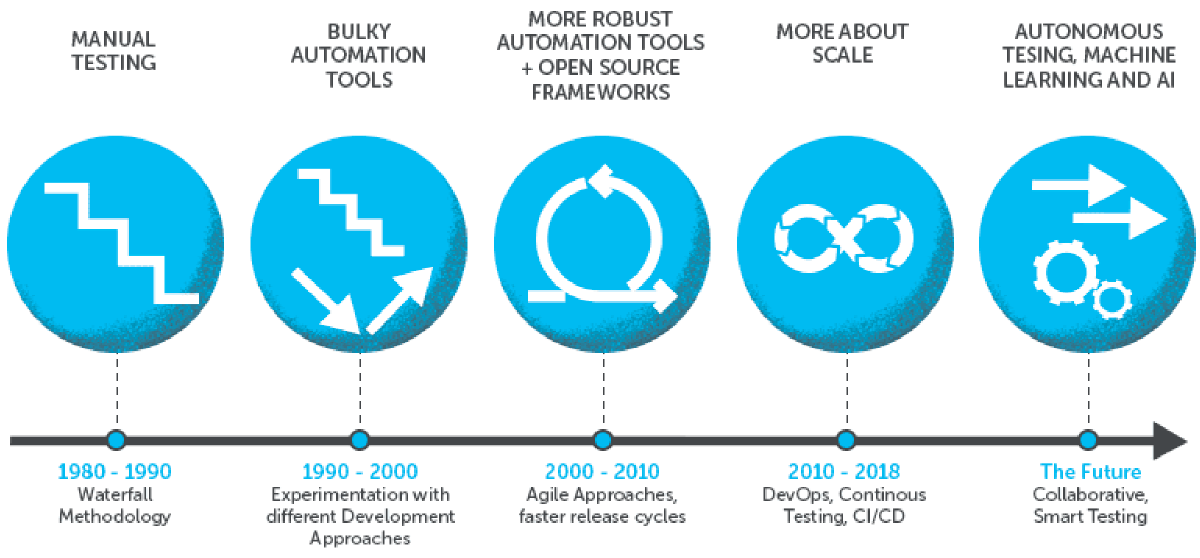
To cope with all these difficulties, various improvements are done in quality assurance processes. A brief summary of the last 100 years milestones was presented in the World Quality Day [1] as depicted below.


MANUFACTURING
TIME TO PRODUCE
CUSTOMER DEMAND
RATE OF SALES
The Age of Inspection
Walter Shewart's inspection approach
1924
1938
Cutting waste
Taiichi Ohno applied in Toyota
Quality management
Joseph Juran redesigned W Edward Deming's approaches
1951
1962
Quality circles
Teamwork encourages even more
Basic tools
Dr Kaoru Ishikawa introduced 7 tools of quality
1969
1970
Production workflow
Toyota production system became a standard
Defect-free plan
Philip Crosby introduced zero-defect policy
1979
ZERO DEFECTS
1986
Six Sigma
Motorola implemented Six Sigma
Organisational excellence
Business Excellence Model designed
1990
1995
Change management
Lean production applied in service sectors
Agile manifesto
The agile methodology
2001
2019
Industry 4.0
Big Data, IoT, AI

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.

4 Evolution of Software Testing

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


MANUAL TESTING
BULKY AUTOMATION TOOLS
MORE ROBUST AUTOMATION TOOLS + OPEN SOURCE FRAMEWORKS
MORE ABOUT SCALE
AUTONOMOUS TESING, MACHINE LEARNING AND AI
1980 - 1990
Waterfall Methodology
1990 - 2000
Experimentation with different Development Approaches
2000 - 2010
Agile Approaches, faster release cycles
2010 - 2018
DevOps, Continous Testing, CI/CD
The Future
Collaborative, Smart Testing

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.

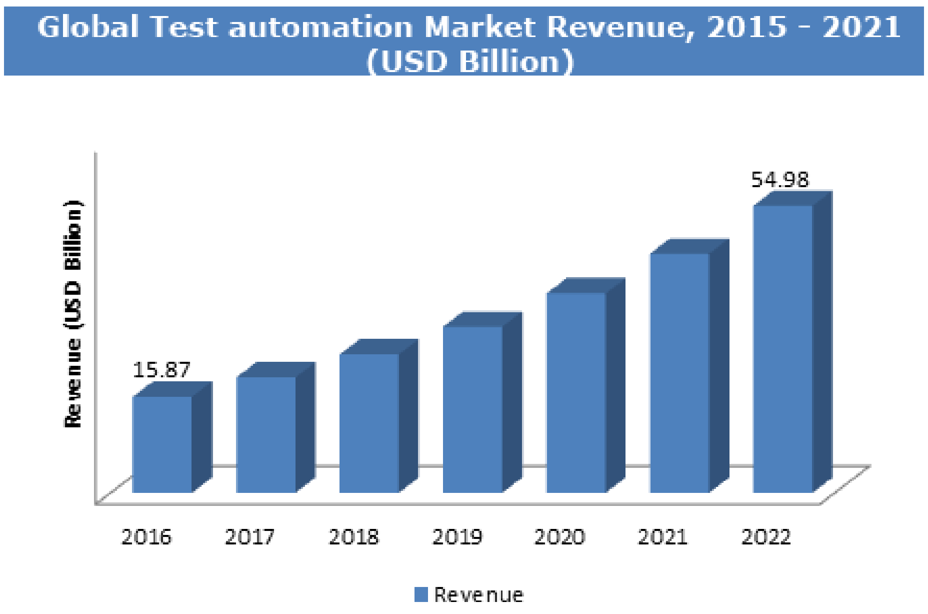
4.1 Test Automation

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.

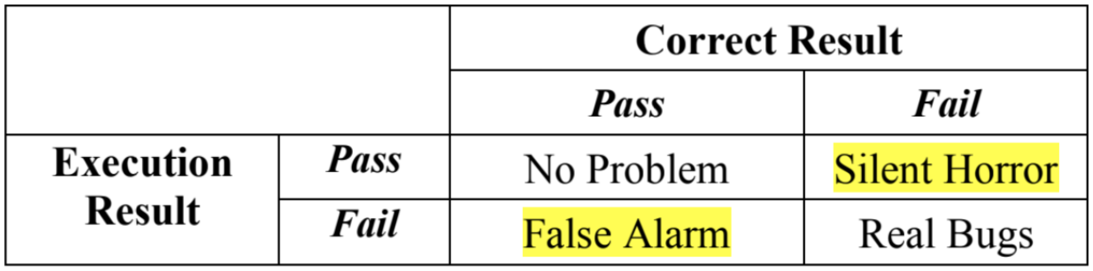
Coming to execution duration, no one would argue that machines outperform humans in calculating complex problems. Executing a single test case by a human may take minutes where it can be executed by automation in seconds. Moreover, we can run a lot of tests in parallel on machines with the help of automation.

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].



		Correct Result	
		Pass	***Fail***
Execution Result	***Pass***	No Problem	Silent Horror
	Fail	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

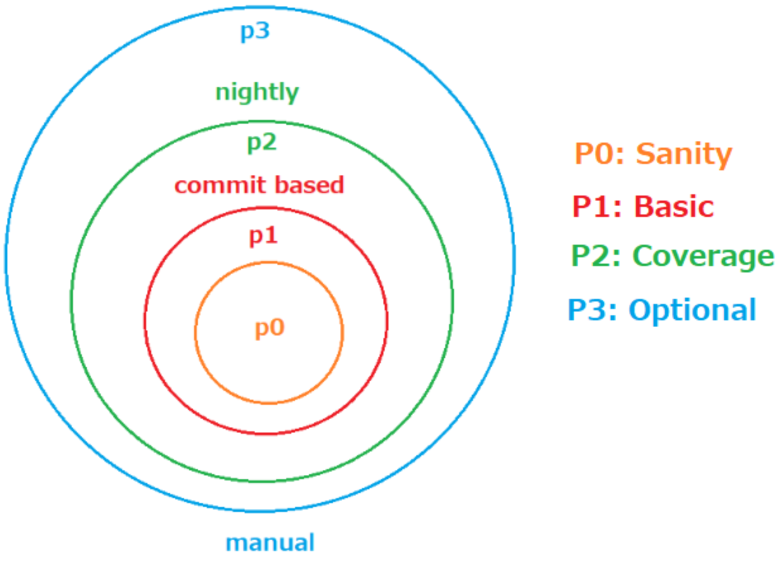
4.2 Agile Testing

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.

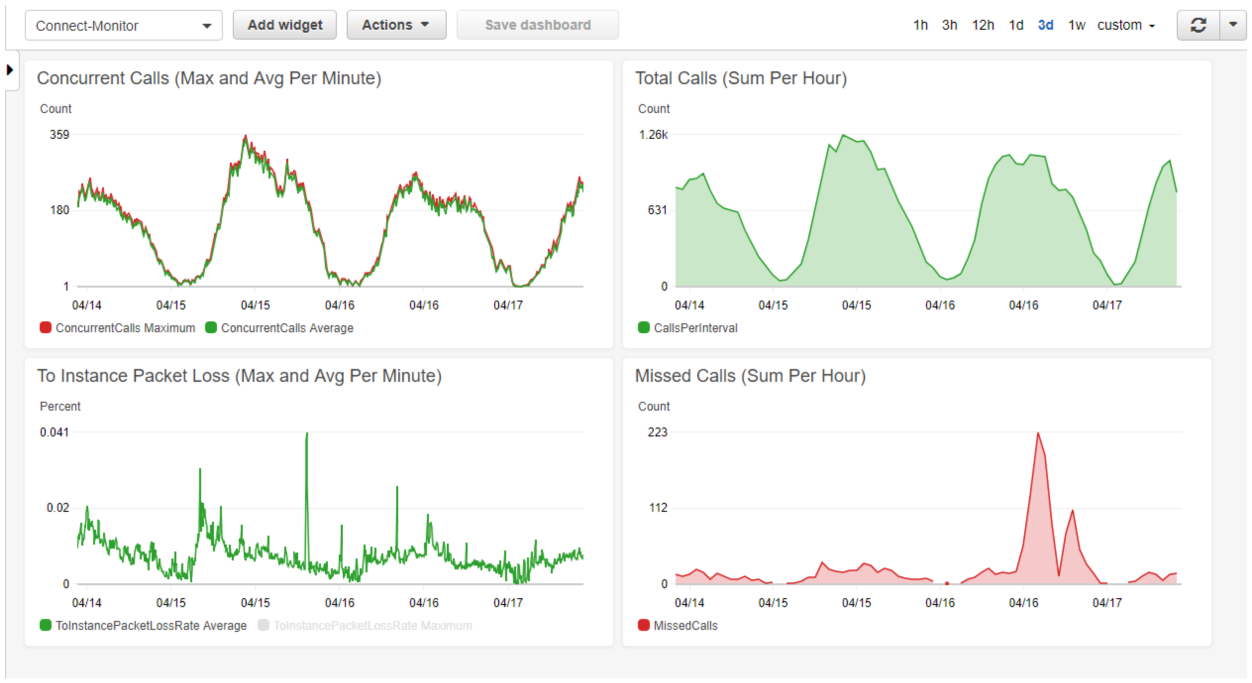
4.3 Continuous Testing

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.

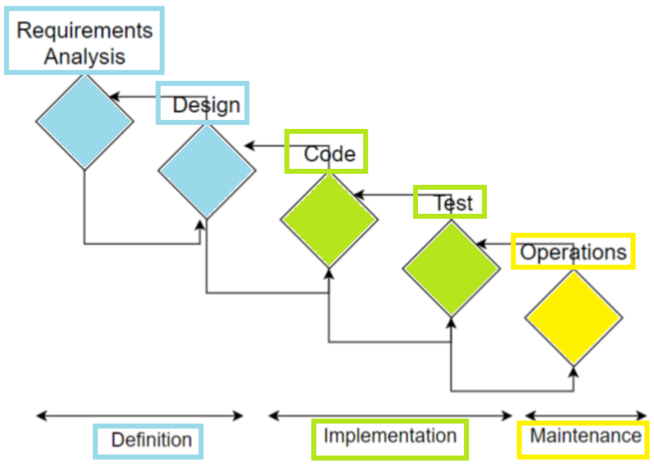
5 Machine Learning in Software Testing

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.

5.1 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.

5.2 Implementation Stage

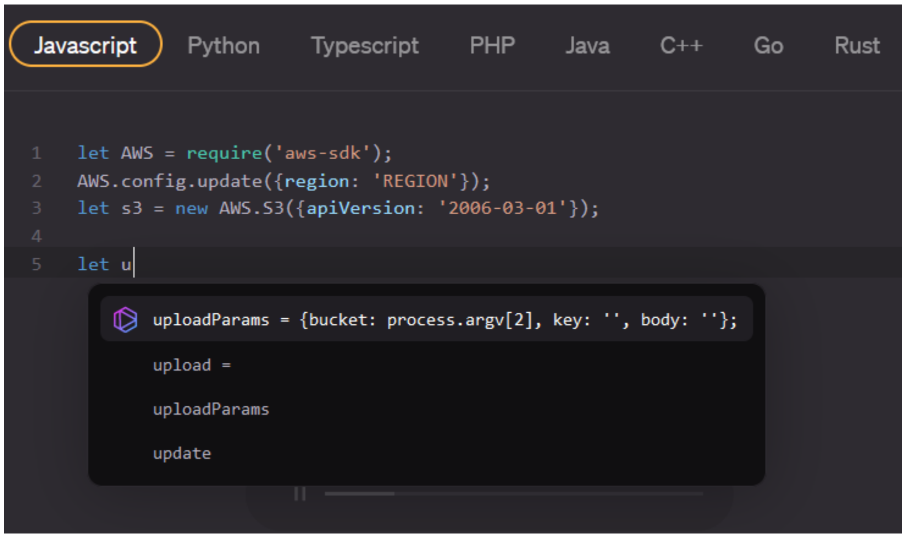
We can also support test implementation with ML. After the problem is defined with inputs and outputs, the needed operations are figured out. DeepCoder [6] follows the same approach. Here is an example of input and output in a scenario, in which negative numbers are filtered and listed in a reverse order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



```
let AWS = require('aws-sdk');
AWS.config.update({region: 'REGION'});
let s3 = new AWS.S3({apiVersion: '2006-03-01'});

let u
```

```
uploadParams = {bucket: process.argv[2], key: '', body: ''};
upload =
uploadParams
update
```

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(elseAndImags,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].

5.3 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:

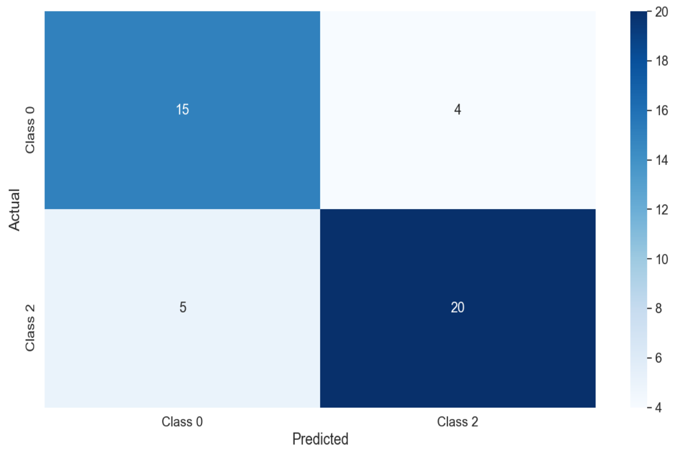
- 889 bugs exported from Jira
- 3 severity levels were defined in our project

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.

6 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.

7 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)