SUP'COM
Higher School of Communication of Tunis

End-Of-Studies Internship's Report

Automated Exception Logs Classification Using Natural Language Processing

Realized By :

Ms. Hadil BEN AMOR

Project Supervisors :

Mr. Ridha BOUALLEGUE (SUP'COM)
Ms. Doniazed BEN NSIR (Cognira Tunisia)

Work proposed and carried out in collaboration with:

COGNIRA

Academic year :
2022 - 2023

Higher School Of Communication Of Tunis
SUP'COM
Technological City of Communications, 2083 - Elghazala - Ariana - Tunisia
Contact. +216 70 240 900 – website : www.supcom.mincom.tn

I hereby authorize **Ms. Hadil BEN AMOR** to deposit her end-of-studies project report.

Academic Supervisor

Ridha BOUALLEGUE

I hereby authorize **Ms. Hadil BEN AMOR** to deposit her end-of-studies project report.

Professional Supervisor

Doniazed BEN NSIR

Dedication

My most heartfelt thoughts go to the ones that matter the most for each one of us. The ones that never fail to surprise you with their endless love:

My parents, who raised me and guided me through my entire journey, who believed in me and never failed to alleviate the burdens of life. You have been my pillars of strength throughout my journey. Your support, and sacrifices have made me the person I am today. I am forever indebted to you for everything you have done for me, and I hope I make you proud in every step of my life. Your oldest daughter, loves you to the infinity and beyond.

My siblings, Moutie, Racil and Ahmed, who made sure to be supportive in every step in my life despite their young ages, I love you so much. Your unwavering love and guidance have been invaluable to me, and I am grateful to have you in my life.

My friends that I cherish the most and for whom I wish all the best in their future endeavors. To everyone who has supported me throughout my journey, your love, encouragement, and kindness have been instrumental in making me the woman I am today. To those who held me back, thank you for helping me grow stronger.

My dear grandparents 'Slima and Ali', who passed away two years ago and I would like to acknowledge their memory. Their love, affection, and prayers have always been a source of strength and comfort to me, and I am grateful for the time we had together. May they rest in peace.

Thank you all endlessly

Acknowledgements

Special thanks goes to Mr. Ridha BOUALLEGUE, my academic supervisor, for his helpful support and oversight, as well as his wise remarks, suggestions, patience and mentorship throughout the synchronisation period of the internship. Also I adress huge thanks to Ms. Doniazed BEN NSIR my professional supervisor, for the help she has given me throughout the internship, for handing me the keys to the professional world and for pushing me forward which lead me to reach my full potentials.

Thanks to all of my professors throughout my schooling. Many thanks goes to Cognira Tunisia team for welcoming me as a member of the team and for giving me a huge added value in such a short notice of four months.

Abstract

When executing test cases, we often encounter some bugs or errors. The latter are traceable through their associated exception logs that need to be classified in order to distinguish the actual bugs from the not actual ones. The objective of this work is to classify the exception logs through natural language processing and deep learning. The developed code will extract the exception logs from 'Allure Report', the graphical user interface component of the framework 'Allure' used by the Quality Assurance (QA) team in visualizing their tests results, after every running build in Jenkins pipeline and explore multiple layers of analysis, delving beyond surface-level categorization through deep classification.

The 'Allure Report' contains logs classified into two main categories 'Product defects' and 'Test defects' and it met the needs in term of broad error classification. However, the limitation lied in refined classification to help the QA team discover the real bugs from the flaky ones. The contribution of this work resides in developing an intelligent solution that classifies the exception logs into different categories, generate Key Performance Indicators (KPIs) that evaluate the accuracy and the dynamic nature of the automated testing process. These KPIs will provide insights into the performance of the automated system, helping to assess its effectiveness and reliability in identifying and classifying the exception logs. Then, a new dashboard will be generated. The newborn visualization tool is named 'QA Intelligent Dashboard'.
The new dashboard will be integrated by the development team into Jenkins pipeline to automatically extract the logs after every new build run.

Keywords:

Data collection, Natural Language Processing (NLP), Deep Learning (DL), Automation, Jenkins, Selenium, Behavior Driven Development (BDD), Cucumber, Gherkin, Page Object Model (POM), Key Performance Indicators (KPIs)

Contents

List of Figures

List of Tables

List of Abbreviations

AI: **A**rtificial **I**ntelligence

IAI: **I**ndustrial **A**rtificial **I**ntelligence

QA: **Q**uality **A**ssurance

NLP: **N**atural **L**anguage **P**rocessing

KPI: **K**ey **P**erformance **I**ndicator

SDLC: **S**oftware **D**evelopment **L**ife **C**ycle

UML: **U**nified **M**odeling **L**anguage

BDD: **B**ehavior-**D**riven **D**evelopment

POM: Page **O**bject **M**odel

G-W-T: Given-**W**hen-**T**hen

VPN: Virtual **P**rivate **N**etwork

CI/CD: Continuous **I**ntegration/**C**ontinuous **D**eployment

GUI: Graphical **U**ser **I**nterface

IaC: Infrastructure **a**s **C**ode

NLTK: Natural **L**anguage **T**ool**K**it

ML: Machine **L**earning

DL: Deep **L**earning

CNN: Convolutional **N**eural **N**etwork

DNN: Deep **N**eural **N**etwork

CSV: Comma **S**eperated **V**alues

HDF: Hierarchical **D**ata **F**ormat

JSON: Java**S**cript **O**bject **N**otation

API: Application **P**rogramming **I**nterface

General Introduction

Artificial Intelligence (AI) solutions have gained more and more popularity, after every discovery, to mimic human intelligence. The 1956 Dartmouth workshop, aimed to explore the potential for creating intelligent machines and simulate human-like intelligence, marks a milestone in the history of AI. Actually, it marked the moment that AI gained its name and mission. Since then, AI was used to solve industrial and research problems to help humans attaining their goals by imitating their behaviors and intelligence. Industrial artificial intelligence (IAI), is the application of AI to industrial use cases like the movement and storage of goods, supply chain management, advanced analytics, automation and robotics in manufacturing.

Our project applies IAI in the customer retail industry. In the customer retail world, promotions refer to the various strategies used by retailers to encourage customers purchase their products or services. These promotions can take many forms, such as discounts, free gifts, loyalty programs, and other incentives that are designed to attract and retain customers. Promotions take part of the main strategies to enlarge a new customer segment, keep old customers happy, and maximize profit at the same time. Cognira offers a solution to its clients, that helps them perfectly manage their promotions. A key operation to keep Cognira's clients happy is to deliver a high quality solution after involving capturing tests results and reporting bugs and errors by the QA engineers. The latter create additional tests to verify these issues. For that, it's important to enhance the effectiveness of the automated testing and optimize the test execution and analysis phase.

QA team members use 'Allure Report' to visualize their tests results and gain insights from it. In 'Allure Report', testing results are divided into two main types: product defects and test defects. The first, are issues or bugs that are found in the software after it has been released to production. These defects are usually reported by end-users or customers who have used the software. Product defects can be caused by a variety of factors, including coding errors, design flaws, or incorrect functionalities. For example, having an 'Assertion Error', a message returned after verifying if the code behaves as expected or testing if a condition is true, is a product defect. The second type is test defects which are issues or bugs that are found during the testing process and they are spotted when executing test cases. They are usually reported in the testing phase and are fixed by developers before the

software is released to production. For example, clicking on a non existent element raises a 'NoSuchElementException', an exception in programming that is thrown when a program tries to access an element in a data structure such as an array, list, or a set, and the element does not exist, is considered as a test defect.

The problem is that the current classification of defects takes too much time for the QA team to discover the real bugs since this type of classification has a low rate of revealing the actual bugs. Every single day for example, the pipeline runs many builds, which will create an 'Allure Report' with the results for each one of them. The number of test cases may vary up to 120 per build. QA engineers need to test all the defected ones manually to figure out the real bugs and evaluate their automated tests performance. This situation is considered as a time consumer.

Given this context, the aim of this project is to develop an AI algorithm to classify the exception logs using deep learning, after processing them using natural language processing. Then develop an automation script to access the Jenkins pipeline, collect the exception logs from the 'Allure Report' of every new build. The outcomes will be presented in a novel format called the 'QA Intelligent Dashboard' which is designed to present the classification results in a visually appealing and user-friendly manner. By utilizing this new format, we aim to make the results easily understandable and accessible to users. The project emphasizes on the need for continuous monitoring and feedback to fine-tune the automated testing process. By regularly analyzing testing results and collecting feedback, QA team could make adjustments to optimize test cases, and ensure accurate and reliable test outcomes. By enhancing the effectiveness of the automated test and optimizing the testing process, the project aims to achieve better test efficiency, improve the accuracy of bug detection, and ultimately deliver a high-quality software product.

This report consists of three different chapters. The first one concerns the scope statement which covers the project's scope, the project's owner, and the used methodology. The second focuses on the requirements analysis and specifications. This chapter will include the project overview, requirements specifications and describe both automation and AI parts. The third chapter, will discuss the realization and implementation of the project from data collection, data processing, AI algorithm development till preparing the KPIs and generating the 'QA Intelligent Dashboard'. Finally, a general conclusion is presented to summarize the work.

Chapter 1

Scope Statement

Introduction

This chapter contains the project charter, the project owner description, the problem statement explanation with the project objectives, and the methodology followed to accomplish the work that was done.

1.1 The Project Charter

The project charter contains an overview of the scope statement [1]. In this section, the project context, description and project roles are defined and well described.

1.1.1 Project Context

Cognira has the QA department which is responsible for ensuring that its solutions meet certain quality standard and the desired level of functionalities before being delivered to the clients. In order to visualize the testing results, Cognira's QA engineers use a tool named 'Allure Report'. It takes the test results from the console and displays the failures classified into test defects and product defects. Upon using 'Allure Report', the QA engineers expectations were met through it that they decided to stick with it. As their test cases and scenarios grew throughout complexity, the current solution wouldn't be efficient to use. This is because it doesn't reflect the performance of the automated tests and the real bugs rate. Also, the need for the report is changing since Cognira's QA team now, are opting for the automation and they would like to increase the accuracy of their automated tests intelligently by getting better classification of defects to determine trustfully the real bugs from the not actual ones.

1.1.2 Project Description

Our 'QA Intelligent Dashboard' is an AI solution dedicated to visualize the exception logs classification. Using automation, our solution accesses Jenkins dashboard to go through the 'Allure Report' of the last build execution, extract its exception logs, process them using NLP and classify them using DL to generate new more detailed observations. The results of this classification are displayed in the 'QA Intelligent Dashboard' where QA engineers can easily access that information as well as get insights on additional KPIs.

1.1.3 Project Roles

Throughout the entirety of the project, the engineer is supposed to set its main goal, state its needs and requirements and make a theoretical design of the solution with great consideration to the technical feasibility, the project execution, and project documentation.

1.2 The Project Owner

1.2.1 Presentation of Cognira

Cognira was founded in 2015. It is a multi-national company with offices in three global locations which are Atlanta, Georgia in the United States of America, Tunis in Tunisia, Lille in France, and employees in London and Istanbul. Since opening the Tunis office in 2017, it grew exponentially in terms of number of employees to reach more than 85 employees by the end of the first quarter of 2022. Globally, Cognira made a significant growth since its foundation, so significant that it ranked as one of the top 5000 growing companies in the United States two years in a row (2019 and 2020). Cognira's clients are particularly big retailers who are looking for a solution provider that understands the technical world as well as the retail world so the overall interaction would be as easy as possible for them. [2]

1.2.2 Field of Expertise and Solutions

Cognira provides three different solutions for its clients depending on their need.

1.2.2.1 Cognira's AI Promotion Solution

It forecasts demand of all the products accurately. It also reduces inventory costs and decreases waste with machine-learning promotional forecasts that accurately reflect demand and continuously improve over time. [3]

1.2.2.2 Forecast Service

It provides the accurate demand forecasts desired by the customer without the complexities of managing the system and the forecasting team. [4]

1.2.2.3 Assortment and Allocation Decisions Optimization

With the assortment and allocation decisions optimization solution, retailers get the most advanced and scientific answers to the questions that concern them most - what and how much to buy, and when and where to place products - helping to reduce loss and increase sales. [5]

1.3 Study of The Existing

In the QA field, the primary methods employed to capture and analyze test results are predominantly based on two solutions: the console output and the 'Allure Report'. These approaches play crucial roles in gathering information and generating insights regarding the outcome of the testing process.

1.3.1 Console Output

The console output with its textual format serves as a fundamental and widely adopted method for displaying test results. It plays a crucial role in providing insights into the pipeline execution process. The console output is a real-time log that includes information such as build status, test results, and error messages. By using the console output, developers can quickly diagnose issues, debug the pipeline, and discover the testing results. In particular, when running automated tests as part of the pipeline, the console output can display the test results, including pass/fail status, test duration, and any error messages or stack traces. The console output is a powerful tool that can help in improving the quality and reliability of the pipeline and software by providing real-time visibility into the execution process. Figure 1.1 shows a sample of the console output.

```
Creating artifact for the build.
Artifact was added to the build.
[Pipeline] }
[Pipeline] // script
[Pipeline] mail
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] }
[Pipeline] // podTemplate
[Pipeline] End of Pipeline
Finished: UNSTABLE
```

Figure 1.1: Console Output

1.3.2 Allure

1.3.2.1 Definition

The 'Allure Report' emerges as a comprehensive and visually appealing reporting tool in the QA domain. 'Allure' is a flexible, open-source and lightweight multi-language test reporting tool. It provides clear graphical reports and allows everyone involved in the development process to extract the maximum of information from the everyday testing process.

Figure 1.2: Allure Logo

1.3.2.2 Test Cases Specificity

In 'Allure Report', features and stories are used to organize test cases into a hierarchical structure.

- Feature: is a high-level business goal or objective that the software being tested is intended to achieve. It can be broken down into one or more stories.

- Story: is also known as a user story or a user scenario, represents a single user action or interaction with the software. It can be further broken down into one or more test cases.

By organizing test cases into features and stories, you can easily see which tests are related to which features and which stories. This makes it easier to understand the overall testing progress and to identify areas that may require additional testing.
'Allure' framework does not impose a maximum limit on the number of test cases that can be included in a test report. However, the amount of data that 'Allure' can handle will depend on the system's resources where the report is generated and stored. If there are a large number of test cases, it is recommended to divide them into smaller test suites to make it easier to analyze, manage and maintain the results.

1.3.2.3 Components

Allure Report's components are:

- Suite: is a collection of related test cases. It can contain multiple test cases that share a common purpose, such as testing a specific feature, functionality, or module of an application. Test suites can be used to organize test cases in a hierarchical manner to provide a clear and structured view of the tests. Each suite can contain multiple test cases.
- Graphs: are graphical representations of test suites in the test report. The test cases could be grouped based on their status, priority, or other criteria, which makes it easier to identify the areas of the application that require attention.
- Environment variables: are a way to provide additional information about the environment in which tests are executed.

Figure 1.3 represents an 'Allure Report' overview and figure 1.4 shows its graphs.

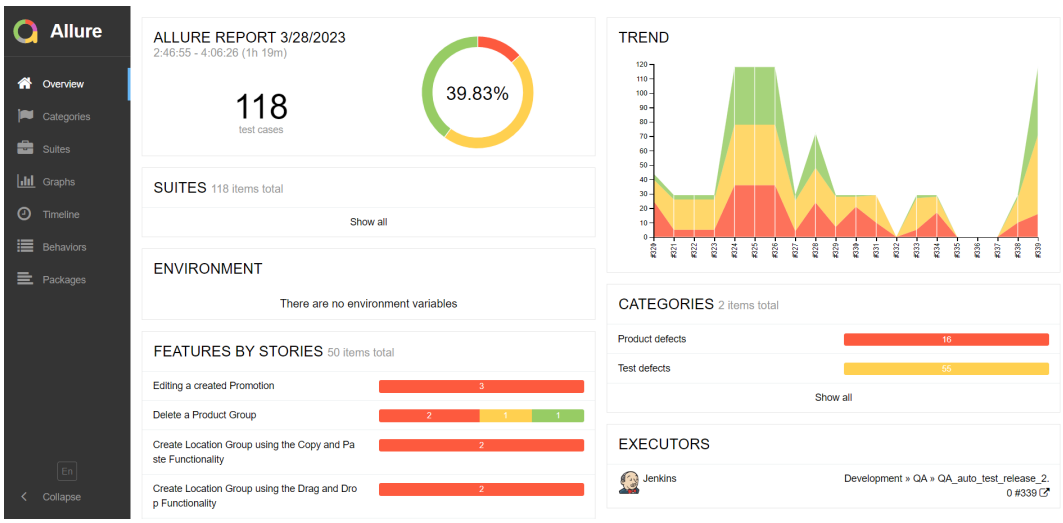

Figure 1.3: Overview of Allure Report

Figure 1.4: Graphs in Allure

Those solutions are powerful tools for understanding and interpreting test results. While the console output offers a quick and concise overview, 'Allure Report' enhances the analysis by engaging representation of the testing outcomes. For that, the solution used by Cognira's QA team to discover the test cases results is 'Allure' since its report is more interactive than reading a text output.

1.4 Problem Statement

Cognira's business is getting bigger with more releases and deployments, confirming if a test result is an actual bug or not is now considered as a time waste. Every day there is at least two builds of the pipeline and an 'Allure Report' is generated for each of them. The QA team needs to test the defects in every 'Allure Report', that may be at least 30, without having

an idea about the accuracy of their automated tests. QA team needs a new solution that classifies the exception logs extracted from 'Allure Report', into more detailed categories to differentiate the actual bugs from the not actual ones and evaluate the performance of their tests.

The project will rely on automation and NLP to achieve this goal. Our contribution will be the generation of a more specific and detailed dashboard for the QA team, optimize the testing process and improve bugs identification. By implementing advanced automation and leveraging AI techniques, the project aims to reduce testing time while enhancing the efficiency of bugs recognition. This optimization enables the QA team to allocate their time and resources more efficiently, focusing on critical areas and executing a larger volume of tests within the available time frame.

1.5 Project Goals and Objectives

The solution to be developed must solve the discussed problems. Therefore, the project's goals are as follows:

- **Data collection from 'Allure Reports'**
  The data collection must be done from various 'Allure Reports' by accessing the QA pipeline through Jenkins dashboard.
- **Exception logs classification**
  One of the main problems that were met in 'Allure Report' is that it provides only binary classification (product defects and test defects). A deeper and a well defined multi-class classification algorithm will be developed.
- **QA Intelligent Dashboard Generation**
  The trained model needs to be integrated within an automation script and output the KPIs to be generated into the new dashboard, the 'QA Intelligent Dashboard'.

1.6 Project Deliverables

- **Source code:**
  At the end of this internship, the source code of the different parts indicated is to be delivered. This deliverable includes the well reviewed and validated source code.
- **Requirements specifications:**
  Requirements specifications must be prepared according to the project charter in order to specify the different tasks needed with an estimation of capacity and to know the functional expectations.

- **Source code documentation:**
  The AI algorithm architecture and the implemented code must be accompanied by a documentation with all the research results that demonstrate the different flows and choices.

- **Final presentation:**
  During the internship, a progress report in the form of a presentation is expected on monthly basis in order to offer an overview of the project status, milestones reached, and difficulties encountered to Cognira's QA team along with the group of interns.

1.7 Software Development Life Cycle

Software Development Life Cycle (SDLC) is a conceptual framework or process that considers the structure of the stages involved in the development of an application from its initial feasibility study to its deployment in the field and maintenance [6]. Developing a software solution involves a set of multiple phases. In this context, different methodologies were designed to better enhance the SDLC process workflow implementation, yet, information technology expertise introduced a better approach to the SDLC process: the Agile Software Development Cycle.

1.7.1 Agile Software Development Methodology

For this project, it is essential to develop an application that is both testable and stable, while also ensuring scalability. These characteristics need to be present at all times during the development life cycle. Also, the implementation of frequent updates must be easy. Having all of these requirements to be taken into consideration, Cognira opted that the Agile methodology would be the best method for managing and organizing the project.

The Agile methodology is a project management approach that involves breaking the project into phases and emphasizes continuous collaboration and improvement. Teams follow a cycle of planning, executing, and evaluating. The Agile methodology is perfectly adapted to the continuous changing needs of customers, allowing a better process for software developments. In fact, Agile allows reversing back to previous steps and responds to the evolving enhancements and priorities encountered while developing the application.

1.7.2 Scrum Framework

For more consistency while implementing the Agile methodology, Cognira's team is adopting the scrum framework as a way to support agile implementation and enhance its approach on developing complex and long-term projects allowing the continuous monitoring of consecutive

deployments.
Scrum is an adaptable, fast, flexible and efficient Agile framework. It is designed to deliver value to the customer throughout the development of the project. Scrum's main objective is to satisfy the customer's needs through an environment of transparent communication, collective responsibility and continuous progress. The development starts from a general idea of what needs to be built, by developing a prioritised list of features (product backlog) that the product owner wants to achieve [7]. Figure 1.5 represents the scrum cycle.

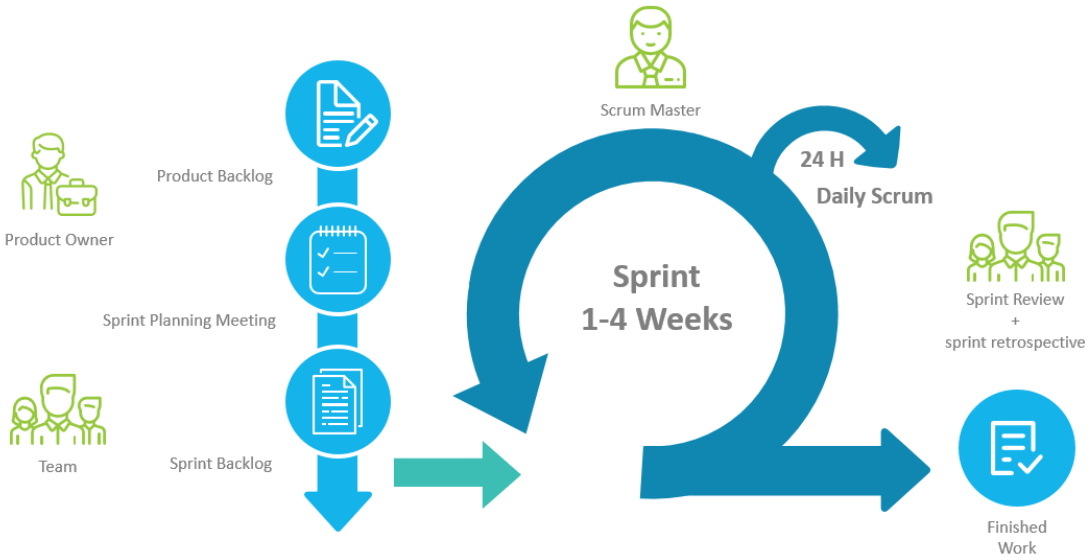

Figure 1.5: Agile Scrum Process[8]

1.7.2.1 Scrum Roles

Upon working with scrum, a given team member should take part in one of these groups or take one of these roles:

- **Scrum master**

  He helps to facilitate the project workflow to the whole team by ensuring that the framework of the scrum is respected. The scrum master is committed to the values and practices of the scrum, but must also remain flexible and open to the possibilities of the team and open to opportunities for them to improve the workflow. [9]

- **Quality Assurance team**

  It is a group of professionals responsible for ensuring that a product or a service meets the expected quality standards. The main goal of the QA team is to identify defects, errors, and other issues in a product or service before it is released to customers. The QA team is typically involved in the entire product development life cycle, from the initial design phase to the final release. They perform various tests to ensure that the

product meets specific quality requirements and standards. QA team works closely with other teams, such as development, design, and product management.
Cognira's QA team is a self-organizing cross-functional team [10]. The team is responsible for meeting the sprint target, and its performance is critical to the success of Scrum.

- **Product owner**

  He is a member of the agile team. He is responsible for defining the stories and for organizing the backlog of the team to order the execution of program precedence while keeping the conceptual and technical integrity of the components for the team. [11]

1.7.2.2 Scrum Meetings

- **Sprint review meeting**

  The scrum team and stakeholders discuss what was done throughout the sprint as well as what has changed in their environment during this meeting. The participants discuss what to do next based on these updates. The Product Backlog can be tweaked to accommodate new possibilities. The sprint review should not be limited to a presentation as it is a working session. [12]

- **Daily meeting**

  It is a 15-minute meeting for the scrum team. To keep things simple, it happens at the same time and place every working day of the sprint. They engage as members if the product owner or scrum master is actively working on items in the sprint backlog. [13]

- **Code review process**

  In order to assure the good quality of the final product, a code review process occurs at the end of every sprint or following the creation of each component inside the project. As figure 1.6 demonstrates, the author submits the code to be reviewed. The reviewer have the ability to improve the code and to make remarks and comments on it. Afterwards, he can either approve it or not. In case it gets approved, the code is submitted and the author moves on to the next agile iteration. If not, further modifications are performed on the code until it is good enough to be submitted.

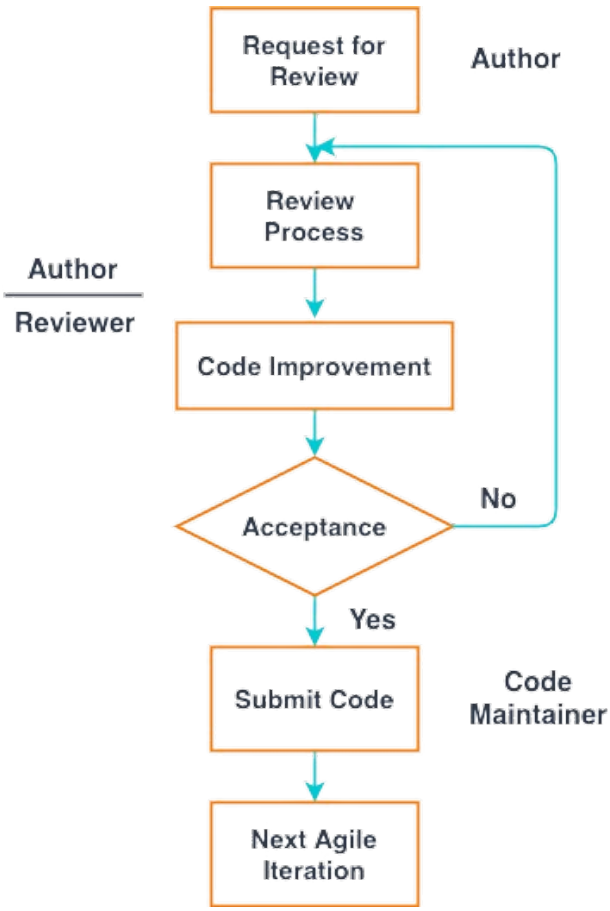

Figure 1.6: Code Review Process[14]

1.7.3 Regression Testing

To make sure that new changes introduced in each sprint do not affect the existing functionalities of the application, regression testing is needed. In Agile, regression testing ensures that previous functionalities of the application works effectively and new changes in a sprint have not introduced new bugs. Regression tests should be employed whether there is a small localized change to the software or a larger change.

In software testing, a regression bug refers to a defect that is introduced into previously working software code as a result of changes made in newer versions or updates. Regression bugs can occur when new features are added, code is modified, or configurations are changed, and can lead to unexpected failures in the software. In the context of automation, a regression bug can occur when automated test cases that were previously passing, now fail due to changes made in the application. This can happen if the automated test scripts are not updated to reflect the changes in the application or if the changes made in the application were not thoroughly tested for their impact on existing functionalities. To prevent regression bugs in automation, it's important to:

- Have a robust and comprehensive test suite that covers all the critical functionalities of the application
- Regularly review and update automated tests to ensure they reflect any changes made in the application.
- Have a well-defined and structured testing process that includes thorough regression testing after each new release or update.

1.8 Project Plan

1.8.1 Project Realization Overview

The workload of the project realization consists of:

- Phase 1: 'Allure' context understanding by reading the documentation provided by Cognira. The main aim of this phase is to understand the test types, the visualization tool as well as the general workflow of exception logs extraction.
- Phase 2: Automate the exception logs extraction from 'Allure Reports' by accessing QA Jenkins pipeline. Through this phase, the data collection is performed.
- Phase 3: Ensure the reliability and quality of the data while processing it by performing tasks such as data cleaning, data augmentation and feature engineering.
- Phase 4: Develop the AI algorithm and try different architectures while searching for optimizers and metrics. This phase is running while following up the performance of the model till it meets the needs.
- Phase 5: Integrate the trained AI model into the automation script to ensure the extraction of new information and the automated prediction.
- Phase 6: Generate the 'QA Intelligent Dashboard' that displays the classification results through various KPIs.

1.8.2 Project Progress Chronogram

The following measures were taken in order to keep track of the project execution:

- Daily stand-up meeting so that interns could update their supervisors with the work progress

- Weekly deep dive follow-up in order to track the overall work done for the week
- Monthly work progress presentation
- Code review after pushing the code of a certain sprint to the version control management tool

The timetable designed to track the project's progress is depicted in the Gantt chart[15] presented in table 1.1. In order to make sure that the project features are implemented in the best way, sticking as much as possible to this timeline was important. The work was divided into sprints. JIRA is a project management software that was used to follow the project's track. In JIRA, a sprint is called also an 'epic'[16].

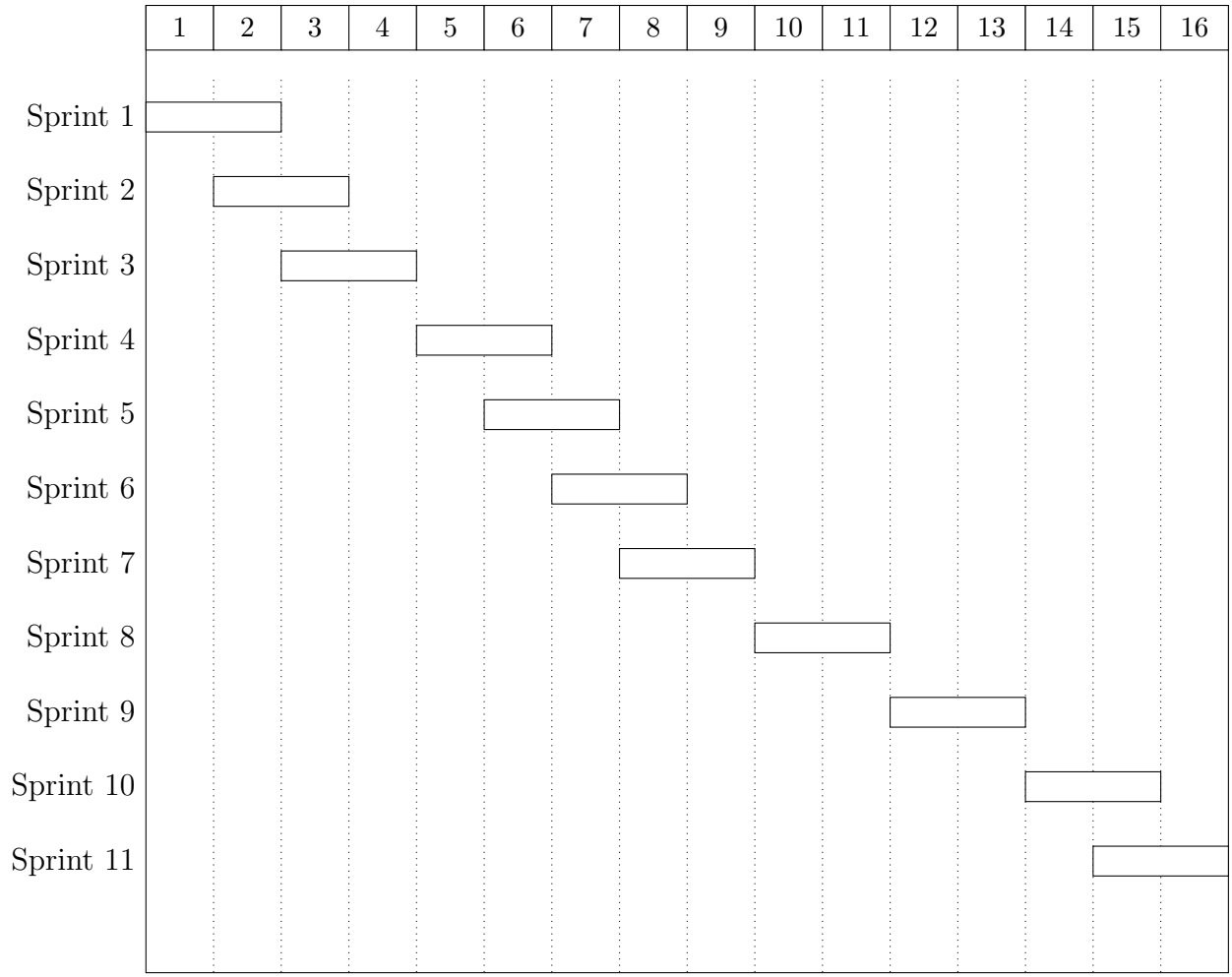

Table 1.1: Gantt Diagram

These are the descriptions of each sprint :
✓Sprint 1: Cognira's 'PromoAI' solution exploration.
✓Sprint 2: Training on Cognira QA development stack.
✓Sprint 3: 'Allure Report' exploration, discovering the exception logs coming from Cognira's releases, starting from the categories and test results to graphs.

✓Sprint 4: Data collection using an 'Allure Report' local URL.
✓Sprint 5: Automate data collection through Jenkins access and multiply the number of 'Allure Reports'.
✓Sprint 6: Explore various NLP techniques for textual data processing.
✓Sprint 7: Develop the AI model to classify the exception logs and predict the different categories of test failures.
✓Sprint 8: Implement the trained model into the automated data extraction script.
✓Sprint 9: Design the new report and define the KPIs such as charts and statistics.
✓Sprint 10: Automate the 'QA Intelligent Dashboard' generation.
✓Sprint 11: Write the report.

Conclusion

In this chapter, a detailed description of the project is demonstrated along with a description of the necessary workload that should take place in order to finish the project within the required expectations. The following chapter is dedicated to introduce the requirements analysis and specifications.

Chapter 2

Project Requirements

Introduction

This chapter serves to discover the development environment and to explore the requirements specifications, employing the Unified Modeling Language (UML) to capture and communicate system requirements. Then, the focus is shifted to discover the automation methodologies and frameworks and touch upon the role of AI, which brings advanced capabilities and intelligence to our system.

2.1 The Development Environment

The working environment plays an important role in order to realize the project. It helps in delivering the product in good quality and helps to ensure a good work process for the developer. Our working environment is shown as below:

2.1.1 Hardware Environment

As this is a software project, the only hardware that will be used is the laptop used for the development. Table 2.1 contains a description of it:

Laptop	DELL INSPIRON 3542
RAM	16 GB
Storage	1 TeraByte
Processor	Intel i5-4210U 2.7 GHz

Table 2.1: Hardware Environment

2.1.2 Software Environment

A wide variety of software tools were used while working on the project. The main software tools used for the project realization are presented through table 2.2:

Desired functionality	**Designated tool**
IDE	Pycharm editor is an integrated development environment used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester and integration with version control systems.
Project management	JIRA is a proprietary issue tracking tool developed by Atlassian that includes bug tracking and agile project management features.
Version control	Bitbucket is a Git-based source code repository hosting service.
Computing environment	Jupyter notebook is a web-based interactive computing platform. The notebook combines live code, equations, narrative text, visualizations. We used it to develop, train and save the AI models.

Table 2.2: Software Environment

2.2 Requirements Specifications

2.2.1 Modeling Language

The project's objects were modeled using the Unified Modeling Language (UML). It's a modeling language for designing and implementing software architecture. It is made up of a number of diagrams that depict the system's boundaries, structure, behavior, as well as the components that make it up. [17]

2.2.2 Actors Specifications

There are two actors that interact with our system.

- **QA engineer**: this actor is responsible for executing the 'Runner file' to start the system.
- **Allure-Jenkins plugin**: allows to automatically generate the 'Allure Report', that contains the exception logs, and attach it to the build during Jenkins job run.

2.2.3 Use Case Diagram

We expose, in this section, the project design by modeling the use case diagram that will help to better understand the mechanism of the implementation of the software. Figure 2.1 represents the use case diagram of the project.

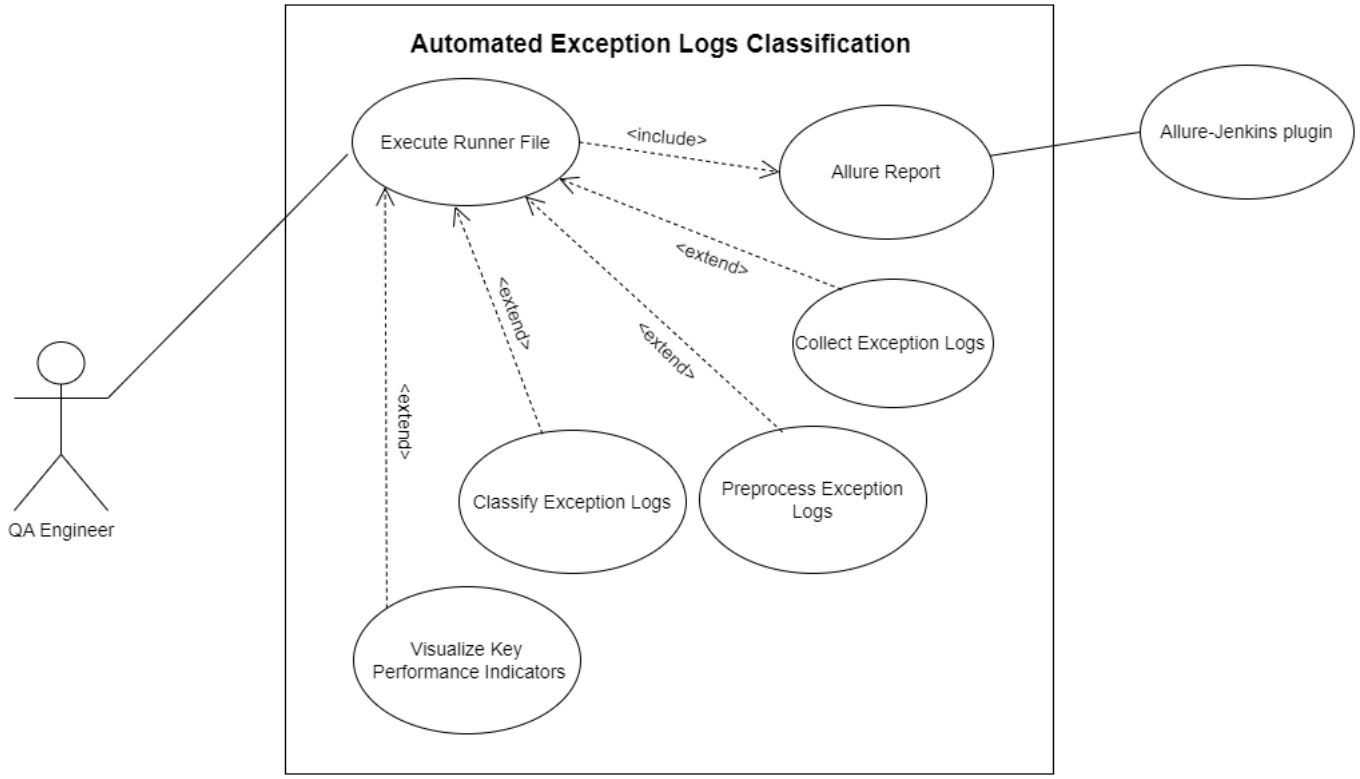

Figure 2.1: Use Case Diagram

2.2.4 Use Case Textual Description

The textual description of the use case is a description of the chronology of the actions. It allows to clarify the flow of the functionality and to indicate possible constraints. In the following, we present the textual descriptions of some use cases in order to have a global vision of the functional behaviour of our application. Table 2.3 resumes the use case of automated exception logs classification.

Use case description: automated exception logs classification
Identification Title: automated exception logs classification Goal: automate the exception logs extraction and classify them using deep learning Principal actor: QA engineer Secondary actor: Allure-Jenkins plugin Summary: allure-Jenkins plugin generates Allure Report and attach it to build during Jenkins job run. This 'Allure Report' contains the exception logs to be extracted, processed using NLP and classified using deep learning. From the predictions, the KPIs are prepared and visualized into the generated dashboard'.
Sequencing The use case starts when the QA engineer executes the runner file to launch its start
Pre-conditions The existence of an 'Allure Report'
Nominal scenarios 1. The data is collected from the Allure Report. 2. The collected exception logs are processed using NLP. 3. The processed exception logs are classified. 4. The KPIs are prepared thanks to the predicted results. 5. The KPIs are visualized in the generated 'QA Intelligent Dashboard'
Post-conditions The QA Intelligent Dashboard will remain alive until the QA engineer exit the view.

Table 2.3: Use Case: Automated Exception Logs Classification

2.2.5 Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how, and in what order, a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Figure 2.2 represents the sequence diagram of the project.

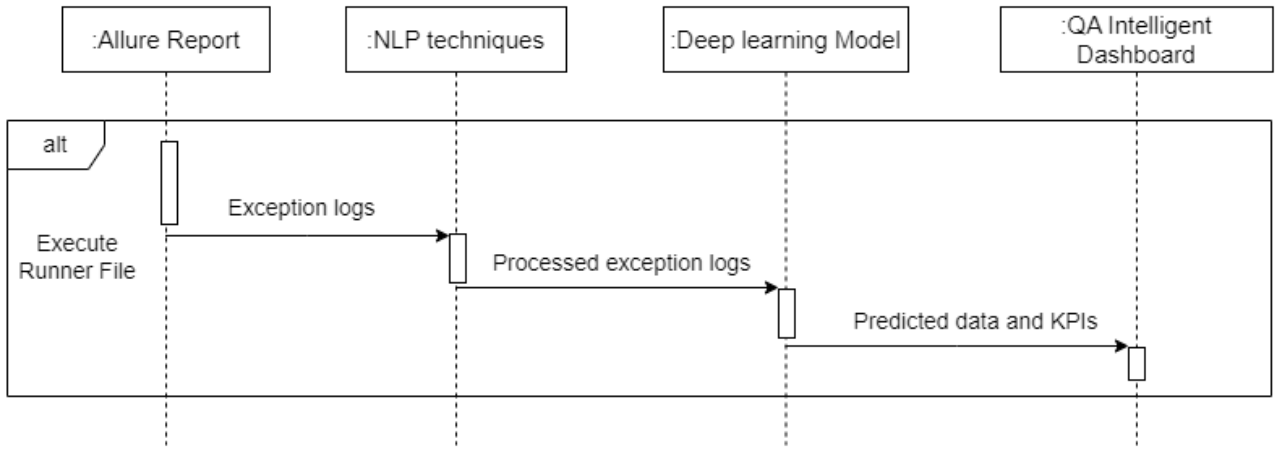

Figure 2.2: Sequence Diagram

2.3 Automated Testing Methodologies and Frameworks

Automation refers to the use of technology to perform tasks and processes that were traditionally done manually by humans. In response to the growing need for automated testing and in order to meet Cognira's expectations in terms of the project's usability, an in-depth study has been conducted to best adjust the benchmarking and methodologies and frameworks to be used.

2.3.1 Behavior Driven Development

Behavior Driven Development (BDD) is an agile software development methodology in which an application is documented and designed around the behavior a user expects to experience when interacting with it.
By encouraging developers to focus only on the requested behaviors of an application, BDD helps to avoid bloat, excessive code, unnecessary features or lack of focus. A typical project using BDD would begin with a conversation between the developers, managers and the customer to form an overall picture of how a product is intended to work. It offers the ability to enlarge the pool of input and feedback to include business stakeholders and end users who may have little software development knowledge. The expectations for the product's behavior are then set as goals for the developers, and once all of the behavior tests are passed, the product has met its requirements and it is ready for delivery to the customer. Figure 2.3 represents how does BDD work.

Figure 2.3: Behavior Driven Development Cycle [18]

2.3.2 Page Object Model

Page Object Model (POM) is a design pattern used in test automation to create a more maintainable and scalable test framework [19]. In POM, the web pages of an application are modeled as classes, and each class encapsulates the behavior and properties of a specific page. The class exposes methods that perform actions on the page and properties that expose the state of the page. The main benefits of using POM in test automation are:

- Maintainability: by encapsulating the behavior and properties of a page in a class, changes to the page can be made in a single location, rather than being scattered throughout the test code.

- Reusability: POM allows the reuse of code across tests, as page classes can be used by multiple tests.

- Scalability: as the number of pages in an application grows, POM can help manage the complexity by breaking down the pages into smaller, more manageable classes.

Figure 2.4 shows the difference between using POM and not using it.

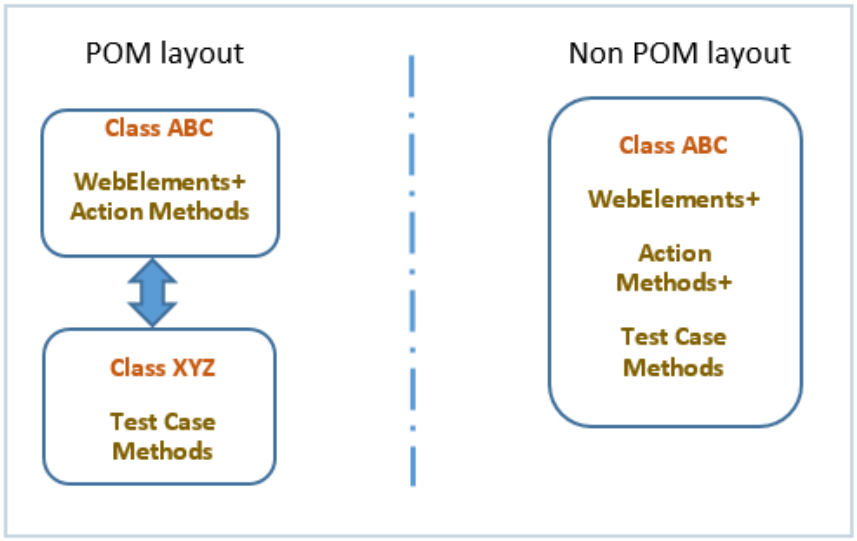

Figure 2.4: Structure Comparison: With and Without POM in Automated Testing[20]

2.3.3 Cucumber

Cucumber is a tool that supports BDD. It reads executable specifications written in plain text and validates that the software does what those specifications say. The specifications consists of multiple examples, or scenarios where each scenario is a list of steps for Cucumber to work through.

Figure 2.5: Cucumber Logo

Cucumber verifies that the software conforms with the specification and generates a report indicating success or failure for each scenario. In order that Cucumber understand the scenarios, they must follow some basic syntax rules, called Gherkin.

2.3.4 Gherkin

Gherkin is a human-readable language used to write executable specifications for software systems. Gherkin uses a simple syntax that allows developers, testers, and business stakeholders to collaborate on defining the expected behavior of a software system. It uses keywords such as 'Given', 'When' and 'Then' to describe the preconditions, actions, and expected outcomes of a feature or scenario. Gherkin scenarios are typically written in plain english and can be easily understood by non-technical stakeholders. Figure 2.6 shows an example of a scenario:

```
Feature: As a user I want to access Jenkins and extract data

Scenario:  access Jenkins
    Given the user has launch the browser
    And open jenkins
    Then enter credentials
    Then click on login button
    And the home page is opened
    Then get allure URLs
    And extract exception logs from the wanted number of allure
    Then transform product csv into a dataframe
    And transform test csv into a dataframe
    Then clean product errors dataframe
    And clean test errors dataframe
    Then join the dataframes
    Then save data
```

Figure 2.6: : Gherkin Language- Feature File (.feature)

2.3.5 Selenium

Selenium automates web browsers. It is the most famous for enabling rapid, repeatable web-app testing, which allows developers to ship new releases faster and with confidence. Selenium is an open-source tool providing a single interface that lets you write test scripts in programming languages like Ruby, Java, NodeJS, PHP, Perl, Python, and C#, among others.

Figure 2.7: Selenium Logo

2.3.6 Test Automation Structure

- Feature files: are typically written in a human-readable language, such as Gherkin. They contain scenarios, which are a series of steps that describe the actions a user would take to accomplish a task. These steps are written in a specific format called 'Given-When-Then' (GWT) that helps to organize and structure the test cases.

- Steps definitions: are code blocks that define the actions needed to be taken for each step in the GWT format. They are typically implemented using a programming language. The steps definitions map to the GWT steps in the feature file, and their

implementation describes the automated actions.

- Runner file: is a script or program that runs multiple automated test cases or suites. It may include the necessary folders and related allure commands. The runner file eliminate the repeatability of common steps and avoid the waste of time when dealing with more dynamic and complex structure for automated testing.
  Figure 2.8 represents our project architecture:

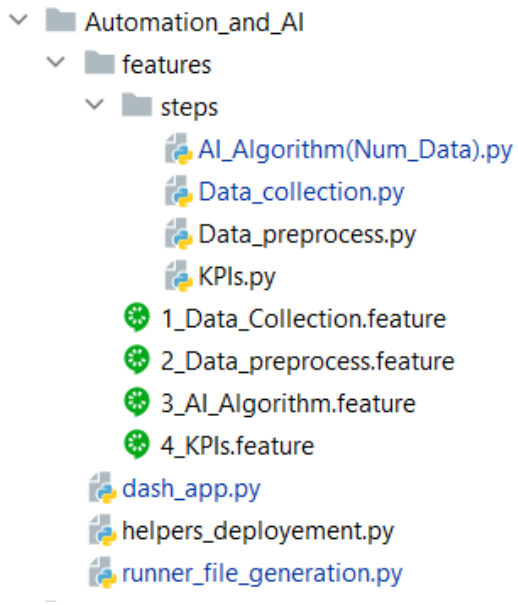

Figure 2.8: Test Automation Architecture

2.3.7 Jenkins

The exception logs are extracted from an 'Allure Report' found in Jenkins dashboard. For security reasons, the access to Jenkins requires being connected to Cognira's Virtual Private Network (VPN) and having your own Microsoft Azure credentials.

Jenkins is an open-source automation tool used to automate software development processes, such as building, testing, and deploying software applications. It allows software developers to build and test their code continuously and automatically, providing real-time feedback on build status and enabling early detection and resolution of issues. It is highly customizable and extensible, allowing developers to add various plugins to suit their specific needs. It supports a wide range of programming languages, source code management systems, build tools, and testing frameworks, making it a popular choice for continuous integration (CI) and continuous deployment (CD) pipelines. Jenkins is written in Java and can run on various operating systems, including Windows, Linux, and macOS. It has a large community of

users and contributors who actively develop and maintain it which makes it a user-friendly when encountering an issue.

Figure 2.9: Jenkins Logo

Cognira's Jenkins is configured and managed 'as code' which means that the configuration and management of the Jenkins environment are done through code, rather than a Graphical User Interface (GUI) or manual configurations. This approach is often referred to as 'Infrastructure as Code' (IaC).

2.4 Data Processing and AI Algorithm

2.4.1 Data Processing

Data processing refers to the overall process of transforming raw data into useful information. This involves a variety of tasks, such as data cleaning, data transformation, and data analysis. Data processing can be a complex process that involves various tools and techniques, depending on the nature and scope of the data. It is a task of converting data from a given form to a much more usable and desired form, making it more meaningful and informative, using machine learning algorithms, mathematical modeling and statistical knowledge.

In summary, data processing is a broader term that focuses on extracting meaningful insights and patterns from the data to make informed decisions and predictions. Figure 2.10 represents the data life cycle from collection till the final output.

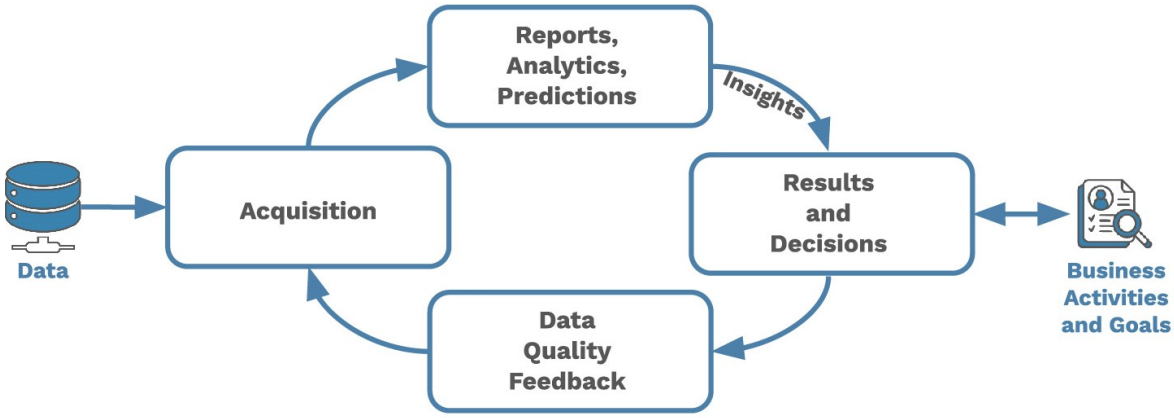

Figure 2.10: Data Life Cycle[21]

For our project, we went through various NLP techniques that gave our data more clearness. For the sake of this, some packages have been installed and imported such as:

- Natural Language Toolkit (NLTK): is a suite of libraries and programs for symbolic and statistical natural language processing for english, written in the Python programming language.

- Word tokenize: in Python tokenization, basically refers to splitting up a larger body of text into smaller lines and words. Word_tokenize is a function in Python that splits a given sentence into words using the NLTK library. Also, it removes the tabulations, new line indexes and remains the punctuation and special characters like $!

- Stop words: are common words and basically ignored by typical tokenizers. By default, NLTK includes a list of 40 stop words, including: 'a', 'an', 'the', 'of', 'in', etc. The stop_words in NLTK are the most common words in data.

- Punkt: is an unsupervised trainable model, which means it can be trained on unlabeled data. It is used in place of removing punctuation from a string in Python by using the 'String' module, which provides a constant string of all American Standard Code for Information Interchange (ASCII) punctuation characters, and then filter out those characters from the string.

2.4.2 AI Algorithm

An AI algorithm refers to a specific set of instructions or rules designed to enable systems to perform tasks or solve problems. These algorithms form the core computational methods used in various AI applications and technologies.
AI algorithms can be categorized into different types based on their approaches and functionalities. Here are a few common types:

- Machine Learning (ML) Algorithms: enable machines to learn patterns and make predictions or decisions without explicit programming. They include techniques such as linear regression, logistic regression, decision trees, random forests and support vector machines.

- Deep Learning (DL) Algorithms: are a subset of machine learning algorithms that use neural networks with multiple layers to learn complex representations from data. Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs) are widely used deep learning algorithms.

2.4.2.1 Hyper-parameters

In DL, hyper-parameters are parameters that are set by the user before training the neural network and are not learned from the data during training. These parameters affect the behavior and performance of the neural network by controlling aspects such as its architecture, learning rate, regularization, optimization algorithm, and batch size [22]. The hyper-parameters to be tuned are the number of epochs, batch size, optimizer, learning rate and activation function.

- Epochs: defines the number of times that the learning algorithm will work through the entire training data set. One epoch means that each sample in the training data set has had an opportunity to update the internal model parameters.

- Batch size: is a hyper-parameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.

- Optimizer: is an algorithm or a method used to change the attributes of the neural network such as weights to reduce the loss.

- Learning rate: controls how much to change the model in response to the estimated error each time the model weights are updated.

- Activation function: defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.

2.4.2.2 Overfitting Prevention

Overfitting is a fundamental issue in supervised ML which prevents us from perfectly generalizing the models to well fit observed data, as well as unseen data [23]. To prevent overfitting, we included Early stopping and Dropout layers.

- Early Stopping: is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such method updates the learner so as to make it better fit the training data with each iteration. Up to a point, this improves the learner's performance on data outside of the training set.

- Dropout: randomly sets input units to 0 with a frequency of rate at each step during training time. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged. Adding Dropout layers has never changed the number of trainable parameters as dropout does not have any variables/weights that can be frozen during training. Both figures 2.11 and 2.12 demonstrate that the addition of dropout layers has no impact on the number of trainable parameters.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 128)               384

 activation (Activation)     (None, 128)               0

 dropout_3 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 64)                8256

 activation_1 (Activation)   (None, 64)                0

 dropout_4 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 32)                2080

 activation_2 (Activation)   (None, 32)                0

 dropout_5 (Dropout)         (None, 32)                0

 dense_7 (Dense)             (None, 6)                 198

=================================================================
Total params: 10,918
Trainable params: 10,918
Non-trainable params: 0
```

Figure 2.11: Model With Dropout Layers

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 128)               384

 dense_9 (Dense)             (None, 64)                8256

 dense_10 (Dense)            (None, 32)                2080

 dense_11 (Dense)            (None, 6)                 198

=================================================================
Total params: 10,918
Trainable params: 10,918
Non-trainable params: 0
```

Figure 2.12: Model Without Dropout Layers

2.4.2.3 Evaluation Metrics

Metrics are quantitative measures that are used to evaluate the performance of a model on a particular task. Different metrics are used depending on the specific task and the type of model being used [24]. Below, are the metrics we used to evaluate our classification model:

- Accuracy: is the number of correctly predicted data points out of all the data points. This metric is used to measure the algorithm's performance in an interpretable way. It is usually determined after the model parameters and it is calculated in the form of a percentage.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \times 100\% \tag{2.1}$$

  The accuracy is represented by a matrix called Confusion Matrix [25]. We opt for the accuracy metric since it perfectly describes what we want to follow.

- Loss: is a value that implies how poorly or well a model behaves after each iteration of optimization. It is calculated by the loss function. In our project, we used 'Sparse Categorical cross-entropy' loss function since our classes are mutually exclusive, that means each sample belongs exactly to one class.

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \times \log \hat{y}_i \tag{2.2}$$

  The loss metric chosen, 'Sparse Categorical Cross-Entropy', measures the difference between two probability distributions. In the case of classification problems, it is often used to compare the predicted probability distribution over a set of classes to the true probability distribution of the labels.

Conclusion

In conclusion, this chapter provided an extensive overview of the project, focusing on key aspects such as project UML diagrams, automation concepts, and AI components. Through this analysis, we have gained valuable insights into the various elements that contribute to the success of our project.

Chapter 3

Project Realization

Introduction

To ensure that the goals outlined in previous chapters are met, we will go from the initial stages of data acquisition to the fine-tuning of the model's performance. In this chapter, we will delve into the intricate journey of data collection, processing, and the development of the AI models, along with a comprehensive evaluation of its performance.

3.1 Data Collection

The data collection is done through the QA pipeline to iterate through builds history and extract the exception logs from their respective 'Allure Reports'. It is divided into two parts: collect the exception logs and collect the historical records.

3.1.1 Exception Logs Collection

The collected set of data contains roughly 800 exception logs and their corresponding defects: test defect or product defect. The data is then stored into a Comma separated values (CSV) file to be lately explored for data processing. Figure 3.1 presents a segment of the collected exception logs, providing a glimpse into the information gathered during the extraction process.

```
message,type
AssertionError,product
AssertionError: Oops! The Delete button should be disabled,product
AssertionError,product
"""AssertionError: Oops , save button is enabled !""",product
"""AssertionError: Oops , save button not enabled!""",product
AssertionError: Oops! The Delete button should be disabled,product
AttributeError: 'Context' object has no attribute 'seleniumHelpers',product
AssertionError,product
"""AssertionError: Oops , save button is enabled !""",product
AttributeError: 'Context' object has no attribute 'seleniumHelpers',product
AttributeError: 'Context' object has no attribute 'InputData',test
AttributeError: 'Context' object has no attribute 'Merchant_Cookies',test
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0),test
```

Figure 3.1: Collected Data

We also relies on data augmentation to increase the number of exception logs categories, hence the data size and the amount of training data available as input for our future model. To augment our textual data, we opt for:

- Random Insertion: in this technique, a new word such as 'issue' is randomly inserted into the sentence, which can help the model handle previously unseen words.
- Random Deletion: this technique involves randomly removing words from the exception log, which can help the model learn to handle missing words.
- Random Swap: two words are swapped in the message of exception log to create a new one.
- The use of a new pipeline to obtain the testing outcomes of a deployment following specific updates.

3.1.2 Historical Records Collection

As an important segment in our classification problem, we checked the historical records that will help us in classifying regression bugs. We checked if the status of each suite has been changed or not. For each suite, we added a label = 0, 1 with 1 indicates that the current status has been changed from the previous one, and assign it to its exception log. To accomplish this goal, we followed these steps:

1. Iterate over the list of suites in the 'Allure Report'
2. Disable the 'Passed' icon because suites with 'Passed' as a current status don't belong to any existent exception log.
3. Collect the suite and its exception log.

4. Check the history and label the suite.

Figure 3.2 displays a sample of a suite's historical records.

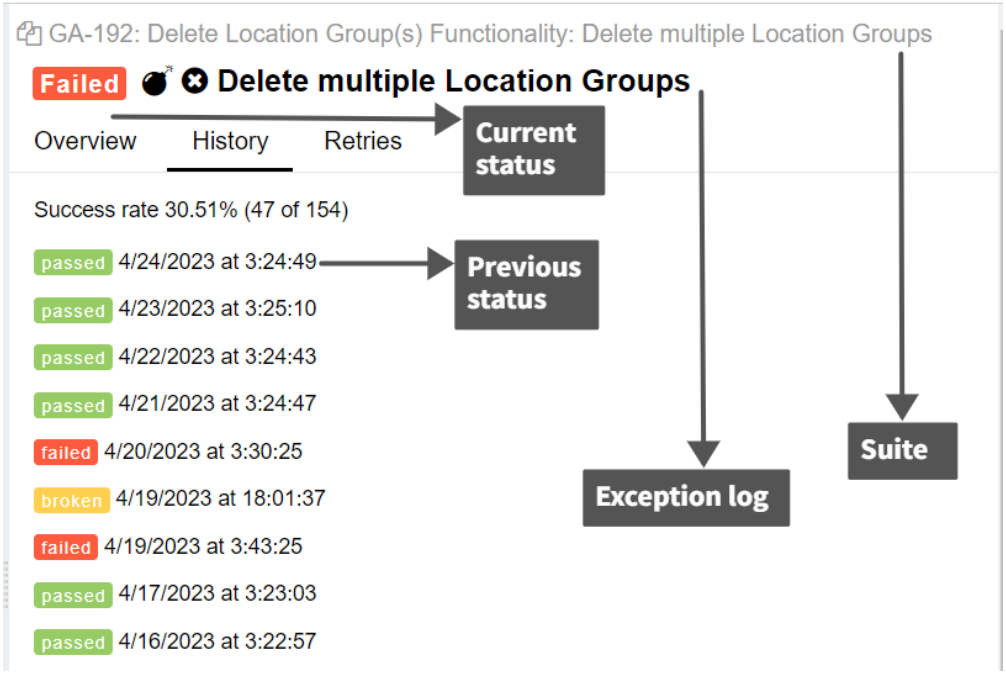

Figure 3.2: Suite's Historical Records

The set of data collected after seeking for historical records is saved into a CSV file, with three columns- message, suite, and label.

In the context of data analysis, it is often necessary to combine data from multiple sources to get a complete picture of the situation. This process is called 'Data Integration'. In this case, we are talking about joining two CSV files that were previously saved. The first, collected from the exception logs along with their types and the second was collected from suites' histories. By combining these two CSV files based on their common column, ['message'], we were able to create a complete, solid data set that contains information on both the exception logs and the tests that were running.
In order to combine these CSV files, there are multiple tools and software available for the task, including Structured Query Language (SQL) databases or data analysis software like Python, which was utilized in our particular case. These tools enable us to efficiently join and merge the CSV files, facilitating the integration of data and enabling further analysis. Figure 3.3 represents a subset of the final data.

```
message,suites,label,type
KeyboardInterrupt,Compare the whole grid: API to Config file Promotion Grid,1,test
AttributeError: 'Context' object has no attribute 'seleniumHelpers',GA-191 : Clone a Location Group,0,test
AssertionError,Verify the Layout of LG PopUP,0,product
```

Figure 3.3: Final Data

The resultant CSV file obtained from the data integration process holds significant value as it serves as a foundation for further analysis and decision-making. It is worth noting that the entire data collection process has been seamlessly integrated into the automation script.

3.2 Natural Language Processing Techniques

We have applied the following data processing techniques to our exception logs:

- Remove Punctuation
- Remove Stop Words
- Remove Duplicated Rows

Figure 3.4 illustrates an instance of an exception log, while figure 3.5 showcases the log after the removal of punctuation. Finally, figure 3.6 displays the exception log with stop words excluded.

```
"AttributeError: 'Context' object has no attribute 'InputData'"
```

Figure 3.4: Initial Exception Log

.

```
'AttributeError Context object has no attribute InputData'
```

Figure 3.5: Exception Log without Punctuation

.

```
"['AttributeError', 'Context', 'object', 'attribute', 'InputData']"
```

Figure 3.6: Exception Log without Stop Words

3.3 Feature Engineering

Feature engineering is a critical step in AI as it helps to improve the performance of models by creating new features that can capture more information from the data. This process involves using the domain knowledge to identify key patterns and relationships in the data set and creating new features that can capture these relationships. To fulfill feature engineering, we employed:

- Feature extraction: involves identifying the most relevant features in the data set and creating new features based on these features. For that, this entails extracting the primary error from the exception log because the exception log's message includes extraneous words that are not relevant to the main error. Consequently, we conducted a thorough examination of the exception logs to pinpoint the precise location of the main error.

- Feature transformation: incorporates transforming the data into a new representation that is more suitable for the model. For that purpose, we encoded all categorical columns.
  In the context of multi-class classification, it is necessary to convert class labels into a numerical format that can be effectively utilized by the model. This is achieved through a technique called 'One-Hot Encoding'. Through it, each class label is transformed into a binary vector with a length equals to the number of classes. Each element in the vector corresponds to a distinct class, where the index representing the class label is assigned a value of 1, while the remaining elements are set to 0.
  Even if the class labels are initially represented as integers, employing 'One-Hot Encoding' is still crucial for training a multi-class classification model. This approach ensures that the model does not assign an unintended order or significance to the integer labels, which may not accurately reflect the true nature of the data. By utilizing 'One-Hot Encoding', we can appropriately prepare the class labels for effective model training and interpretation. Figure 3.7 illustrates a segment of the training labels, while figure 3.8 specifically showcases the first element of the training labels, which belongs to the 2nd class, after undergoing the process of 'One-Hot Encoding'.

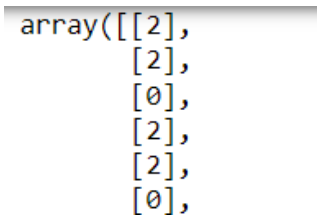

Figure 3.7: Sample of Training Labels

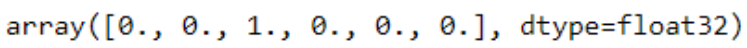

Figure 3.8: The Effect of One-Hot Encoding

- Feature selection: comprises selecting the most important features and discarding the less important ones. For that, we selected the useful columns to be faded to the model.
- Define 'Quality Assurance Issues': this operation was done using a switcher that replaces the previously extracted main error by its respective QA issue. We figured out the different QA issues of an error thanks to the QA team expertise and referring to Selenium documentation. The final set of QA issues is summarized in the table 3.1:

Quality Assurance Issue	Definition
Mismatched Values	occurs when there is a discrepancy between the expected and actual values of a variable or parameter in a software system.
Server Related Bug	error that occurs on the server-side of a client-server software system. This bug can impact the overall functionality, performance, and security of the system.
Regression Bug	happens when a feature was tested and validated however it is now no longer working as expected.
Project Structure Error	happens when the automated testing script has a wrong step.
Maintenance	refers to problems or challenges that arise in maintaining a system or equipment in good working condition. Maintenance issue can vary depending on the type of equipment or system being maintained.
Malformed Payload	occurs when a client sends a request to a server with a payload that does not conform to the expected format or structure. A payload is the data that is sent in a request or response message between a client and a server.
Technical Issue	occurs for various reasons: system takes long time to load for example

Table 3.1: Quality Assurance Issues

The entire process of data processing was integrated into the automation script of data collection to output a processed data, ready for splitting.

3.4 Data Splitting

We partitioned our data into two sets, namely the training set and the test set, with sizes comprising 80% and 20% of the total data, respectively. For each set, we split it into features

X and target Y. Figure 3.9 is a pie chart representing the percentage of each set as follows:

$$X\_train\_percent = \frac{\text{len}(X\_train)}{\text{total\_data}} \times 100 \tag{3.1}$$

$$Y\_train\_percent = \frac{\text{len}(Y\_train)}{\text{total\_data}} \times 100 \tag{3.2}$$

$$X\_test\_percent = \frac{\text{len}(X\_test)}{\text{total\_data}} \times 100 \tag{3.3}$$

$$Y\_test\_percent = \frac{\text{len}(Y\_test)}{\text{total\_data}} \times 100 \tag{3.4}$$

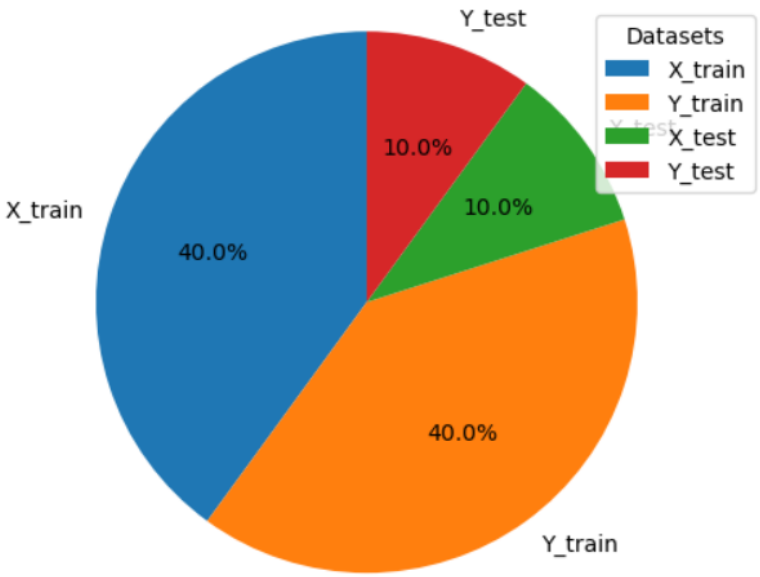

Figure 3.9: Data Split

In order to preserve consistent class distribution, we employed the 'Stratify' parameter in the train_test_split function from the Scikit-learn library. By assigning the target variable's values to this parameter, we ensure that the class distribution is balanced and maintained in both the training and testing sets.
The 'Stratify' parameter acts as a valuable tool for mitigating potential bias and maintaining the integrity of the data during the data split process.

3.5 AI Algorithm Development

In this section, we present two algorithms designed to handle distinct types of data: numerical data and textual data.

3.5.1 Model 1 - Numerical Data

In our study, we made the decision to leverage deep learning[26], specifically utilizing a Deep Neural Network (DNN) model, to handle the numerical data, including encoded error and encoded type.
A neural network is a type of machine learning model that is inspired by the structure and function of the human brain. It consists of layers of interconnected nodes or neurons that process and transmit information. Each neuron takes input from one or more other neurons, performs a computation on that input, and then sends its output to other neurons in the next layer of the network. Figure 3.10 illustrates the neural network architecture where:

- Bias: is a value that is added to the weighted sum of inputs in a neuron before the activation function is applied to adjust the predictions. Bias can help a neural network to better fit the training data by allowing it to model complex relationships between inputs and outputs.

- Basis functions: a set of functions that are used to transform the inputs of a network into a higher-dimensional feature space where they can be more easily separated or classified. The choice of basis functions can have a significant impact on the performance of a neural network, as it determines the complexity and expressive power of the model. Commonly used basis functions in neural networks include polynomial functions, radial basis functions, sigmoid functions, and Fourier series. By using basis functions, a neural network can learn more complex and non-linear relationships between inputs and outputs.

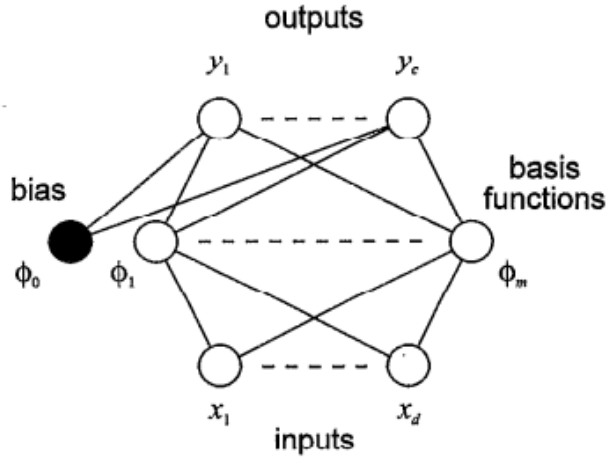

Figure 3.10: Neural Networks Architecture[27]

The layers used in our DNN model are:

- Input Layer: is the first layer of the network, which receives the input data. It defines the shape of the input data that the network expects, which determines the number of neurons in the input layer. Our input shape equals to (n, 2) where n is the number of collected exception logs and 2 refers to: encoded error which is the exception log after processing, and encoded type.

- Hidden Layer(s): is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of the previous layer. This layer is the most commonly used layer in neural networks.

- Output Layer: is the final layer, in a neural network model, that produces the output of the model. The number of neurons in the output layer depends on the type of problem we are trying to solve. For our project, the goal is to predict the class label of an input from a set of more than two classes, the number of neurons in the output layer, is the number of QA issues.

3.5.2 Model 2 - Textual Data

For the textual data, we constructed a machine learning [28] model using scikit-learn library, that provides several tools for working with textual data such as:

- Sklearn.feature_extraction.text for feature extraction.
- Sklearn.naive_bayes for the implementation of Naive Bayes algorithm [29].
- CountVectorizer which counts the frequency of each word in a given text.
- Vectorizer to be fitted to the data and transforms it into a feature matrix in order to prepare exception logs for analysis.

The subsequent part focuses primarily on the algorithm for numerical data, while the inclusion of a cutting-edge solution for textual data remains optional and is intended for further enhancements in our work.

3.6 Model's Performance

The performance of an AI model can be defined as how well it can accomplish the task it was designed for. It is usually measured by metrics depending on the main goal the model is aiming.
In our classification problem, the performance of the model is measured by how accurately it can classify new instances into their respective categories. Generally, the better the model performs, the higher its performance metric will be.

The following set of hyper-parameters - epochs = 200, batch size = 32, and optimizer = Adam [30] with a learning_rate of 0.001 - yielded the best performance. The resulting model achieved an accuracy of 98.65% and a loss value of 0.15.
We opt for a Confusion Matrix to compute the model's performance with numerical values as shown in figure 3.11.

Figure 3.11: Confusion Matrix

The following graph 3.12 shows the variation of our two metrics, accuracy and loss, during the training process of the model. The x-axis represents the number of training epochs, and the y-axis represents the values of the metrics.
The plot contains four lines: training accuracy, validation accuracy, training loss and validation loss. The training accuracy and training loss lines show how the model is performing on the training data, while the validation accuracy and validation loss lines show how the model is generalizing to new data.

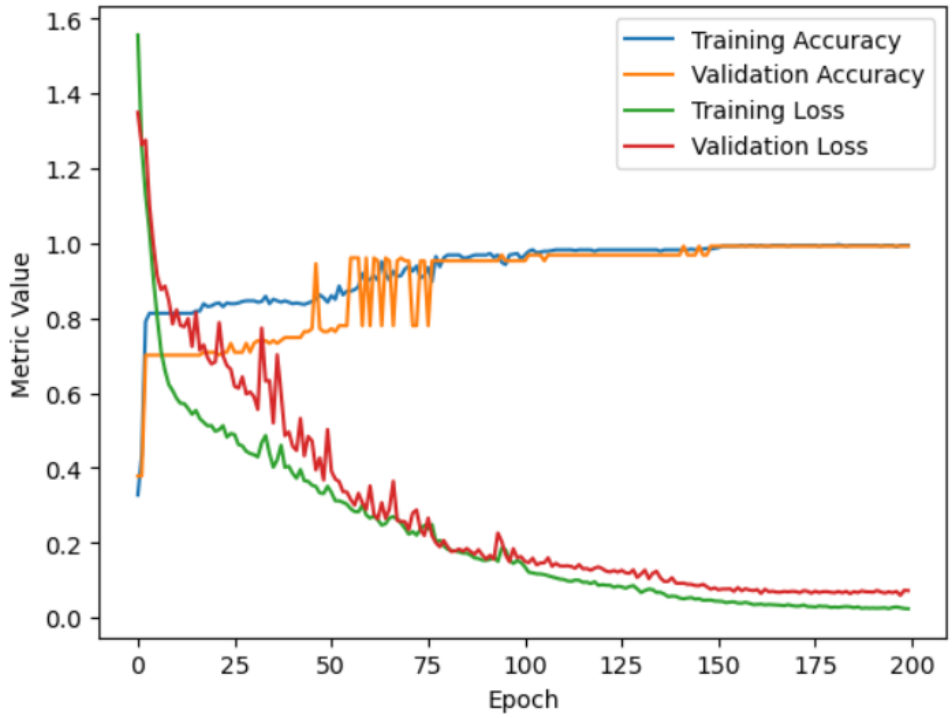

Figure 3.12: Training Progress of Model- Variation of Accuracy and Loss over Epochs

Analyse: the training accuracy line is increasing over the epochs, while the training loss line is decreasing. This indicates that the model is learning and improving its predictions on the training data. The validation accuracy didn't stop improving and didn't decrease while the training accuracy continues to increase. Both the training and validation accuracy lines are increasing and staying close to each other, which indicates that the model is learning and generalizing well. Similarly, the training loss line is decreasing over the epochs, while the validation loss line is staying relatively low.
The spikes in the training accuracy and loss curves and validation accuracy and loss curves indicate the model's perturbations during learning. These perturbations can occur for various reasons and can provide valuable insights into the behavior and performance of the model. By increasing the number of epochs, the model had more opportunities to learn and adjust its parameters, potentially improving its accuracy and reducing the loss. However, it is important to find the right balance when determining the number of epochs for training. If the number of epochs is too low, the model may not converge to an optimal solution. On the other hand, if the number of epochs is too high, the model might start to over fit the training data, capturing noise.
Spikes in the accuracy and loss curves during training and validation highlight perturbations in the model's behavior and provide insights into its performance. That helps us in finding the optimal number of epochs.
Overall, the plot helps us in monitoring the training progress of our model and ensure the stability and effectiveness of the trained model.

3.7 QA Intelligent Dashboard

3.7.1 Key Performance Indicators

A dashboard is an essential tool for monitoring and analyzing business performance. By incorporating a trained AI model into the dashboard, we can gather insights and generate predictions that can help us make informed decisions. The Key Performance Indicators (KPIs) displayed on the dashboard will provide a quick overview of the tests performance, enabling us to identify areas that require attention and take action accordingly.
KPIs are specific metrics used to measure progress and success in achieving desired outcomes. In order to generate our 'QA Intelligent Dashboard', the steps we went through were:

- Integrate the trained model: after saving the trained model as a '.h5' file extension, a convenient method for saving and loading models in the Hierarchical Data Format (HDF5) format, the need for integration was crucial. Then, we got predictions automatically after exception logs' extraction without any user intervention.

- Generalize output: associate each QA issue with a particular software issue, which in turn falls under a category of required investigation.

    1. Software Issues is broad enough to cover both bugs, technical issues and code defects, while also emphasizing on the fact that these are issues that need to be addressed in the software development process.

    2. Required Investigation name emphasizes that all issues need to be investigated to determine their root cause and potential impact on the system, regardless of whether they are actual bugs, non-bugs, or undefined. It also suggests that a thorough investigation is necessary before reporting any issue as an actual bug or not actual bug or undefined. It's crucial to emphasize that the term 'Undefined' denotes a newly identified category of predicted QA Issues that has not been previously defined.

    The table 3.2 below shows the software issues associated with each QA issue and specifies the investigation required for each one.

Quality Assurance Issue	Software Issue	Required Investigation
Mismatched Values	Bug	Actual Bug
Server Related Bug	Bug	Actual Bug
Regression Bug	Bug	Actual Bug
Project Structure Error	Code Defect	Not Actual Bug
Maintenance	Code Defect	Not Actual Bug
Malformed Payload	Code Defect	Not Actual Bug
Technical Issue	Technical Issue	Not Actual Bug

Table 3.2: QA and Software Issues with Investigation Categories

- Generate JavaScript Object Notation (JSON) file: this format is the best fit in our case since JSON data consists of one or more key-value pairs, where the key is always a string and the value can be a string, number, boolean or array. We generated the JSON file to: store the data in a structured format, share it once the team needs to share data between different systems or applications and use it as test fixture. Actually, by using a JSON file to represent test data, we can easily compare expected results whenever needed.

- Prepare statistics: calculate the rates of undefined, actual bugs and not actual bugs to have an overview of the output's distribution:

  1. Undefined_rate = round((`len(Undefined)` / (`len(Actual Bugs)` + `len(Negative Bugs)` + `len(Undefined)`)) × 100)

  2. Actual_bugs_rate = round((`len(Actual Bugs)` / (`len(Undefined)` + `len(Not Actual Bugs)` + `len(Actual Bugs)`)) × 100)

  3. Not_Actual_Bugs_rate = round((`len(Not Actual Bugs)` / (`len(Actual Bugs)` + `len(Undefined)` + `len(Not Actual Bugs)`)) × 100)

  Where len(Actual Bugs), len(Not Actual Bugs) and len(Undefined) represent the length (or number of elements) in the lists of Actual Bugs, Not Actual Bugs and Undefined respectively and round function rounds the parameter value to the nearest integer.

- Prepare charts: having some visualizations may be more comfortable for some engineers to figure out the majority of the issues they are going to deal with. Figure 3.13 shows

the predicted QA issues from an 'Allure Report', figure 3.14 represents the number of various software issues and figure 3.15 gives the required investigation.

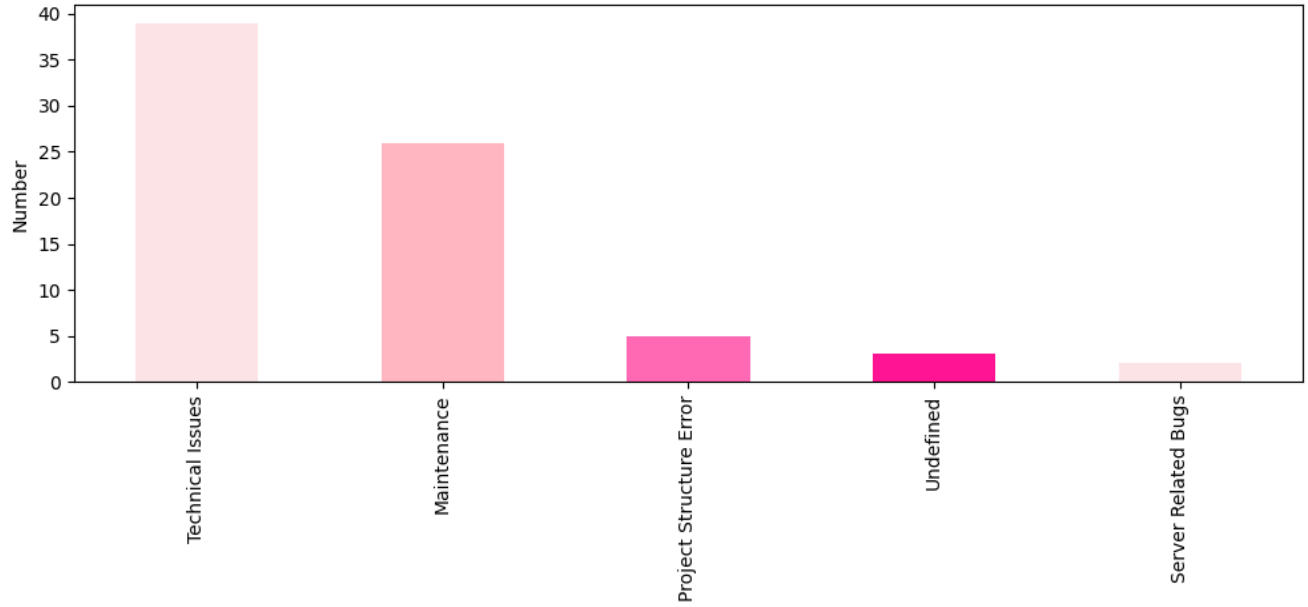

Figure 3.13: Prediction- QA Issues

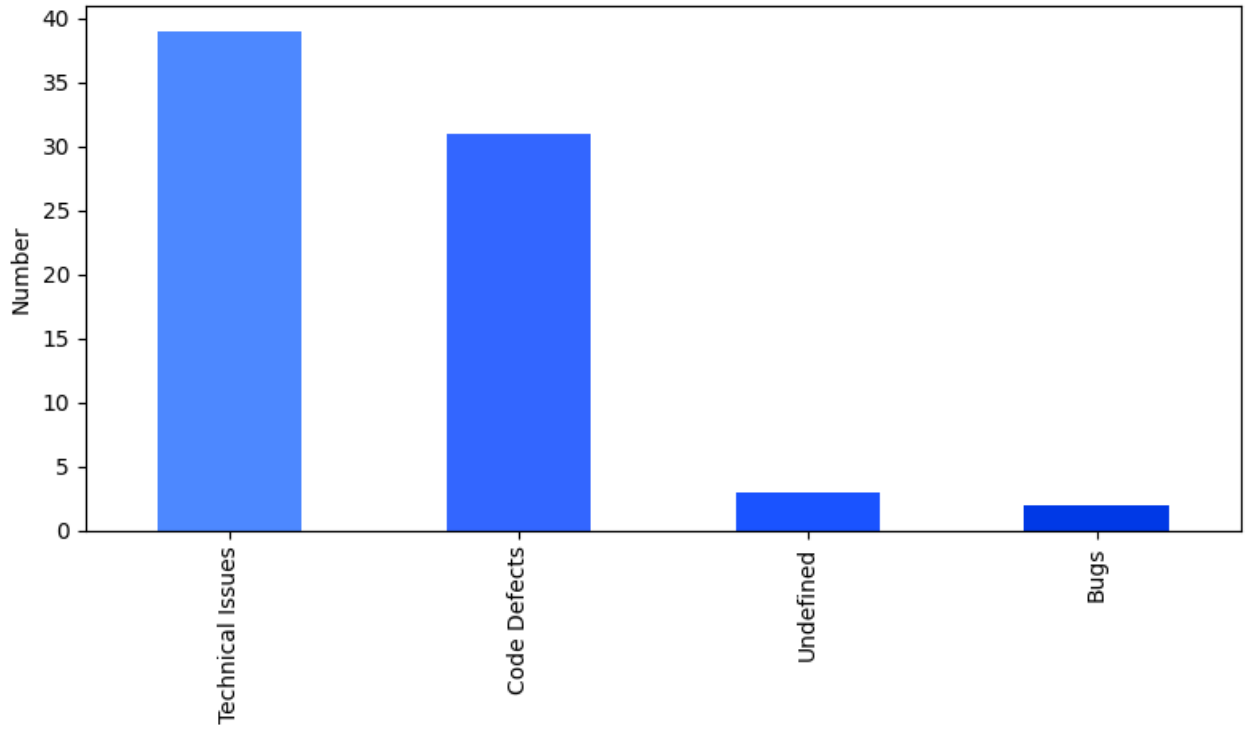

Figure 3.14: Assigned Software Issues

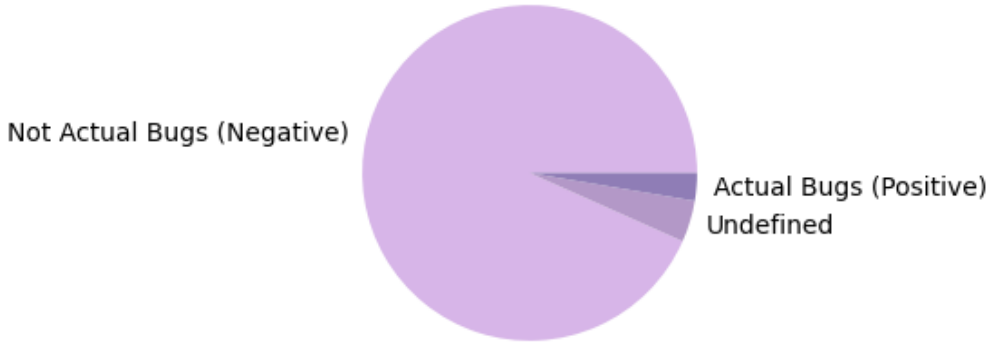

Figure 3.15: Final Required Investigations

3.7.2 Dashboard Generation

Dashboard generation refers to the process of creating a GUI that displays the KPIs in an easily digestible format. The dashboard can be generated automatically through various tools and softwares. In our case, we opt for Plotly Dash.
Plotly Dash is a Python framework for building interactive web-based dashboards, data visualizations, and applications. It is built on top of Flask, React.js, and Plotly.js, and provides a high-level API for creating dashboards with a user-friendly interface.

Figure 3.16: Plotly Dash Logo

Dash allows users to create interactive web applications with a Python backend that can easily be deployed on the web. It supports a wide range of visualizations, including line charts. Additionally, it allows users to customize the look and content of the dashboard with CSS styling and JavaScript callbacks.

3.8 Additional Work

As the required work of the project was done earlier than expected, an additional part was added to the project, which is holding edge cases when there is no 'Allure Report'.
During pipeline execution, issues that lead to an aborted or broken build may occur. In such cases, no 'Allure Report' is generated, resulting in the loss of relevant execution information. However, the console log always exists and can be used to retrieve output information.

In Jenkins, the 'Full Log' refers to the comprehensive output generated by a build or job in the console log. This log provides detailed information about the job or build, including warnings, errors, and the status of each step. As the build or job executes in Jenkins, the

console log is updated with real-time output information, which can be accessed through the Jenkins web interface. This information is useful for troubleshooting build issues and identifying areas for improvement. By default, Jenkins retains console log output for a set period before automatically deleting it. However, users can choose to archive console log output for a specific build or job, enabling them to review the output later. Figure 3.17 contains an aborted build (ID: #535), where no 'Allure Report' is generated.

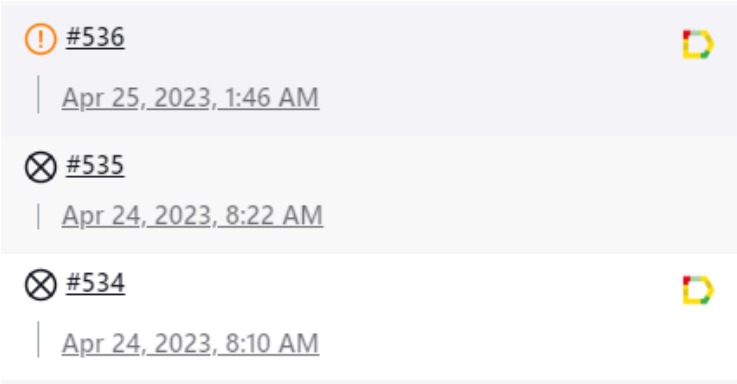

Figure 3.17: Aborted Build

Despite the tiny number of aborted builds, we tried to eliminate any loss of information, so we push the boundaries and added the exception logs extraction from the console output using automation and our AI model for textual data, so we:

- Access the console output of the build and extract the text
- Keep only the lines which contain an exception log and discard those of installations and setups.
- Predict the QA issue using the AI model for textual data where the only input is the exception log, because we don't have the defects type in the console output.

3.9 Conclusion

In this chapter, we have gone through data processing, data splitting, and the construction of the AI algorithm which are all crucial components in developing an accurate and effective solution. The performance of the model, in terms of accuracy and loss value, reflects the success of the aforementioned steps. We discussed the generation of a dashboard that can provide a user-friendly interface for visualizing the model's output and facilitating the decision-making based on the insights gained from the KPIs it contains.

General Conclusion

In this project, titled 'Automated Exception logs Classification using Natural Language Processing', our primary goal was to develop an AI-based solution that could effectively classify exception logs, thereby reducing the not actual bugs rate. We defined the Key Performance Indicators (KPIs) based on the model's output. Additionally, we created a user-friendly interface called the 'QA Intelligent Dashboard' to display the classification results in an easily understandable manner.

To achieve our objective, we employed natural language processing techniques and deep learning algorithms to process and classify exception logs obtained from the 'Allure Report' of Jenkins pipeline builds. By integrating our system into a developed automation script, we were able to automate the collection and processing of exception logs, minimizing the need for manual execution, which is typically time-consuming. The project successfully fulfilled its objectives by significantly improving the depth of classification for exception logs and providing detailed insights to the QA team regarding the performance of their automated tests. The 'QA Intelligent Dashboard' proved to be a valuable tool for the QA team, enabling them to visualize and analyze the classification results through charts and statistics. Throughout this project, we encountered and addressed various challenges and limitations associated with applying AI in industrial applications. Developing an effective AI solution necessitated careful consideration of data collection methodologies, feature engineering techniques, and model selection processes as well as security and scalability. By navigating these challenges, we gained valuable insights into the practical implementation of AI in real-world scenarios.

Overall, the project illustrates the immense potential of AI solutions in enhancing the efficiency and results accuracy in information technology, particularly within the domain of quality assurance. The generated 'QA Intelligent Dashboard' has the potential to be further improved and expanded with more features facilitators for QA. We recommend further research into integrating our solution with other existing tools and systems used to enhance the overall testing and quality assurance processes. This collaboration could yield even more significant improvements and advancements in the field of AI-driven quality assurance.

Bibliography

[1] G. Blokdyk. *Project Charter: The Definitive Handbook*. South Carolina, United States: CreateSpace Independent, 2017.

[2] E. Koester. *Inc. 5000*. https://www.inc.com/inc5000/2019. Accessed: 2023-02-27.

[3] Amanda Hilgers. *Cognira website: promo management*. https://cognira.com/promotion-management-solution-for-retail/. Accessed: 2023-02-27.

[4] A. Hilgers. *Cognira: FaaS*. https://cognira.com/forecast-as-a-service/. Accessed: 2023-02-27.

[5] A. Hilgers. *Cognira: Assortment Allocation*. https://cognira.com/assortment-allocation/. Accessed: 2023-02-27.

[6] Nayan B Ruparelia. "Software development lifecycle models". In: *ACM SIGSOFT Software Engineering Notes* 35.3 (2010), pp. 8–13.

[7] I. Nuinaja. *Scrum framework definition*. https://www.digite.com/agile/scrum-methodology. Accessed: 2023-03-01.

[8] L. Quartie. *Scrum visualized*. https://powerslides.com/powerpoint-business/project-management-templates/agile-scrum-process/. Accessed: 2023-05-03.

[9] J. Sutherland. *Scrum master definition*. https://www.atlassian.com/agile/scrum/scrum-master. Accessed: 2023-03-01.

[10] Edward F McDonough III. "Investigation of factors contributing to the success of cross-functional teams". In: *Journal of Product Innovation Management: An International Publication of the Product Development & Management Association* 17.3 (2000), pp. 221–235.

[11] D. Leffingwell. *Product owner definition*. https://www.scaledagileframework.com/product-owner/. Accessed: 2023-03-01.

[12] R. Ming. *Sprint review definition*. https://www.scrum.org/resources/what-is-a-sprint-review. Accessed: 2023-03-01.

[13] R. Ming. *Daily scrum standup definition*. https://www.scrum.org/resources/what-is-a-daily-scrum. Accessed: 2023-03-01.

[14] O. Jainai. *Activity diagram of code review process*. https://svitla.com/blog/code-review-and-its-important-role-in-software-development. Accessed: 2023-03-01.

[15] S.A. Mubarak. In: *Bar Gantt Charts*. 2010. URL: https://www.researchgate.net/publication/319359193_BarGanttCharts.

[16] A. Regata. *Definition of JIRA epic*. https://www.javatpoint.com/jira-epic. Accessed: 2023-03-02.

[17] K. Hamilton. *Learning UML 2.0*. VORT Yotsuyasakamachi Bldg. 1F, Tokyo, Japan: Oreilly, 2006. URL: https://www.oreilly.com/library/view/learning-uml-20/0596009828/ch01.html.

[18] A. Barone. *Behavior Driven Development*. https://www.mobileappdaily.com/behavior-driven-development. Accessed: 2023-04-06.

[19] Meral Bozdemir, Turgay Tugay Bilgin, and Kader Nikbay Oylum. "A New Hybrid Software Testing Automation Framework". In: *The European Journal of Research and Development* 3.1 (2023), pp. 200–211.

[20] A. Barone. *Page Object Model*. https://blog.e-zest.com/page-object-model-in-selenium. Accessed: 2023-04-16.

[21] A. Barone. *Data Life Cycle*. https://atrium.ai/resources/data-quality-the-atrium-difference/. Accessed: 2023-05-10.

[22] Matthias Feurer and Frank Hutter. "Hyperparameter optimization". In: *Automated machine learning: Methods, systems, challenges* (2019), pp. 3–33.

[23] Xue Ying. "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022.

[24] Mohammad Hossin and Md Nasir Sulaiman. "A review on evaluation metrics for data classification evaluations". In: *International journal of data mining & knowledge management process* 5.2 (2015), p. 1.

[25] Sofia Visa et al. "Confusion matrix-based feature selection." In: *Maics* 710.1 (2011), pp. 120–127.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[27] Chris M Bishop. "Neural networks and their applications". In: *Review of scientific instruments* 65.6 (1994), pp. 1803–1832.

[28] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9 (2020), pp. 381–386.

[29] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. "Naïve Bayes." In: *Encyclopedia of machine learning* 15 (2010), pp. 713–714.

[30] Ange Tato and Roger Nkambou. "Improving adam optimizer". In: (2018).