

NLP-based classification of test, test debt and test types in project management issues

Neha Durani

Masters Programming and system architecture

60 Credits

Department of informatics

Faculty of mathematics and natural sciences



NLP-based classification of test, test debt and test types in project management issues

© 2024 Neha Durani

NLP-based classification of test, test debt and test types in project management issues

<http://www.duo.uio.no/>

Printing: Reprosentralen, Universitetet i Oslo

Abstracts

Context

Software testing is a particularly important phase of the software development process which if not carried out properly causes issues. The proper identification and classification of these issues is essential to keep them under control and to ensure software quality. To maintain a robust testing process addressing test issues and debts is necessary so project managers can manage risks more effectively.

Objective

The objective of this thesis is to develop and evaluate modern machine learning approaches in automatic classification of test and test debt issues as well as identifying different test types within test issues in software projects. The objective is achieved by using transformers, rule-based approach and large language models. Through automatic classifications of these issues, the studies aim to reduce manual work required in managing test quality and test debt.

Method:

My research is quantitative analysis. The entire process involved data collection, preprocessing, classification using transformers, categorization based on ISTQB standards, and classification through large language models and comparative analysis of these classifications with already existing methods. So, the research design leans heavily towards a quantitative approach.

Results:

This thesis finds out the transformer-based approaches (DistillBert and Roberta) are excellent in classifying with high accuracy of (91%, 87.83% respectively) for test and 81% for test debt issues. The thesis finds out that functional and change testing related issues are most common in projects. And the approach suggested in this thesis has almost 6.58% higher accuracy than SVM and 80% higher

accuracy than TextCNN. Also, the model trained on the data tagged by developers has higher (3.85%) accuracy than the model trained on data tagged by researchers.

Conclusion:

The results from the thesis suggested that use of advanced machine learning techniques can help in identification of test issues and test debt that can help the project managers to identify the issues describing sub-optimality in testing that can create issues in future. These advanced machine learning techniques have higher accuracy than already existing approaches. Also, the attempt to classify the test issues in different test types has good potential to identify which type of testing triggers more issues and debt. Finally, the usage of datasets self-tagged by developers and researchers tagged yields almost similar results for classifying test debt.

Acknowledgements

This thesis outlines the conclusion of my degree in informatics, which was both challenging and rewarding at the same time. Throughout the journey there were some individuals without their assistance, this thesis was not possible.

I am sincerely thankful to my supervisor Antonio Martini for their valuable advice, support, guidance and help throughout the year. Special thanks to my co-supervisor Karthik Shivashankar for his excellent support and guidance in the whole technical part of thesis. Without their help and mentorship, it was not possible to do the whole work.

To my family who supported me throughout this journey, thank you!

Neha Durani

Oslo, May 2024

Contents

Table of figures:.....	ix
List of tables:	x
Chapter 1	1
1. Introduction.....	1
1.1 Research problem:	3
1.2 Research questions:	4
1.3 Structure:	5
Chapter 2:	7
2. Background.....	7
2.1 Software Testing and test issues:	9
2.2 Type of testing:	10
2.3 GitHub open-source projects:	11
2.4 Technical debt:	12
2.4.1 Test debt:	13
2.5 Natural language processing:	14
2.6 Machine learning:	14
2.7 Deep learning:	16
2.8 Related work for issues classification:	17
Chapter 3:	19
3. Methodology	19
3.1 Research design:	19
3.2 Research Methodology.....	22
3.2.1 Quantitative Component:	23
3.3 Data Collection:	24

3.3.1 GitHub Archives:.....	24
3.4 Data pre-processing:.....	25
3.5 Quantitative analysis workflow:.....	28
3.5.1 Text classification:.....	29
3.5.2 Implementing and Analyzing Transformer Models for Classification Tasks:.....	30
3.5.3 Classification of Testing Types using rule-based approach.	31
3.5.4: Classifying GitHub Issue into Testing Type using language model....	32
Classifying issues using traditional models and comparison with transformer models:.....	33
3.5.5 Comparison of datasets:.....	34
3.6 Data Ethics:	34
Chapter 4.....	36
4. Results	36
4.1 Collected data:	36
Final Dataset:.....	39
4.2 RQ1 Classifying project management issues as test related issues using transformers:	39
4.2.1 Transformer-based models (RoBERTa, DistilBERT) in classifying project management issues as test debt related issues:.....	45
4.3 RQ2: Most common types of testing issues in the dataset from GitHub	52
4.4 RQ3: To what extent are large language models able to classify project management test issues as specific testing types?.....	56
4.5 RQ4: Does the use of transformers improve the identification of testing debt over traditional methods (SVM, Text CNN)?	59
4.6 RQ5: How does the performance of transformer-based models trained on data with ground truth (labeled by developers) compare to the performance on	

manually labeled datasets (researchers) in identifying test debt in issue tracking systems?	62
Chapter 5.....	67
5.1 Discussion:.....	67
5.2. List of contributions:.....	78
5.3. Implication of research and practice:	79
5.4. Threats to validity:.....	80
5.5. Limitations:	82
Chapter 6.....	83
6.1 Conclusion:	83
Bibliography	85
Appendix.....	91
External materials:	91

Table of figures:

Figure 1 Overall research design for thesis.	21
Figure 2 Detailed research design with colors indicating the corresponding phase from overall design.	22
Figure 3 Data collection using GitHub archives.	25
Figure 4 Dividing data to 2 datasets.	27
Figure 5 Data cleaning.	28
Figure 6 Initial extracted data with multiple event types.	36
Figure 7 Dataset before pre-processing.	37
Figure 8 Dataset after pre-processing and cleaning.	38
Figure 9 Distilled Bert Validation loss, accuracy, and training loss test/not_test issues.	41
Figure 10 RoBERTa Validation loss, accuracy and training loss test/not_test issues.	43
Figure 11 DistilBERT Validation loss, accuracy, and training loss on testDebt/not_testDebt.	46
Figure 12 RoBERTa Validation loss, accuracy, and training loss on testDebt/not_testDebt.	50
Figure 13 keywords for identifying types of tests issues.	52
Figure 14 Regex for identifying types of tests issues.	53
Figure 15 Pie-chart indicating percentage distribution of test types in dataset..	55
Figure 16 Output scheme and prompt template for OpenAI model.	56
Figure 17 Pie-chart indicating percentage distribution of test types in dataset..	58
Figure 18 DistillBert Validation loss, accuracy, and training loss on testDebt/not_testDebt. labelled by researchers.	64

List of tables:

Table 1 Hits per google scholar search.....	8
Table 2 Summary of data collected.	38
Table 3 Distilled Bert performance on training data test/not_test issues.....	42
Table 4 Distilled Bert performance on test set	42
Table 5 RoBERTa performance on training data for test/not_test issues	44
Table 6 RoBERTa performance on test set	44
Table 7 DistilBERT performance on training data for testDebt/not_testDebt dataset	47
Table 8 DistilBERT performance on test set for testDebt/ not_testDebt dataset.	47
Table 9 RoBERTa performance on training data for testDebt/not_testDebt	49
Table 10 RoBERTa performance on test set for testDebt/ not_testDebt dataset.	50
Table 11 Classification of test types using rule-based classification.	55
Table 12 Classification of test types using large language models.....	57
Table 13 Comparison of LLM and rule-based approach.....	58
Table 14 SVM performance on training data for testDebt/ not_testDebt dataset.	60
Table 15 SVM performance on test data for testDebt/ not_testDebt dataset.....	60
Table 16 TextCNN performance on training data for testDebt/ not_testDebt dataset.	61
Table 17 TextCNN performance on test data for testDebt/ not_testDebt dataset.	62
Table 18 External dataset structure and labelling criteria.	63
Table 19 DistillBert performance on training data manually labelled by researchers.	64
Table 20 DistillBert trained on dataset labelled by researchers' performance on test dataset.	65
Table 21 Comparison of results of models trained on different datasets on test data.	66

Chapter 1

1. Introduction

Software is a set of machine-readable instructions that are given to computers to perform specific tasks. With various software's surrounding our lives, we are more dependent on software than ever before[1]. With such a significant role in our lives it is especially important to ensure the quality of them. Software testing is one of the frequently used methods to evaluate and ensure the quality of software [2]. The two main goals of testing are to make sure that the system that is developed is according to customer needs and to identify bugs. Testing is an important form of quality assurance[3]. To ensure the quality there should be well defined testing process which can be used by development team to monitor control as well as adjust the testing activities[4]. Any compromises in the software testing can cause huge loss[5, p. 137].

Testing is a systematic process which ensures that the system operates in accordance with the specifications. Despite its systematic nature, the process is prone to challenges that manifest as '**test issues**' which are recorded as tasks in project management tools. These issues refer to any differences between the actual and expected outcome of software under test. These issues can occur due to lack of coordination, reduction in test time, insufficient user participation, unsatisfactory understanding, insufficient team and inadequate documentation[3]. Identification and resolution of test issues are fundamental for supporting the software development lifecycle's integrity. If these issues remain unaddressed, they can lead to unreliable and insecure software that fails to meet user or market needs.

Testing is carried out throughout the development life cycle to ensure adequacy in software quality. However, when time constraints or resources are insufficient than required, companies usually reduce the tasks and practices related to software testing[2]. Because of this project member may take some decisions that can achieve fast delivery and some competitive advantage but these decisions are

very critical as they can directly affect the quality of software offered to customers[1]. These technical decisions taken in software projects may bring short-term advantages, but they are in long-term may be detrimental to project quality, and these sub-optimal decisions are defined as technical Debts.

Technical debt is a metaphor reflecting the implementation or adoption of sub-optimal solutions to achieve short-term benefits while sacrificing the long-term maintainability and evolvability of software[6]. This is the term first introduced in 1992 and was inspired by the financial theory of economics[7]. Some analyses have addressed TDs in other phases of the software development lifecycle (for example Tests, requirements, documentation) and provided solutions to manage them in software projects. For example, Technical Debts that are related to software testing are known as Test Debts. They arise when inadequate decisions regarding testing activities (for example lack of tests, test estimation errors)[1].

Identification of diverse types of technical debts helps the project manager to manage and prioritize the efforts and resources needed for them [6]. After identifying technical debts of various kinds, doing some statistics on those types can give us valuable results such as the mean time of solving debts or debts that get solved easily and early or debts that never get solved. Additionally, understanding the lead time required to solve diverse types of technical debt can provide valuable insights into the impact of technical debt on project timelines and resource allocation [8].

The technical debt can be present in more than one form in a software. Some technical debts are issues at the code level, for example: duplicated code, poor coding standards, and unnecessary lines of code, poor logic, or lack of comments in code. These are known as code smells and code analysis techniques can be adopted to identify these types of debts[9].

The other form of technical debt has large granularity and can't be detected using code analysis for example decision to postpone the automation of tests and using manual testing instead because of time constraints or to save resource, can accumulate test debt. While manual testing can be effective and fast in early

stages, the lack of automation becomes a block as the project scope becomes a border, increasing the time and resources required for testing exponentially[10].

This thesis aims to explore modern and new machine learning based approaches to identify and classify test issues and test debt in issue tracking system GitHub as a complement to already existing approaches. The identification and classification not only help in improved issue management but also enhance test quality. This research used the issues data available in open-source projects in GitHub as it is widely used to record, track, and prioritize various kinds of issues in software projects. As various approaches to identify technical debt already exist my research is mainly focused on issues and debt related to tests.

Though, many attempts has been made to understand test debt[9][10],but the dimensions like identification, classification of the test issues and test debt and types of test issues in software projects has not been discovered well. As testing play an important role in software development and there is huge effect of test debt on software project, so corresponding management techniques for example identification and classification of test issues and test debt deserve more focus for developing high-quality software in industrial contexts and to avoid lack of testing[1]. Despite the existence of some methods that manually identify test debt in code comments or from developer's discussion, or using some conceptualized vocabulary but these approaches are either manual or utilize traditional machine learning and manual dataset creation which does not contain ground truth. My methodology is an automated approach that will utilize powerful machine learning approaches to identify as well as see effectiveness of these approaches and comparison of the approaches used with traditional machine learning methods. This approach can be used by developers, testers, quality assurance team and project managers to automatically identify test and test debt issues that would save time.

1.1 Research problem:

Based on the literature as discussed above the research problem is defined as follows:

“Classifying issues reported by developers in test, test debt issues and different test types by using machine learning and large language model’s techniques to automate the analysis of textual data”.

1.2 Research questions:

RQ1: How accurate are transformer-based models (DistilBERT, RoBERTa) in classifying project management issues as **test** related issues?

RQ1.1: How accurate are transformer-based models (DistilBERT, RoBERTa) in classifying project management issues as **test debt** related issues?

RQ2: What are the most common types of testing issues in the dataset from GitHub?

RQ3: To what extent are large language models able to classify project management test issues as specific testing types?

RQ4: Does the use of transformers improve the identification of testing debt over traditional methods (SVM, Text CNN)?

RQ5: How does the performance of transformer-based models trained on data with ground truth (labeled by developers) compared to their performance on manually labeled datasets (researchers) in identifying test debt in issue tracking systems?

Above is the list of 5 research questions that are aimed to give answers to the research problem defined above. RQ1 and RQ1.1 are closely related to each other as they aim to classify the issues from the projects that are related to test and then use those issues to relate them to test debt. In these two research questions transformers-based approaches are used to classify the issues in categories (test, not test, test debt, not test debt). It is interesting to find that which of the issues are related to only tests and which are related to both test and test debt. For example, a bug find during testing is a test issue while lack of automated testing or flaky tests are test debt while outdated test scripts are both test and test debt issue as they may fail during the testing cycle and their continued use is test

debt. Also, two different transformer models are used, so their results are also compared to see which is more efficient and accurate and to provide diverse perspectives and validate robustness across two different models.

RQ2 is focused in finding different types of test (functional, non-functional, structural testing and testing related to changes) based on book foundation of software testing ISTQB certification [5]. RQ3 aims to provide information about the classification of test issues to specific testing types by using large language models which is **gpt-4-0125-preview**. It is focusing on how large language models can help in classifying the test issues as a specific test type. Also, a comparative analysis of these approaches is done as both are performing the same tasks but one is manual and other is automated approach.

RQ4 is comparing the results from research done on identifying self-admitted technical debt in issue tracking systems using machine learning[12] by Li et al., to find out if the use of modern machine learning approach i.e transformers is better than traditional machine learning approaches like SVM and Text CNN used in the previous research. This research question is also aiming to perform evaluation of our classifier to see how well it is generalizing to the unseen data as compared to previously done studies.

Finally, RQ5 is related to comparing two datasets. And how the choice of dataset labelled on ground truth by developers and the datasets labelled based on experience by researchers affect the training process of model and what are the performances of corresponding model on test data.

1.3 Structure:

Chapter 2 – Background

This chapter provides in-depth background information of all the concepts being utilized in the performing analysis and in writing results.

Chapter 3 – Methodology

The methodology chapter describes the methodological choices that were made during the thesis which include the research design and methodology, methods of data collection, cleaning, and analysis as well as research ethics.

Chapter 4 – Results

This chapter describes in detail the results obtained for each research question in order.

Chapter 5 – Discussion

This chapter describes findings from the results in chapter 4 combined with background from chapter 2 and related work from chapter 1 along with future work.

Chapter 6 – Conclusion

The main outcomes of the thesis are discussed in this chapter.

Chapter 2:

2. Background

This chapter will describe essential background theories and review past research findings and lay out ground to clarify the goals of this research. Firstly, an introduction of software testing and test issue is given that describes why testing is important in software development life cycle and what are test issues. Secondly, major concepts like technical debt and test debt are discussed briefly. Then what are various types of testing are being discussed. It is followed by a brief introduction of machine learning describing what are traditional and modern machine learning approaches and what transformers and large language models are. Lastly, an introduction of GitHub is presented as it will be relevant for the research context of the thesis.

A review of existing literature review is carried out to gain insights of the current state of knowledge on test issues, test debt and their relation to technical debt. The literature review helps to determine whether the topic is worth studying, and it provides insight into ways in which the researcher can limit the scope to a needed area of inquiry[35]. The literature review helped in finding the existing knowledge and figure out what areas are worth studying and how much research is already done in them.

“A literature review means locating and summarizing the studies about a topic.[35, p. 74]”

The first step of literature was identifying key words, which would be useful in locating materials in an academic library at a college or university. These key words helped in identifying a topic and result from preliminary readings. And the 2nd step is using those keywords in major libraries and computerized databases, and it is suggested to focus initially on journals and books related to that topic[35]. General databases, including Google Scholar, Web of Science, EBSCO, ProQuest, and JSTOR, cover a broad range of disciplines. In this mainly on most

famous search engine which is Google scholar ¹ is used and search. Table 1 shows the initial search terms and their corresponding results. The 1st row indicates the search term, and some search terms are within quotation marks and indicate exact search. The use of logical gates AND OR is utilized to get more concise and broad results. OR is used to get the articles that include any of the search terms while AND is used to get papers which are more relevant and have all the mentioned terms. There are 2 search results in table 1 where the search terms can be in any part of the document and this result has a lot of irrelevant information. The other one is with the search terms occurring only in the title of the document. It resulted in giving truly relevant data for the research that were utilized in doing the data collection. This data collection helped in figuring out the research problem which further assisted in writing research questions.

Search term	Search with words occur anywhere in article	Search with words occur in title	Include citations?
"Test issues" And " Test debt" AND "technical debt" AND "Software engineering"	3	0	Yes
Machine learning "test issues" OR "Test debt" OR "technical debt"	8950	44	Yes
Machine learning AND “test issues” OR "Test debt" OR "technical debt"	6 400	11	Yes

Table 1 Hits per google scholar search.

¹ <https://scholar.google.com/>

Most of the papers that were relevant were freely available on IEEE, ResearchGate, Wiley, ACM, and ScienceDirect. Then after Skimming through initial group of articles and chapters from some relevant books and those articles, chapters and paper were collected that were central to the topic. Throughout this process, a lot of information was found that made a useful contribution in understanding of literature.

2.1 Software Testing and test issues:

In the field of software development, the creation of code is just one part of the story. It is fundamental to make sure that this code does exactly what it is supposed to do as described by the customers. This is where software testing comes into play, resulting in the quality control mechanism of the software development process [4].

According to ISTQB:

“Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.”

It is an integral part of SDLC and often encompasses 50% of expenditure incurred in the whole process[13]. The complexity and time consumed during the process is highly dependent upon the complexity of the software. The time and resources needed depend upon the complexity of the project. However, the process of completing a well-tested software application is a challenge, collectively known as testing issues. These issues include a wide range of issues that can arise during the testing phase, from straightforward bugs in the code to more complex system-wide failures[14]. The complexity of testing issues makes them a significant challenge for both the developers and the testers.

Testing issues can have many forms, for example the individual simple piece of code that remained untested, and they are relatively easy to find and correct. However, the complexity of individual code defects is dependent on the complexity of more systemic testing issues. For example, integration issues occur

when the separately tested components conflict upon being combined revealing the problem that individual unit test has failed to test.

Furthermore, test issues expand beyond the technical level to include procedural and methodological challenges. Insufficient test coverage, for example, where parts of the application are not adequately tested due to lack of time or resources, can leave weaknesses unexplored. Additionally, performing more manual testing due to a lack of automated tests can lead to inconsistencies and inefficiency, preventing the development process and increasing the likelihood of human error[5].

As the indicator of quality, the testing must be a detailed process and the testing issue must be identified as well as prioritized and resolved as soon as possible. However, the traditional manual identification and classification of these issues are labor-intensive and prone to low quality. So, this thesis seeks to develop an automated approach using the latest advancements in machine learning that will analyze textual data from GitHub issues for classification of those issues.

2.2 Type of testing:

Software testing is of several types, each with a distinct purpose and each ensures quality and reliability of the software in different ways. We must consider all the different types of testing since testing only the functionality of the component or system may not be sufficient at each level to meet the test objectives[5]. According to the ISTQB certification book, there are four major types of testing each having more types. These are Functional testing, non-functional testing, structural testing, and testing related to changes and their definitions according to ISTQB certification book are as follow:

Functional testing is related to exploring and classifying issues related to testing the functions of a system or component or in simple testing what the system is supposed to do. It usually concentrates on aspects such as requirements-based testing and business-process-based tests. This type of testing emphasizes the importance of prioritizing tests based on risk criteria. Function (or functionality)

testing can be conducted based on ISO 9126, focusing on suitability, interoperability, security, accuracy, and compliance[5].

Non-functional requirements focus on quality characteristics or non-functional aspects of the system. Non-functional testing contains performance testing, load testing, stress testing, usability testing, maintainability testing, reliability testing and portability testing and some other types also. It evaluates 'how well' the system works[5].

Structural testing is a way of measuring the thoroughness of testing through the coverage of a set of structural elements or coverage items. It usually classifies issues that pertain to the structure of the system or component[5].

The final type of testing is testing related to changes that categorize issues related to testing changes in the software, including confirmation testing (re-testing) and regression testing. It also clarifies the purpose of regression testing as proving that modifications have not caused adverse effects.

2.3 GitHub open-source projects:

As discussed above, testing is an extremely vital component of the software development process. The issue reports are one of the important documents to track and fix bugs. Many systems provide issue reporting; however GitHub is one of the most popular and helps to trace back the issue to the developer[15].

GitHub, as a hub for collaborative software development, plays a crucial role in identifying and analyzing various kinds of issues in projects reported by software teams. These issues also include test issues. The platform's issue tracking feature helps in the documentation of bugs and errors and other methodological issues encountered during testing but also serves as an important communication channel.

For this thesis a rich dataset of real-world testing challenges is created, enabling an automated approach to classify and comprehend test issues at a rapid pace. The dataset contains the issues that are tagged by the developers themselves in

the issue description from different projects in the GitHub. The use of GitHub does not only streamline the identification of test issues and debt, but also contributes to the wider goal of improving software quality and reliability.

2.4 Technical debt:

As briefly discussed in the introduction Technical debt (TD) is a metaphor reflecting the implementation or adoption of sub-optimal solutions to achieve short-term benefits while sacrificing the long-term maintainability and evolvability of software[6]. This is the term first introduced in 1992 and was inspired by the financial theory of economics[7].

TD can have some short periods provide benefits for software development teams [16]. For instance, if the development team chooses to implement solutions that aren't optimal to have a faster delivery time i.e., a development team may find it advantageous to take some TD to ensure that they can deliver a business value faster. Although, as it was stated, TD is harmful when examining the medium-to-long-term perspective. This has led developers to believe that TD must be actively managed [17].

In the beginning, this term was just limited to be used in the context of code only but later it was extended to all the phases of the software development life cycle and the other types like design debts, architecture debt, code debts, testing debts etc also emerged[18]. The technical debt can be deliberate or non-deliberate. Non-deliberate technical debts are those that occur unintentionally i-e architecture decay and they are not tagged by anyone. While the other type is one intentionally incurred by the developer and they usually make notes on it (in comments, issues, description) for example using the words refactor etc. These deliberate technical debts are called the self-admitted technical debts[19].

Due to the high importance of this topic because of its potential to improve software quality, enhance project success rates, reduce resource allocation, mitigate risks, and enable informed decision-making throughout the software development process a lot of studies have been done to prioritize, detect and

manage technical debt and to identify different kinds of technical debt. Most common types of technical debt studied so far are Design debt, code debt, architecture debt, test debt, documentation debt, defect debt, infrastructure debt, requirements debt, build debt, people debt, service debt and versioning debt.

Technical debt can have negative impacts on the reliability and long-term evolution of software systems. It can also increase the cost and time required to implement new features or fix. Therefore, it is important to manage technical debt effectively by making it visible using some techniques, performing cost-benefit analysis, and controlling it according to the business goals [20]. A common metric used to measure technical debt is the technical debt ratio, which is the ratio of the repair cost of software against the overall build cost. A lower technical debt ratio indicates higher code quality and lower probability of technical debt[21].

The classification of technical debt in issue trackers, comments, or other labeling systems is valuable for identifying, prioritizing, and managing technical debt effectively. It enhances communication, aids resource allocation, and enables analysis to drive continuous improvement in software development projects[22].

2.4.1 Test debt:

In software development lifecycle if the time to deliver the project is less or available resources are less than what is needed or become scarce testing is usually the first in row for its activities to be withdrawn or reduced. If defects cannot be found, products quality can be affected[2].

In a software development lifecycle process as discussed above

“Aborted, or reduced activities that can provide short-term benefits, but can be detrimental to the project in the long run, are considered Technical Debt (TD).” When TDs impact testing activities, they are known as Test Debt. There are several studies dealing with test debt, however, current solutions often deal with specific types of tests (e.g., exploratory, and automated tests) and do not address the whole software testing process[10], [23].

Test technical debt (or test debt for short) occurs due to shortcuts i.e., wrong, or non-optimal decisions regarding testing. It is a test dimension of technical debt[2]. For example, when unit tests are completely absent (this scenario is common in legacy projects), it contributes to test debt. It is because not conducting unit testing is a shortcut. Although there is a short-term benefit of speeding up development, there is a long-term impact associated with that choice[1].

2.5 Natural language processing:

In this thesis, Natural Language processing is being used to classify GitHub issues to ‘test’ and ‘test debt’. NLP is a component of artificial intelligence that helps computers understand, interpret, and manipulate human language. It fills the gap between human language and computer understanding[24]. , the goal is to use NLP on GitHub issues to classify whether the issues are test and test debt issues or not. To achieve this purpose, NLP is combined with machine learning and an artificial neural network. Both of which are sub-fields of AI that will be outlined in this section, as well as in the forthcoming sections of the thesis.

To measure test issues and debt from the issues data from GitHub huge amount of data needs to be analyzed. Traditional algorithms might have low accuracy in classifying different types of issues from dataset. It can be especially hard to determine the context of an issue without a broad understanding that NLP and machine learning.

2.6 Machine learning:

Machine learning is a branch of artificial intelligence (AI) that addresses the issue of how to construct machines that improve automatically through experience without being explicitly programmed. It is today's rapidly growing technical fields, which are connected to computer science and statistics, and at the forefront of artificial intelligence and data science [25]. Learning refers to the ability to improve the results of a task when exposed to more data. Training in machine learning is a process of learning a machine learning model to make

predictions or decisions based on data. During the training process, the model progressively adjusts its parameters to minimize errors in its predictions [12]. Another term is featurizing engineering that is concerned with finding relevant variables to use from the raw data (which will be selected from the data from GitHub). More specifically, it involves using domain knowledge to select useful features (e.g., attributes, properties, and so forth) from the data. As a result, the raw data can be transformed into more useful data that can be utilized efficiently with machine learning[25].

There are three main approaches for machine learning which are Supervised machine learning, un-supervised machine learning and reinforcement learning. In supervised machine learning model learn to map input to output based on input-output pairs. In this kind of learning we have labelled training data and collection of training examples to infer a function. It is a task-driven approach that is used when certain goals are identified to be accomplished from a certain set of inputs. Most common example of supervised learning is classification and regression[26].

Text classification is one of the major tasks in machine learning that involves categorizing text into one or more predetermined categories. It is commonly used for tasks such as sentiment analysis, spam detection, topic labeling, and more. The process typically involves converting text data into a format that a machine learning model can understand and then training the model on a dataset where the categories are already labelled, and then using the model to predict the categories of new, unseen texts. A supervised machine learning approach along with deep learning for text classification has been employed in this thesis.

Traditional machine learning models used to classify the text into technical debt, or its subtypes are CNN, SVM, KNN and many more[19]. This thesis also compared the neural network-based machine learning approach transformers to traditional machine learning methods to see how well the neural network-based machine learning approach performed as compared to traditional machine learning approaches.

SVM is a supervised learning algorithm that utilizes robustness and effectiveness in high-dimensional spaces, which is often employed in text classification tasks due to the large vocabulary size. SVM can locate the hyperplane that is suitable for separating different categories in the feature space. It's especially effective for text classification because of its ability to handle both linearly separable and non-linearly separable data, making it a useful option for various text data structures[27].

CNNs is an effective technique for text classification tasks. It comprises of a convolutional layer over the text data, which is used to automatically locate spatial hierarchies of features. CNNs are especially good at picking up on local and spatial patterns in data which ability makes CNNs exceptionally useful for tasks where local context and text structure play a critical role in determining the meaning or category of the text [28].

2.7 Deep learning:

Deep learning is a innovative technology in the field of natural language processing. It is a subset of machine learning that uses neural networks with many layers (that is why called deep) to model complex patterns in data. Unlike traditional machine learning, deep learning automatically learns features directly from the data. This ability makes them extremely useful in classification of unstructured data.

Transformers are the most recent advancement in deep learning which consists of an encoder and a decoder architecture. The input set in the form of vectors, keys, and queries is passed through a linear layer and then through the scaled dot-product attention, which uses a SoftMax function. The results are joined, passed through a linear layer, and finally to the decoder architecture.

The decoder architecture has transformer blocks with masked multi-head attention to ensure that each consecutive output has knowledge of the previous input [29]. So, a variation of the transformer block can be used for text classification which has been utilized in this thesis.

The most popular transformer models for classification tasks are DistillBert, GPT, and RoBERTa. DistillBert, which stands for Distilled Bidirectional Encoder Representations from Transformers. It has been widely used in many natural language processing tasks due to its robust performance in sentence-level tasks. GPT, also known as Generative Pre-trained Transformer, is a language model that can generate coherent and relevant text. Roberta is an extension of BERT, which applies additional pre-training techniques to improve performance on downstream tasks[30].

2.8 Related work for issues classification:

In the past there were many methods that were used to classify different types of debts in code, comments as well as on issues from open-source projects. One of them is using conceptualized vocabulary to automatically classify different types of debts [31] in code comments. This method focuses on making a vocabulary and then using that vocabulary on code comments to see in which category that debt falls. The classification part was done using a tool **eXcomment**. This method is applicable for self-admitted technical debts that the developers have intentionally added such as using words like TO-DO's or refactor etc. But a lot of effort is utilized to make and update the vocabulary and is purely dependent on the skill of people making the vocabulary.

Another approach, the maximum entropy classifier was also used to automatically identify self-admitted technical debts [19]. The study used 10-fold cross project validation, nine open-source projects for training, and one project for validation. The results showed that the NLP improved accuracy compared with previous pattern-based detection used in the previous studies.

Wattanakriengkrai et al., used N-gram IDF [32], with multi-classification techniques, and a model was built that can classify a comment as design debt, requirements debt, or non-SATD. The datasets from the previous studies(eXcomment) were used here. A RF machine learning algorithm was applied to classify target comments. The result of the study was that N-gram IDF topped traditional techniques like bag of words (BoW). But this study just focused

on design and requirement debt. Yu et al., used another method for SATD in which there were two types of debts [33] i-e the easy to find and the hard to find and the concept was very simple the easy to find were those which were defined in the clearly in the comments while hard to find were those that needed experts to accurately identify them. Supervised machine learning is used to present the comment to the experts for identifying SATD that is not identified automatically. After that, the comments identified by the experts are reused by updating the model and through continuous training. Some of the techniques were also applied by Fahid et al., on the commit messages to see what their effects on SATD were on. In that study the relationship of SATD with the code commits was analyzed [34].

Chapter 3:

3. Methodology

3.1 Research design:

The most common types of the research designs in the field of computer science, IT and software engineering are descriptive, comparative, experimental, and quasi-experimental, cross-sectional, longitudinal, case study, exploratory and explanatory research design.

Experimental research on the other hand enables the research to have control of all the factors (variables) that could affect the results or outcomes of the research. It uses a statistical approach to establish the cause-and-effect' relationship between a group of variable factors in a study[36]. **Comparative study design** is used to understand the relationship among multiple objects, ideas, or parameters. This is achieved by observing and analytically comparing various aspects of these entities. The objective of **quasi-experimental** is to identify the cause-and-effect relationships between different variables. In a true experimental setup, researchers can assign subjects randomly to either the experimental or control groups, respectively. The objective of **correlational research design** is to determine the extent of the relationship between two or more variables that are closely related, using statistical analysis[36]

My research is based on finding the test related issues in project management and how they are related to test debt along with classification of test issues to different test types and this research utilized more than one above mentioned approaches. For RQ1 comparative approach is to see the effectiveness of two different transformer-based approaches in classification of issue to test and test debt. The data collection was mainly from the GitHub archives and data annotation is done by the developers. Comparative research design was suitable for these questions as the thesis is evaluating the performance of transformer-

based models that can help in providing diverse perspectives and validating robustness across varying architectures.

For RQ2 a quantitative approach is being applied and the dataset is analyzed to find different types of testing. For this rule based classification (regex) is used which is based on definition of different types of testing from ISTQB certification book[5]. Data collection was already done in the previous step and data-annotation is done using regex approach.

For RQ3 large language model is used to see how it classified the test types according to given prompts and how well is the accuracy of the model w-r-t transformers. In this question comparative analysis is used to see comparison between the rule based and machine learning based approach.

In RQ 5 comparison of transformer-based models with traditional methods (CNN, SVM) in identifying testing debt is done. For this comparative study design is used to see how well the traditional and modern machine learning approaches performed of the data from GitHub and on the data from other sources[12].

Finally, In RQ6 comparison of performance of models trained on 2 differently labelled datasets is done, utilizing comparative study design in this question too.

Figure 1 shows the overall research design which starts with preliminary exploration that was focused on finalizing the research questions and literature review that was relevant to the studies. In the next step in figure 1 is data collection and preparation that was focused on collecting and cleaning data from GitHub. After data collection is model training and testing where the machine learning, rule based, and traditional models were trained and tested. Following that is comparative analysis of the models where the outcomes from the model were tested. Finally, the comparative analyses are fed into the synthesis and conclusion step, from where the conclusion is drawn.

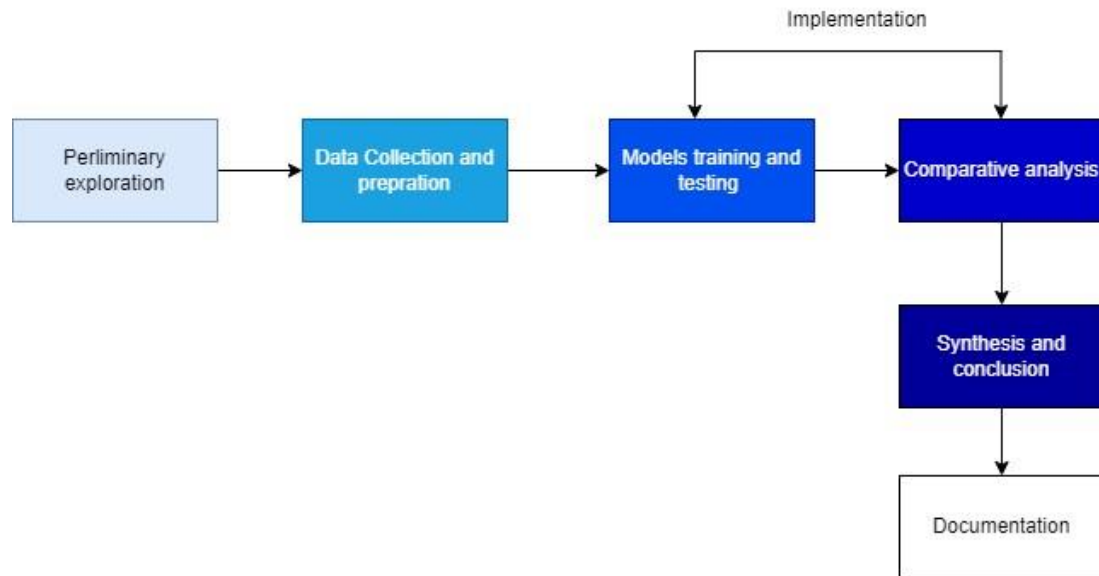


Figure 1 Overall research design for thesis.

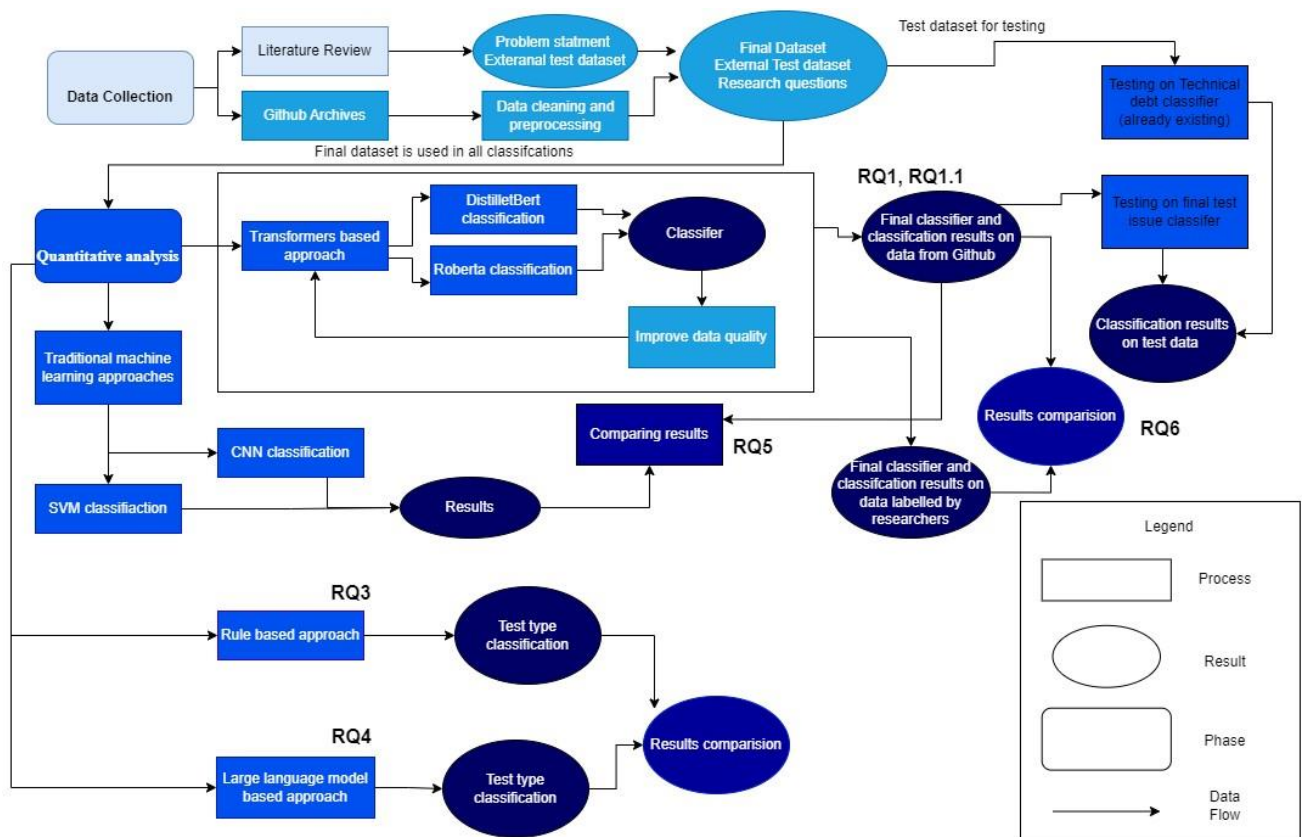


Figure 2 Detailed research design with colors indicating the corresponding phase from overall design.

Figure 2 explains the whole research design in detail. Each color indicates the step it belongs to in the overall design. The process starts with data collection which is done using the literature review and GitHub archives. The other processes are classification of test and test debt and test types using different models and the comparative analysis of the results. Figure 2 highlights the all the elements of quantitative analysis utilized in each research question. There are 4 types of quantitative analysis done in figure 2. The 1st one is transformer-based approach that resulted in two classifiers which are used for RQ1 and RQ5. The other two are rule based and large language model-based approaches used for classification of test types, and they answered RQ2 and RQ3. Lastly there are traditional machine learning approaches which answered RQ 4.

3.2 Research Methodology

“A research study is a process of pursuing ways to find the science behind natural and social events by study, experiment, observation, and comparison. The pursuit of knowledge through a systematic way of finding explanations for a phenomenon constitutes research methodology[37]”.

There are three types of methodologies for research that are quantitative research, qualitative research and mixed research[36]. Quantitative research is concerned with numerical results while qualitative research is related to words. While mixed methods research is a unique type of research that combines the factors of both quantitative and qualitative.

*“A **quantitative approach** seeks to quantify the research problem by emphasizing objective measurements using mathematical, computational, statistical and numerical analysis of data collected from documentation, surveys and questionnaires, in ways that ensure that the study has attributes of repeatability, transparency and credibility[36].”*

*“The **qualitative research approach** is concerned with unstructured, or semi-structured, non-numerical data[38] . It seeks to understand a specific population’s experiences based on the individual perceptions (or words) of members of that study population. It uses narrative, not numeric, data[36]”.*

*“A **mixed-methods research** is a type of research methodology with its own philosophical assumptions and inquiry methods. This methodology includes philosophical assumptions to give guidance for the data collection and analysis from different sources in a single study[39]”.*

The mixed-methods approach stands as a unique methodology within the realm of research especially in the field of computer science, software engineering and data security[36]. These methods are mostly used where a standalone research method, for example quantitative or qualitative is not sufficient.

This thesis involved data collection, preprocessing, classification using transformers, categorization based on ISTQB standards, classification through large language models and lastly using traditional machine learning approaches. So, the research design leans heavily towards a quantitative approach as shown in figure 2.

The qualitative components of the thesis are:

3.2.1 Quantitative Component:

Data Collection and Preprocessing: Collecting and preparing GitHub issue data involves quantitative methods to handle large datasets, which is done using data collection and cleaning in Python.

Initial Classification & Further Categorization: Utilizing transformers to classify issues as test/not test and then to test-debt/ not test-debt and then categorize them into types of testing based on large language model is a quantitative analysis, focusing on measurable data.

Evaluation and Validation: Evaluating the performance of the models using statistical metrics (accuracy, precision, recall, F1 score) is fundamentally quantitative.

Analysis (using LLM): By using Large Language Model (LLM) for classifying test issues into different types of tests based on textual data, the process joins both advanced computational techniques and the understanding of textual context.

3.3 Data Collection:

3.3.1 GitHub Archives:

The data collection is done using the GitHub Archives, which is a widespread resource that captures a snapshot of GitHub activity. As the number of open source projects are increasing tremendously, it is the biggest online software repository host with ten millions of the publicly available git-version controlled repositories[40]. In GitHub, events are actions that take place on the platform that can initiate workflows or give notification to users or update the repository state. These events are important in GitHub's automation capabilities and only the issueEvents were used in this thesis. These are the events that are triggered when issues are opened, closed, edited or commented on. Only issue events are selected as the goal is to classify the issues based on the developers labelling. The dataset used in my research consists of the open-source projects from **01/01/2015** to **02/01/2023** shown as in figure 3. This massive dataset was created by mining every issueEvent and then it was filtered to select issue labels, focusing on the issues that had been explicitly tagged by developers, consisting of various categories including test, debt, test debt and test types.

A Python-based script was created to automate the extraction of data as shown in figure 3. This script interacted with the GitHub API, allowing for the precise retrieval of issue data including id, type, created at, repo, repo URL, action, title, labels, body. Particular attention was paid to the labels assigned by the developers, as these tags contain rich metadata that can indicate the presence of

test labels, test debt, technical debt, and test types. The entire process of data collection is shown in the flow chart in figure 3 below:

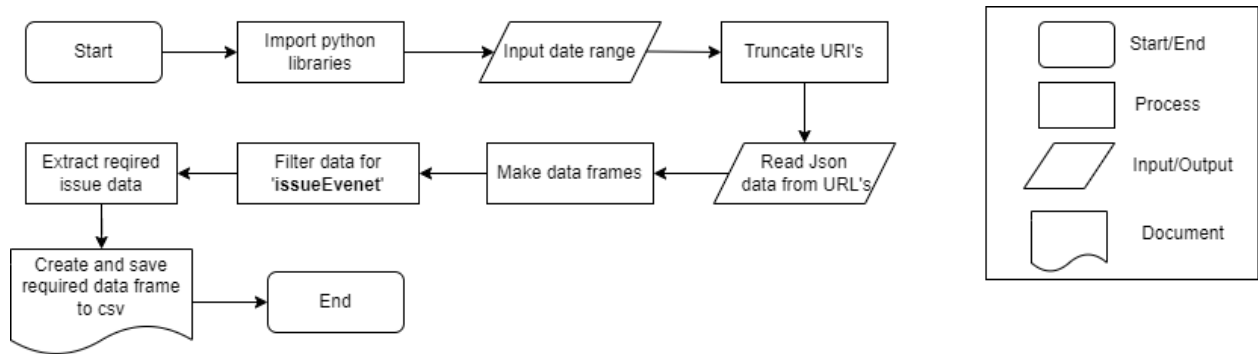


Figure 3 Data collection using GitHub archives.

The process starts with importing libraries, selecting the date ranges, and then performing some truncation and then filtering and saving only the relevant data as shown in figure 3. This process helped to get all the issueEvents within a specific date range which contained all the issues that were labelled '**test**' by the developers. This process is the 1st step in dataset creation that will be utilized further in the thesis.

From the initial data extraction from GitHub archive in figure 3 where all the data with issueEvents was loaded the next step is shown in figure 4 where that big data is divided into 4 sets based on the labels they have. This step resulted in 4 datasets **test**, **not test**, **test debt**, and **not test debt**. These are the datasets that will be used in training and testing phases of thesis after pre-processing.

3.4 Data pre-processing:

The raw data collected by the GitHub Archives went through initial pre-processing, which was done in Python. Python's robust libraries, specifically pandas for data manipulation and NLTK for natural language processing, were used to refine and enhance the dataset according to thesis requirement. Irrelevant columns were removed to rationalize the dataset for analysis, while

tokenization and other text preprocessing techniques were employed to refine the textual data for machine learning algorithms.

The main focus of the data pre-processing was to clean the dataset to remove any unnecessary information, while making sure that the remaining data was completely relevant to the research questions at hand. The entire process involved the thorough examination of each data point to establish its relevance to the study's objectives. During the data preparation phase, any instances of missing or incomplete information were identified and issues with incomplete missing or irrelevant information were removed. This proactive approach helped in the subsequent analysis that was based on data that was both comprehensive and robust. In instances where data gaps were identified, additional information was taken directly from the GitHub platform to augment and enrich the dataset. And for those issues if sufficient information was not found they were removed from the data. Further preprocessing included deduplication, where issues with matching titles were removed from the data. Another important thing done in this phase was to remove non-English phrases from the data to ensure that data is in only one language as shown in figure 5. The data in the other languages was not translated in English to ensure high accuracy. Translating the other data in language other than English and then using an English trained model is not recommended, as well as reducing the accuracy of the model 16% it resulted in worsening performance of the model, as the data first needs to be translated[41].

Before the preprocessing phase, the dataset was comprised of 7,260,844 GitHub issues covering many irrelevant columns. This is the sum of all the four datasets from figure 4. All four datasets have an unequal number of issues which need to be balanced so that the models will learn in unbiased fashion. So, the datasets went to pre-processing which removed some irrelevant and duplicated issues which reduced the dataset size. After that the datasets were balanced and combined into two datasets which are (test, not test) dataset and (test debt, not test debt) dataset.

After the data preprocessing phase, which included data cleaning, removing duplicates, and the removal of non-English text the total number of issues were 3,651,769 which was then balanced to make data even and to have an equal number of 'test' and 'not_test' labeled issues, totaling 45,195 entries each for both classes while 'test debt' and 'not test debt' were balanced, with 247 issues for each class, amounting to a total of 494 issues considered for the analysis. This reduction ensures that the dedicated dataset, removing any irrelevant information and standardizing the entries for effective analysis, is sufficient. This data is without the test data used for evaluation.

The step-by-step process of test/not test and testDebt/not testDebt datasets creation and data cleaning is shown in figures 4 and 5. Figure 4 describes how the data is divided into 4 csv files each containing data with different labels. While figure 5 is describing the whole data cleaning process in a step-by-step manner.

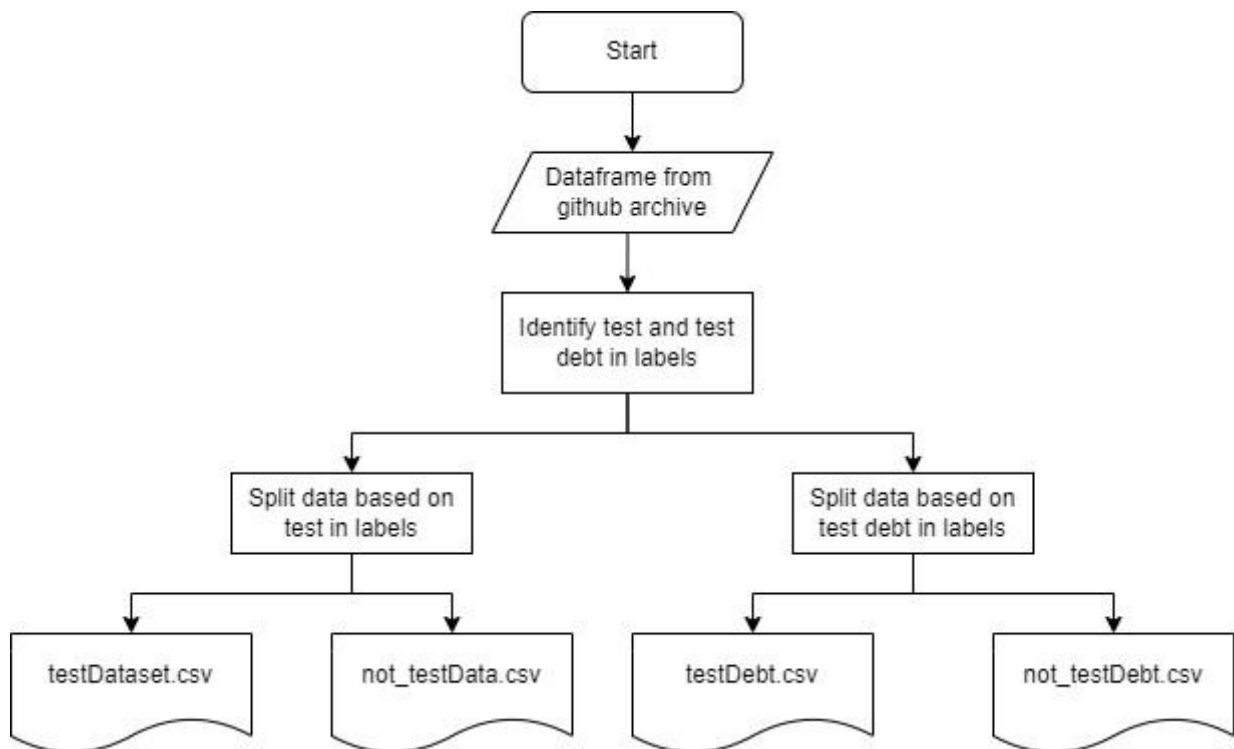


Figure 4 Dividing data to 2 datasets.

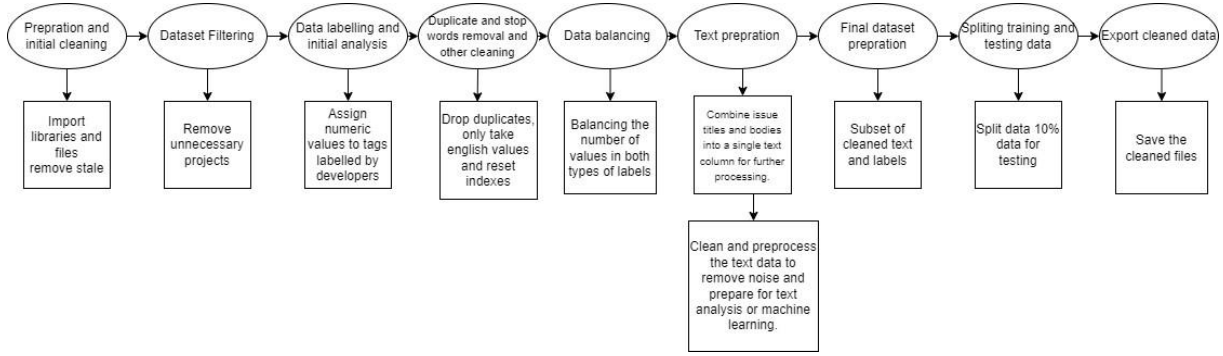


Figure 5 Data cleaning.

3.5 Quantitative analysis workflow:

After pre-processing the dataset was ready for machine learning tasks. Both datasets (test/not test and test debt/not test debt) were stored in separate csv files and for the final step the data was converted to numerical form so that it can be utilized in machine learning tasks. Now the goal is to make use of a model that will learn to associate the labels (test, not test, test debt , not test debt) with the issue description. This will use supervised machine learning where the model will learn to associate the issue with the labels done by developers.

The analysis started with the use of state of art transformer model, which are a sophisticated type of neural network architecture that transforms or changes an input sequence into an output Two types of transformer models are used, and their data is compared with each other.

Now we have 2 classifiers that will classify the issue to test and test debt related issues. Next the test related issues are further categorized to different test types using two different approaches .One is rule-based classification of the test issues into different test types based on ISTQB book [5] and second is by using large language models.

Next the results of classification done using transformer is compared to traditional machine learning approaches to see how well the proposed model is performing with respect to already existing approaches. This step showed the effectiveness of my approach with respect to already existing approaches.

The dataset is created from scratch for this thesis using labelling done by developers themselves but there are already some existing datasets where labelling is done by researchers based on some indicators. Final step was comparing the transformer model trained on the data set which was already labelled by researchers[12] with the transformer model trained on the data set which was labelled by developers. The entire process is described step by step below.

3.5.1 Text classification:

There are three types of text classification systems available. One is rule based classification and the other is machine learning based text classification and last is hybrid approach[42].

Rule based text classification utilizes a manual system of languages to classify the text into different categories and groups. The system is instructed to categorize the text into a specific category based on the content in the text by utilizing semantically relevant textual elements. Each rule includes an antecedent or pattern and a projected category. These systems can be improved over time and are understandable to humans. However, there is a need for expertise in the subject[43].

The machine learning approach on the other hand does not utilize manually crafted rules. Instead, it learns to classify the text based on past observations. By using per-labeled data as examples it learns how to associate a specific tag to the data. A tag is pre-defined category that the given text will fall into[42]. Most common type of traditional machine learning models are SVM, naïve bayes and neural network.

The hybrid approach combines a machine learning-trained base classifier with a rule-based system to enhance the results. These systems can be easily fine-tuned by incorporating specific rules for those conflicting tags that haven't been properly modeled by the base classifier[42].

My research utilized both machine learning and rule-based approaches. The machine learning approaches are used to classify data in test/not test and test debt/not test debt. Furthermore, machine learning using large language models is utilized to further classify the test issues in further types of tests. While the rule-based approach is also employed to classify the test issues into various test types.

3.5.2 Implementing and Analyzing Transformer Models for Classification Tasks:

This section will give a detailed description of the implementation, training, and evaluation of transformer models, focusing on the DistilBERT and RoBERTa model, for classifying project management issues related to testing in GitHub datasets. The process aims to establish the efficacy of transformer models in classification tasks. These models were used to classify issues into test and test debt.

The implementation of transformers model is done using PyTorch[44] which is popular machine learning library used in computer vision and natural language processing. It is powerful with respect to efficient memory usage and processing is done NVIDIA GeForce RTX 2080 Ti GPUs to make sure that training is efficient with CUDA acceleration. It provided parallel computing platform to enable operations to be computed on GPU's rather on CPU's to reduce the training time[45]. The '**transformers**' and '**datasets**' libraries from hugging face are also used because they are state-of-art tools that provide pre-trained models as well as data processing tools. The finalized data set consisting of labelled GitHub issues was loaded and prepared using panda's framework and then split into training, validation, and test sets using an **80-10-10** split to ensure multiple data exposure during model training and evaluation.

DistilBERT model which is pre-trained on uncased text is fine-tuned for required a binary classification task. The model configuration is adapted to categorize texts into two categories: 'test-related' and 'non-test-related' issues and 'testDebt' and 'notTestDebt'. Same is done using RoBERTa model. The models were then trained in 5 epochs with different learning rates and batch sizes were adjusted be optimized with the computational resources available. The training process

involves periodic evaluation to monitor performance and adjust the parameters as needed. Then evaluation of the models was done using the parameters such as accuracy, precision, recall, and F1-score, that give us a complete understanding of the model's performance on both training and validation datasets.

Following the training phase, the model was carefully tested on unseen data from the test set to see how well it is generalized on unseen data. Additionally, a separate test set is also used to further validate the model's performance and to increase the model's validity. The classification results are analyzed to determine the model's effectiveness in identifying test-related issues. The analysis process involved comparison of the predicted labels to the actual labels to determine the accuracy and other metrics and the comparison of DistillBert and RoBERTa performance and resource usage. The detail of this is described in the results section.

3.5.3 Classification of Testing Types using rule-based approach.

This step involves classification of different testing types based on project management issues sourced from GitHub based on rule-based approach. The rules were defined using the definition of testing types from ISTQB book which are as follow:

Functional Testing:

“Functional tests are based on functions described in documents or understood by the testers and may be performed at all test levels (e.g. test for components may be based on a component specification). Function (or functionality) testing can, based upon ISO 9126, be done focusing on suitability, interoperability, security, accuracy, and compliance[5, p. 48].”

Non-functional Testing:

“Testing is related to the quality characteristics or non-functional attributes of the system. Include aspects like performance testing, usability testing, and

reliability testing. Leverage the ISO 9126 standard's quality characteristics and sub-characteristics for classification[5, p. 50].”

Structural Testing:

“Testing that pertains to the structure of the system or component. Highlight the 'white box' nature of structural testing and the use of control flow models. Emphasize the measurement of testing thoroughness through coverage of structural elements[5, p. 52].”

Testing related to Changes:

“This is related to testing changes in the software, including confirmation testing (re-testing) and regression testing. Clarify the purpose of regression testing as verifying that modification has not caused unintended adverse side effects”[5, p. 56].

The dataset utilized in this step only included the issues that were related to test and non-test issues were removed in this step. This data was loaded and inspected using Pandas, to understand the distribution of the four above-mentioned testing types using regex.

Given the nature of the data and the classification task, this is simple yet an effective approach to classify issues text based on the presence of specific keywords associated with different types of testing (functional, non-functional, structural, change-related). The keywords were extracted from the definitions of the testing types above. This method assumed if a certain keywords are present, it identifies the testing type, which is often practical in a well-defined domain such as software testing[43], [46].

3.5.4: Classifying GitHub Issue into Testing Type using language model

In this step machine learning model in particular language model from OpenAI namely **gpt-4-0125-preview** is used to classify the GitHub test issues to different testing types. This classification also aligned with ISTQB guidelines to identify different types of testing including Functional, Non-Functional, Structural, and

Testing related to Changes. This approach performs the same task as rule based approach described in section 3.5.3 but it is automatic approach while the rule based is manual.

The data used in this process is test issue data only. Non-test issues are not used here. The process starts by utilizing python library pandas for data manipulation and LangChain for integration of language model[47]. LangChain framework is used for developing applications powered by LLMs. Using this model for classification is a time taking process as this model has billions of parameters and they require significant resources so concurrent.futures and ThreadPoolExecutor library is used for parallel processing to enhance computational efficiency. They are used to speedup processing time as these modules provide high-level interface for asynchronously executing callable[48].

The model **gpt-4-0125-preview** is the most powerful model currently available because of its ability to understand and generate human-like text based on the input provided to model. To use model ChatOpenAI instance is initialized with specified parameters to ensure that the model generates responses at a temperature-controlled temperature setting, affecting the randomness of response generation.

The next step is to instruct the model to perform tasks which is done using '**custom prompts**'. These prompts are instructing the model to generate only the required test types and not others ensuring that the system understands its role in classifying texts into predefined testing categories as per ISTQB standards. Then the model processes each issue text through a pipeline 'LLMChain', this process involves the transformation of the text into structured outputs based on the classification categories (testing types).

Classifying issues using traditional models and comparison with transformer models:

To compare proposed transformer models with related work done by Li et al., the models from the studies which are SVM and Text CNN are also used. These

models were trained on training dataset from GitHub and tested on separate test data . After classification the results were compared with results of transformers model to see the effectiveness of the modern approaches w-r-t the previously done approaches.

3.5.5 Comparison of datasets:

After use of each classification techniques, the data set from the studies done by Li et al. [12], is compared dataset created in this studies. These datasets are different based on tagging. One is tagged by developers and the other is tagged by researchers. The transformer model (DistillBert) is trained using both the datasets separately and is then also tested to see the model trained on which dataset gives better results in term of accuracy and generalizability.

3.6 Data Ethics:

While conducting the research, great attention was paid to ethical considerations regarding the use of the data. All data used in this research was collected from public repositories on GitHub, for which consent is not required for accessing the data. However, keeping the privacy concerns in mind the personal identification information was not used and only the data relevant to the issues was used and thus the confidentiality of the individual contributor was maintained.

The data collection strictly followed the terms of service provided by GitHub. The data was retrieved using the GitHub API and is solely used for academic research purposes. At no point did the data be utilized for commercial gain and care to comply with the platform's regulations on data usage[49].

In addition to GitHub Archive data, my research used an external dataset created by Li et al., which is publicly available for research purposes. This dataset, named "SATD Issue Tracker Data," has been used in process of validating the results as well as in comparing the transformer approach with already existing approaches, specifically in the context of self-admitted technical debt (SATD) [12].

The "SATD Issue Tracker Data" was obtained from its GitHub repository ², where it was free to be used by anyone who provided proper credits. Despite these terms, this thesis admits Yikun Li's considerable contributions and references their work accordingly. Furthermore, by using this dataset, I complemented my primary GitHub data, enabling a comparative analysis and validation of findings across various sources. In this way the study results were strengthened, ensuring they were not solely based on a single data source.

To ensure the reproducibility of the research, all the methodologies used are properly documented as well as the code used in analysis is made available through GitHub repository. All the datasets created are also made available and no sensitive information is being used in the research[50].

² <https://github.com/yikun-li/satd-issue-tracker-data>

Chapter 4

4. Results

This chapter contains the results of the thesis. Each section in chapter provides results based on the research questions. The collected data section gives an overview of the data collected from the GitHub archives after it has been extracted and processed. The results from the collected and processed data are used for all the research questions RQ1, RQ2, RQ3, RQ4 and RQ5.

4.1 Collected data:

The initial data extracted from GitHub has multiple events as shown in figure 6. But the data of our interest was only related to issueEvents as discussed in detail in section 2.3 and 3.3.1. This event type relates to how projects manage tasks, bugs, enhancements, and other activities that are crucial for project management.

	id	type	actor	repo	payload	public	created_at	org
0	2489368070	PushEvent	{'id': 9152315, 'login': 'davidjhulse', 'grava...	{'id': 28635890, 'name': 'davidjhulse/davesbin...	{'push_id': 536740396, 'size': 1, 'distinct_si...	True	2015-01-01 00:00:00+00:00	NaN
1	2489368072	PushEvent	{'id': 5581438, 'login': 'jmoon018', 'gravatar...	{'id': 26392647, 'name': 'jmoon018/rshell-unit...	{'push_id': 536740397, 'size': 1, 'distinct_si...	True	2015-01-01 00:00:00+00:00	NaN
2	2489368089	CreateEvent	{'id': 6352424, 'login': 'christoferpeterson',...	{'id': 28677542, 'name': 'christoferpeterson/V...	{'ref': 'master', 'ref_type': 'branch', 'maste...	True	2015-01-01 00:00:01+00:00	NaN
3	2489368095	PushEvent	{'id': 66577, 'login': 'JakeWharton', 'gravata...	{'id': 5152285, 'name': 'square/okhttp', 'url'...	{'push_id': 536740405, 'size': 8, 'distinct_si...	True	2015-01-01 00:00:01+00:00	{'id': 82592, 'login': 'square', 'gravatar_id'...

Figure 6 Initial extracted data with multiple event types.

Each payload in the issue event had 86 fields in total but only relevant fields were used which are id, type, created_at, repo, repo_url, action, title, labels and body. The data then undergoes some analysis and the issues tagged by developers

as test and test debt was labelled as test and test debt class and the others were labeled as not test or not test debt class. The assigning of the test and not test as well as test debt and not test debt is done using regex which tag the issues based on presence of particular keyword in them i-e ‘test’ ,‘not test’ and ‘test debt’, ‘not test debt’ as shown in figure 4 . For example if the issue text is “Add more tests for function 1 it lacks functionality that could be a debt”. This issues belongs to both test and test debt. While an issue with text ‘Add more functionality it could cause harm’ is neither test nor test debt. A numerical value is also assigned to each issue as 0 and 1 to be able to use in machine learning tasks. After this step dataset was filtered to only three columns that are issues text, label, and class as shown in figure 7:

	text	label	class
0	Add Tests for System.Drawing.CommonThis issue ...	1	test
1	ACCTests for packetfabric_port_resource<!-- P...	1	test
2	[Text Analytics] AbstractiveSummarizeBatchConv...	1	test
3	See Entity recordsSee the story in the [releas...	1	test
4	Editor crash when click mesh instance 3d with ...	1	test
...	...	...	...
90385	Solis Energy Today does not reset correctly**D...	0	not_test
90386	検索ボタンを押さなくても勝手に絞り込まれる# 仕様\n- タグを選択した時点で絞り込みが行わ...	0	not_test
90387	How to enable new home structure and navigatio...	0	not_test
90388	Package Request: waydroidRun android apps on W...	0	not_test

Figure 7 Dataset before pre-processing.

Finally, the data was ready for the next step which was data cleaning and pre-processing. The aim was to remove irrelevant and unprocessed data for example data in language other than English, removing stop words, data with empty fields and punctuation marks. The detailed process is described in section 3.4 and whole process is shown in figure 5 Also, the data after pre-processing looks like the data in figure 8:

	text_clean	label
0	add tests for systemdrawingcommonthis issue tr...	1
1	acctest for packetfabricportresource please k...	1
2	text analytics abstractivesummarizebatchconven...	1
3	see entity recordssee the story in the release...	1
4	editor crash when click mesh instance 3d with ...	1
...	...	...
90385	solis energy today does not reset correctlydes...	0

Figure 8 Dataset after pre-processing and cleaning.

The summary of data collected from GitHub archive is in table 2. This table shows the total number of initial issues as well as issues after data preprocessing and after balancing. The data is divided into two types (test/not-test) and (test-debt/ not-test-debt). Now the data is ready to be used for classification.

Total number of initial issues	7,260,844
Total number of issues after removing un-necessary data	3,651,769
No of issues (test/not-test) after pre-processing and balancing	90,390
No of issues (test-debt/ not-test-debt) after processing and balancing	494

Table 2 Summary of data collected.

Other than my primary dataset additional external dataset is also used in some of the research questions from a previous study[12].

Final Dataset:

After labelling and initial extraction of irrelevant data and data preprocessing the dataset is finalized which comprises 90884 issues with 90,390 issues related to test and not test while 494 related to test debt and not test debt. This is the primary dataset created and used throughout the thesis. One complimentary dataset is also utilized in the thesis.

This dataset comprises 4,200 issues, which break down into 23,180 individual issue sections sourced from seven open-source projects, making it particularly rich in examples of self-admitted technical debt (SATD) across different platforms and development contexts. That dataset is manually tagged by the researchers about what kind of debt is the issue. This dataset is used for comparative analysis of transformers technique with traditional machine learning approaches as well as comparison of how dataset labelling process effect the accuracy of the models.

4.2 RQ1 Classifying project management issues as test related issues using transformers:

Automatic classification of test issues helps in identifying areas with high risks and areas with more issues regarding testing which helps the testing team, developers, and project managers to manage issues more efficiently. It organizes the allocation process and makes sure that issues are directed to the appropriate team members quickly and accurately without going into each issue description separately and reading them.

DistillBert:

To find out how good transformer-based model in classification of test debt, it is trained to see whether the issue is test related or not. Then the model's performance is evaluated on several metrics, including accuracy, precision, recall, and F1 score, across both a validation set and a separate test set.

The choice of the model at start is '**distilbert-base-uncased**' which is a distilled version of the BERT base model. The choice is made based on its uncased nature as 'test' and 'Test' are same for this model[51]. This model is smaller and faster than BERT, which was pretrained on the same corpus in a self-supervised fashion. The model was fine-tuned on a binary classification task to determine whether GitHub issues are test-related or not.

The training and testing was done on one date set from GitHub using 80-10-10 approach Before model training the primary dataset created from GitHub was tokenized into a format that the model can understand and work with during the training phase. In this analysis using DistilBERT for classifying GitHub issue texts, tokenization was done using the **AutoTokenizer** from the transformer's library. This tokenizer is specifically tuned for the DistilBERT-base-uncased model, ensuring compatibility and optimal performance. The tokenizer is applied to the dataset in batches to convert text data into tokens and ensures that sequences are padded to a consistent length or truncated based on the model's maximum sequence length requirement.

The model training involved 5 epochs which indicate the number of full cycles through the training set with a learning rate of $5e-5$ and a batch size of 16 for training and 32 for evaluation. The model was not set to automatically save, preventing overfitting alongside monitored evaluation.

The model showed improvement over 5 epochs, with a starting training loss of 0.337800, decreasing to 0.045800 by the final epoch i-e epoch 5 as shown in figure 9. Validation loss initially decreased but started to rise after the third epoch, indicating potential overfitting as shown in graph in figure 9. This is a common challenge in machine learning, where the model learns the training data too well, such as noise and anomalies, which do not appear to be accurate on unseen data[52]. The accuracy of model keeps on increasing with the number of epochs. table 3 below shows in detail the performance of distilled Bert model on over the 5 epochs.

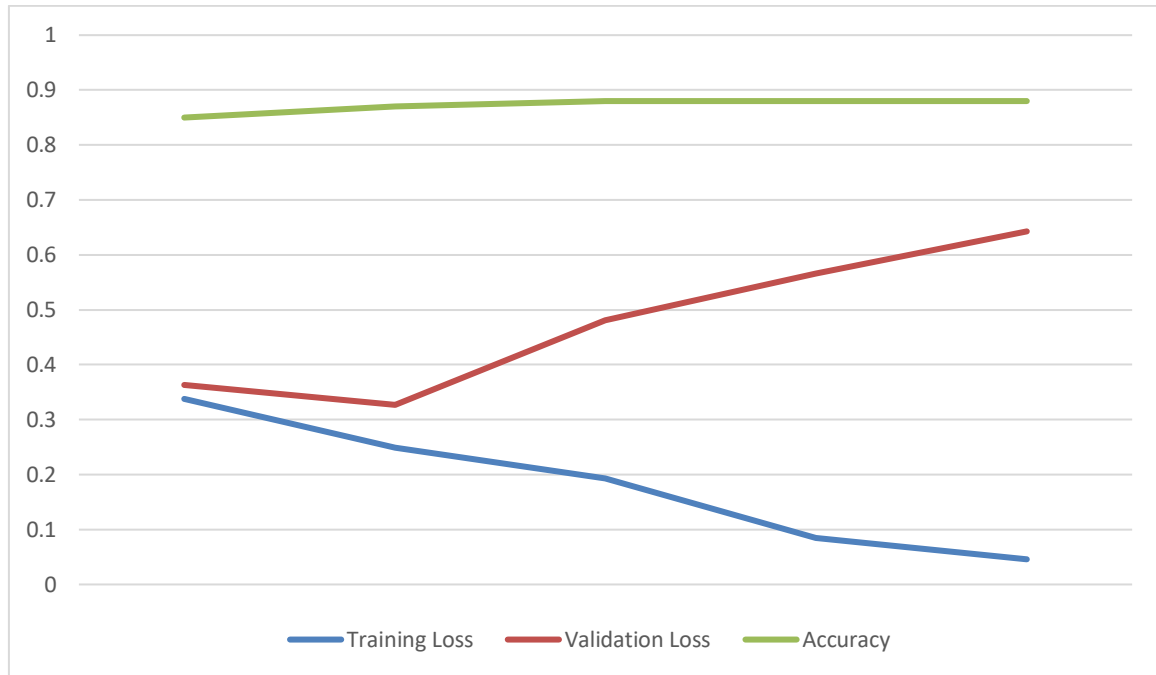


Figure 9 Distilled Bert Validation loss, accuracy, and training loss test/not_test issues.

Epoch	Training Loss	Validation Loss	Accuracy	F1 Score	Precision	Recall
1	0.337800	0.362775	85%	0.856565, 0.860176	0.877870, 0.840296	0.836270, 0.881019
2	0.248900	0.326933	87%	0.876592, 0.882150	0.908439, 0.853574	0.846902, 0.912706
3	0.192900	0.481332	88%	0.879121, 0.884245	0.909682, 0.856685	0.850547, 0.913638

4	0.084800	0.566562	88%	0.886454, 0.884377	0.888347, 0.882462	0.884569, 0.886300
5	0.045800	0.642783	88%	0.887611, 0.890272	0.908946, 0.870326	0.867254, 0.911153

Table 3 Distilled Bert performance on training data test/not_test issues.

Evaluation on test data gives an unbiased evaluation of the model performance on new and unseen data that helps in checking the generalization of the model as well as to see that model has not memorized the training data but has learned to recognize the pattern to classify new data.

While the model was tested on separate test set (10% of original dataset) which was separated from the data before training to see the accuracy and generalizability of model. On a separate test set, the model achieved an accuracy of 0.960045, with precision, recall, and F1 scores indicating high generalizability as shown in table 4 below.

F1 scores for both classes in the test set were nearly identical, suggesting balanced performance across the binary classification tasks.

Metric	Value
Accuracy	96%
F1 Score	0.959749, 0.960336
Precision	0.971073, 0.949426
Recall	0.948686, 0.971500

Table 4 Distilled Bert performance on test set

RoBERTa base:

The 2nd transformer model used is RoBERTa- base because it maximizes the model performance, and the resources allow for handling a larger and more complex data[53]. The processing time of this model is more as compared to previous model. Also, this model demands higher computational resources.

The model tokenization and the matrices are same as Distilled Bert with only difference of using AutoTokenizer with Roberta-base. The primary dataset was used for training and testing. The training loss in the beginning is 0.465 and ended at 0.322, indicating a continuous decrease in training loss. Validation loss and accuracy metrics are also monitored to assess model performance on unseen data. The best accuracy achieved on the validation set is approximately 87.7%. The model achieved F1 scores of 0.874 for one class and 0.880 for the other, which demonstrates a balanced performance across both classes. The highest precision (0.894 for one class and 0.862 for the other) and the recall (0.855 and 0.899 respectively), suggested that the model effectively recognized relevant features for classification without significant bias toward one class. The results from model training are shown in the table 5 :

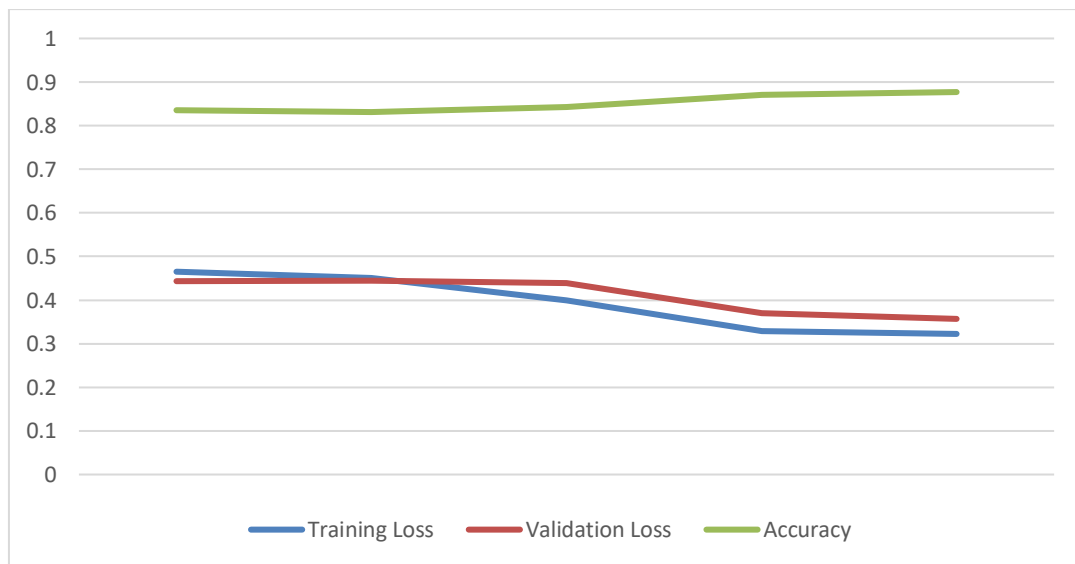


Figure 10 RoBERTa Validation loss, accuracy and training loss test/not_test issues.

Epoch	Training Loss	Validation Loss	Accuracy	F1 Score	Precision	Recall
1	0.4651	0.4433	83.52%	0.8231, 0.8457	0.8869, 0.7958	0.7679, 0.9023
2	0.4505	0.4445	83.12%	0.8132, 0.8460	0.9087, 0.7785	0.7359, 0.9262
3	0.3987	0.4392	84.30%	0.8335, 0.8515	0.8862, 0.8086	0.7867, 0.8992
4	0.3295	0.3697	87.09%	0.8697, 0.8720	0.8766, 0.8653	0.8629, 0.8789
5	0.3224	0.3569	87.71%	0.8742, 0.8798	0.8939, 0.8617	0.8554, 0.8987

Table 5 RoBERTa performance on training data for test/not_test issues

The model achieved accuracy of 87.83% on the test dataset (10% of original dataset), indicating strong generalization from training to unseen data. The model also demonstrated balanced performance across both classes. Test class showed slightly higher effectiveness in recall, indicating that the model is very capable of identifying true positives for test class.

Metric	Value
Accuracy	87.83%
F1 Score	0.8751, 0.8814
Precision	0.9042, 0.8552
Recall	0.8478, 0.9092

Table 6 RoBERTa performance on test set

The comparative analysis between the two models is discussed in more detail in the discussion section.

4.2.1 Transformer-based models (RoBERTa, DistilBERT) in classifying project management issues as test debt related issues:

Classifying the issues as test debt helps in identifying the areas of the project which are okay now but are at the risk to cause issue in future. Highlighting these issues can significantly affect project management as well as helps in prioritization that further helps to address issues that need immediate attention.

DistilBERT:

The same model DistillBert as above in RQ 1 is now utilized on test debt dataset. The dataset available for training is a relatively small dataset, so additional dataset is also utilized as a complement to see how the model trained on small dataset generalizes on unseen data. Furthermore the results from the test debt model were also compared to technical debt [54] from a previous study done by Shivansker et al., and test model from RQ 1 that are trained on bigger datasets and that have very good accuracy and generalizability on unseen data. This is done to see what are the results of the test debt model trained on small dataset as compared to combination of the models that were trained on huge data and have good generalization and accuracy.

The test debt model findings and results from training data are in table 7. All the parameters for training, testing and tokenization is kept same as RQ1 for DistillBert, The constant decrease in training loss in each suggests a effective learning and model optimization. Despite the overall improvement in training metrics, the validation loss started to increase after the third epoch, which is potentially indicating overfitting. The model's accuracy peaked in the fourth epoch at 0.8182 but showed a slight decrease in the fifth epoch along with an increase in validation loss, further supporting the possibility of overfitting. While for precision and recall there are mixed trends, with maximum precision in the

fifth epoch for the second label but decreasing for the first label, indicating some trade-offs being made by the model between detecting different classes.

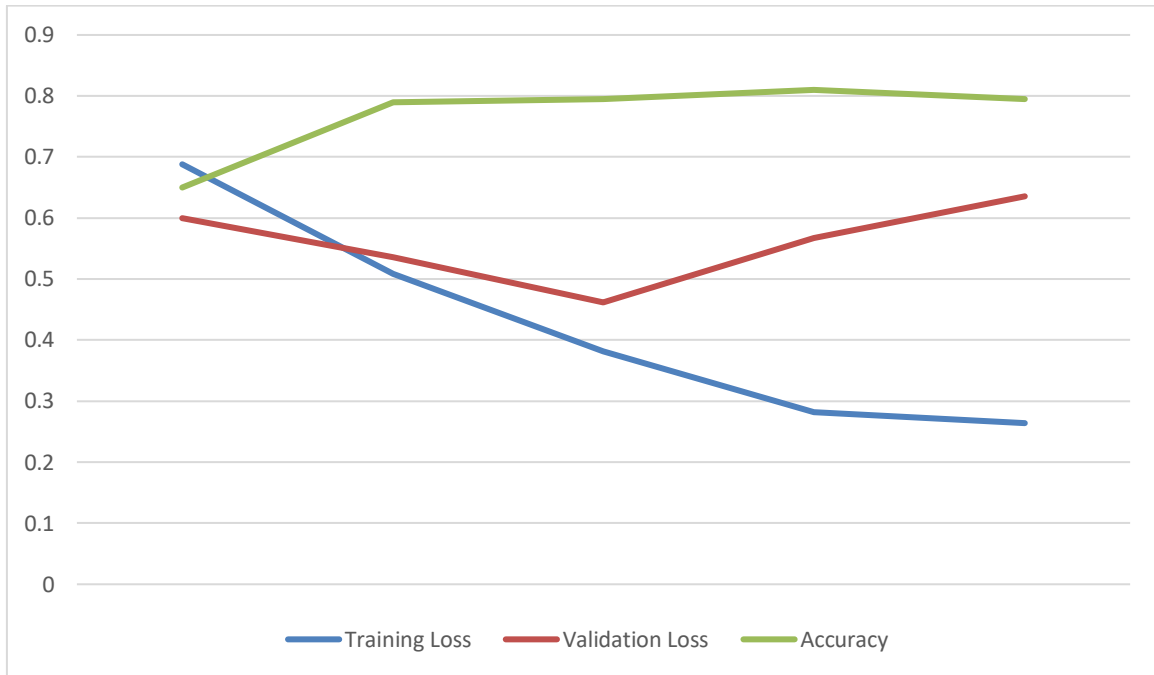


Figure 11 DistilBERT Validation loss, accuracy, and training loss on testDebt/not_testDebt.

Figure 11 shows the trend of accuracy, validation loss and training loss during the training in 5 epochs. The extreme decrease in training loss suggests that the model is learning over the course of time and is minimizing the loss between actual and predicted outcomes. The validation loss is decreasing initially but increasing slightly in the last steps, giving signs of some overfitting. Accuracy on the other hand is increasing over time indicating a superior performance of model.

Epoch	Training Loss	Validation Loss	Accuracy	F1 Scores	Precision	Recall
1	0.6881	0.5996	65%	[0.7059, 0.5946]	[0.5806, 0.8462]	[0.9, 0.4583]
2	0.5084	0.5356	79%	[0.7692,	[0.7895,	[0.75,

				0.8163]	0.8]	0.8333]
3	0.3815	0.4618	79%	[0.7805, 0.8085]	[0.7619, 0.8261]	[0.8, 0.7917]
4	0.2815	0.5676	81%	[0.7895, 0.84]	[0.8333, 0.8077]	[0.75, 0.875]
5	0.2640	0.6356	79%	[0.7568, 0.8235]	[0.8235, 0.7778]	[0.7, 0.875]

Table 7 DistilBERT performance on training data for testDebt/not_testDebt dataset

The model performance on test set is as follow:

Metric	Value
Accuracy	81%
Precision	0.816
Recall	0.812
F1 Score	0.814

Table 8 DistilBERT performance on test set for testDebt/ not_testDebt dataset.

As described earlier, the dataset for training was not large enough so the results might be a bit different in real scenarios. To see how the results of this model are as compared to the models that are trained on huge dataset as well as have particularly superior performance of the same dataset is tested on technical debt model and test model and then combined to true positives to see what the accuracy of the models is combined. In this way we can see how the results of this ensemble method are as compared to our approach. As the technical debt model

will give indication of either the issue is related to technical debt or not and test model will give indication if the issue is related to test or not their ensemble will indicate the presence of test debt. For this testing purpose a entirely different external data from previously done studies by Li et al. set is utilized to avoid data leakage[12]. This dataset is tagged by researchers manually.

Results of '**test model DistillBert**' on additional dataset: The model from RQ 1 tested using the external dataset. The true positives from this model indicated that they are test related issues. This test model has accuracy of 88% with 0.83 precision. While the recall and f1 score is 0.93 and 0.87 respectively.

Results of '**technical debt model**' on additional dataset: A technical debt model from previously done studies by Skryseth et al. [54] is used and external dataset is tested on this model. The true positives from this model indicated an accuracy of 50% with 0.50 precision. While the recall and f1 score is 0.84 and 0.63 respectively.

Results of both '**models combined**' on additional dataset: Then result of both models i-e true positives are combined. For example cases where both models gave true positives . Those indicate the issues that are both test and technical debt . Combined accuracy is 50% with 0.25 precision. While the recall and f1 score is 0.50 and 0.33 respectively.

These results suggest a few things. 1st that test debt model using DistillBert has similar or higher individual accuracy when tested on its own dataset compared to the test model on the additional dataset. The test debt model results on the external additional dataset suggested that it generalizes well, possibly even better than the technical debt model. The reason could be that the features and patterns it learns are more effective across different data environments and technical debt model covers a wide variety of debts including test, code, architecture and other debt and not only test debt. Also, the drop in combined performance of two models suggested that simply merging model predictions isn't effective[55]. This could be due to conflicting prediction criteria or the models not

being cooperative in their decision boundaries. More details are discussed in the discussion section.

Roberta:

The Roberta model is applied with the same configurations as the configuration on test issues dataset on the test debt data now. The training and testing were done on one date set using 80-10-10 approach and then for further testing external dataset is utilized as well as the combination of model is utilized again as in DistilBERT. The results of the model training are as follow in table 9:

Epoch	Training Loss	Validation Loss	Accuracy	F1 Score	Precision	Recall
1	0.6881	0.5996	65%	[0.706, 0.595]	[0.581, 0.846]	[0.900, 0.458]
2	0.5084	0.5356	79%	[0.769, 0.816]	[0.789, 0.800]	[0.750, 0.833]
3	0.3815	0.4618	79.5%	[0.780, 0.809]	[0.762, 0.826]	[0.800, 0.792]
4	0.2815	0.5676	81.2%	[0.789, 0.840]	[0.833, 0.808]	[0.750, 0.875]
5	0.2640	0.6356	79.5%	[0.757, 0.824]	[0.824, 0.778]	[0.700, 0.875]

Table 9 RoBERTa performance on training data for testDebt/not_testDebt

These results show a continuous improvement in learning across epochs, though the validation loss increase in later epochs suggests potential overfitting in last 2 steps as shown in the figure 12. Especially, the model's precision and recall metrics show a balanced ability to identify relevant instances and minimize false positives.

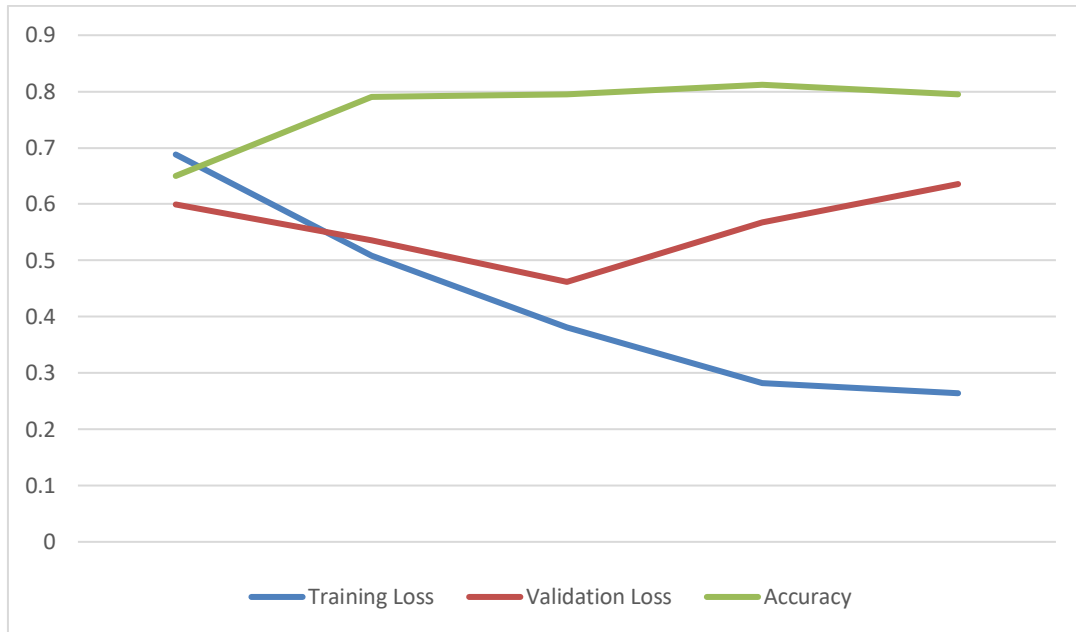


Figure 12 RoBERTa Validation loss, accuracy, and training loss on testDebt/ not_testDebt.

After training the model was tested on separate test data which is 10% of original dataset to see the performance on unseen data. The results are shown in table 10 below. Like DistillBert this model was also trained on not big data, so again the results of the test model and technical debt model are compared with this model too.

Metric	Value
Accuracy	81%
Precision	[0.821, 0.810]
Recall	[0.852, 0.773]
F1 Score	[0.836, 0.791]

Table 10 RoBERTa performance on test set for testDebt/ not_testDebt dataset.

As described earlier in DistillBert test debt model, again the dataset for training was not large enough so the results might be a bit different in real scenarios. To see how the results of this model are as compared to the models that are trained on huge dataset as well as have very good performance of the same dataset is tested on technical debt model which is already done in above section and test model and then combined to true positives to see what the accuracy of the models is combined. In this way we can see how the results of this ensemble method are as compared to our approach. For this testing purpose a entirely different external data from previously done studies by Li et al. set is utilized to avoid data leakage[12]. This dataset is tagged by researchers manually.

Results of '**test model DistillBert**' on additional dataset: The model from RQ 1 tested using the external dataset. The true positives from this model indicated that they are test related issues. This test model has accuracy of 86% with 0.85 precision. While the recall and f1 score is 0.85 and 0.85 respectively.

Results of '**technical debt model**' on additional dataset: A technical debt model from previously done studies by Skryseth et al. [54] is used and external dataset is tested on this model. The true positives from this model indicated an accuracy of 50% with 0.50 precision. While the recall and f1 score is 0.84 and 0.63 respectively.

Results of both '**models combined**' on additional dataset: Then result of both models i-e true positives are combined. For example, cases where both models gave true positives. Those indicate the issues that are both test and technical debt. Combined accuracy is 75% with 0.76 precision. While the recall and f1 score is 0.74 and 0.75 respectively.

The combined results from both these models (test model with Roberta and technical debt model) give relatively satisfactory results as compared to previous (test model with DistilBERT and technical debt model) but are still not as good as test debt model. This suggests that the classification performance given by this model is quite like technical debt and test model combined.

4.3 RQ2: Most common types of testing issues in the dataset from GitHub

To identify the most common type of testing problems a rule-based approach is utilized on the textual dataset. For this research question only issues of test types are utilized to classify them further in testing types. So, any non-test issues were removed from the dataset. The rules are defined based on the definition of each testing type in ISTQB certification book[5]. The rules defined are as follow in the figure 13:

```
functional_keywords = ['function', 'requirements-based', 'business-process-based',  
'priority',, 'functionality', 'requirements(-|\s)based', 'business(-|\s)process',  
'specifications','user stories', 'acceptance criteria', 'feature(s)?', 'use  
case(s)?', 'scenario(s)?', 'functional']  
  
non_functional_keywords = ['performance', 'usability ', 'reliability', 'ISO 9126',  
'performance', 'usability', 'reliability', 'ISO 9126', 'scalability', 'efficiency',  
, 'maintainability', 'portability', 'non(-|\s)functional', 'load ', 'stress',  
'security', 'compatibility']  
  
structural_keywords = ['structural ', 'white-box', 'control flow', 'coverage',  
'structural', 'white(-|\s) box', 'control flow', 'coverage', 'unit', 'integration',  
, 'path', 'code coverage', 'cyclomatic complexity', 'internal structure']  
  
change_related_keywords = ['change', 'confirmation', 'regression', 'side  
effects','change ', 'confirmation ', 'regression ', 'side effects', 'patch ',  
'release ', 'modification ', 'update ', 'backward compatibility ', 'version ',  
'hotfix ', 'functional regression','non(-|\s)functional regression']
```

Figure 13 keywords for identifying types of tests issues.

```

import re

def classify_test_debt(text_clean):
    # Compile regex patterns
    functional_pattern = re.compile('|'.join(functional_keywords), re.IGNORECASE)
    non_functional_pattern = re.compile('|'.join(non_functional_keywords), re.IGNORECASE)
    structural_pattern = re.compile('|'.join(structural_keywords), re.IGNORECASE)
    change_related_pattern = re.compile('|'.join(change_related_keywords), re.IGNORECASE)

    # Check for each category using regex search
    if functional_pattern.search(text_clean):
        return 'Functional Testing'
    elif non_functional_pattern.search(text_clean):
        return 'Non-functional Testing'
    elif structural_pattern.search(text_clean):
        return 'Structural Testing'
    elif change_related_pattern.search(text_clean):
        return 'Testing related to Changes'
    else:
        return 'Unclassified'

```

Figure 14 Regex for identifying types of tests issues.

The above-mentioned keywords in figure 13 were utilized in regex which is in figure 14 and then applied to test issues to identify which type the test issue belongs to base on specific keyword or combination of keywords. For example there is an issue with the description ‘add **tests** for systemdrawingcommonthis issue tracks porting some set of tests from monos test suite covering the portions of systemdrawing that we support on net core monos test cases are in this folder <https://github.com/commonmonotree/master/mcs/class/systemdrawing/test> we most likely want to convert the most useful tests from all of the sections here with the exception of systemdrawingdesign which we arent going to support right now on net core it is mainly related to designer winforms support monos tests use nunit so we will need to convert them to xunit when copying them additionally ive identified that there will need to be some **functional changes** made to the **tests** themselves as they do not pass against the net framework implementation we consider the net framework implementation to be the **compatibility** baseline so we should change the tests to accomodate it rather than the other way around the test failures seemed mainly related to very small subtle differences in things like floatingpoint precision color values offsets etc we should do the following when we have both windows and unix implementations up and running ensure

that all tests have enough leniency to accomodate floatingpoint differences and other minor inconsistencies where unavoidable ensure that all minor inconsistencies are acceptable or otherwise very difficult problematic to fix ensure that all minor inconsistencies are listed somewhere so that we can mention it both in the tests and in the user documentation hughbe qmfrederik mareksafar current status code coverage note that there is a large amount of internal and debugonly code which distorts these numbers when the coverage is generally high we can clean out a lot of dead code and then get more accurate data date branch coverage line coverage 6262017 33 25 7112017 49 54 7172017 55 60 822017 58 66 8252017 62 71 9212017 646 753 namespaces and coverage as of 9212017 systemdrawing 763 systemdrawingdrawing2d 938 systemdrawingimaging 861 systemdrawingprinting 628 systemdrawingtext 96

' – *issue from GitHub dataset.*

The text is unstructured and vague due to data cleaning. But the issue is related to functional test so the regex should assign it functional test tag after classification because it has keywords like 'functional changes', 'compatibility baseline' which are related to functional testing.

In total there were 72,330 issues in dataset and from these only 36269 were test issues. The rest were non test issues, so they were removed from the dataset. After applying the rule-based approach the classification of different type of test issue is as follow in the table 11:

Testing Type	Number of issues
Functional Testing	6,571
Non-Functional Testing	4,328
Structural Testing	5,855
Testing Related to Changes	6,367

Unclassified	11,109
--------------	--------

Table 11 Classification of test types using rule-based classification.

The classification is also shown in the chart below figure 15:

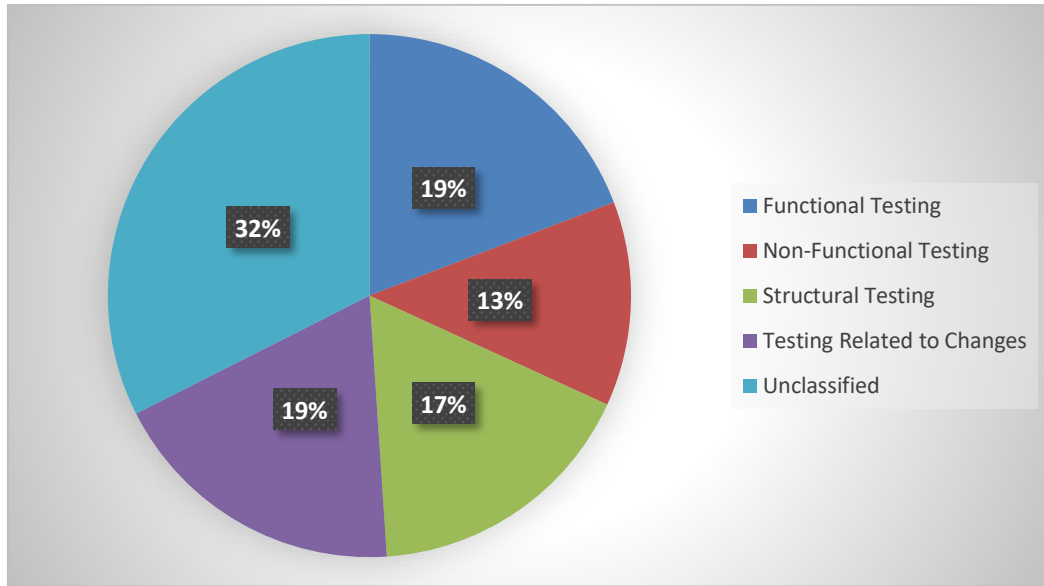


Figure 15 Pie-chart indicating percentage distribution of test types in dataset.

There is dominance in the Functional and change testing here which are concerned with verifying what system does is according to business requirements and confirmation as well as regression testing respectively. Next structural testing also referred as white box testing is most common in the dataset indicating almost 17% of the issue are related to structural testing. At last is the non-functional testing related issues. Although the importance of non-functional testing is significant[56] still there are least amount of issues related to non-functional testing. Potential reasons for this distribution of testing types are explained in the discussion section.

This classification is a quick and fast approach for classification. However, sizable portion of dataset (32%) remained unclassified using this approach suggesting that some issues did not contain any of the predefined keywords, highlighting potential limitations in the keyword set or the complexity of the

issue contexts. This approach also does not care about the semantics or meaning of the full issue description so the classification could be affected by that as well. Since ground rule was not available for the dataset, manual check of how good this approach is done and is described in the discussion section-

4.4 RQ3: To what extent are large language models able to classify project management test issues as specific testing types?

In this step only a subset of the test issue data (only 10,000 issues) were used for classification as using LLM is very expensive to be implemented for a large dataset (\$10.00 / 1M tokens). So only 10,000 entities from the test issues dataset (total: 36269) were used in this step. A language model (ChatOpenAI) is used with GPT-4. The model (**gpt-4-0125-preview**) is initialized with specified parameters such as temperature zero to ensure deterministic behavior. The temperature was set to zero to control the randomness in the results. The higher the value the more random the output, while a lower value makes it more deterministic[57]. Then a classification chain is created, that combines the model with a structured output to classify GitHub issues based on pre-defined prompt templates. The prompts asked the model to classify the issues based on definitions from ISTQB certification book[5]. The output scheme and prompt template given to model are shown in the figure 16 below:

```
# Define the output schema
class TestingTypeClassification(BaseModel):
    """Classification of GitHub issue text with testing type."""
    classification: str = Field(..., description="The testing type of the GitHub issue text, based on ISTQB categories.")

# Define the prompt template
PROMPT_TEMPLATE = """\
Classify the following GitHub issue text into the appropriate testing type based on ISTQB guidelines.

GitHub Issue Text:
{text}

Testing Type:
"""
```

Figure 16 Output scheme and prompt template for OpenAI model.

For example, one of the issues in the dataset is as follows:

‘performance test tools in repo i think it would be valuable to have some infrastructure in place to allow for performance testing this would let us see where this library compares to other rabbitmq clients and will help quantify performance increases and potential decreases version to version or while developing i dont have any specific tools in mind and will begin researching if this idea is given the green light do you think this idea is worth adding this idea was inspired by your change that reduces the number of syscalls 33 and i am curious how much of a performance increase was achieved by the reduction of syscalls kind of test issue is that’ – *issue from GitHub dataset*

Now this issue is related to performance testing which according to ISTQB is a type of functional testing [5, p. 50]. So, the model should classify this issue to non-functional test based on the ISTQB guidelines.

Then the classification chain is invoked and ThreadPoolExecutor with 10 workers is used to reduce the processing time. The classification results from large language model are as follows in table. There was a total of 10000 issues.

Testing Type	Number of issues
Functional Testing	2827
Non-Functional Testing	1125
Structural Testing	405
Testing Related to Changes	4912
Unclassified	731

Table 12 Classification of test types using large language models.

The classification is also shown in figure 17 below:

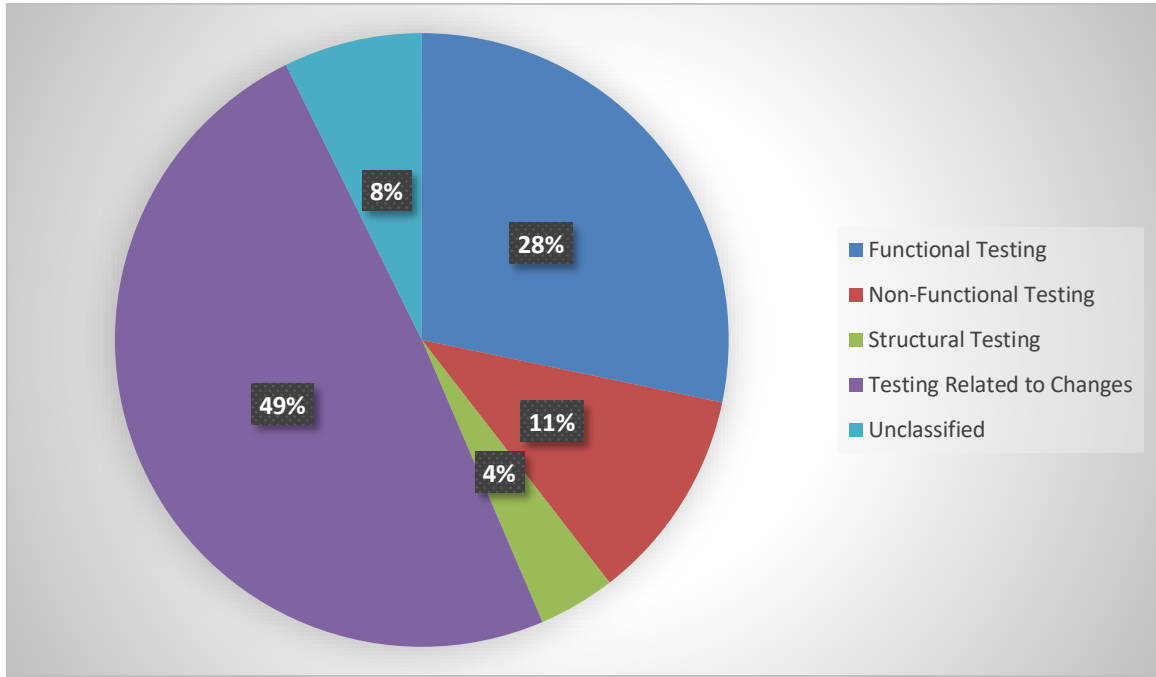


Figure 17 Pie-chart indicating percentage distribution of test types in dataset.

Testing Type	Number of issues (LLM)	Number of issues (rule based)
Functional Testing	2827	6,571
Non-Functional Testing	1125	4,328
Structural Testing	405	5,855
Testing Related to Changes	4912	6,367
Unclassified	731	11,109

Table 13 Comparison of LLM and rule-based approach.

Table 12 shows the distribution of the testing types is almost identical to rule based classification with testing related to changes (49%) and functional testing (28%) more dominant than non-functional(11%) and structural testing(4%). However, the number of issues in each category is quite different from the rule-based approach as shown in table 13. Also, the number of unclassified issues is much less than rule-based approach indicating its flexibility in classification as

compared to rule-based approach. It indicates that LLM are better at generalizing patterns and identifying issues that do not fit neatly into predefined rules. However, this machine learning based classification is expensive, time-taking and the accuracy of the results depends on the real categories the issues belong to. So, there are a few limitations to this approach which are discussed in the discussion section along a manual evaluation of how good the approach is also described in discussion section. But overall, this approach seems to classify more issues to test types than rule-based approach. Also, the semantics and meaning of the words in the issue description are taken into consideration with this model.

4.5 RQ4: Does the use of transformers improve the identification of testing debt over traditional methods (SVM, Text CNN)?

To test how much the use of transformer model has improved the process of classification a previous study done by Li et al., is utilized [33] where self-admitted technical debt was identified using machine learning approach. But the approaches utilized are traditional machine learning. In this approach the types of technical was labelled manually by the researchers based on their experience. This dataset had a total of 4200 technical debt issues from which almost 600 issues were test debt issues. This approach utilized traditional machine learning approaches such as SVM and Text CCN to classify the technical debt into distinct categories. The same approach is used and trained and tested on primary GitHub dataset to see how much transformer model has improved the classification of test debt issues as compared to traditional machine learning models.

SVM:

The training of SVM model was done using the test debt dataset and the testing is done using a separate dataset to avoid data leakage. The model used a radial basis function (RBF) kernel, which is optimized by a grid search approach that focuses on hyperparameters such as C (penalty parameter) and gamma (kernel coefficient). The values of C and the gamma are set to 10 and 0.01. These values are chosen based on their performance in cross-validation. The vectorization is

done using TF-IDF vectorization. The accuracy of the model on the training data is as follows in the table 14:

Metric	Value
Accuracy	73%
Precision	0.6739
Recall	0.7750
F1 Score	0.7209

Table 14 SVM performance on training data for testDebt/ not_testDebt dataset.

The accuracy of this model on training data is relatively less compared to transformers approach both for DistillBert and RoBERTa. But the training time is amazingly fast as compared to transformers approach. Where in transformer the model took 70% more time for training. The performance of model on test dataset is given below in table 15:

Metric	Value
Accuracy	76%
Precision	0.7727
Recall	0.7727
F1 Score	0.7727

Table 15 SVM performance on test data for testDebt/ not_testDebt dataset.

From these matrices it is clear that transformers have higher accuracy indicating higher overall performance. In precision the transformers again outperform the performance of the SVM model. It shows that transformers gave a higher number of true positives. Better recall of transformer model indicates that transformer model correctly identifies a higher percentage of all actual positives. Finally, F1

Score is also high for Transformers model, which points to a better balance between precision and recall for transformers as compared to SVM.

Overall, transformer model shows a 6% higher accuracy, 5.61% higher precision, 5.08% better recall and 5.34% better f1 score as compared to SVM model for test debt.

TextCNN:

The other traditional model utilized is TextCNN. For this model TensorFlow tokenizers and pad sequence are used. Then the embedding and 1D convolution of the layers is also performed using TensorFlow. The model is compiled with the Adam optimizer and binary cross-entropy loss. The model training is done in 5 epochs. The results of the model training are as follows in the table 16:

Epoch	Accuracy	Training Loss	Validation Accuracy	Validation Loss
1	53%	0.6904	0.4607	0.6871
2	86%	0.6374	0.7528	0.6634
3	96%	0.5987	0.7416	0.6383
4	95%	0.5474	0.8315	0.6062
5	96%	0.4902	0.8315	0.5611

Table 16 TextCNN performance on training data for testDebt/ not_testDebt dataset.

The performance on training data for Text CNN is almost like transformer model with few epochs outperforming transformers model. The accuracy in the final three epochs is very high as compared to transformers model indicating better performance of TextCNN in training phase. The performance of model on test data set is as follows in the table 17:

Metric	Value
Accuracy	45%
F1 Score	0.4097
Precision	0.4471
Recall	0.378

Table 17 TextCNN performance on test data for testDebt/ not_testDebt dataset.

Although the model had extremely superior performance on training data but the model's performance on unseen data is very poor as compared to transformers as well as compared to SVM. Overall, transformer model shows a higher accuracy (almost 80% more accurate) , higher precision, better recall and better f1 score as compared to TextCNN model for test debt

4.6 RQ5: How does the performance of transformer-based models trained on data with ground truth (labeled by developers) compare to the performance on manually labeled datasets (researchers) in identifying test debt in issue tracking systems?

To see how the accuracy of the model change based on the choice of datasets the transformer model is trained on external dataset obtained from previously done research[12] by Li et al. The parameters of the model (num of epochs, choice of tokenizer and transformer model) are kept the same as before in RQ 1.1 to do a fair comparison. In this external dataset each self-admitted technical debt was further classified into diverse types of technical debt i-e test, code, and documentation debt. This classification was done manually by the researchers based on definitions, their experience which makes it different from the dataset used in the thesis as shown in Table 18. As the dataset had diverse types of technical debts tagged in it only the test debt data from the full dataset is used based on our requirement and to compare the results.

Type	Indicator	Definition	#	#	%
Architecture debt	Violation of modularity	Because shortcuts were taken, multiple modules became inter-dependent, while they should be independent.	46	87	2.7
	Using obsolete technology	Architecturally-significant technology has become obsolete.	41		
Build debt	Over- or under-declared dependencies	Under-declared dependencies: dependencies in upstream libraries are not declared and rely on dependencies in lower level libraries. Over-declared dependencies: unneeded dependencies are declared.	25	64	2.0
	Poor deployment practice	The quality of deployment is low that compile flags or build targets are not well organized.	39		
Code debt	Complex code	Code has accidental complexity and requires extra refactoring action to reduce this complexity.	30	1246	38.0
	Dead code	Code is no longer used and needs to be removed.	121		
	Duplicated code	Code that occurs more than once instead of as a single reusable function.	40		
	Low-quality code	Code quality is low, for example because it is unreadable, inconsistent, or violating coding conventions.	856		
	Multi-thread correctness	Thread-safe code is not correct and may potentially result in synchronization problems or efficiency problems.	40		
	Slow algorithm	A non-optimal algorithm is utilized that runs slowly.	159		
Defect debt	Uncorrected known defects	Defects are found by developers but ignored or deferred to be fixed.	25	25	0.8
Design debt	Non-optimal decisions	Non-optimal design decisions are adopted.	935	935	28.5
Documentation debt	Low-quality documentation	The documentation has been updated reflecting the changes in the system, but quality of updated documentation is low.	342	486	14.8
	Outdated documentation	A function or class is added, removed, or modified in the system, but the documentation has not been updated to reflect the change.	144		
Requirement debt	Requirements partially implemented	Requirements are implemented, but some are not fully implemented.	67	96	2.9
	Non-functional requirements not being fully satisfied	Non-functional requirements (e.g. availability, capacity, concurrency, extensibility), as described by scenarios, are not fully satisfied.	29		
Test debt	Expensive tests	Tests are expensive, resulting in slowing down testing activities. Extra refactoring actions are needed to simplify tests.	28	338	10.3
	Flaky tests	Tests fail or pass intermittently for the same configuration.	83		
	Lack of tests	A function is added, but no tests are added to cover the new function.	158		
	Low coverage	Only part of the source code is executed during testing.	69		

Table 18 External dataset structure and labelling criteria.

The training parameters are kept the same as in RQ 1.1. The performance of the model on training data is as follows in Table 19:

Epoch	Training Loss	Validation Loss	Accuracy	F1 Scores	Precision	Recall
1	0.6103	0.1907	96.36%	[0.96, 0.9667]	[1.0, 0.9355]	[0.9231, 1.0]

2	0.1645	0.1478	96.39%	[0.96, 0.9667]	[1.0, 0.9355]	[0.9231, 1.0]
3	0.0687	0.1380	96.36%	[0.96, 0.9667]	[1.0, 0.9355]	[0.9231, 1.0]
4	0.0466	0.1109	98.18%	[0.9804, 0.9831]	[1.0, 0.9667]	[0.9615, 1.0]
5	0.0302	0.1158	96.36%	[0.96, 0.9667]	[1.0, 0.9355]	[0.9231, 1.0]

Table 19 DistillBert performance on training data manually labelled by researchers.

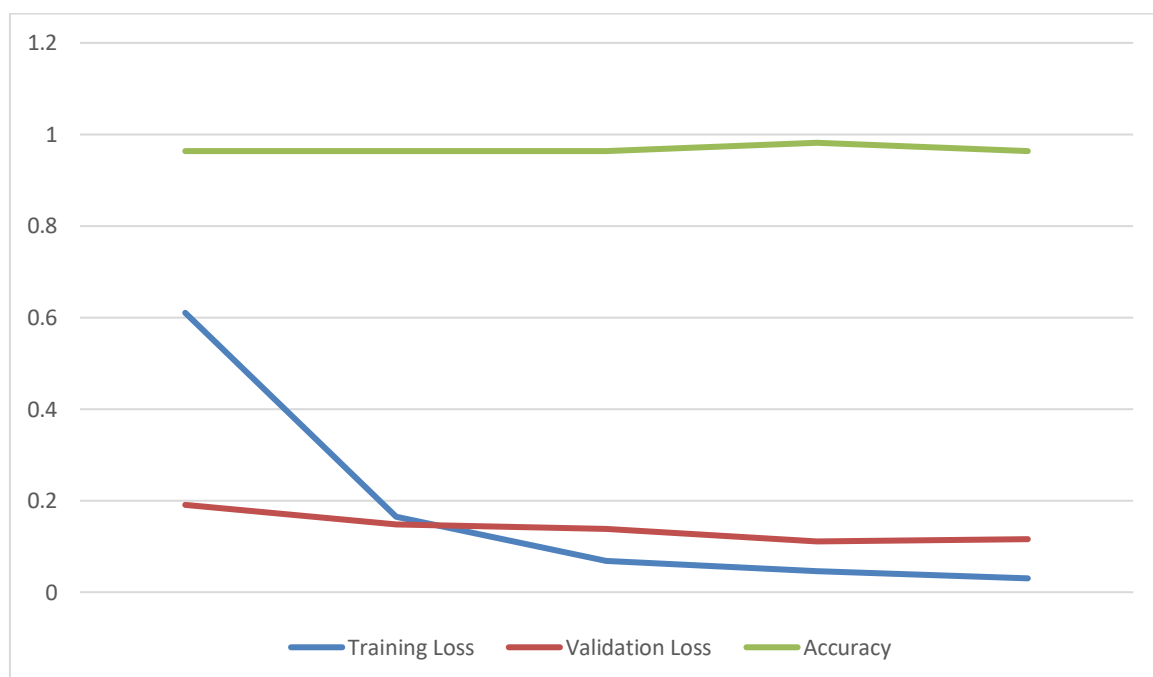


Figure 18 DistillBert Validation loss, accuracy, and training loss on testDebt/not_testDebt. labelled by researchers.

The model has slightly higher performance in training data with good generalization and no overfitting in 1st four epochs but a slight reduction in training and a slight increase in validation loss in last epoch indicates a small

fluctuation in model's performance shown in Figure 18. Overall, the model has quite superior performance even better than model trained on the dataset with ground truth. Now for the test dataset which was ten percent of the whole dataset and was not part of the training dataset the model's performance is as follows:

Metric	Value
Accuracy	78%
F1 Score	0.79
Precision	0.75
Recall	0.84

Table 20 DistillBert trained on dataset labelled by researchers' performance on test dataset.

Table 20 shows that the model, when tested on unseen data, shows very good results with accuracy of 78%. The accuracy of the model is slightly less (almost 3%) as compared to the previous model in RQ 1.1 where accuracy was 81% as shown in Table 21 . Same is for the other parameters i-e F1 score and precision except for recall which is slightly higher than the previous model. The transformer that choice of dataset does not really affect the accuracy of model and the transformer model is good in classification of test debt irrespective of the labelling done by developers or researchers.

Metric	Performance on test dataset (Labelled by researchers)	Performance on test dataset (Labelled by developers)
Accuracy	78%	81%
F1 Score	0.79	0.816

Precision	0.75	0.812
Recall	0.84	0.814

Table 21 Comparison of results of models trained on different datasets on test data.

Chapter 5

5.1 Discussion:

In this chapter, I will go through the insights gathered from my study and this chapter is structured around all the research questions involved. For each of the questions, I will dive into the key findings and what the question means.

I will discuss what I learned from the results, reflecting on how they answer the research questions and how these findings contributed to current research. I will also consider the contribution and limitations of my study as well as threats to validity—acknowledging any factors that might affect the reliability of the conclusions—and discuss the broader implications of the findings.

Finally, I will wrap up some thoughts on the potential directions for future research suggested by my work, pointing out opportunities to build on what is being discovered. This chapter aims to tie together the research in a clear and approachable way, highlighting its impact and the next steps it inspires.

RQ1: How accurate are transformer-based models (RoBERTa, DistilBERT) in classifying project management issues as test related issues.

Transformers have gained a lot of attention in NLP for text classification tasks. Because of their ability to capture self-attention and ability to handle large dependencies between words make them extremely useful in classification tasks[58]. The aim of this research question is to see the effectiveness of two transformers-based models which are DistilBERT and RoBERTa.

The DistilBERT model showed a strong performance on classifying GitHub issues as test-related or not test related, with especially robust results on the separate test set with accuracy of 96%.

This suggested that the model can generalize from training to test data and is effective, even though the initial signs of overfitting occurred in the validation loss.

The DistilBERT transformer model proved to be a powerful tool for classifying GitHub issues based on the presence of testing-related content. The results demonstrate the potential of transformer models in automating aspects of software development project management, particularly in understanding and coordinating the nature of project issues. Further studies could investigate additional fine-tuning or a more extensive parameter search to enhance model robustness and reduce overfitting tendencies observed during validation.

The RoBERTa model also revealed a strong capacity to classify test issues within GitHub dataset, achieving high accuracy and maintaining an excellent performance across independent test sets with accuracy of 87.83%. RoBERTa, with its robust training and generalization properties, is suitable for complex text classification tasks such as identifying test debt in issue tracking systems. Future work may explore additional tuning and regularization techniques to mitigate overfitting and enhance model reliability for broader deployment scenarios.

Talking about resource usage, specifically focusing on the utilization of machine learning nodes (such as GPUs), gives an important assessment of the computational efficiency and scalability of both models DistilBERT and RoBERTa. DistilBERT is a smaller, faster, and lighter version of BERT and has half transformer layers than BERT. Each layer of this model consists of a multi-head self-attention mechanism and a feed-forward neural network same as transformers, but with reductions in the dimensionality and complexity to enhance computational efficiency. DistilBERT offers a reduction in size and increase in speed by approximately 40% and 60% respectively, compared to BERT, while maintaining up to 97% of its language understanding capabilities. This makes it highly suitable for environments with strict latency or memory constraints[59].

Roberta on the other hand enhances BERT by optimizing its pre-training approach. RoBERTa has the same architecture as BERT, but with multiple layers of bidirectional transformers. Every layer has a multi-head self-attention

mechanism and a position-wise fully connected feed-forward network. It also differs in its training method. It is trained on a much larger dataset and for more iterations compared to BERT. It also changes the BERT's masking pattern that is applied dynamically on the training data rather than static masking. The extensive training helps in better generalization of the different NLP tasks[60].

So, for this studies the DistillBert model has showed almost 9% high accuracy for classifying test and not test related issues as well as it was better in terms of speed and took less training time because of its faster speed. So, the DistillBert model is better in classifying the issues related to test and not test.

The key findings of this research question are:

- Among DistillBert and RoBERTa, DistillBert has better performance in terms of speed and accuracy for classification of test issues in project management.

RQ 1.1 How accurate are transformer-based models (DistilBERT, RoBERTa) in classifying project management issues as test-debt related issues.

The aim of this research question is to find out the ability of transformers-based approach in classifying test debt and not test debt.

The DistillBert showed excellent results in identifying the test debt and non-test debt issues in the dataset with accuracy of 81% on test dataset. RoBERTa on the other hand showed the same accuracy of 81% on test dataset.

The resource usage for both models is a bit different in case of test debt with RoBERTa taking more time in training compared to DistillBert which makes DistillBert more effective in classification of test debt issues. However, these results are based on a small dataset for both training and testing.

The results of transformer-based approach suggest a strong potential of using this approach in specialized NLP tasks within software engineering. Both models (RoBERTa and DistilBERT) had particularly good accuracy of 81% on test dataset with good generalization properties.

To address the issue of small training dataset for test debt classification I used two pre-trained transformer models one technical debt classifier[54] from the work done by Skryseth et al. and other test issue classifier from RQ1. These models are trained on huge datasets and have very good accuracy and generalization on unseen data separately. By combining the true positives of these models, the aim was to increase the accuracy and overall performance and compare it with test debt classifier.

But the combined accuracy was very low indicating simply selecting the best individual models does not necessarily lead to a better combined result[55]. This could be due to conflicting prediction criteria or the models not being cooperative in their decision boundaries. Right now, in both models and in combining the models there is no mechanism to take in account the confidence that a classification decision is made correctly or not. So, more work needs to be done to use model assembling techniques that can weigh the output of each model based on its confidence and that might increase the performance of the model. Each prediction will have a confidence score that will be based on the accuracy of the model or certainty of its predictions or its performance on validation dataset. This might improve the results of combining the results from 2 models to improve prediction accuracy and robustness.

The key findings from this research question are:

- DistillBert and RoBERTa are equally good in classification of test debt issues in term of accuracy while for resource usage DistillBert outperforms RoBERTa.
- Simple ensemble of two models doesn't always guarantee better results.

RQ2: What are the most common types of testing issues in the dataset from GitHub?

This research question is focused on classifying test issues into different test types. The test types are based on ISTQB certification book.

The most common types of testing issues in the dataset are functional (19%), change (19%), structural (17%) and non-functional (13%) testing related issues respectively.

These results are based on rules defined from the ISTQB certification book. Now there are potential reasons why there is this distribution like this. The top common types are functional and change testing which matches the industry standards where functional software testing is one of the most straightforward and essential types of quality assurance for engineers to determine whether all of the components of a piece of software operate according to its functional requirements[61]. One reason for the high ratio of these types could be more test automation in this type of testing. According to testing statistics 2023 almost 78% of regression and functional testing is automated[62]. Previous studies done by C. Persson et al., and K. Wiklund et al. revealed that the big reason of having technical debt is test automation. Because there is no clear, detailed, guidelines on how to design, implement, and maintain an automated test execution system so that technical debt be kept on an acceptable level[63], [64]. As, test debt is type of technical debt it could be the reason that due to high percentage of test automation in functional and change testing the issues related to functional and change testing are more. However, the data used for this research question is not purely test debt related data with small portion issues related to test debt and major portion is only test issue, so the reason is not very obvious and more work is needed to be done to evaluate the relationship between these.

Moving towards structural testing, the percentage is 17%. It is relevant to mostly code and less number that could indicate that term related to these are mostly code related in issues description and label and not test related, so the rules defined are not very good at capturing these issues. Lastly, non-functional testing issues although are as much important as functional issues but there exists evidence that Non-functional tend to come second class to functional testing[65]. This is due to lack of documentation of non-functional requirements in documentation. As, Agile software engineering emphasizes on “continuous attention to technical excellence”[66] . These methods focus solely on immediate

feedback and suggest the adequate availability of skilled software developers to mitigate potential quality risks. Unfortunately, agile values and principles often seem to be adopted naively [67], i.e lacking proper documentation. This lack of documentation of non-functional issues indicates their low visibility in the dataset.

Also, 32% of the dataset remained unclassified indicating potential limitations of rule-based approach. This shows that this approach simply breaks when faced with issues that don't match the predefined rules. Also the rules defined doesn't account for the context or semantic meaning beyond simple pattern matching, which might lead to misclassifications or omissions[68]. So, rule-based approach alone is not a particularly good classification technique, and this is potential limitation of this approach and utilizing it with machine learning and making hybrid model can give more reasonable results.

The key finding from this research question are as follows:

- Rule based approach is faster and simpler approach to classify test issues into different test types.
- This is rigid approach that doesn't account semantic and meaning of the words and breaks easily if no rule match giving a lot of unclassified issues.

RQ3: To what extent are large language models able to classify project management test issues as specific testing types?

The aim of this research is to see to what extent LLM can classify the test issues to different test types based on ISTQB definitions. The LLM (Large Language Model) tagging approach that uses advanced natural language processing techniques to classify GitHub issues into different testing types gives very promising results of classification.

The most common types of testing issues according to in the dataset are change (49%), functional (28%), structural (4%) and non-functional (11%) testing related issues respectively. The distribution of the types is remarkably similar to rule-based model.

However, for large datasets this process is expensive, that's why the whole data set was not utilized. The reduction in the number of unclassified issues indicates that the model has a better ability to classify issues than the rule-based approach based on a smaller number of unclassified issues. However, this classification does not indicate anything about accuracy of the results due to lack of ground truth available for specific test types available. However, some manual evaluation showed that this model is better than rule based approach and is described below.

Although the classification is balanced as compared to however the results of the model are also influenced by the prompts provided. For example, if more emphasis is on one type there is possibility that model assigns more issue to that kind. For example, in the results shown in Table 12 almost half of the issues are related to 'testing related to changes' as it has more textual emphasis as compared to other types. Other concerns regarding the use of LLMs, specifically GPT-4, are the potential for bias in the model's training data, which can influence classification outcomes. Making sure that the model is trained on a vast and representative dataset is especially important to reduce this risk. The theoretical estimates are not empirically tested, but they provide a structured approach to understand the potential impacts and efficiencies the models may offer.

A fundamental limitation of both rule-based and LLM approach discussed in this research is the reliance on theoretical analysis and synthetic data due to the lack of real-world data for empirical testing. This limitation may affect the generalizability of the models to real world scenarios. For the dataset that is used for rule based and llm approach there were only a few issues (almost 1-2%) that were tagged based on ground truth that caused limitation to test which model performed the best based on accuracy, recall and other matrices. However, the results of the model were compared with each other to see how they classified the into different types which showed that agreement rate between these models is 10.68%. This comparison is just based on a small portion (30%) of the full dataset as LLM approach was just applied on small dataset. The low agreement rate indicates that two models understood and classified data in different ways as the

rule-based model does not have any semantic knowledge and it classified just based on set of keywords while the other model has vast knowledge of semantic, but accuracy of results of later might be biased based on training data.

Because of lack of ground truth a small part of the dataset (only 50 issues) is manually analyzed to classify those issue into different types of test to see which approach has classified the issues better. Rule based approach classified only 8 of those 50 issues and the rest remained unclassified because they did not contain any keywords in them that were defined in that approach. Of those 8 issues only 4 issues were correctly classified. llm model classified all the issues in certain test types and none remained unclassified and 30 out of 50 are classified correctly. For the rest that are not correctly classified. The main pattern is not correctly classified issues was that they labelled as change testing instead of functional testing and the potential reason of this biasness is already discussed which is due to the more contextual emphasis in the prompt. However, this evaluation suggested that llm are way better than classification of test types than rule-based approach.

So, there is a major opportunity for future research to explore the application and applicability of these models in live environments with the data labelled based on ground truth to evaluate the results, which could provide valuable feedback for iterative improvements and practical insights into the performance of these models.

Before applying the above-mentioned approach, an attempt was made to classify the issues into several types using llama 2 large language model which is also from the family of transformers-based model because of its advanced capabilities in generating language-based models for classification. Despite lofty expectations, the model mostly generated off-topic outputs that were not relevant to testing even though in prompts the model was instructed to give only specific test type. One reason could be the training data used for llama 2. If the training data did not have clear example of the test types it is possible that model could not classify them properly in specified test types.

The key take aways from this research question are:

- The large language models classified the issues better than rule-based approach based on number of unclassified issues.
- The classification of rule-based approach is biased based on the prompts as well as by training data.

RQ4: Does the use of transformers improve the identification of testing debt over traditional methods (SVM, Text CNN)?

The aim of this research question is to see how well the transformer models performed as compared to traditional machine learning models.

The transformers showed exceptionally good performance (accuracy 81%) than both SVM (accuracy 76%)and TextCNN (accuracy 45%) because of its advanced attention mechanism and ability to capture the context as well as semantic of the text issue.

The ability of transformers to process the whole issue at once gives better results than local receptive fields of CNN and high-dimensionality space of the SVM model. Also, transformers have high flexibility to be adapted in wide range of natural language processing tasks, but they require significant computational power making them less accessible in an environment with limited hardware or in real time applications.

The performance of SVM is also overall good giving a particularly good performance on test data set with accuracy of 76% as well because the feature space was clearly definable, and the dataset was also not extensive. The training time of the model was also less as compared to transformer model. So, for the dataset where quick, and interpretable results are required the SVM model is particularly good.

Text CNN are also powerful for when the local patterns or keywords in dataset indicate the output but their performance on test data showed their in-efficiency in classification task for test debt (accuracy 45%). But this model requires less

training data and resources as compared to transformers. Also feature engineering is especially important in CNNs and size of dataset does not matter much for CNNs, and they have better ability to handle large datasets unlike SVM.

Li et al. discussed in their paper about classification of technical debt that SVM among the traditional machine learning model the best one with average accuracy of 86% when tested on manually labelled dataset. TextCNN on the other hand is less accurate 68% when tested on same test dataset. The results of this thesis agree with the work done by Li et al. about the suitability of models based on the classification task.

In practice, the choice of model is purely dependent on requirements, for example, size of training dataset available, number of computational resources as well as model interoperability. For future research a hybrid model combining the good characters can give promising results using less resources. For example, SVM or Text CNN are used to preprocess data or extract features and then using those features or preprocessed data into a transformer model could optimize both performance and computational efficiency. As less time would be utilized for training data.

The key take aways from this research question are:

- Transformer based models have extremely better performance than traditional machine learning method including 6.85% better performance than SVM and with respect to TextCNN transformers are 80% better.
- The training time for transformers is two times more than traditional machine learning algorithms even on small dataset because of complex architecture and attention mechanisms.

RQ5: How does the performance of transformer-based models trained on data with ground truth (labeled by developers) compared to the performance on manually labeled datasets (researchers) in identifying test debt in issue tracking systems?

The aim of this research question is to see how the accuracy of transformer model changes based on selection of data set labelled by researchers and developers.

Comparing the dataset is quite interesting as one dataset was based on ground truth that that developers have self-admitted in the issues text. So, this labelling is considered as ground truth and the dataset is 100% correctly labelled. But the other data set, although is carefully analyzed by the researchers and reason of labelling and indicators of test debt were also given for each issue it is still not based on the ground truth but on the experience and understanding of the researcher.

The transformers model showed a particularly good performance for developers labelled (81%) and researchers labelled (78%). Table 21

The training accuracy for the model labelled by the researchers is slightly better than the training accuracy for the model labelled by the developers. But the difference is slight. However, for the accuracy of the models on the unseen data the training accuracy for the model labelled by the developers has higher accuracy than the other. Again, the difference was exceedingly small.

The dataset creation for manually tagging is a lengthy process and a lot of expertise in subject, time and effort is required and still the data is not expected to be 100% correctly labelled. This accuracy is affected by the nature of data and subjectivity involved in the data. As the researchers involved in labelling are not developers and variation in interpretation of the issue text among different researchers can lead to inconsistencies. Also, manual tagging is more time consuming and more effort seeking making from humans which further complicates the effort to ensure complete accuracy.

Li et al., Soliman et al. and Avgeriou et al. suggested in their paper regarding classification of self-admitted technical debt that dataset labelled by researchers tends to achieve better generalizability and high accuracy if the size of dataset is small and moderate[12]. Their words show that systematic labelling processes done by researchers can improve training as the data provided is clear and

consistent. Compared to the work done by Li et al., Soliman et al. and Avgeriou et al. this thesis used a small size dataset and suggested that the dataset labelled by the developers have better accuracy and generalizability than the data labelled by researchers.

The key take aways from this research question are:

- Developers labelled data has better generalizability and accuracy in classification of test debt when trained and tested on small dataset.
- The difference in accuracy and generalizability for data labelled by developers and researchers is quite small.
- Taking in consideration time and effort using dataset tagged by developers in issue trackers is better option as it is both time saving and based on ground truth.

5.2. List of contributions:

The key contribution from this thesis are as follows:

- Trained and tested two transformer-based models (DistillBert and RoBERTa) for classification of test issues in issue tracking system with accuracy of 96% and 87.83% on test sets.
- Trained and tested two transformer-based models (DistillBert and RoBERTa) for classification of test debt issues in issue tracking system with accuracy of 81% on test set for both models.
- Provided a straightforward and simple rule-based classifier for classification of test types in test issues from issue tracking systems.
- Provided a large language model-based approach for classification of test types in test issues from issue tracking systems.
- Provided evidence that discovered that simple ensemble of two better performing models does not guarantee better combined results.
- Evidence that transformer-based approaches (DistillBert, RoBERTa) are better than traditional machine learning approaches.

- Evidence that supports that choice of data labelling for datasets used for training the models done by developers is better than labelling done by the researchers.
- Made all the classifiers and training as well as testing data available to be used in practice and for future research.

5.3. Implication of research and practice:

Based on the findings from the work, this thesis suggested the following directions for researchers and software practitioners:

Recommendation for research:

- The researchers can use this automatic classification of issues to conduct further study to measure and prioritize the test issues and test debt in issue tracking systems.
- To enable research in this area, the datasets are made available publicly ³.
- Although this study experimented generalizability of transformer-based approach across issues from issue tracking systems the scope of this study is still limited to only one issue tracking system which is GitHub. Thus, further research can investigate the applicability of this approach to other projects and other issue tracking systems.
- In this thesis the results of two models are ensembled for RQ 1.1. These were the models that had particularly good performance independently, but the results were not very good so for future using model ensemble techniques that can weigh the output of each model based on its confidence and that might increase the performance of the model. Each prediction will have a confidence score that will be based on the accuracy of the model or certainty of its predictions or its performance on validation dataset.
- As mentioned in limitation of work the results of the rule based and large language model both are not tested against real world data that is based on ground truth so there is a major opportunity for future research to explore the application and applicability of these models in live environments with the data

³ <https://github.com/nehadurani996/TestDebt>

to evaluate the results, which could provide valuable feedback for iterative improvements and practical insights into the performance of these models.

- As transformer model has excellent accuracy but they are training time and resources that make them hard to be implemented in real time applications. For future research a hybrid model combining the good characters of traditional models that require less training time can give promising results using less resources. For example, SVM or Text CNN to be used to preprocess data or extract features and then using those features or preprocessed data into a transformer model could optimize both performance and computational efficiency. As less time would be utilized for training data.

Recommendations for software practitioners:

- Tool developers can use the classifiers from the research in their toolsets and dashboards to experiment their applicability in practice from live projects.
- The classification approaches can help project managers and developers to evaluate the quality of their work and projects. For example, project managers can use these to keep track of test issues and test debt in issue tracking systems along evolution.

5.4. Threats to validity:

Validity is defined as the extent to which a concept is accurately measured in a quantitative study[69]. Three aspects of the validity are discussed for this thesis are internal, external and construct validity.

Internal validity:

Internal validity determines whether the changes observed in the conditional variable result from changes in the independent variable or some other factor[70]. To ensure the classifiers correctness the dataset used for training and testing needs to be correctly labelled which is resolved using the labelling done based on ground truth. This is considered in this thesis where the issues are labelled by the developers in the issue tracking system. Also, the labelling criteria for test types is based on the testing standard book for ISTQB

certification. However, there are many factors that determine if the issue is related to test, test debt or certain test type that may not be captured by in the text and causing the model not to capture that aspect of the issue which is threat to internal validity.

External validity:

External validity refers to the degree to which the findings of a study can be generalized to other populations and contexts[70] which applies to generalizability of the given approaches. Because the classifiers are trained on the dataset created from the issue tracking system and tested on the test set separated from that set there could be the issue that the models generalized only on those datasets. To avoid this issue the models are tested on external dataset from already done research on SATD[12].

Construct validity:

Construct validity refers to the degree to which a research study accurately measures the theoretical construct it intends to measure[70]. The construct validity is done by measuring the accuracy of machine learning models for classification using matrices like accuracy, precision, recall and F1 score. These are well established matrices for these kind of approaches and has been done in the previously done approaches by Skryseth et al., Shivashankar et al. [54], Li et al. [12] and Yu et al. [33]. However, there is a doubt if the developers actually tagged the issue correctly. There is a threat in the fact that, if many issues are tagged based on misunderstanding, the model would learn that and spread it to the tagged issues. However, comparing researchers-developers results lower the probability of that threat.

Reliability:

All the related data including python scripts and datasets are made available as supplements to increase the reliability of the thesis so that the data can be used to replicate the same results. But choice of data, analysis and alternative methodology can lead to different results and findings.

5.5. Limitations:

One of the limitations of this study is lack of comprehensive dataset focused on test debt. This constraint restricts the depth of analysis within the study of test debt and may also have affected the accuracy of the results of transformer model for test debt.

Another fundamental limitation of this thesis is in both rules based and LLM approach. These approaches are only based on theoretical analysis and synthetic data because of the lack of real-world data for empirical testing. This limitation may have an impact on generalizability of the models to real-world scenarios. For the dataset there were only a few issues (almost 10%) that were tagged based on ground truth that caused limitation to test which model performed the best based on accuracy, recall and other matrices. Also, different types of test data were unbalanced which might lead to biased statistical analysis.

The other limitation is overfitting problem in training phases of the transformers model which was tried to be solved using early stopping but has affected the accuracy of model, but the overfitting was in last training phases and had negligible effect because models showed excellent generalization in testing phases. The large language model was expensive to implement for the full dataset so only a portion of the dataset was used for that model.

Chapter 6

6.1 Conclusion:

This thesis finds out the transformer-based approaches (DistillBert and Roberta) are excellent in classifying the GitHub issues to test and not test related issues. Both models have high accuracy (**DistillBert 96%**) and (**RoBERTa 87%**) with excellent ability in generalization on unseen data. The sub question of first question is also relevant to transformers-based approach and how good are they in classifying the GitHub issues to test debt related issues. For test debt to the transformers model showed exceptionally satisfactory results (**DistillBert 81%**) and (**RoBERTa 81%**) on test dataset. The generalization of these models was also particularly good on unseen data from previously done studies. This automatic classification can reduce the manual work, specially for the extensive projects with a lot of issues.

The rule-based approach proposed in this thesis identified that the classification of 4 types of testing is **Functional (19%)**, **Non-Functional (13%)**, **structural (17%)**, and **testing related to changes (19%)**. The approach was quick and easy, but the number of **unclassified issues (32%)** was very high and semantics and meaning of the issue description was not considered.

The large language model-based approach proposed in this research identified that the classification of 4 types of testing is **Functional (28%)**, **Non-Functional (11%)**, **structural (4%)**, and **testing related to changes (49%)**. This approach was more flexible than the previous one with a very few percent of **unclassified issues (8%)**, but this approach is time taking, expensive and results can be affected by the bias in training data.

The fourth question asked about does the use of transformers improve the identification of testing debt over traditional methods (SVM, Text CNN). The answer is yes. The transformer model is almost **6.58% better than SVM** and **80% better than TextCNN in terms of accuracy**. This suggested that use of latest

transformer-based machine learning approach is quite better than traditional approaches, but the training time of the transformer-based machine learning approaches is much longer than the traditional approaches.

Lastly how different is the performance of transformer-based models trained on data with ground truth (labeled by developers) change compared to their performance on manually labeled datasets (researchers) in identifying test debt in issue tracking systems. The answer is there is only a slight difference in accuracy of both models when tested on the unseen dataset. The model trained on the data tagged by **developers** has **(3.85%) higher accuracy** than the model trained on data tagged by **researchers**. So, using the data tagged by developers in issues trackers can save time of manual labeling and gives better accuracy.

Bibliography

- [1] H. Mohanty, J. R. Mohanty, and A. Balakrishnan, Eds., *Trends in Software Testing*. Singapore: Springer Singapore, 2017. doi: 10.1007/978-981-10-1415-4.
- [2] C. Gaston, N. Kosmatov, and P. Le Gall, Eds., *Testing Software and Systems: 31st IFIP WG 6.1 International Conference, ICTSS 2019, Paris, France, October 15–17, 2019, Proceedings*, vol. 11812. in Lecture Notes in Computer Science, vol. 11812. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-31280-0.
- [3] M. Salahat *et al.*, “Software Testing Issues Improvement in Quality Assurance,” Mar. 2023, pp. 1–6. doi: 10.1109/ICBATS57792.2023.10111145.
- [4] “Software testing: a research travelogue (2000–2014) | Future of Software Engineering Proceedings.” Accessed: Mar. 22, 2024. [Online]. Available: <https://dl.acm.org/doi/10.1145/2593882.2593885>
- [5] Dorothy Graham, Erik van Veenendaal, Isabel Evans, and Rex Black, *Foundation of software testing ISTQB certification*, 4th ed.
- [6] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, “Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162),” p. 29 pages, 2016, doi: 10.4230/DAGREP.6.4.110.
- [7] W. Cunningham, “The WyCash portfolio management system,” *SIGPLAN OOPS Mess.*, vol. 4, no. 2, pp. 29–30, Dec. 1992, doi: 10.1145/157710.157715.
- [8] “Neural Network-based Detection of Self-Admitted Technical Debt: From Performance to Explainability: ACM Transactions on Software Engineering and Methodology: Vol 28, No 3.” Accessed: May 12, 2023. [Online]. Available: <https://dl.acm.org/doi/10.1145/3324916>
- [9] K. Dai, “Identifying technical debt through issue trackers,” University of British Columbia, 2018. doi: 10.14288/1.0374920.
- [10] “(PDF) Technical Debt in Test Automation.” Accessed: Mar. 26, 2024. [Online]. Available: https://www.researchgate.net/publication/254034665_Technical_Debt_in_Test_Automation
- [11] “The Risks of Acceptance Test Debt | Cutter Consortium.” Accessed: Mar. 26, 2024. [Online]. Available: <https://www.cutter.com/article/risks-acceptance-test-debt-416216>

- [12] Y. Li, M. Soliman, and P. Avgeriou, "Identifying Self-Admitted Technical Debt in Issue Tracking Systems using Machine Learning." arXiv, Feb. 04, 2022. doi: 10.48550/arXiv.2202.02180.
- [13] "5 Main Software Testing Issues and Methods to Resolve Them," Testbytes Softwares. Accessed: Mar. 28, 2024. [Online]. Available: <https://www.testbytes.net/blog/software-testing-issues/>
- [14] "Common Testing Problems: Pitfalls to Prevent and Mitigate." Accessed: Mar. 28, 2024. [Online]. Available: <https://insights.sei.cmu.edu/blog/common-testing-problems-pitfalls-to-prevent-and-mitigate/>
- [15] S. Bajaj, S. Raj, S. Mantri, and K. Wadibhasme, "Github in software testing," *5th International Conference on Software Security (ICSS 2019)*, 21 -22 December 2019, Sydney, Australia, pp. 221–231, 2019.
- [16] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, Mar. 2015, doi: 10.1016/j.jss.2014.12.027.
- [17] Y. Guo *et al.*, "Tracking technical debt — An exploratory case study," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 528–531. doi: 10.1109/ICSM.2011.6080824.
- [18] "Managing technical debt in software-reliant systems | Proceedings of the FSE/SDP workshop on Future of software engineering research." Accessed: May 04, 2023. [Online]. Available: <https://dl.acm.org/doi/10.1145/1882362.1882373>
- [19] E. da S. Maldonado, E. Shihab, and N. Tsantalis, "Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1044–1062, Nov. 2017, doi: 10.1109/TSE.2017.2654244.
- [20] "Investigating the impact of design debt on software quality | Proceedings of the 2nd Workshop on Managing Technical Debt." Accessed: May 04, 2023. [Online]. Available: <https://dl.acm.org/doi/10.1145/1985362.1985366>
- [21] "The financial aspect of managing technical debt: A systematic literature review - ScienceDirect." Accessed: Mar. 30, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584915000762?casa_token=P1l-NF3Wf_AAAAAA:WZPFWk9zHQ0jGYpIM7Ut1V965-ZYvIVFQpSg8-oWnRI2fERH1ljt7BHQwTMyLByr733Z06Loj_c
- [22] A. Martini, T. Besker, and J. Bosch, "Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations," *Science of Computer Programming*, vol. 163, pp. 42–61, Oct. 2018, doi: 10.1016/j.scico.2018.03.007.
- [23] "(PDF) Exploratory Testing as a Source of Technical Debt | Marco Torchiano - Academia.edu." Accessed: Mar. 30, 2024. [Online]. Available:

https://www.academia.edu/19900388/Exploratory_Testing_as_a_Source_of_Technical_Debt

- [24] “Natural Language Processing (NLP): What it is and why it matters.” Accessed: Apr. 01, 2024. [Online]. Available: https://www.sas.com/no_no/insights/analytics/what-is-natural-language-processing-nlp.html
- [25] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, Jul. 2015, doi: 10.1126/science.aaa8415.
- [26] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” *SN COMPUT. SCI.*, vol. 2, no. 3, p. 160, Mar. 2021, doi: 10.1007/s42979-021-00592-x.
- [27] A. Sun, E.-P. Lim, and Y. Liu, “On strategies for imbalanced text classification using SVM: A comparative study,” *Decision Support Systems*, vol. 48, no. 1, pp. 191–201, Dec. 2009, doi: 10.1016/j.dss.2009.07.011.
- [28] H. Phan, M. Krawczyk-Becker, T. Gerkmann, and A. Mertins, “DNN and CNN with Weighted and Multi-task Loss Functions for Audio Event Detection,” Aug. 2017.
- [29] A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: Apr. 01, 2024. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
- [30] “Text classification.” Accessed: Apr. 01, 2024. [Online]. Available: https://huggingface.co/docs/transformers/tasks/sequence_classification
- [31] “A Contextualized Vocabulary Model for identifying technical debt on code comments | Semantic Scholar.” Accessed: May 04, 2023. [Online]. Available: <https://www.semanticscholar.org/paper/A-Contextualized-Vocabulary-Model-for-identifying-Farias-Mendon%C3%A7a/317494fbce92d2753c50ff232ca89d3c2c2b4c71>
- [32] S. Wattanakriengkrai *et al.*, “Automatic Classifying Self-Admitted Technical Debt Using N-Gram IDF,” in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2019, pp. 316–322. doi: 10.1109/APSEC48747.2019.00050.
- [33] Z. Yu, F. M. Fahid, H. Tu, and T. Menzies, “Identifying Self-Admitted Technical Debts with Jitterbug: A Two-Step Approach,” *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1676–1691, May 2022, doi: 10.1109/TSE.2020.3031401.

- [34] M. Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang, "Automating Change-Level Self-Admitted Technical Debt Determination," *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1211–1229, Dec. 2019, doi: 10.1109/TSE.2018.2831232.
- [35] J. W. Creswell and J. D. Creswell, "Research Design: Qualitative, Quantitative, and Mixed Methods Approaches".
- [36] Uche M. Mbanaso, Lucienne Abrahams, and Kennedy Chinedu Okafor, "Chapter 6 Research Philosophy, Design and Methodology."
- [37] M. Mkansi and E. A. Acheampong, "Research philosophy debates and classifications: students' dilemma," 2012, Accessed: Feb. 22, 2024. [Online]. Available: <https://uir.unisa.ac.za/handle/10500/24076>
- [38] M. Mkansi and E. A. Acheampong, "Research philosophy debates and classifications: students' dilemma," 2012, Accessed: Feb. 26, 2024. [Online]. Available: <https://uir.unisa.ac.za/handle/10500/24076>
- [39] "Designing and Conducting Mixed Methods Research," SAGE Publications Inc. Accessed: Feb. 22, 2024. [Online]. Available: <https://us.sagepub.com/en-us/nam/designing-and-conducting-mixed-methods-research/book241842>
- [40] V. Markovtsev and W. Long, "Public git archive: a big code dataset for all," in *Proceedings of the 15th International Conference on Mining Software Repositories*, in MSR '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 34–37. doi: 10.1145/3196398.3196464.
- [41] Matts, "To translate or not to translate, best practices in non-English sentiment analysis," Medium. Accessed: Mar. 06, 2024. [Online]. Available: <https://towardsdatascience.com/to-translate-or-not-to-translate-best-practices-in-non-english-sentiment-analysis-144a53613913>
- [42] "Text Classification: What it is And Why it Matters," MonkeyLearn. Accessed: Mar. 20, 2024. [Online]. Available: <https://monkeylearn.com/text-classification/>
- [43] "Understanding Text Classification in Python." Accessed: Mar. 20, 2024. [Online]. Available: <https://www.datacamp.com/tutorial/text-classification-python>
- [44] "PyTorch," PyTorch. Accessed: Apr. 12, 2024. [Online]. Available: <https://pytorch.org/>
- [45] "NVIDIA cuda software and gpu parallel computing architecture | Proceedings of the 6th international symposium on Memory management." Accessed: Apr. 12, 2024. [Online]. Available: <https://dl.acm.org/doi/10.1145/1296907.1296909>
- [46] "Document Understanding - Keyword Based Classifier." Accessed: Apr. 12, 2024. [Online]. Available: <https://docs.uipath.com/document-understanding/automation-cloud/latest/user-guide/keyword-based-classifier>

- [47] “Introduction | 🦜🔗 LangChain.” Accessed: Apr. 13, 2024. [Online]. Available: https://python.langchain.com/docs/get_started/introduction/
- [48] “concurrent. futures — Launching parallel tasks,” Python documentation. Accessed: Apr. 13, 2024. [Online]. Available: <https://docs.python.org/3/library/concurrent.futures.html>
- [49] “GitHub Terms of Service,” GitHub Docs. Accessed: Mar. 22, 2024. [Online]. Available: <https://docs.github.com/en/site-policy/github-terms/github-terms-of-service>
- [50] “The European Code of Conduct for Research Integrity - ALLEA.” Accessed: Mar. 22, 2024. [Online]. Available: <https://allea.org/code-of-conduct/>
- [51] “distilbert/distilbert-base-uncased · Hugging Face.” Accessed: Apr. 16, 2024. [Online]. Available: <https://huggingface.co/distilbert/distilbert-base-uncased>
- [52] “What is Overfitting? - Overfitting in Machine Learning Explained - AWS.” Accessed: Apr. 16, 2024. [Online]. Available: <https://aws.amazon.com/what-is/overfitting/>
- [53] “FacebookAI/roberta-base · Hugging Face.” Accessed: Apr. 18, 2024. [Online]. Available: <https://huggingface.co/FacebookAI/roberta-base>
- [54] D. Skryseth, K. Shivashankar, I. Pilán, and A. Martini, “Technical Debt Classification in Issue Trackers using Natural Language Processing based on Transformers,” in *2023 ACM/IEEE International Conference on Technical Debt (TechDebt)*, May 2023, pp. 92–101. doi: 10.1109/TechDebt59074.2023.00017.
- [55] D. Abbott, “Combining models to improve classifier accuracy and robustness,” *Information Fusion - INFFUS*, Jan. 1999.
- [56] “Towards a theory of shared understanding of non-functional requirements in continuous software engineering | Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings.” Accessed: Apr. 24, 2024. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3510454.3517069>
- [57] “in AWS langchain Bedrock, how do I set temperature? · Issue #14083 · langchain-ai/langchain,” GitHub. Accessed: Apr. 24, 2024. [Online]. Available: <https://github.com/langchain-ai/langchain/issues/14083>
- [58] “Transformers for Text Classification,” Paperspace Blog. Accessed: May 10, 2023. [Online]. Available: <https://blog.paperspace.com/transformers-text-classification/>
- [59] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.” arXiv, Feb. 29, 2020. doi: 10.48550/arXiv.1910.01108.

- [60] Y. Liu *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” arXiv, Jul. 26, 2019. doi: 10.48550/arXiv.1907.11692.
- [61] P. Azorin, “The Most Important Types of QA Testing for Software Development,” The Startup. Accessed: Apr. 24, 2024. [Online]. Available: <https://medium.com/swlh/the-most-important-types-of-qa-testing-for-software-development-3e21810f1bdb>
- [62] “Software Testing Statistics 2023 | 99firms.” Accessed: Apr. 24, 2024. [Online]. Available: <https://99firms.com/blog/software-testing-statistics/>
- [63] C. Persson and N. Yilmazturk, “Establishment of automated regression testing at ABB: industrial experience report on ‘avoiding the pitfalls,’” in *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, Sep. 2004, pp. 112–121. doi: 10.1109/ASE.2004.1342729.
- [64] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, “Technical Debt in Test Automation,” in *Verification and Validation 2012 IEEE Fifth International Conference on Software Testing*, Apr. 2012, pp. 887–892. doi: 10.1109/ICST.2012.192.
- [65] G. Robiolo, E. Scott, S. Matalonga, and M. Felderer, “Technical Debt and Waste in Non-functional Requirements Documentation: An Exploratory Study,” in *Product-Focused Software Process Improvement*, X. Franch, T. Männistö, and S. Martínez-Fernández, Eds., Cham: Springer International Publishing, 2019, pp. 220–235. doi: 10.1007/978-3-030-35333-9_16.
- [66] “Manifesto for Agile Software Development.” Accessed: Apr. 24, 2024. [Online]. Available: <https://agilemanifesto.org/>
- [67] “Becoming Agile: A Grounded Theory of Agile Transitions in Practice | IEEE Conference Publication | IEEE Xplore.” Accessed: Apr. 24, 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/7985657>
- [68] M. Ozalbant, “What is the Rule Based Classification and why we use??,” Medium. Accessed: Apr. 24, 2024. [Online]. Available: <https://medium.com/@meli.ozalbant92/what-is-the-rule-based-classification-and-why-we-use-e4c6f6364156>
- [69] R. Heale and A. Twycross, “Validity and reliability in quantitative studies,” *Evidence-Based Nursing*, vol. 18, no. 3, pp. 66–67, Jul. 2015, doi: 10.1136/eb-2015-102129.
- [70] SooleenAbbas, “The Significance of Validity and Reliability in Quantitative Research,” Sago. Accessed: May 19, 2024. [Online]. Available: <https://sago.com/en/resources/blog/the-significance-of-validity-and-reliability-in-quantitative-research/>

Appendix

External materials:

The dataset creation scripts, and final code written and used in the thesis has been published in a GitHub repository for reusability purposes. This link to the repository is as follows:

<https://github.com/nehadurani996/TestDebt>