A PROJECT REPORT ON

Test Case Identification Utility Based on Code Check-ins

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)

SUBMITTED BY

Jinesh Parakh	Roll No. 41126
Aditi Mantri	Roll No. 41142
Swatej Sonawane	Roll No. 41168
Utkarsh Gurav	Roll No. 41173

DEPARTMENT OF COMPUTER ENGINEERING PUNE INSTITUTE OF COMPUTER TECHNOLOGY DHANKAWADI, PUNE – 43

SAVITRIBAI PHULE PUNE UNIVERSITY
2021-2022

Certificate

This is to certify that the project report entitles

Test Case identification utility based on code check-ins

Submitted by

Jinesh Parakh	Roll No. 41126
Aditi Mantri	Roll No. 41142
Swatej Sonawane	Roll No. 41168
Utkarsh Gurav	Roll No. 41173

is a bonafide student of this institute and the work has been carried out by him/her under the supervision of **Dr. S. S. Sonawane** and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of **Bachelor of Engineering** (Computer Engineering).

Dr. S. S. Sonawane	**Dr. G.V. Kale**
Internal Guide	Head
Dept. of Computer Engineering	Dept. of Computer Engineering

Place: Pune
Date:

ACKNOWLEDGEMENT

It gives us great pleasure in presenting the project report on 'Test Case identification utility based on code check-ins'.

We would like to take this opportunity to thank **Dr. S. S. Sonawane** for giving us all the help and guidance we needed. We are really grateful to her for her kind support. Her valuable suggestions were very helpful.

We are also grateful to **Dr. G.V. Kale**, Head of Computer Engineering Department, PICT for his indispensable support, suggestions.

Jinesh Parakh
Aditi Mantri
Swatej Sonawane
Utkarsh Gurav

ABSTRACT

In terms of quality control, software testing is a crucial activity in software development. For system testing, software testing necessitates the use of test cases. Based on the code changes on version control systems, there might be a possibility of a negative impact on the existing functionalities. Regression testing is an essential activity to assure that software code changes do not adversely affect existing functionalities. Running each test case and determining its validity becomes extremely time-intensive in large-scale systems.

As a solution to this problem, we present a pattern analysis and AI/ML-based method for identifying appropriate test cases based on the JIRA ID of the code check-in to provide early and most relevant feedback to all the stakeholders.

Keywords: *Quality Control, Software Testing, Pattern Analysis, Machine Learning, NLP*

Contents

List of Abbreviations

Abbreviation	Full form
UI	User Interface
CLI	Command-line Interface
ML	Machine Learning
AI	Artificial Intelligence
NLP	Natural Language Processing
POC	Proof of Concept
STLC	Software Testing Life Cycle
SDLC	Software Development Life Cycle
QA	Quality Assurance
PMs	Project Managers

Table 1: Abbrevations

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Motivation

The issues that QA engineers experience during the testing portion of the software development life cycle inspired this problem statement. The engineers had to manually match the tokens of the code base commits to the tokens of the test cases before running the matched test cases. Some test cases in this scenario include tokens that don't match but are nonetheless impacted by the code modifications. Finding these test cases by hand is time-consuming, therefore automating them is one of the key goals.

Instead of running all of the test cases to locate the ones that are impacted, we propose the use of Machine Learning models to cut down on the time it takes to find the ones that are relevant.

1.2 Problem Definition & Objectives

1.2.1 Problem Definition

Design an automated pattern analysis and AI/ML-based relevant test case identifier based on code check-ins for a JIRA ID.

1.2.2 Objectives

- To understand, analyze and study the current testing techniques and their benefits and downsides.
- To understand the existing Veritas specific infrastructure used for testing and its pros and cons.
- Extracting dependency analysis information(like code documentation, git history, function dependency graphs, etc.) to match labels and consequently explore ways to improve it.
- To develop a pattern analysis and AI/ML-based tool that can locate priority test cases to execute based on code check-ins for a certain JIRA ID.
- After executing the indicated relevant test cases, create a thorough report that serves as feedback to the stakeholder.

CHAPTER 2

LITERATURE SURVEY

2.1 Summary

The Following table shows the papers referred for literature survey :

No.	Paper	Published Year
1.	F. Normann. Test case selection based on code changes.	2019
2.	Mojtaba Ghaleb Taher Briand Lionel Pan, Rongqi Bagherzadeh. Test case selection and prioritization using machine learning: A systematic literature review.	2021
3.	Aly. Gomaa, Wael Fahmy. A survey of text similarity approaches.	2013
4.	A. Hammad H. Hourani and M. Lafi. The impact of artificial intelligence on software testing.	2019
5.	X. Xue Y. Pang and A. S. Namin. Identifying effective test cases through k-means clustering for enhancing regression testing.	2013
6.	Microsoft Corporation. Speed up testing by using test impact analysis (tia).	2018
7.	B. Nguyen S. Dhanda E. Nickell R. Siemborski A. Memon, Z. Gao and J. Micco. Taming google-scale continuous testing.	2017
8.	What is test load balancer (tlb)?	Article
9.	Infinitest - the continuous test runner for the jvm.	Article

Table 2.1: Literature survey summary

2.2 Overview

The literature review for this project entails investigating various studies done for test case selection based on code check ins. Several publications have been researched that propose the use of Artificial Intelligence techniques to enhance software testing.

2.2.1 Test Case Selection Based on Code Changes [1]

- The author has provided a comparative analysis for the test case selection.
- The comparative analysis is of the plugins, STARTS, Ekstazi and jdeps. The paper provides pros and cons for each of these plugins and has suggested and used jdeps for test case selections for optimal results.

2.2.2 Test Case Selection and Prioritization(TSP) Using Machine Learning: A Systematic Literature Review [2]

- This research paper has provided a literature survey for the various ML algorithms that can be used for TSP as well as for priority-based searching.
- The focus is on Continuous Integration(CI) along with ML for TSP. The paper provides a detailed thesis of the techniques, features, evaluation metrics and reproducibility for ML-based TSP.

2.2.3 A Survey of Text Similarity Approaches [3]

- This research paper is a literature survey that discusses the existing work on text similarity by partitioning it into 3 major approaches - String-based, Corpus-based and Knowledge-Based similarities.
- A small portion of the paper also discusses Hybrid Similarities Measures.

2.2.4 The Impact of Artificial Intelligence on Software Testing [4]

- The authors of this paper have done a thorough literature survey on AI algorithms and approaches employed in the selected papers and references.
- Along with that, the software testing scope area and associated components, were stressed in the summary.

2.2.5 Identifying Effective Test Cases through K-Means Clustering for Enhancing Regression Testing [5]

- To improve regression testing, the authors suggested a test case categorization approach based on k-means clustering.

- The article discovered that using the statement coverage requirement first improves the performance of the clustering-based strategy.

2.2.6 Test impact analysis in visual studio [6]

- With articles dating back to 2009, Microsoft began working with test case selection, which they call test impact analysis (TIA).
- The TIA feature has subsequently been retained in Visual Studios. When a commit enters the CI flow, the feature will choose the tests that are required to validate the commit.
- Previously failed tests, impacted tests, and newly introduced tests are all tracked by this feature.
- When it doesn't know how to choose tests, the feature has a safe fallback option that runs them all.
- When utilising TIA, they do not support parallelization of tests or code coverage estimations for updates.

2.2.7 Taming Google-Scale Continuous Testing [7]

- The paper indicates how Google began implementing test case selection by exploiting correlations between code, developers, and test cases because they are unable to test their code changes individually due to resource constraints.
- They present their broad solution to the problem, highlighting how they used data and machine learning to choose test cases.

2.2.8 Test Load Balancer (TLB) [8]

- TLB is written in Java and works with a variety of build servers, including Jenkins.
- TLB is a tool that may be used to split subsets of tests onto separate computers, both virtual and physical, and execute the tests in parallel without any test overlap, i.e. no test is run more than once and no test is skipped.

2.2.9 Infinitest [9]

- Infinitest is a Java-based plugin for IntelliJ and Eclipse that is used by developers to automatically test their code every time they make a change.
- When a file with dependencies is saved, it implements test case selection and runs tests.
- This plugin was subjected to limited testing, however, it was found to be relatively slow and to take a significant amount of computing resources.

2.3 GAP Analysis

- Techniques to identify labels with same meaning but different words haven't been researched yet, a research in this area will significantly improve the efficiency in finding relevant test cases.
- DAG has not been extensively used in the field of testing and can provide significant results if explored more.
- Most of the research is done to improve the accuracy and not to reduce costs incurred while building the models.

CHAPTER 3

SOFTWARE REQUIREMENTS SPECIFICATION

3.1 Introduction

3.1.1 Project Scope

- The goal of this project is to design an automated test case identifier based on code check-ins for a certain JIRA ID.
- Various types of dependency analysis information like code documentation, git history, function dependencies are to be extracted and processed as pre-processing for test case recommendation.
- Pattern Analysis approaches and various AI / ML - based algorithms and approaches are to be explored to make an efficient automated testing system.

3.1.2 User Classes and Characteristics

- Technical Users: Users, such as the developers of the system, who have the knowledge of the product built and the internal tools and algorithms used.
- Non-technical Users: Have little to no knowledge of the underlying technologies of the product / tool.

3.1.3 Assumptions and Dependencies

Assumptions

- There is a JIRA ID associated with every code check-in that happens after a developer makes a code change that has to be tested.
- Data in the form of git history and commits is available in order to collect the dependency analysis information for the entire source code and to later incorporate AI/ML - based test case recommendation systems.
- Every test case has its associated labels and components manually given while designing the test case.
- Code base is in programming languages compatible with the open source libraries being used for extraction of dependency analysis information.do

Dependencies

- The training of the system necessitates the presence of a reasonably competent CPU or GPU, as well as the necessary libraries.
- Dependency Analysis Information for the entire source code is necessary to extract useful data from it in order to run pattern analysis algorithms to pull out labels.
- Part of the system dealing with various functionalities is written in Python, and needs Python version 3.7 or higher installed.

- Some parts of the system uses Java and java based libraries and needs Java 8 installed.
- Other libraries such as Doxygen, JIRA Python API etc are also essential.

3.2 Functional requirements

3.2.1 Extracting Dependency Analysis Information

a. Description: In order to identify and recommend the affected test cases for a particular code modification, we require dependency analysis information.

   Components like git history, git commits associated with the particular JIRA ID, code documentation, function dependencies in the code base etc. is an example of the data that needs to be extracted and processed before giving it as an input to the Automated Test Case Recommender.

b. Response Sequence: Fig.3.1 shows the workflow used to extract the various dependency analysis information to be given to the next module.

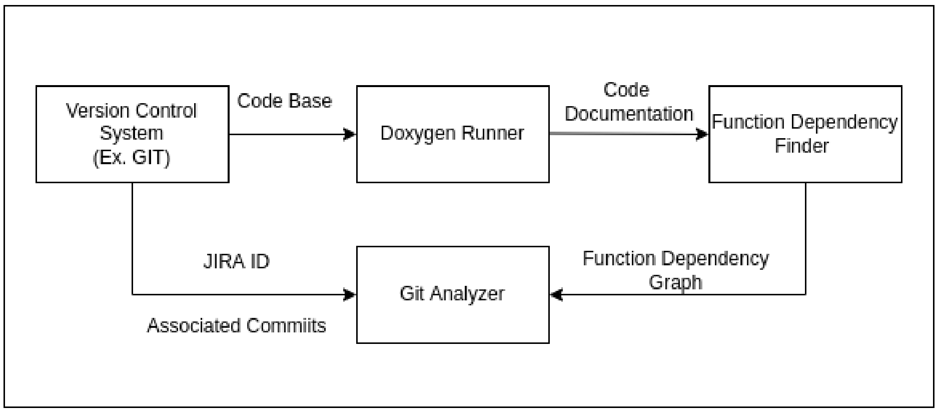

Version Control System (Ex. GIT)
Code Base
Doxygen Runner
Code Documentation
Function Dependency Finder
JIRA ID
Git Analyzer
Function Dependency Graph
Associated Commiits

Figure 3.1: Extraction Dependency Analysis Information Workflow

c. Functional requirement:

   i. A version control system (like GIT) should be in place.
   ii. A JIRA ID should be associated with the checked-in code and should be available as input.
   iii. Previous commits associated with the JIRA ID should be available for retrieval to aid the pattern analysis and weightage algorithms.
   iv. Code base should be written in a programming language compatible with the open source libraries being used to extract the code documentation.

3.2.2 Automated Test Case Identifier

a. Description: When a developer checks in a modification, such as a code change or a bug/defect patch in a file in the code base, the product's functionalities are affected. The QA engineers must test the affected functionalities. Currently, it is done manually, with developers and QA engineers collaborating to identify the affected functionality.

   Automated Test Case Identifier would run pattern analysis on the affected code and related files and aided with ML based algorithms recommend test cases to be run to test the affected code.

b. Response Sequence: JIRA ID from the CLI, JSON files of the associated commits and function dependencies and the Test Case Suite are passed to the ML Model to predict the relevant test cases as shown in the Fig.3.2

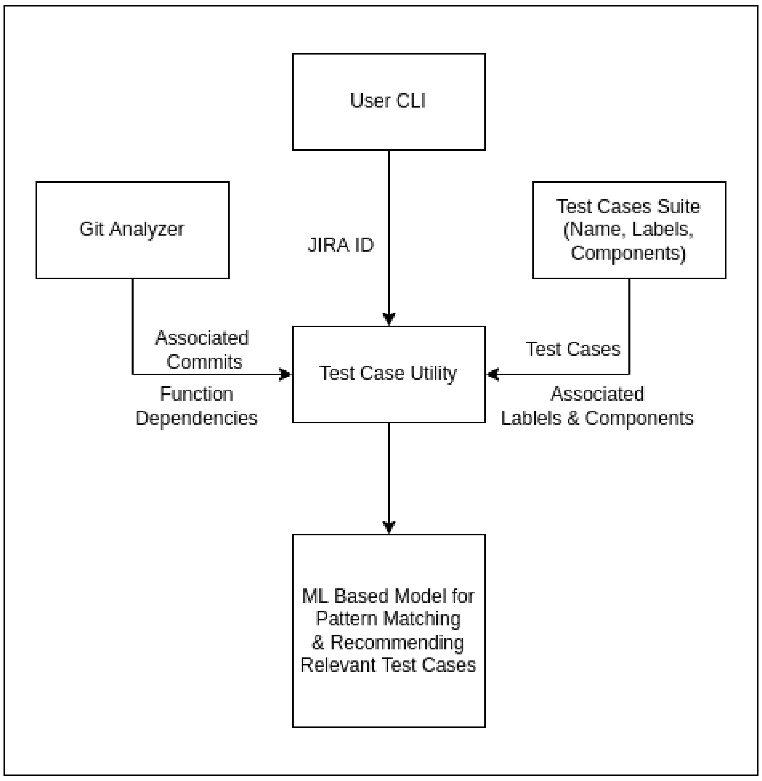

User CLI
Git Analyzer
JIRA ID
Test Cases Suite (Name, Labels, Components)
Associated Commits
Function Dependencies
Test Case Utility
Test Cases
Associated Lablels & Components
ML Based Model for Pattern Matching & Recommending Relevant Test Cases

Figure 3.2: Automated Test Case Identifier Workflow

c. Functional requirement:

   i. Changes in code should be checked-in by the developer.
   ii. Dependency Analysis Information should be extracted and pre-processed.
   iii. Test Cases should be available with their associated labels and components.

3.2.3 Automate Test Case Run and Generate Report

a. Description: Even after recognizing the affected functionalities and test cases to be run, QA engineers would have to manually run them and extract the proper analysis. This system would automatically run the test cases recognized by the previous system and spontaneously generate a detailed report of the test run as feedback to the stakeholders.

b. Functional Requirements:

   i. In order for the automatic test runs to be quick and efficient, users should possess the essential system configurations and libraries.

ii. In case of breakdowns, errors should be conveyed to the user so that they are prompted to try again or report the breakdown.

3.3 External interface requirements

3.3.1 User Interfaces

- The system in its initial prototype is supposed to be a Command Line Interface (CLI) on Jenkins Portal of Veritas.
- A Flask API and a basic webpage using HTML, CSS and JS has been built just to facilitate demonstration.

3.3.2 Hardware Interfaces

- The system would run on a Veritas specific server.
- To avoid problems, the server must be available at all times and capable of handling several sessions at once.

3.3.3 Software Interfaces

- The client side is the Jenkins UI portal specific to Veritas.
- A clean and user friendly GUI is expected to enhance the user experience.

3.4 Non-functional requirements

3.4.1 Performance Requirements

- The test cases recommended by the system should not be excessive but should be sufficient to test the affected code.
- Even if not deployed for use, the prototype should be scalable to handle larger code bases as well with the same efficiency.
- Response time for identifying the test cases, running them and generating a detailed report should be minimum.
- The system should be able to handle a large number of requests from multiple users as well.

3.4.2 Safety Requirements

- Error reduction must be prioritized and in case of any system breakdown, the user must be alerted immediately.
- The system should include a means for users to minimise their own as well as the system's problems, ensuring that the system functions smoothly and error-free.

3.4.3 Security Requirements

- Only users affiliated with the organisation should have access to the tool.
- Identified Users should not have difficulty accessing data they are verified to use.
- Unauthorized access to the code base or the report generated should not be allowed under any circumstances.

3.4.4 Software Quality Attributes

The following are some key quality characteristics that will be important to objectively judge the end result and feasibility of the product:

- Efficiency: Storage, transmission routes, and completing the assigned task on time are all examples of efficiency.
- Portability: The system should be able to run on devices other than the one on which it was built or intended to run.
- Usability: The user should be able to operate the system in a straightforward and unrestricted manner. It should be simple to understand and use, and it should be capable of performing a variety of tasks.
- Accuracy: The system should be accurate in the many functions it performs and should adhere to the functional requirements that have been established.

- Testability: The developers must be able to expose the software system to a variety of tests in order to detect problems and ensure that the system is functioning properly.
- Reliability: It should consistently generate correct and desirable outcomes under all conditions and produce suitable outputs.
- Robustness: The system should be tolerant of failures caused by both external and internal factors and should not fail.
- Maintainability: The system should be capable of detecting and correcting problems as needed. It should also be easy to extend and adjust features.

3.5 System requirements

3.5.1 Data Requirements

- JIRA ID of the code check-in.
- Git history of the entire code base.
- Test Cases with Labels and Components

3.5.2 Software Requirements

- Python Programming language and essential libraries
- Java Programming Language and essential libraries
- Windows/Linux OS as the primary OS of implementation and Eclipse IDE for development.

3.5.3 Hardware Requirements

Sr. No.	Parameter	Minimum Requirements	Justification
1	CPU Speed	2 GHz	Support the operations of the UI and APIs
2	RAM	8GB	For training and running the system
3	GPU	12 GB NVIDIA Tesla K80	For NLP purposes

Table 3.1: Hardware Requirements

3.6 Analysis Models: SDLC Model to be applied

For this project, we have decided to follow the Agile SDLC model. Rather than a top-down procedure with a single sequence of phases, the Agile SDLC methodology emphasises collaborative decision-making and development over several short cycles or sprints.
We are eager to capitalise on some advantages of the Agile paradigm, namely,

- Product is developed fast and frequently delivered.
- Project is divided into short and transparent iterations.
- The development process incorporates the validity of functional requirements.
- It continuously gives attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcome.

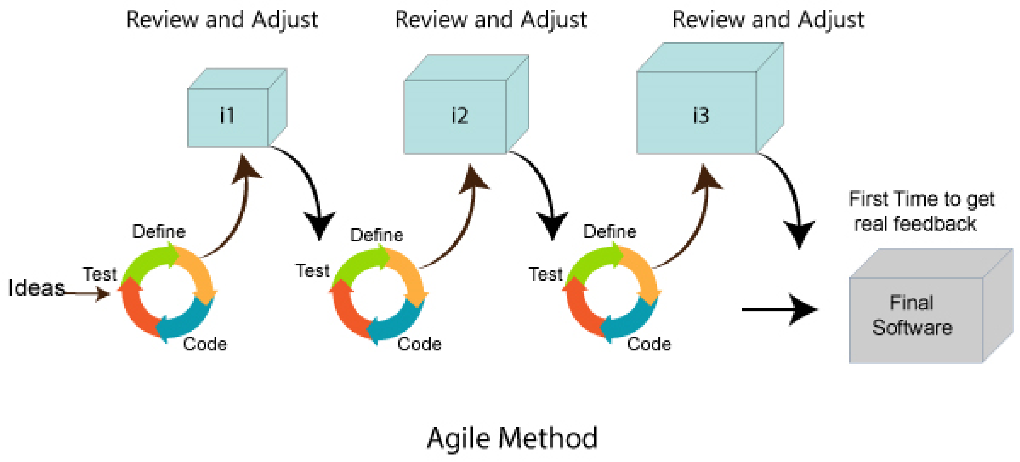


Figure 3.3: Agile Method [10]

Each iteration of our Agile SDLC will contain the following steps:

1. **Requirement gathering and analysis**
   During this step, the demands of each iteration must be obtained and analysed. We generate an initial requirements document to assist us in the subsequent phases based on our interactions with stakeholders and the requirements for designing such an application.

2. **Design based on requirements**
   We create some design documents based on the current iteration's needs to help us with our workflow during this phase.

3. **Construction/iteration**
   During this phase, we as developers will begin to develop the functionalities based on the requirements in the estimated time frame. Following stakeholder review, this phase will consist of minor and steady increments in the product build.

4. **Testing**
   This step involves testing and quality assurance of the functions added in this iteration, as well as the validity of the features produced in previous iterations.

5. **Deployment**
   During the deployment phase, we ensure that the functions established up to this point interact with one another as needed and that the released product does not break in the actual world.

6. **Feedback**
   The stakeholders provide comments on the current iteration, and the next iteration is prepared based on the feedback.

CHAPTER 4

SYSTEM DESIGN

4.1 System Architecture

The proposed system comprises of a system that receives a JIRA ID from the system user, pre-processes the commits connected with the JIRA ID, and delivers the necessary test cases that are expected to be executed to test the modifications specific to the JIRA ID. Pre- processing steps involve fetching the git commit history along with extracting the function dependency call graph through a function dependency finder utility as shown in the fig below(Fig.4.1) :

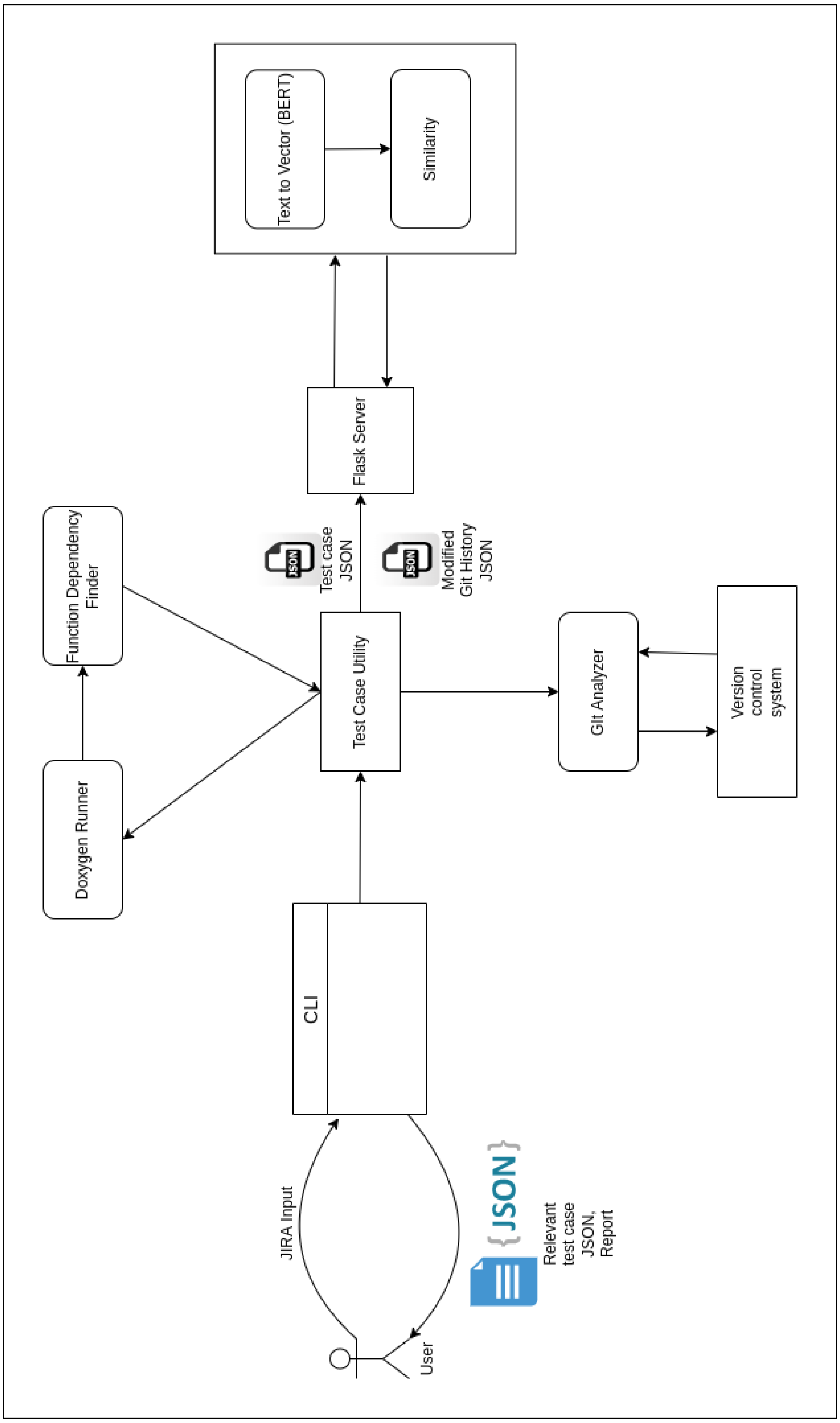


Figure 4.1: System Architecture Diagram

4.2 Flowchart

Fig 4.2 shows the overall flow of the system for the user.

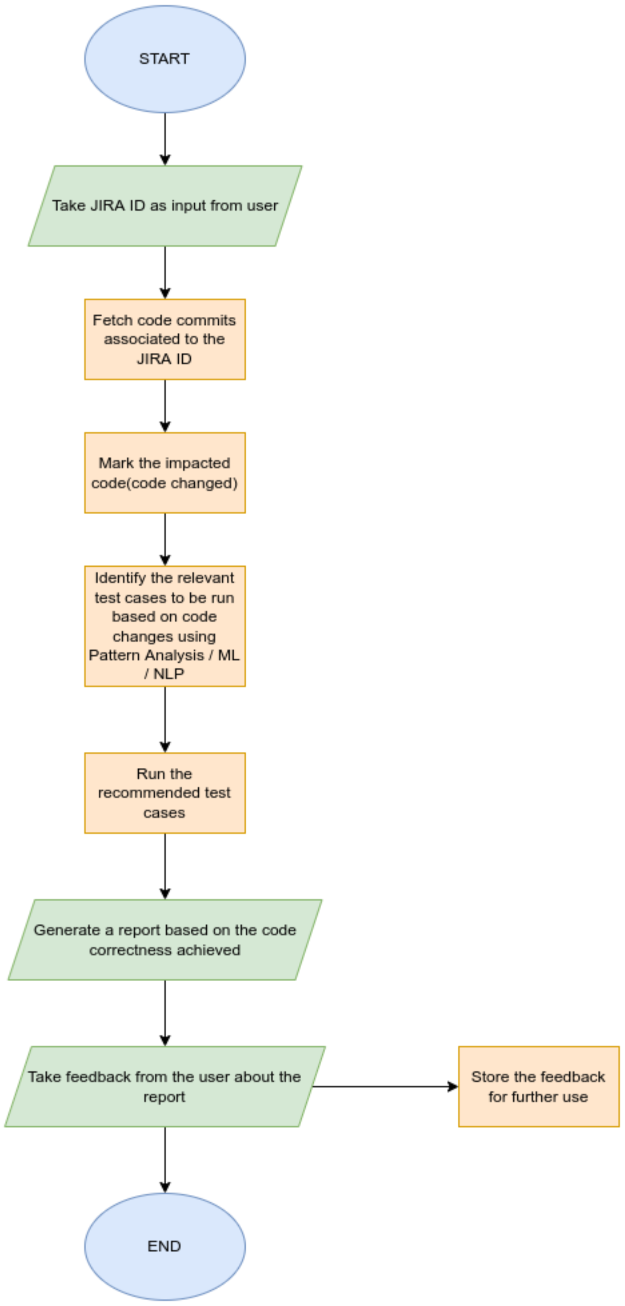

START
Take JIRA ID as input from user
Fetch code commits associated to the JIRA ID
Mark the impacted code(code changed)
Identify the relevant test cases to be run based on code changes using Pattern Analysis / ML / NLP
Run the recommended test cases
Generate a report based on the code correctness achieved
Take feedback from the user about the report
Store the feedback for further use
END

Figure 4.2: Flowchart

4.3 Data Flow Diagram

Fig 4.3 depicts the overview of the flow of data throughout the application's life cycle for the use case. A detailed flow can be seen in the Fig 4.4

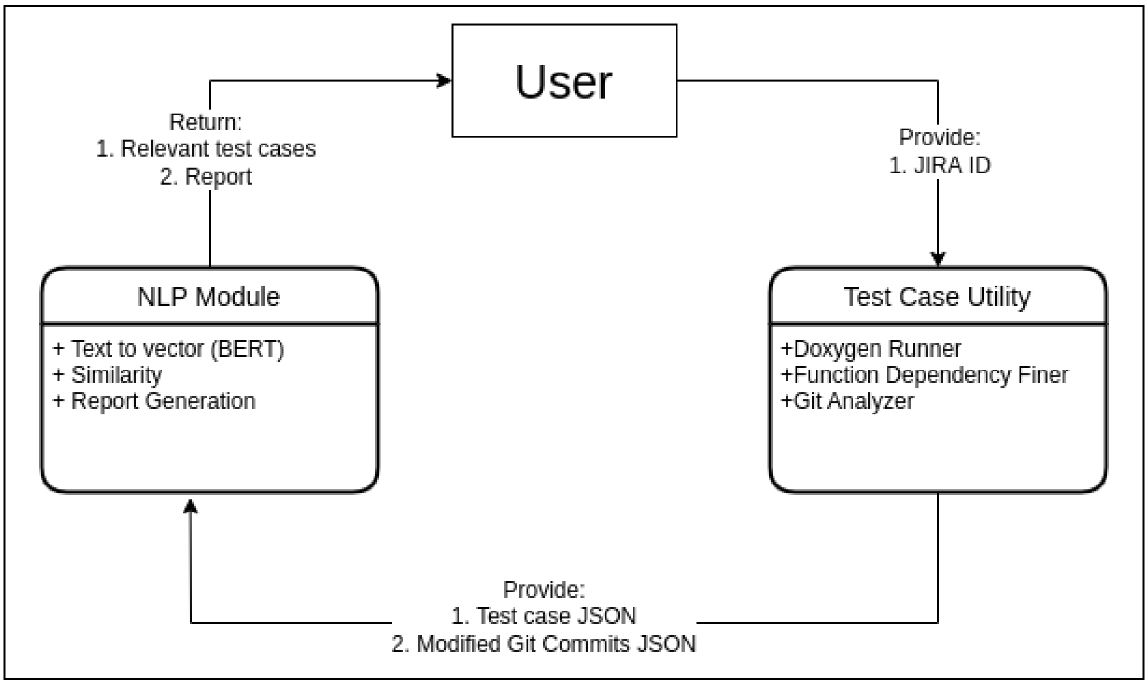


Figure 4.3: Data Flow Diagram

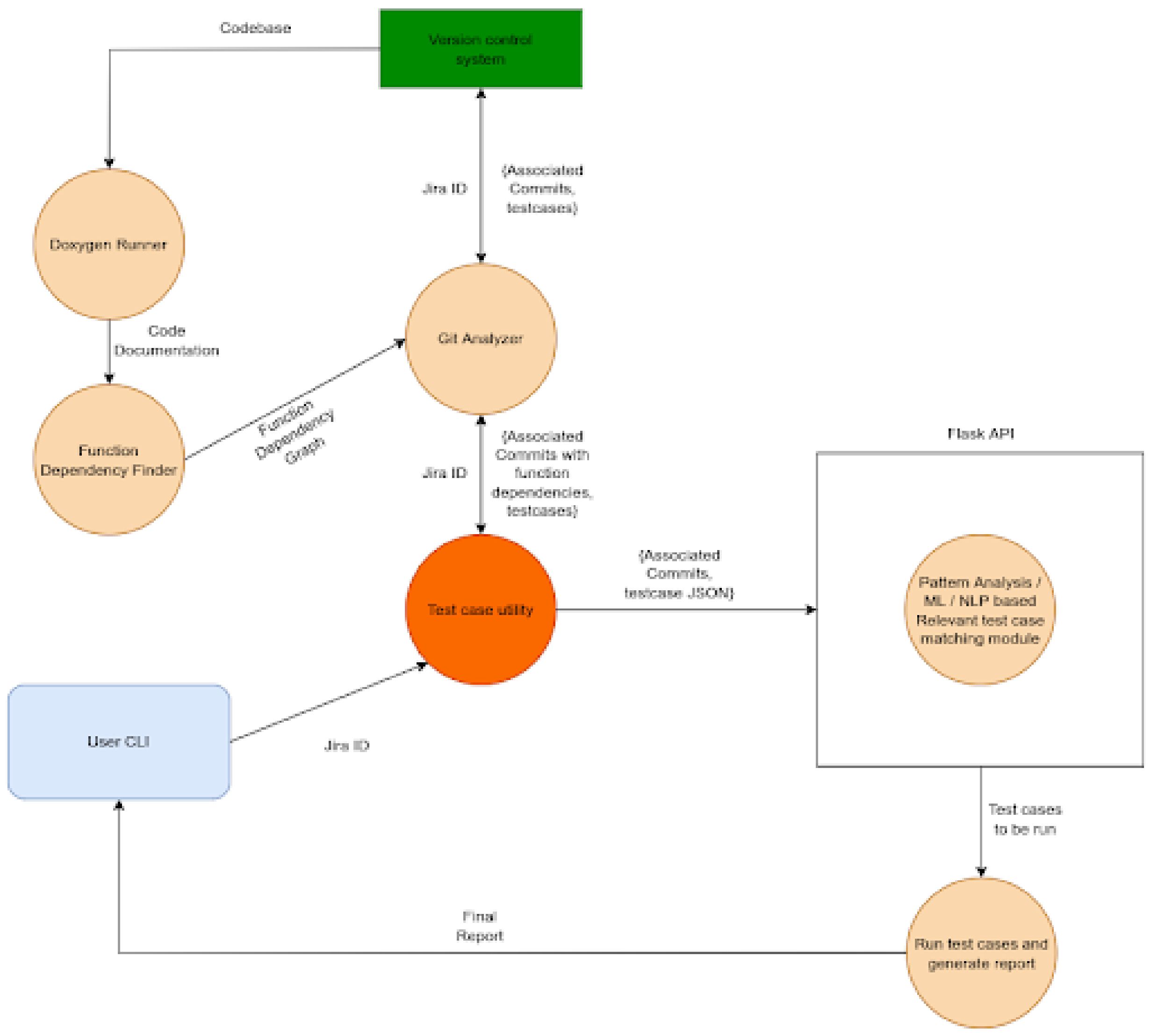

Codebase
Version control system
Jira ID
{Associated Commits, testcases}
Doxygen Runner
Code Documentation
Git Analyzer
Function Dependency Finder
Function Dependency Graph
Jira ID
{Associated Commits with function dependencies, testcases}
Flask API
{Associated Commits, testcase JSON}
Test case utility
Pattern Analysis / ML / NLP based Relevant test case matching module
User CLI
Jira ID
Test cases to be run
Final Report
Run test cases and generate report

Figure 4.4: Detailed Data Flow Diagram

4.4 Sequence Diagram

Fig 4.5 shows the sequence in which the components in the proposed system sequentially interact with each other.

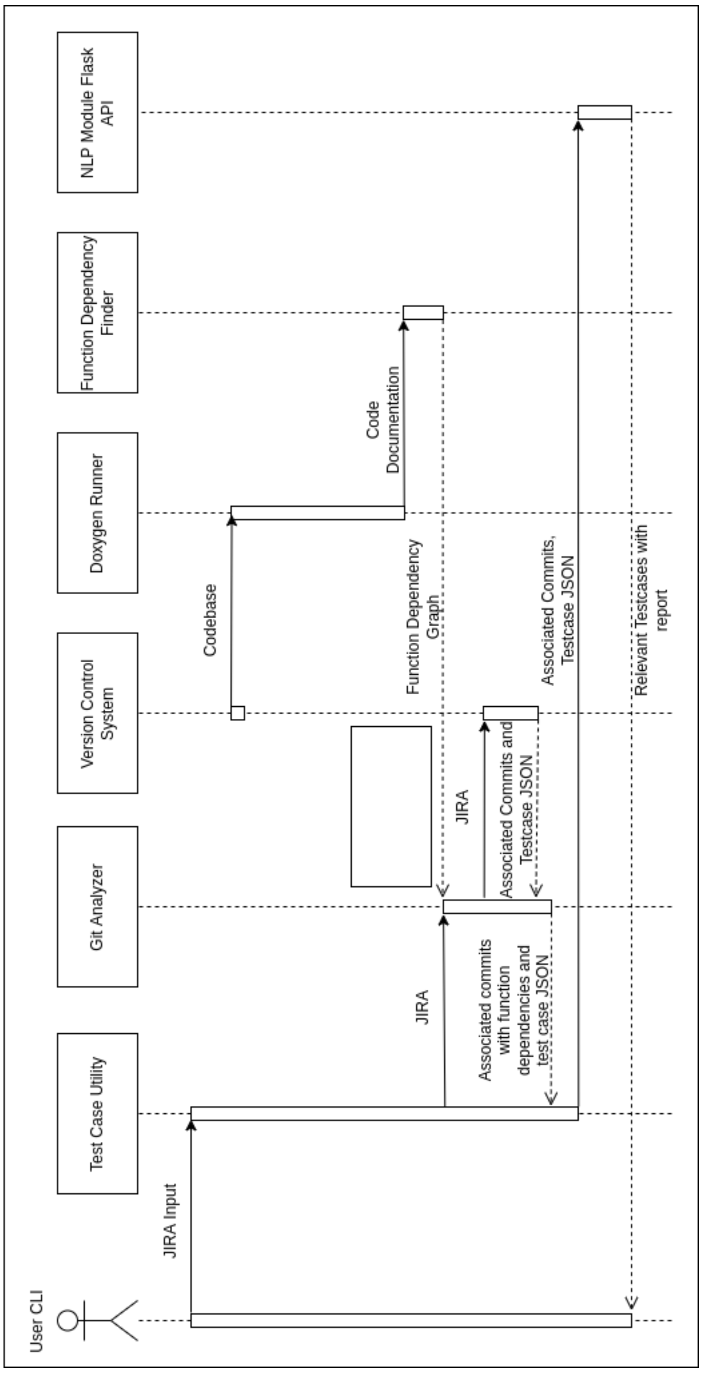

User CLI
Test Case Utility
Git Analyzer
Version Control System
Doxygen Runner
Function Dependency Finder
NLP Module Flask API
JIRA Input
JIRA
Associated commits with function dependencies and test case JSON
JIRA
Associated Commits and Testcase JSON
Codebase
Code Documentation
Function Dependency Graph
Associated Commits, Testcase JSON
Relevant Testcases with report

Figure 4.5: Sequence Diagram

4.5 Activity Diagram

Control Flow of the system from the start point to the end point of the system, showing various decisions paths that exist during the execution of the Test Case identification utility is showcased in the diagram below (Fig 4.6):

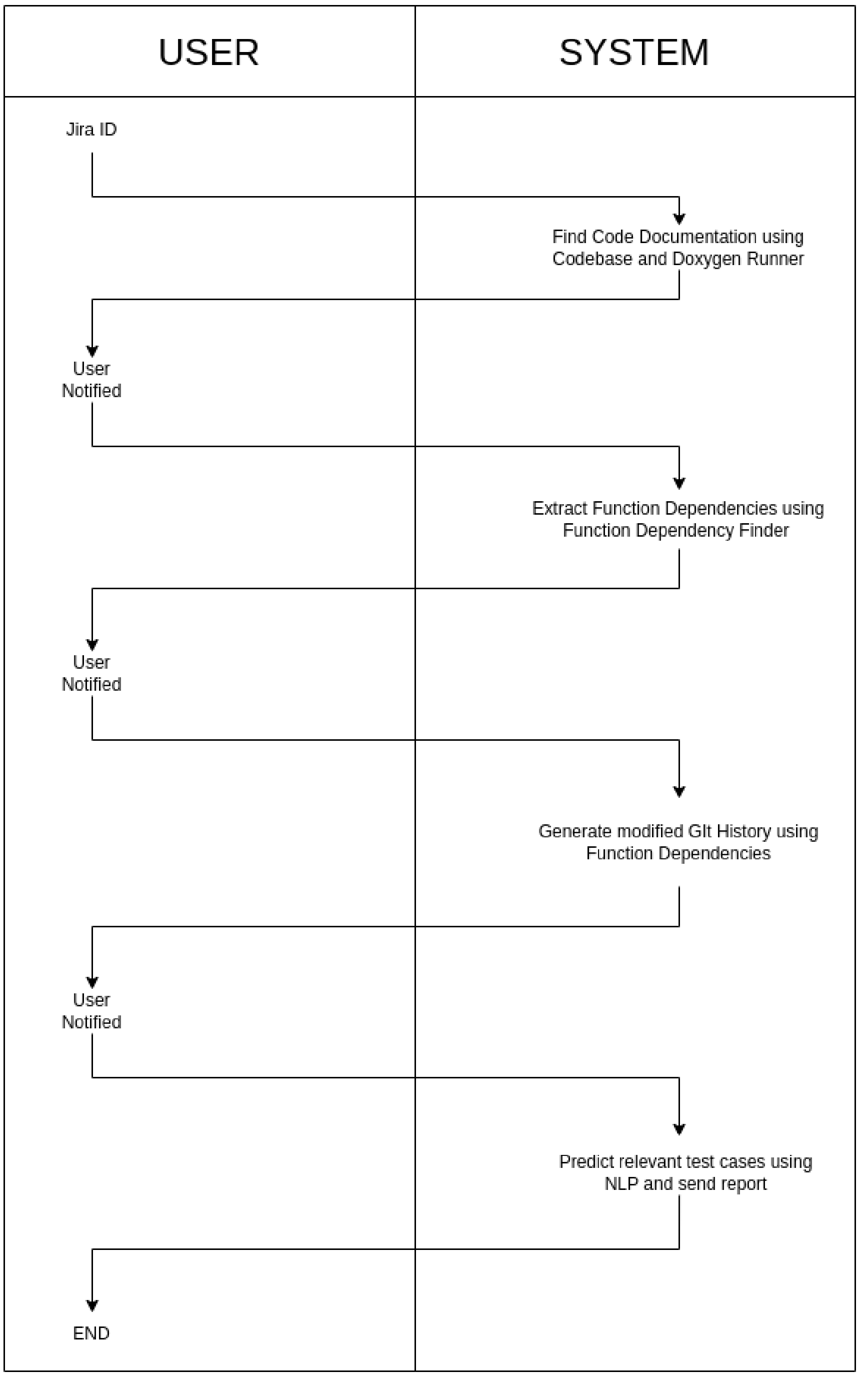

USER
SYSTEM
Jira ID
Find Code Documentation using Codebase and Doxygen Runner
User Notified
Extract Function Dependencies using Function Dependency Finder
User Notified
Generate modified GIt History using Function Dependencies
User Notified
Predict relevant test cases using NLP and send report
END

Figure 4.6: Activity Diagram

4.6 Usecase Diagram

The application's key use cases are depicted below (Fig 4.7), with users acting as actors interacting with various components.

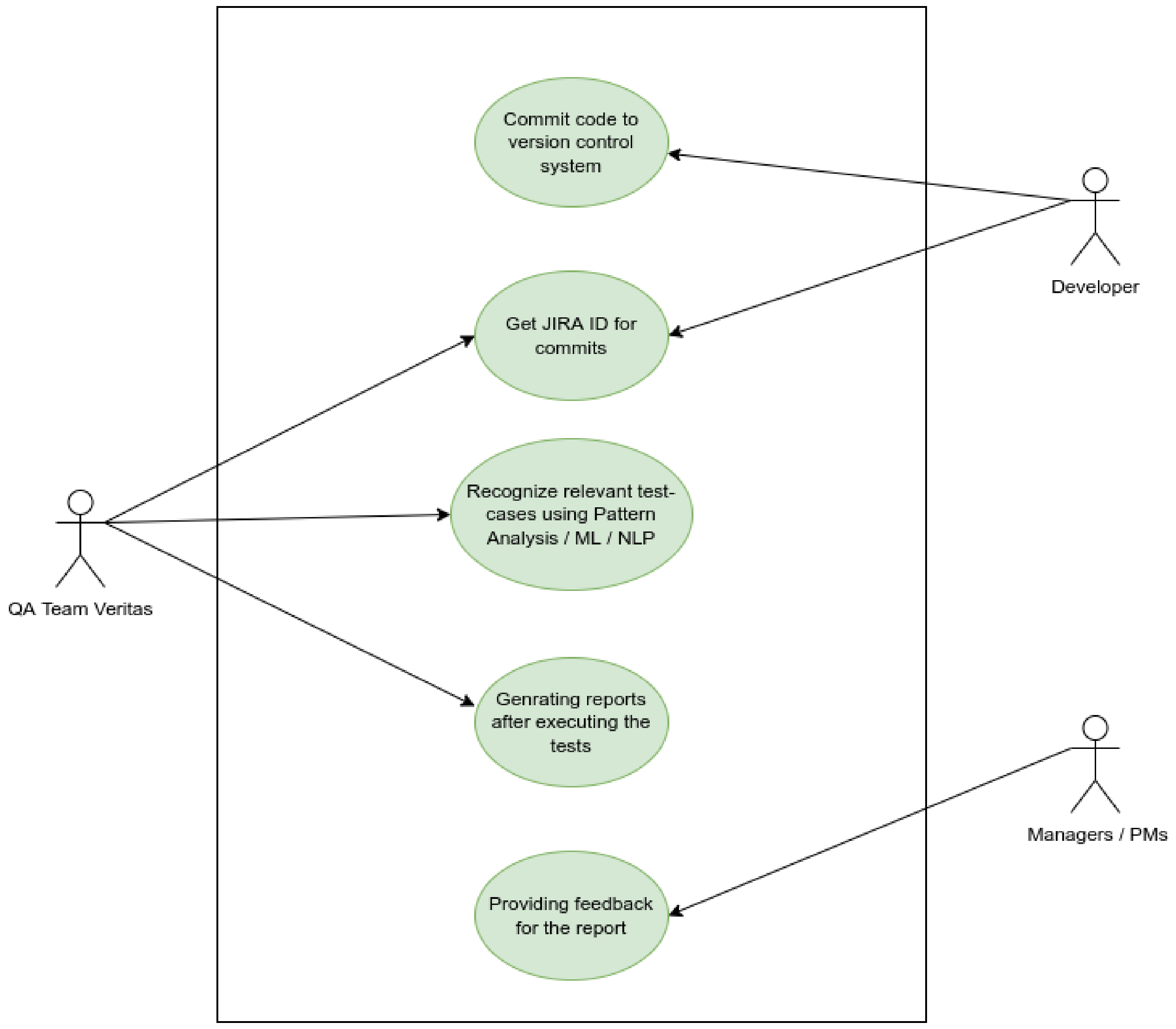

Commit code to version control system
Get JIRA ID for commits
Recognize relevant test-cases using Pattern Analysis / ML / NLP
Genrating reports after executing the tests
Providing feedback for the report
Developer
QA Team Veritas
Managers / PMs

Figure 4.7: Usecase Diagram

CHAPTER 5

PROJECT PLAN

5.1 Project Estimates

5.1.1 Reconciled Estimates

- Cost Estimate: The software used for the application will incur no cost as it is all released under free open source licenses.
- Time Estimate: 9-10 months

5.1.2 Human Resources

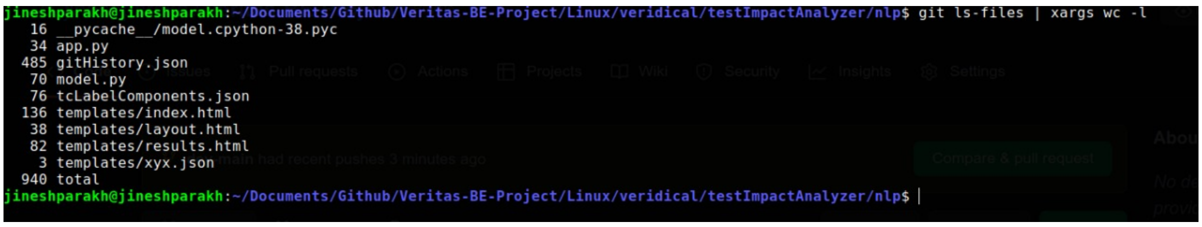
- Number of people : 4
- Skills: Java, Python, Pattern Analysis, NLP, Machine Learning
- Client: Veritas Technologies, LLC
- Stakeholders: Stakeholders for our project team are QA Team Veritas, Scrum Masters, Managers and PMs

5.1.3 Development Resources

- Software:
  1. IDE for development
  2. Python and Java libraries viz. pydriller, jqassistant, etc

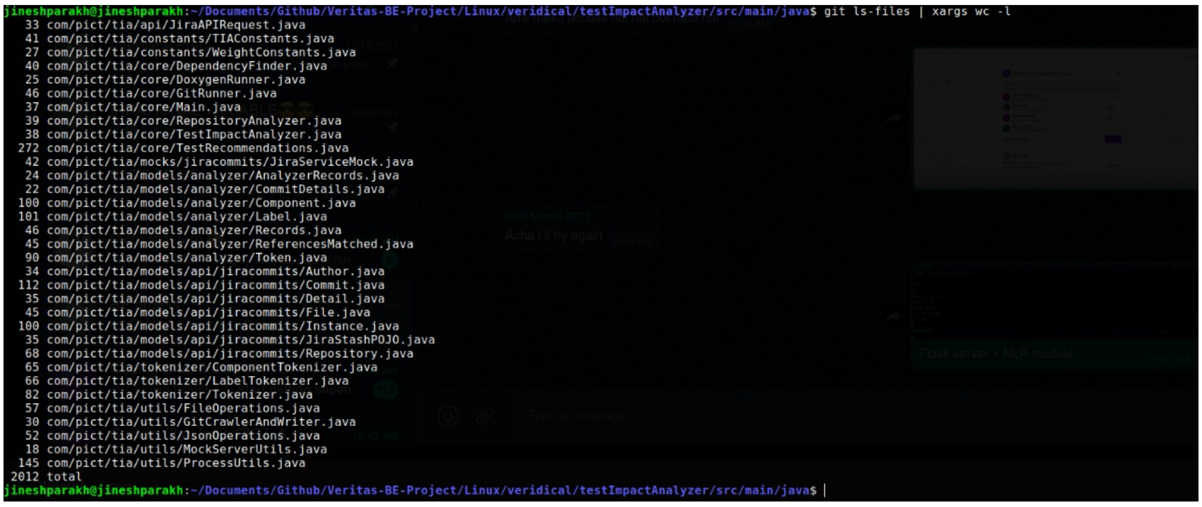
5.1.4 Lines of Code

- Flask Server and NLP Model: 940 lines (as shown in Fig.5.1)
- Core Java Modules: 2012 lines (as shown in Fig. 5.2)
- *BookMyMovie*(repository setup for testing) and its test cases: 2158 lines of code(as shown in Fig.5.3)
- Python Modules: 833 lines (as shown in Fig.5.4)
- Doxygen Runner: 391 lines (as shown in Fig.5.5)



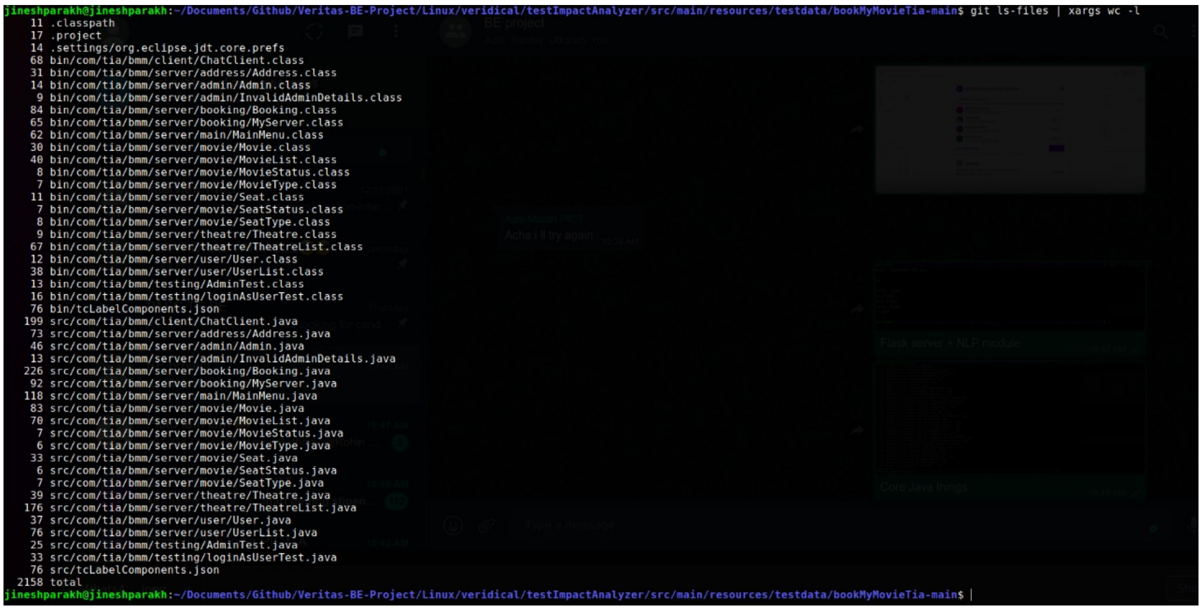
```
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/nlp$ git ls-files | xargs wc -l
  16 __pycache__/model.cpython-38.pyc
  34 app.py
 485 gitHistory.json
  70 model.py
  76 tcLabelComponents.json
 136 templates/index.html
  38 templates/layout.html
  82 templates/results.html
   3 templates/xyz.json
 940 total
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/nlp$ |
```

Figure 5.1: No. of lines - Flask Server and NLP Model



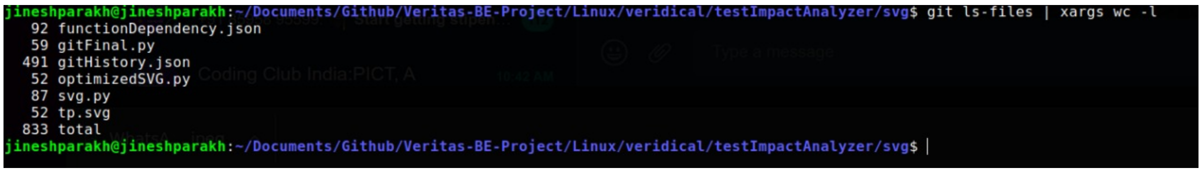
```
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/src/main/java$ git ls-files | xargs wc -l
   33 com/pict/tia/api/JiraAPIRequest.java
   41 com/pict/tia/constants/TIAConstants.java
   27 com/pict/tia/constants/WeightConstants.java
   40 com/pict/tia/core/DependencyFinder.java
   25 com/pict/tia/core/DoxygenRunner.java
   46 com/pict/tia/core/GitRunner.java
   37 com/pict/tia/core/Main.java
   39 com/pict/tia/core/RepositoryAnalyzer.java
   38 com/pict/tia/core/TestImpactAnalyzer.java
  272 com/pict/tia/core/TestRecommendations.java
   42 com/pict/tia/mocks/jiracommits/JiraServiceMock.java
   24 com/pict/tia/models/analyzer/AnalyzerRecords.java
   22 com/pict/tia/models/analyzer/CommitDetails.java
  100 com/pict/tia/models/analyzer/Component.java
  101 com/pict/tia/models/analyzer/Label.java
   46 com/pict/tia/models/analyzer/Records.java
   45 com/pict/tia/models/analyzer/ReferencesMatched.java
   90 com/pict/tia/models/analyzer/Token.java
   34 com/pict/tia/models/api/jiracommits/Author.java
  112 com/pict/tia/models/api/jiracommits/Commit.java
   35 com/pict/tia/models/api/jiracommits/Detail.java
   45 com/pict/tia/models/api/jiracommits/File.java
  100 com/pict/tia/models/api/jiracommits/Instance.java
   35 com/pict/tia/models/api/jiracommits/JiraStashPOJO.java
   68 com/pict/tia/models/api/jiracommits/Repository.java
   65 com/pict/tia/tokenizer/ComponentTokenizer.java
   66 com/pict/tia/tokenizer/LabelTokenizer.java
   82 com/pict/tia/tokenizer/Tokenizer.java
   57 com/pict/tia/utils/FileOperations.java
   30 com/pict/tia/utils/GitCrawlerAndWriter.java
   52 com/pict/tia/utils/JsonOperations.java
   18 com/pict/tia/utils/MockServerUtils.java
  145 com/pict/tia/utils/ProcessUtils.java
 2012 total
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/src/main/java$ |
```

Figure 5.2: No. of lines - Core Java Modules



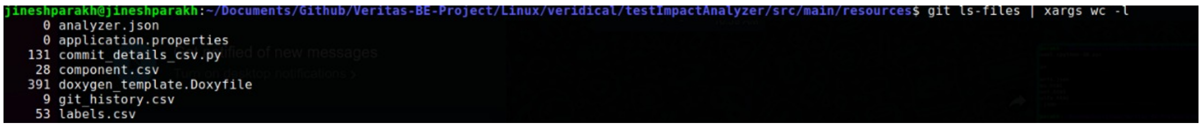
```
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/src/main/resources/testdata/bookMyMovieTia-main$ git ls-files | xargs wc -l
   11 .classpath
   17 .project
   14 .settings/org.eclipse.jdt.core.prefs
   68 bin/com/tia/bmm/client/ChatClient.class
   31 bin/com/tia/bmm/server/address/Address.class
   14 bin/com/tia/bmm/server/admin/Admin.class
    9 bin/com/tia/bmm/server/admin/InvalidAdminDetails.class
   84 bin/com/tia/bmm/server/booking/Booking.class
   65 bin/com/tia/bmm/server/booking/MyServer.class
   62 bin/com/tia/bmm/server/main/MainMenu.class
   30 bin/com/tia/bmm/server/movie/Movie.class
   40 bin/com/tia/bmm/server/movie/MovieList.class
    8 bin/com/tia/bmm/server/movie/MovieStatus.class
    7 bin/com/tia/bmm/server/movie/MovieType.class
   11 bin/com/tia/bmm/server/movie/Seat.class
    7 bin/com/tia/bmm/server/movie/SeatStatus.class
    8 bin/com/tia/bmm/server/movie/SeatType.class
    9 bin/com/tia/bmm/server/theatre/Theatre.class
   67 bin/com/tia/bmm/server/theatre/TheatreList.class
   12 bin/com/tia/bmm/server/user/User.class
   38 bin/com/tia/bmm/server/user/UserList.class
   13 bin/com/tia/bmm/testing/AdminTest.class
   16 bin/com/tia/bmm/testing/loginAsUserTest.class
   76 bin/tcLabelComponents.json
  199 src/com/tia/bmm/client/ChatClient.java
   73 src/com/tia/bmm/server/address/Address.java
   46 src/com/tia/bmm/server/admin/Admin.java
   13 src/com/tia/bmm/server/admin/InvalidAdminDetails.java
  226 src/com/tia/bmm/server/booking/Booking.java
   92 src/com/tia/bmm/server/booking/MyServer.java
  118 src/com/tia/bmm/server/main/MainMenu.java
   83 src/com/tia/bmm/server/movie/Movie.java
   70 src/com/tia/bmm/server/movie/MovieList.java
    7 src/com/tia/bmm/server/movie/MovieStatus.java
    6 src/com/tia/bmm/server/movie/MovieType.java
   33 src/com/tia/bmm/server/movie/Seat.java
    6 src/com/tia/bmm/server/movie/SeatStatus.java
    7 src/com/tia/bmm/server/movie/SeatType.java
   39 src/com/tia/bmm/server/theatre/Theatre.java
  176 src/com/tia/bmm/server/theatre/TheatreList.java
   37 src/com/tia/bmm/server/user/User.java
   76 src/com/tia/bmm/server/user/UserList.java
   25 src/com/tia/bmm/testing/AdminTest.java
   33 src/com/tia/bmm/testing/loginAsUserTest.java
   76 src/tcLabelComponents.json
 2158 total
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/src/main/resources/testdata/bookMyMovieTia-main$ |
```

Figure 5.3: No. of lines - Test repository setup locally and its test cases



```
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/svg$ git ls-files | xargs wc -l
   92 functionDependency.json
   59 gitFinal.py
  491 gitHistory.json
   52 optimizedSVG.py
   87 svg.py
   52 tp.svg
  833 total
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/svg$ |
```

Figure 5.4: Python Modules



```
jineshparakh@jineshparakh:~/Documents/Github/Veritas-BE-Project/Linux/veridical/testImpactAnalyzer/src/main/resources$ git ls-files | xargs wc -l
    0 analyzer.json
    0 application.properties
  131 commit_details_csv.py
   28 component.csv
  391 doxygen_template.Doxyfile
    9 git_history.csv
   53 labels.csv
```

Figure 5.5: Doxygen Runner

5.2 Risk Management

Software development is a broad term that addresses numerous activities such as goal setting, requirement documentation, feature development, infrastructure creation, design selection, and so on. Certain risks are probably inevitable when so many tasks are involved in a project, and so many different people are working on it.

While risk is typically associated with negative epithets, a rational understanding of risk describes it as a crucial component that enables corporations to embrace new opportunities and conquer frontiers, provided risk is managed effectively. As a result, risk management becomes a vital component of building an application.

5.2.1 Risk Identification & Analysis

The risk analysis is performed using the following guidelines:

Risk Probability

a.	High Probability	$75\% \leq x \leq 100\%$
b.	Medium High Probability	$50\% \leq x \leq 75\%$
c.	Medium-Low Probability	$25\% \leq x \leq 50\%$
d.	Low Probability	$0\% \leq x \leq 25\%$

Table 5.1: Risk Probability Levels

Risk Impact

a.	Very High	Catastrophic
b.	High	Critical
c.	Medium	Moderate
d.	Low	Marginal

Table 5.2: Risk Impact Severity

5.2.2 Project Specific Risks

1. Input of the system requires a JIRA ID, unavailability of JIRA ID w.r.t a particular code check in makes the system useless for the system user.

2. System is specific to the Veritas Testing Requirements, test cases not in the format of 'name', 'labels' and 'components' wouldn't work.

3. Use of open source libraries for code documentation makes the system vulnerable to data threats.

5.2.3 Project Schedule

Task Set

Task	**Description**
Task 1	Domain Study - STLC, other techniques and Literature Survey
Task 2	Defining the problem and determine the objectives and usecases
Task 3	Learning relevant libraries and frameworks
Task 4	Designing the system architecture and related diagrams
Task 5	Explore suitable pattern analysis, ML and NLP based algorithms for label matching
Task 6	Implement POCs of different algorithms. Select the one which best fits the use-case
Task 7	Implementation of the selected algorithm for the main prototype and testing
Task 8	Review of project with presentation

Table 5.3: Task set

5.3 Team Organization

The manner in which a team is organized and the mechanisms for reporting are noted. Updates on project improvements are provided to both internal and external guides. Meetings are held on a regular basis to discuss changes and plan for upcoming responsibilities.

5.3.1 Team Structure

The team structure for the project is identified. Roles are defined.

- Dr. S. S. Sonawane: Internal Guide
- Shashank Mahajan: External Guide (Veritas Technologies LLC)
- Jinesh Parakh: NLP Based Approaches, server side applications, documentation
- Aditi Mantri: Pattern Analysis Approaches Experimentation, Server side applications and documentation
- Swatej Sonawane: NLP based Approaches, ML based Recommendation System experimentation, documentation
- Utkarsh Gurav: ML based Recommendation System Approaches, Pattern Analysis Approaches, Documentation

5.3.2 Management reporting and Communication

- Team is always in contact with the internal project guide via Google Meets and the Veritas Mentors via Zoom.
- For successful remote communication, team members also interact using Notion. Chat rooms, e-mails, and online meetings are also utilised for communication.

CHAPTER 6

PROJECT IMPLEMENTATION

6.1 Overview Of the Project Model

6.1.1 Client

The users can directly interact with the model using the Command Line Interface(CLI) and the GUI(made only for demonstration), which is designed using front-end technologies such as HTML, CSS, etc. The recommended test cases are displayed to the user on the GUI, once the user provides the JIRA ID as an input from the CLI.

6.1.2 Server

Flask server is used to host the recommendation module of the project. It comprises of the BERT model and a similarity function. In addition, mock JIRA servers are implemented to enable extraction of essential data from the JIRA ID provided.

6.1.3 Doxygen Runner

The Doxygen Runner uses the Doxygen library to produce documentation for the whole project and makes use of the Graphviz library to display the functional dependency call-graphs in SVG format.

6.1.4 Functional Dependency Finder

The SVG generated is given as an input to the Functional Dependency Finder, which parses the SVG and generates the functional dependencies' inverted call-graphs. These inverted call-graphs are stored as a json file.

6.1.5 Git Analyzer

The project's Git Analyzer Module pulls the necessary data from the mock JIRA servers storing the history of the commits using the Pydriller library. It then integrates the functional dependency json into the extracted information to generate a final git history json having all the required essential data.

6.1.6 Machine Learning Model

Sentence-BERT is used to generate commit wise embeddings of the git history json obtained from the Git Analyzer module and the test case json. The test case json contains the name of the test case, and the labels and components associated with them. Each test case embedding is then matched with all the embeddings of the git history json using a similarity function to identify the most relevant test cases.

6.2 Tools and Technologies Used

a. PYDRILLER: Is a Python framework used to extract useful information from Git Commit History.

b. DOXYGEN: Provides code documentation and generates graphs; used to identify the functional dependencies.

c. GRAPHVIZ: Used to visualize functional dependencies in the form of abstract graphs and networks.

d. SENTENCE-BERT: Is a modification of the pre-trained BERT, used to derive semantically meaningful sentence embedding.

6.3 Algorithms Details

6.3.1 Co-sine Similarity

In terms of context or meaning, Text Similarity must determine how closely two text pieces are connected. Cosine similarity, Euclidean distance, and Jaccard similarity are some of the text similarity metrics accessible. For measuring the similarity of two inquiries, each of these metrics has its unique definition.

Cosine similarity is one of the measures used in Natural Language Processing to compare the text similarity of two documents of different sizes. Text documents are represented as n-dimensional vectors.

The cosine of the angle between two n-dimensional vectors projected in a multi-dimensional space is calculated by the cosine similarity metric. When the Cosine similarity score is 1, it means that two vectors are oriented in the same way and vice versa.

Formula of Cosine Similarity:

A = [1, 1, 1, 1, 0, 1, 1, 2]   B = [1, 0, 0, 1, 1, 0, 1, 0]

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

$$\begin{aligned} A\cdot B &= \sum_{i=1}^{n} A_i B_i \\ &= (1*1)+(1*0)+(1*0)+(1*1)+(0*1)+(1*0)+(1*1)+(2*0) \\ &= 3 \end{aligned}$$

$$\sqrt{\sum_{i=1}^{n} A_i^2} = \sqrt{1+1+1+1+0+1+1+4} = \sqrt{10}$$

$$\sqrt{\sum_{i=1}^{n} B_i^2} = \sqrt{1+0+0+1+1+0+1+0} = \sqrt{4}$$

$$\textit{cosine similarity} = \cos\theta = \frac{A\cdot B}{|A||B|} = \frac{3}{\sqrt{10}*\sqrt{4}} = 0.4743$$

6.3.2 BERT

BERT is an open source machine learning framework for natural language processing (NLP). BERT is designed to help computers decipher confusing words in text by building context via the usage of surrounding content. The BERT framework was trained using Wikipedia text and may be fine-tuned using question and answer datasets.

Bidirectional Encoder Representations from Transformers (BERT) is a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based on their connection. (This is known as paying attention in NLP.)

Previously, language models could only read text input in one of two ways: left-to-right or right-to-left, but not both at once. BERT is one of a kind in that it can read in both directions at once. Bidirectionality is a capability that has been enabled by the invention of Transformers.

Using this bidirectional capacity, BERT is pre-trained on two distinct but related NLP tasks: Masked Language Modeling and Next Sentence Prediction.

Masked Language Model (MLM) training is hiding a word in a phrase and then having the software identify which word was concealed (masked) based on the context of the hidden word. The goal of Next Sentence Prediction training is to have the software predict whether two provided sentences are connected logically and sequentially or are simply random.

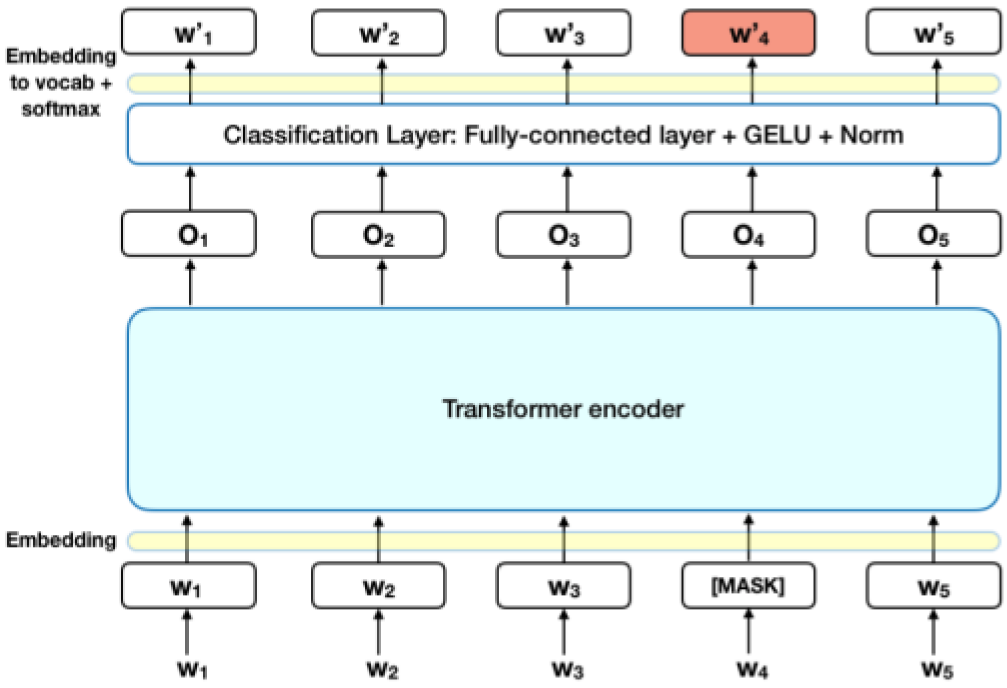
Any NLP approach aims to understand human language in its natural state. This generally includes guessing a word from a list, like in BERT's case. Models are frequently developed using a large pool of specialised, labelled training data. This necessitates the collaboration of linguists to manually label data.

BERT, on the other hand, was pre-trained on a plain text corpus with no labels (namely the entirety of the English Wikipedia, and the Brown Corpus). It continues to learn unsupervised from unlabeled material and improve even when used in real-world applications (ie Google search). Its pre-training serves as a "knowledge" basis on which to build. BERT may then be customised to a user's preferences and adjusted to the ever-growing corpus of searchable material and queries. This process is known as transfer learning.

BERT is made feasible, as previously indicated, by Google's Transformers research. The transformer, which is part of the model, gives BERT a better chance of recognising context and ambiguity in language. As shown in the figure 6.1, rather of processing each word separately, the transformer examines each word in relation to the other words in the phrase. By looking at all surrounding terms, the Transformer aids the BERT model in comprehending the entire context of a term, helping it to better grasp searcher intent.

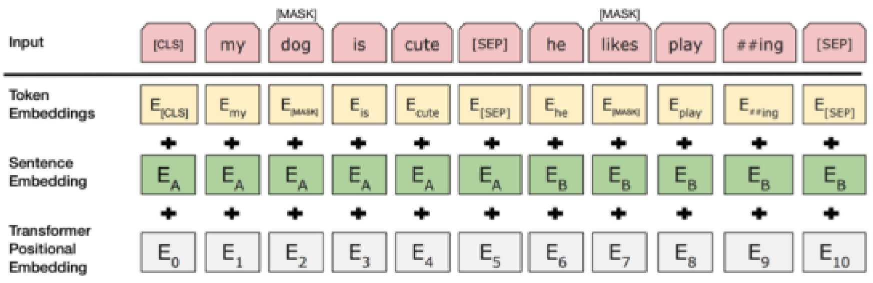
This is in contrast to word embedding, which used older models like GloVe and word2vec to map every single word to a vector that only reflected one dimension, or a sliver, of that word's meaning.

These word embedding models need a large amount of tagged data because all words have a vector or meaning associated to them, they thrive at many broad NLP tasks but fall short when it comes to the context-heavy, predictive aspect of question answering. BERT uses a masked language modelling approach to prevent the word in focus from "seeing itself" that is, having a fixed meaning irre-



W'1
W'2
W'3
W'4
W'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
W1
W2
W3
[MASK]
W5
W1
W2
W3
W4
W5

Figure 6.1: Transformer Encoder



Input
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
[MASK]
Token Embeddings
Sentence Embedding
Transformer Positional Embedding

Figure 6.2: Embeddings

spective of its context. BERT is then forced to only recognise the disguised word based on its context. In BERT, words are defined by their context rather than a fixed identity. The English linguist John Rupert Firth allegedly stated, "You shall know a word by the company it keeps."

BERT is also the first NLP approach that rely only on the self-attention mechanism, which is made possible by the bidirectional Transformers at its heart. This is critical because the meaning of a word can commonly change as a sentence proceeds. Each additional word adds to the overall meaning of the term under consideration by the NLP algorithm. The emphasis word gets increasingly ambiguous as the number of words in a sentence or phrase increases. BERT accounts for the enhanced meaning by accounting for the influence of all other words in a phrase on the focus word and minimising the left-to-right momentum that biases words towards a certain meaning as a sentence progresses by reading bidirectionally.

Figure 6.2 shows how the text is converted into contextualized and meaningful embeddings.

CHAPTER 7

OTHER SPECIFICATIONS

7.1 Advantages and Limitations

Currently, regression testing happens manually where the developers and QA integrate to identify the affected functionality. With this project, we aim to use ML / AI based techniques to automate the testing process. The following are the advantages and limitations for this task:

7.1.1 Advantages

The following are the advantages of the utility built:

- Feedback for freshly built features might take a long time without test automation. Test automation allows you to shorten the feedback cycle and provide faster validation for different stages of your product's development.
- Less time will be spent in validating newly produced features when automation testing approach is done.
- Once the automated test suite is established, reusing tests for new use cases or even other projects is straightforward. This automated test suite can quickly connect to another project, which is a big plus.
- When certain tests fail, automated testing delivers more information than manual testing. Automated software testing along with the application's memory contents, data tables, file contents, and other internal program states, also shows you the memory contents, data tables, file contents, and other internal program states. This assists developers in determining what went wrong.
- During manual testing, even the most experienced testing engineer will make mistakes. Faults can happen, especially when evaluating a complicated use case. Automated tests, on the other hand, can conduct tests with 100 % accuracy since they generate the same result every time they are run.

7.1.2 Limitations

The following are the limitations of the utility built:

- These black box models in machine learning are produced straight from data by an algorithm, which means that people, even those who develop them, do not comprehend how variables are combined to form prediction, even with a list of input variables and their functionalities.
- Although automated testing eliminates the mechanical execution of the testing process, the preparation of test cases is still the responsibility of testing specialists.
- Automated testing might be a time efficient process but requires monetary investment.

- It might be challenging to optimise test cases in regression testing. As the scope of regression testing expands with each sprint, maintaining an increasing number of automated test cases becomes increasingly complex. The regression test cases must be updated to accommodate these changes.

7.2 Future Scope

1. Current test cases have labels and components written manually, they could be automatically generated to reduce more work and increase efficiency.
2. More parameters can be recognized and used for better test case identification.
3. We intend to widen the scope of our research by focusing on the use of AI and machine learning in automated regression testing.
4. We also plan to expand our project by automating additional sorts of test suites, such as unit testing, integration testing, keyword-driven testing, and so on.
5. In addition, we will be working on additional advanced ML techniques which will enable the program to identify the relevant test cases with an improved accuracy and efficiency.

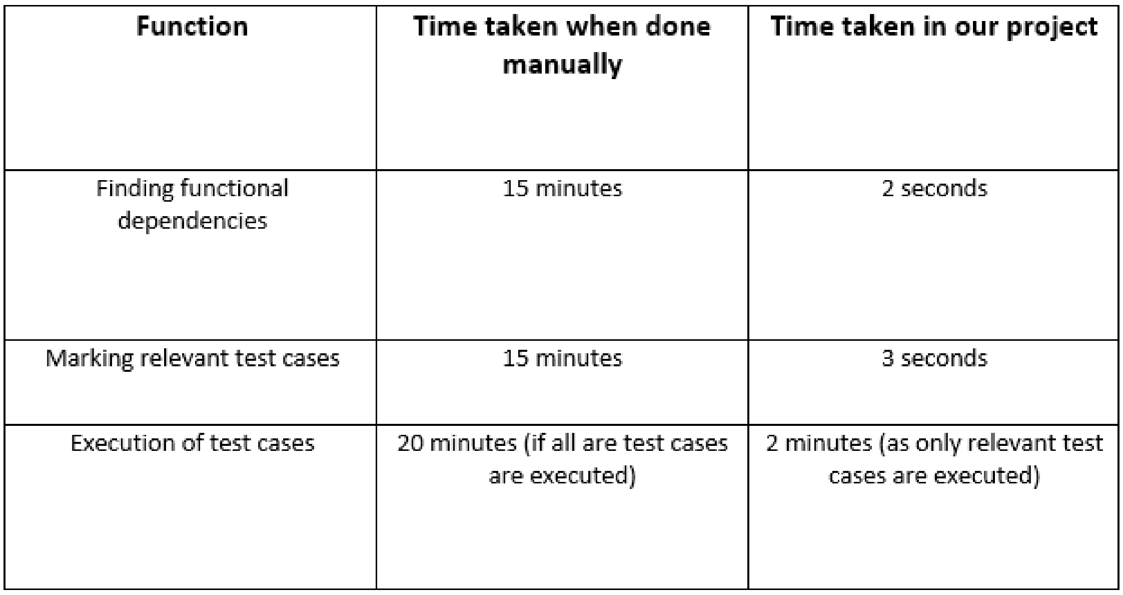
CHAPTER 8

RESULTS

8.1 Comparative Analysis

8.1.1 Manual Time Taken vs Time Taken by our code

- The main motive of this project is to reduce the time taken to manually check all the code changes and according to those changes select the relevant test cases that are to be executed.

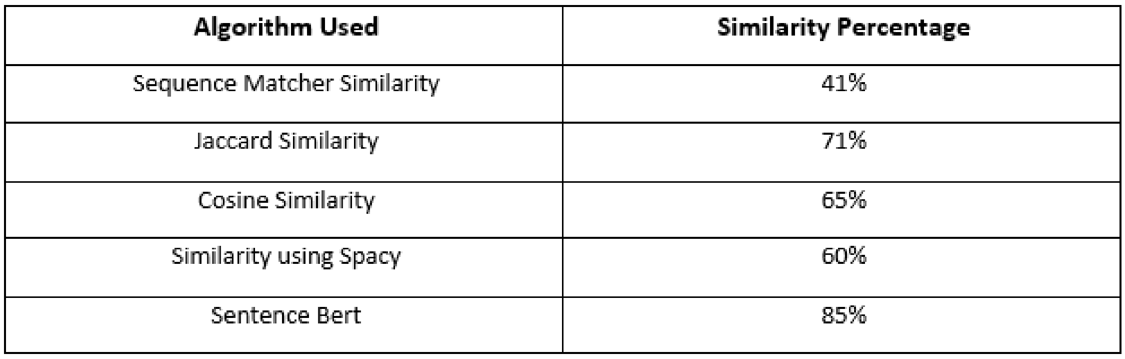


Function	Time taken when done manually	Time taken in our project
Finding functional dependencies	15 minutes	2 seconds
Marking relevant test cases	15 minutes	3 seconds
Execution of test cases	20 minutes (if all are test cases are executed)	2 minutes (as only relevant test cases are executed)

Figure 8.1: Time Comparison for the repository we used

8.1.2 Comparison between various Text Similarity Algorithms used

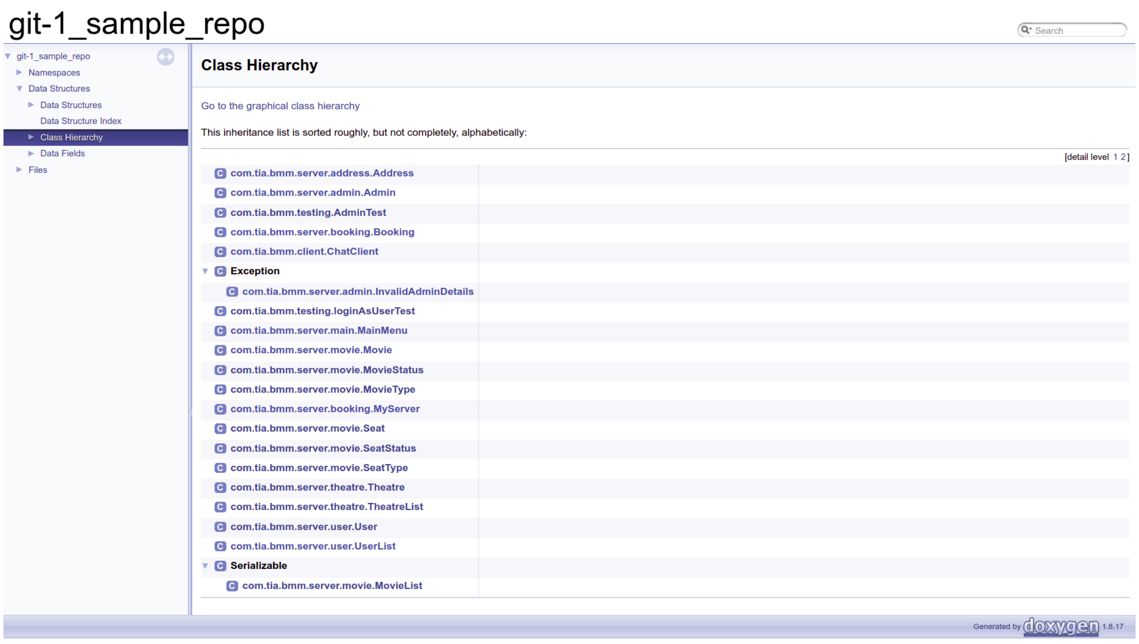
- While working on our project we had worked on various POC's in order to get optimal results. The table below provides an comparative analysis between the algorithms we worked on.



Algorithm Used	Similarity Percentage
Sequence Matcher Similarity	41%
Jaccard Similarity	71%
Cosine Similarity	65%
Similarity using Spacy	60%
Sentence Bert	85%

Figure 8.2: Similarity Analysis for the Algorithms Used

8.2 Screenshots


git-1_sample_repo

Class Hierarchy

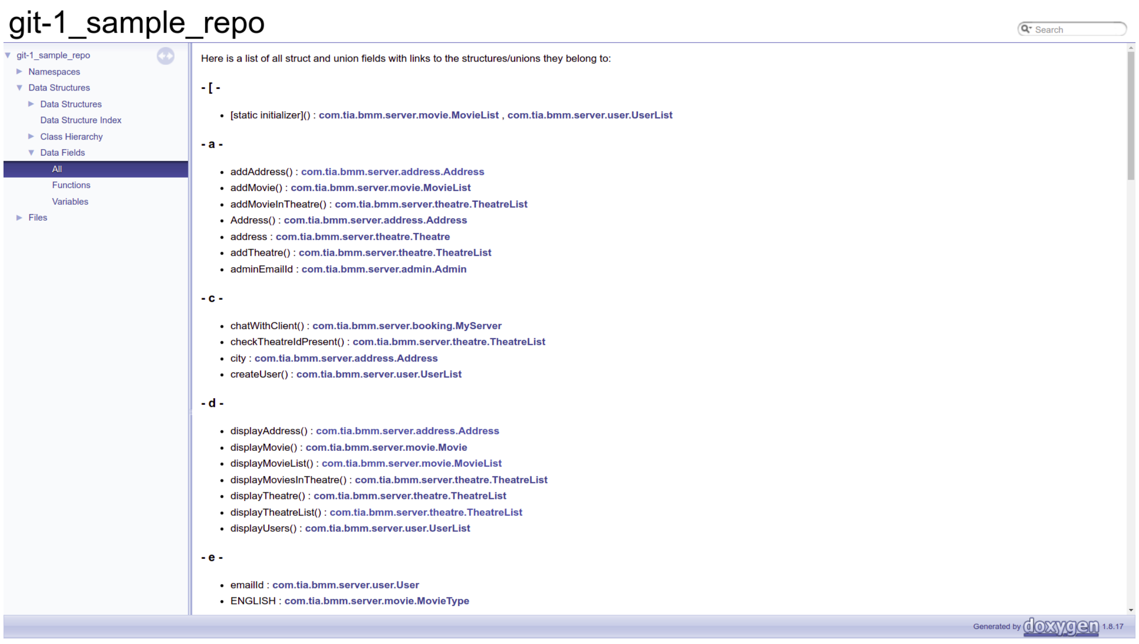
Go to the graphical class hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

[detail level 1 2]

- com.tia.bmm.server.address.Address
- com.tia.bmm.server.admin.Admin
- com.tia.bmm.testing.AdminTest
- com.tia.bmm.server.booking.Booking
- com.tia.bmm.client.ChatClient
- Exception
  - com.tia.bmm.server.admin.InvalidAdminDetails
- com.tia.bmm.testing.loginAsUserTest
- com.tia.bmm.server.main.MainMenu
- com.tia.bmm.server.movie.Movie
- com.tia.bmm.server.movie.MovieStatus
- com.tia.bmm.server.movie.MovieType
- com.tia.bmm.server.booking.MyServer
- com.tia.bmm.server.movie.Seat
- com.tia.bmm.server.movie.SeatStatus
- com.tia.bmm.server.movie.SeatType
- com.tia.bmm.server.theatre.Theatre
- com.tia.bmm.server.theatre.TheatreList
- com.tia.bmm.server.user.User
- com.tia.bmm.server.user.UserList
- Serializable
  - com.tia.bmm.server.movie.MovieList

Figure 8.3: Doxygen Runner Classes



git-1_sample_repo

Search

- git-1_sample_repo
  - Namespaces
  - Data Structures
    - Data Structures
    - Data Structure Index
    - Class Hierarchy
    - Data Fields
      - All
      - Functions
      - Variables
  - Files

Here is a list of all struct and union fields with links to the structures/unions they belong to:

- [-

- [static initializer]() : com.tia.bmm.server.movie.MovieList , com.tia.bmm.server.user.UserList

- a -

- addAddress() : com.tia.bmm.server.address.Address
- addMovie() : com.tia.bmm.server.movie.MovieList
- addMovieInTheatre() : com.tia.bmm.server.theatre.TheatreList
- Address() : com.tia.bmm.server.address.Address
- address : com.tia.bmm.server.theatre.Theatre
- addTheatre() : com.tia.bmm.server.theatre.TheatreList
- adminEmailId : com.tia.bmm.server.admin.Admin

- c -

- chatWithClient() : com.tia.bmm.server.booking.MyServer
- checkTheatreIdPresent() : com.tia.bmm.server.theatre.TheatreList
- city : com.tia.bmm.server.address.Address
- createUser() : com.tia.bmm.server.user.UserList

- d -

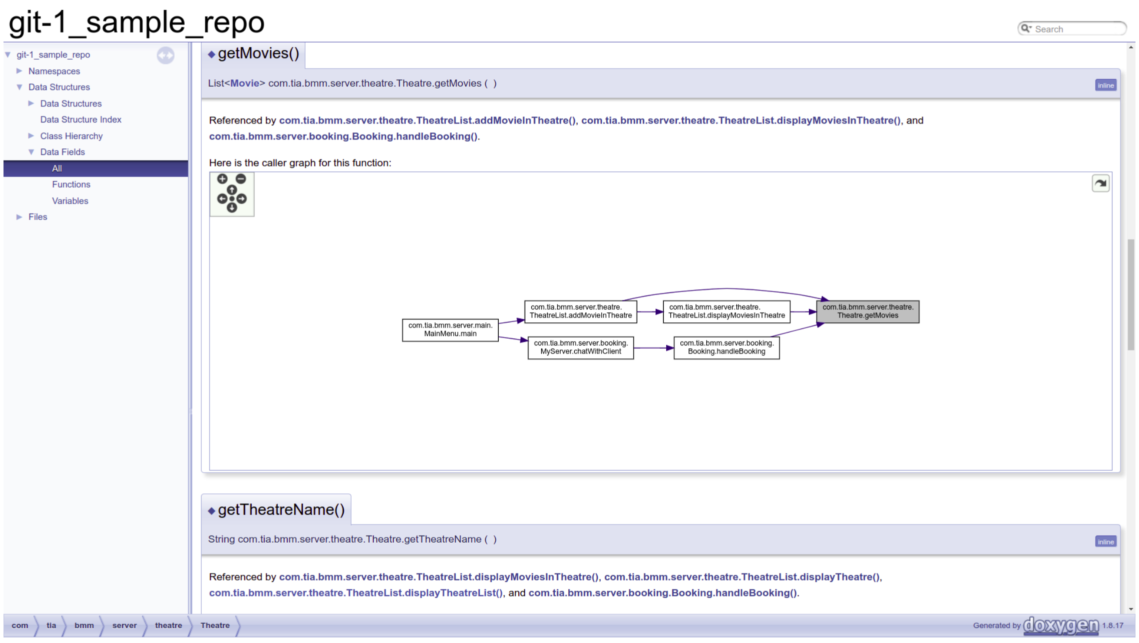
- displayAddress() : com.tia.bmm.server.address.Address
- displayMovie() : com.tia.bmm.server.movie.Movie
- displayMovieList() : com.tia.bmm.server.movie.MovieList
- displayMoviesInTheatre() : com.tia.bmm.server.theatre.TheatreList
- displayTheatre() : com.tia.bmm.server.theatre.TheatreList
- displayTheatreList() : com.tia.bmm.server.theatre.TheatreList
- displayUsers() : com.tia.bmm.server.user.UserList

- e -

- emailId : com.tia.bmm.server.user.User
- ENGLISH : com.tia.bmm.server.movie.MovieType

Generated by doxygen 1.8.17

Figure 8.4: Doxygen Runner Data Fields



git-1_sample_repo

Search

- git-1_sample_repo
  - Namespaces
  - Data Structures
    - Data Structures
    - Data Structure Index
    - Class Hierarchy
    - Data Fields
      - All
      - Functions
      - Variables
  - Files

◆ getMovies()

List<Movie> com.tia.bmm.server.theatre.Theatre.getMovies ( ) inline

Referenced by com.tia.bmm.server.theatre.TheatreList.addMovieInTheatre(), com.tia.bmm.server.theatre.TheatreList.displayMoviesInTheatre(), and com.tia.bmm.server.booking.Booking.handleBooking().

Here is the caller graph for this function:

com.tia.bmm.server.main.
MainMenu.main
com.tia.bmm.server.theatre.
TheatreList.addMovieInTheatre
com.tia.bmm.server.booking.
MyServer.chatWithClient
com.tia.bmm.server.theatre.
TheatreList.displayMoviesInTheatre
com.tia.bmm.server.booking.
Booking.handleBooking
com.tia.bmm.server.theatre.
Theatre.getMovies

◆ getTheatreName()

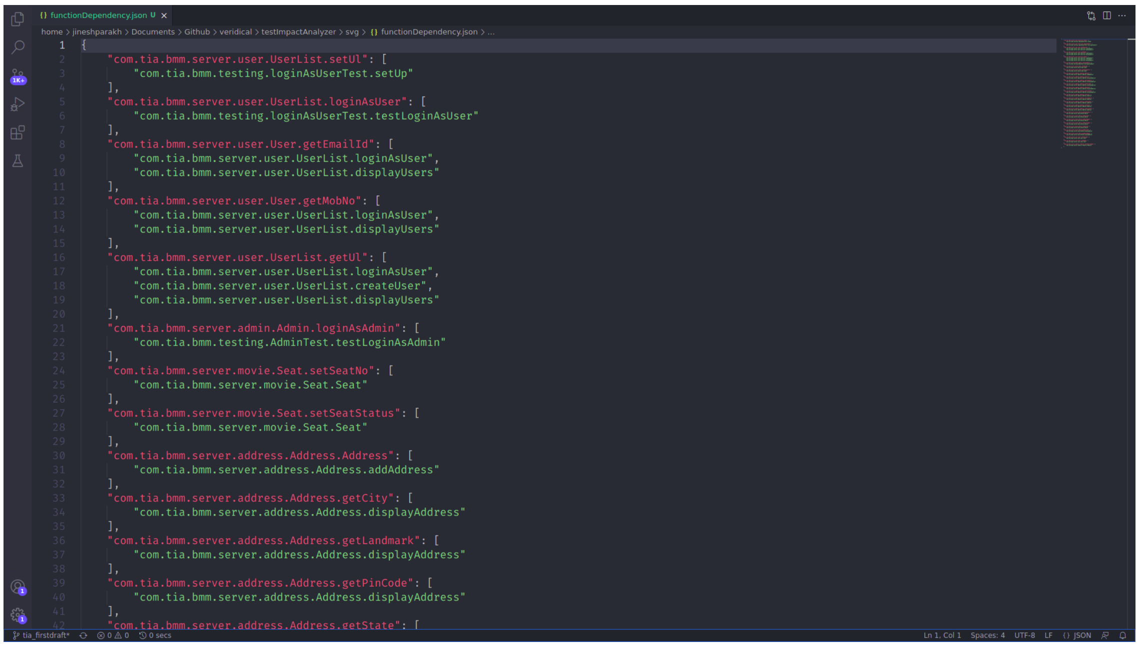
String com.tia.bmm.server.theatre.Theatre.getTheatreName ( ) inline

Referenced by com.tia.bmm.server.theatre.TheatreList.displayMoviesInTheatre(), com.tia.bmm.server.theatre.TheatreList.displayTheatre(), com.tia.bmm.server.theatre.TheatreList.displayTheatreList(), and com.tia.bmm.server.booking.Booking.handleBooking().

com 〉 tia 〉 bmm 〉 server 〉 theatre 〉 Theatre 〉

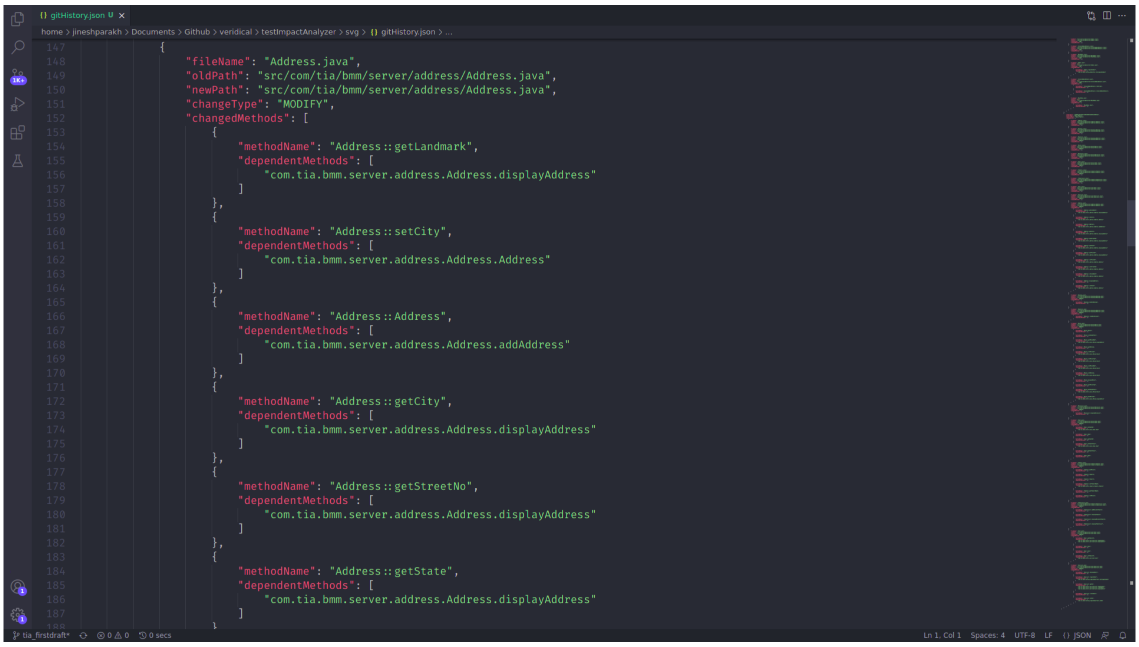
Generated by doxygen 1.8.17

Figure 8.5: Doxygen Runner Call graphs


functionDependency.json U
home > jineshparakh > Documents > Github > veridical > testImpactAnalyzer > svg > functionDependency.json > ...
tia_firstdraft* 0 0 0 secs
Ln 1, Col 1 Spaces: 4 UTF-8 LF JSON

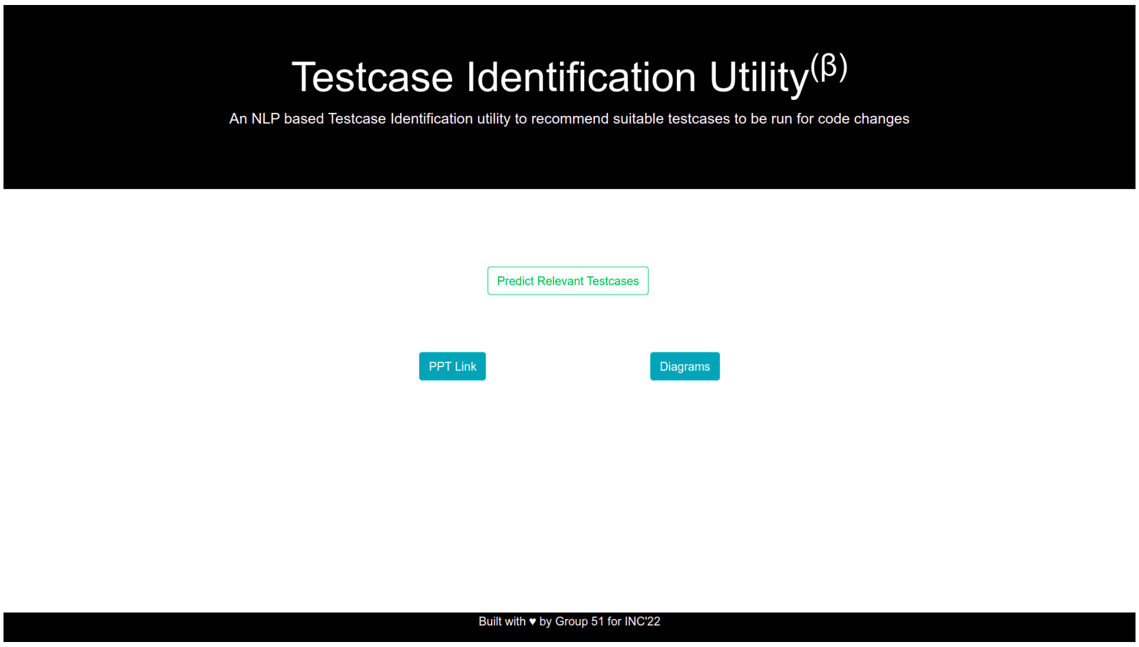
```json
{
    "com.tia.bmm.server.user.UserList.setUl": [
        "com.tia.bmm.testing.loginAsUserTest.setUp"
    ],
    "com.tia.bmm.server.user.UserList.loginAsUser": [
        "com.tia.bmm.testing.loginAsUserTest.testLoginAsUser"
    ],
    "com.tia.bmm.server.user.User.getEmailId": [
        "com.tia.bmm.server.user.UserList.loginAsUser",
        "com.tia.bmm.server.user.UserList.displayUsers"
    ],
    "com.tia.bmm.server.user.User.getMobNo": [
        "com.tia.bmm.server.user.UserList.loginAsUser",
        "com.tia.bmm.server.user.UserList.displayUsers"
    ],
    "com.tia.bmm.server.user.UserList.getUl": [
        "com.tia.bmm.server.user.UserList.loginAsUser",
        "com.tia.bmm.server.user.UserList.createUser",
        "com.tia.bmm.server.user.UserList.displayUsers"
    ],
    "com.tia.bmm.server.admin.Admin.loginAsAdmin": [
        "com.tia.bmm.testing.AdminTest.testLoginAsAdmin"
    ],
    "com.tia.bmm.server.movie.Seat.setSeatNo": [
        "com.tia.bmm.server.movie.Seat.Seat"
    ],
    "com.tia.bmm.server.movie.Seat.setSeatStatus": [
        "com.tia.bmm.server.movie.Seat.Seat"
    ],
    "com.tia.bmm.server.address.Address.Address": [
        "com.tia.bmm.server.address.Address.addAddress"
    ],
    "com.tia.bmm.server.address.Address.getCity": [
        "com.tia.bmm.server.address.Address.displayAddress"
    ],
    "com.tia.bmm.server.address.Address.getLandmark": [
        "com.tia.bmm.server.address.Address.displayAddress"
    ],
    "com.tia.bmm.server.address.Address.getPinCode": [
        "com.tia.bmm.server.address.Address.displayAddress"
    ],
    "com.tia.bmm.server.address.Address.getState": [
```

Figure 8.6: Function Dependency JSON


gitHistory.json U
home > jineshparakh > Documents > Github > veridical > testImpactAnalyzer > svg > gitHistory.json > ...
tia_firstdraft* 0 0 0 secs
Ln 1, Col 1 Spaces: 4 UTF-8 LF JSON

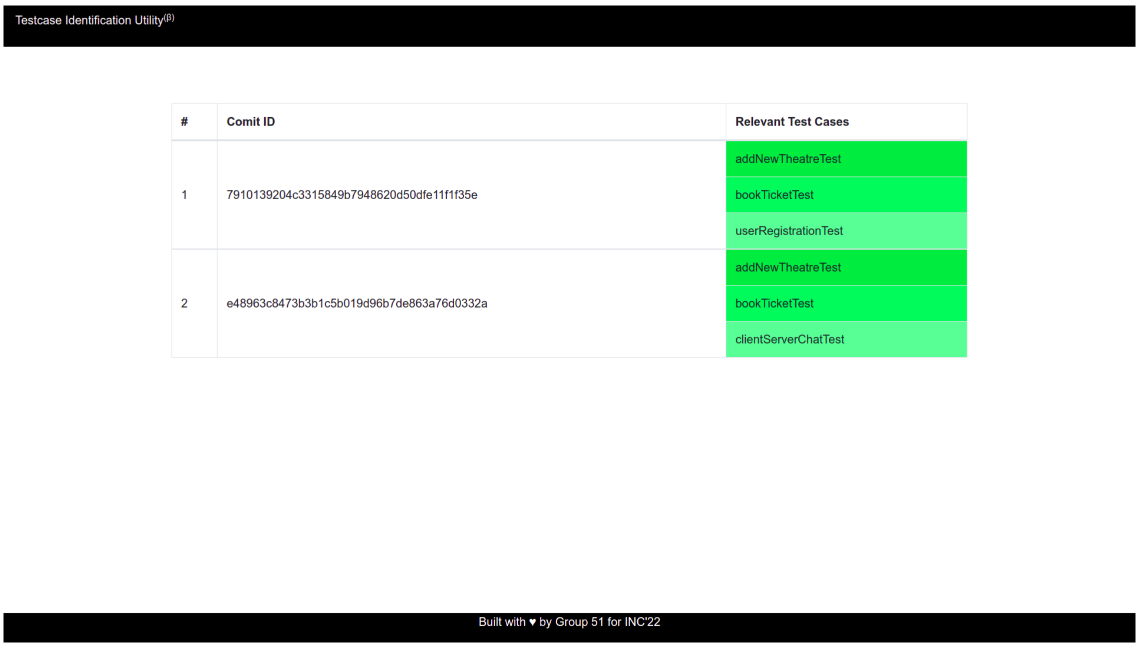
```json
                {
                    "fileName": "Address.java",
                    "oldPath": "src/com/tia/bmm/server/address/Address.java",
                    "newPath": "src/com/tia/bmm/server/address/Address.java",
                    "changeType": "MODIFY",
                    "changedMethods": [
                        {
                            "methodName": "Address::getLandmark",
                            "dependentMethods": [
                                "com.tia.bmm.server.address.Address.displayAddress"
                            ]
                        },
                        {
                            "methodName": "Address::setCity",
                            "dependentMethods": [
                                "com.tia.bmm.server.address.Address.Address"
                            ]
                        },
                        {
                            "methodName": "Address::Address",
                            "dependentMethods": [
                                "com.tia.bmm.server.address.Address.addAddress"
                            ]
                        },
                        {
                            "methodName": "Address::getCity",
                            "dependentMethods": [
                                "com.tia.bmm.server.address.Address.displayAddress"
                            ]
                        },
                        {
                            "methodName": "Address::getStreetNo",
                            "dependentMethods": [
                                "com.tia.bmm.server.address.Address.displayAddress"
                            ]
                        },
                        {
                            "methodName": "Address::getState",
                            "dependentMethods": [
                                "com.tia.bmm.server.address.Address.displayAddress"
                            ]
                        }
```

Figure 8.7: GIT History JSON



Testcase Identification Utility$^{(\beta)}$
An NLP based Testcase Identification utility to recommend suitable testcases to be run for code changes
Predict Relevant Testcases
PPT Link
Diagrams
Built with ♥ by Group 51 for INC'22

Figure 8.8: Flask Server



Testcase Identification Utility$^{(\beta)}$

#	Comit ID	Relevant Test Cases
1	7910139204c3315849b7948620d50dfe11f1f35e	addNewTheatreTest
		bookTicketTest
		userRegistrationTest
2	e48963c8473b3b1c5b019d96b7de863a76d0332a	addNewTheatreTest
		bookTicketTest
		clientServerChatTest

Built with ♥ by Group 51 for INC'22

Figure 8.9: Final Output

CHAPTER 9

CONCLUSION

9.1 Conclusion

Software Testing is a major part of Software Development Life-cycle. During the first phase of this project, a comprehensive research into current methodologies for identifying test cases for regression testing revealed that it was primarily done manually.

Test Case Identification Utility aims to automate the process of recognizing relevant test cases in order to save manual effort and time. This tool would reduce the time taken in both; recognizing the test cases relevant to a particular code check-in and consequently running a much smaller set of test cases. The present methods for identifying test cases are slow and prone to error, techniques for identifying labels with the same meaning but different terms have yet to be researched much, and that the majority of research is focused on improving accuracy rather than cost reduction. The use of AI approaches such as Natural Language Processing (NLP) and Pattern Matching for test case identification has a lot of potential, as several large companies have been working on tools for this for a decade and can help overcome the challenges mentioned above.

Hence, this tool is a beginning to automating the entire testing phase in the SDLC. This tool is all in all a great reducer of monotonous human work, in-turn which will be error free.

CHAPTER 10

REFERENCES

Bibliography

[1] F. Normann. Test case selection based on code changes, 2019.

[2] Mojtaba Ghaleb Taher Briand Lionel Pan, Rongqi Bagherzadeh. Test case selection and prioritization using machine learning: A systematic literature review, 2021.

[3] Aly. Gomaa, Wael Fahmy. A survey of text similarity approaches, 2013.

[4] A. Hammad H. Hourani and M. Lafi. The impact of artificial intelligence on software testing, 2019.

[5] X. Xue Y. Pang and A. S. Namin. Identifying effective test cases through k-means clustering for enhancing regression testing, 2013.

[6] Microsoft Corporation. Speed up testing by using test impact analysis (tia), 2018.

[7] B. Nguyen S. Dhanda E. Nickell R. Siemborski A. Memon, Z. Gao and J. Micco. Taming google-scale continuous testing,, 2017.

[8] http://test-load balancer.github.io/. What is test load balancer (tlb)?

[9] Infinitest http://infinitest.github.io/. Infinitest - the continuous test runner for the jvm.

[10] https://www.javatpoint.com/agile methodology.