# BUG PRIORITIZATION FRAMEWORK FOR BUG FIXING USING META-HEURISTICS

**Enrol. No. (s) - 9919103178, 9919103183, 9919103188**

**Name of Student (s) – Manan Chaudhary, Kartikeya Consul, Megha Jain**

**Name of supervisor(s) – Prof. Chetna Gupta**


**May – 2023**

**Submitted in partial fulfilment of the Degree of**

**Bachelor of Technology**

**in**

**Computer Science Engineering**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY**

**JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA**

# TABLE OF CONTENTS


| | | *Page No.* |
|---|---|---|
| *Declaration* | | *ii* |
| *Certificate* | | *iii* |
| *Acknowledgement* | | *iv* |
| *Summary* | | *v* |
| *List of figures* | | *vi* |
| *List of tables* | | *vii* |
| *List of symbols & acronyms* | | *viii* |
| **Chapter 1 :** | **Introduction** | |
| | 1.1 General Introduction | 1 |
| | 1.2 Problem Statement | 2 |
| | 1.3 Significance/Novelty of the problem | 2 |
| | 1.4 Empirical Study | 3 |
| | 1.5 Brief Description of the Solution Approach | 3-4 |
| **Chapter 2 :** | **Literature Survey** | |
| | 2.1 Summary of papers studied | 5-6 |
| **Chapter 3 :** | **Requirement Analysis and Solution Approach** | |
| | 3.1 Requirement Analysis | 7-9 |
| | 3.2 Solution Approach | 9-11 |
| **Chapter 4 :** | **Modelling and Implementation Details** | |
| | 4.1 Design Diagrams | 12 |
| | 4.1.1 Use Case diagram | 12 |
| | 4.1.2 Class diagram | 13 |
| | 4.1.3 Activity diagram | 14 |
| | 4.2 Implementation details | 15-25 |
| **Chapter 5 :** | **Testing** | |
| | 5.1 Testing Plan | 26-30 |
| | 5.2 Component decomposition and type of testing required | 31-32 |
| | 5.3 Error and Exception Handling | 32-33 |
| | 5.4 Limitations of the solution | 33-34 |
| **Chapter 6 :** | **Findings, Conclusion, and Future Work** | |
| | 6.1 Findings | 35-39 |
| | 6.2 Conclusion | 39-40 |
| | 6.3 Future Work | 40 |
| | **REFERENCES** | 41 |

# DECLARATION

We hereby declare that this submission is our own work titled “Bug prioritization framework for bug fixing using meta-heuristics” and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

**Place:**

**Date:**

**Signature:**

**Name:** Manan Chaudhary

**Enrolment No:** 9919103178

**Signature:**

**Name:** Megha Jain

**Enrolment No:** 9919103188

**Signature:**

**Name:** Kartikeya Consul

**Enrolment No:** 9919103183

# CERTIFICATE

This is to certify that the work titled **"Bug prioritization framework for bug fixing using meta-heuristics"** submitted by **"Manan Chaudhary, Megha Jain, Kartikeya Consul"** in partial fulfilment for the award of degree of B.Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

**Signature of Supervisor** ……..

**Name of Supervisor** ……..

**Designation** ……..

**Date** ……..

# ACKNOWLEDGEMENT

We would like to place on record our deep sense of gratitude to Prof. Chetna Gupta, Professor, Jaypee Institute of Information Technology, India for her generous help, useful suggestions along with her stimulating guidance, continuous encouragement and supervision throughout the course of present work. We also wish to extend our thanks to friends and other classmates for their insightful comments and constructive suggestions to improve the quality of this project work.

**Signature of the Student** ……..

**Name of Student** Manan Chaudhary

**Enrolment Number** 9919103178

**Date** ……..

**Signature of the Student** ……..

**Name of Student** Megha Jain

**Enrolment Number** 9919103188

**Date** ……..

**Signature of the Student** ……..

**Name of Student** Kartikeya Consul

**Enrolment Number** 9919103183

**Date** ……..

# SUMMARY

The bug prioritization proposed framework assists in identifying the most severe bugs to programmers. It includes a feature weighting method that focuses on Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) to find relevant features for severity forecasting. This framework incorporates four stages to operate effectively and utilize performance indicators such as capability ranking, bug severities, and mean bug fixing time to rank programmers. The proposed model also employs classification algorithms such as KNN to predict various levels of bug intensity. Two experiments were conducted to evaluate the effectiveness of the proposed model, which involved predicting bug intensity levels using PSO-KNN and GA-KNN methods. Overall, the model is a promising framework that can enhance the bug triage process and improve the quality of bug fixing.

The proposed model aims to refine a pre-trained framework iteratively using both latest data and sections of historical information. While the model has some issues related to memory, it is a feasible approach for improving the pre-trained framework.

**Signature of Student**

**Name** Manan Chaudhary

**Signature of Student**

**Name** Megha Jain

**Signature of Student** **Signature of Supervisor**

**Name** Kartikeya Consul **Name**

**Date** **Date**

# LIST OF FIGURES


| Fig. No. | Topic Name | Page No. |
|---|---|---|
| Figure 1 | Project workflow | 11 |
| Figure 2 | Use case diagram | 12 |
| Figure 3 | Class diagram | 13 |
| Figure 4 | Activity diagram | 14 |
| Figure 5 | Eclipse platform dataset | 15 |
| Figure 6 | Dropping un-necessary columns | 16 |
| Figure 7 | Tokenization | 16 |
| Figure 8 | Converting into lowercase | 17 |
| Figure 9 | Removing Stopwords | 17 |
| Figure 10 | Removing Punctuations | 18 |
| Figure 11 | Changing ' " ' with blank space ' ' | 18 |
| Figure 12 | Stemming | 19 |
| Figure 13 | Applying TF-IDF | 19 |
| Figure 14 | (a) Eclipse platform GA<br>(b) Eclipse platform PSO | 37 |
| Figure 15 | (a) Firefox platform GA<br>(b) Firefox platform PSO | 38 |
| Figure 16 | (a) Mozilla platform GA<br>(b) Mozilla platform PSO | 38 |
| Figure 17 | (a) Thunderbird platform GA<br>(b) Thunderbird platform PSO | 39 |

# LIST OF TABLES


| Table No. | Table Name | Page No. |
|---|---|---|
| Table 1 | Specific details of logs (bug reports) taken into account for the experiment | 25 |
| Table 2 | Testing plan | 26 |
| Table 3 | Software & hardware items | 30 |
| Table 4 | Component decomposition and identification of tests required | 31 |
| Table 5 | Debugging techniques | 32 |
| Table 6 | Results | 36 |

# LIST OF SYMBOLS & ACRONYMS

| | |
|---|---|
| **ACO** | **Ant Colony Optimization** |
| **PSO** | **Particle Swarm Optimization** |
| **GA** | **Genetic Algorithm** |
| **SQA** | **Software Quality Assurance** |
| **KNN – K** | **Nearest Neighbours** |
| **BPaaS** | **Bug Prediction as a Service** |
| **SVM** | **Support Vector Machine** |
| **NB** | **Naïve Bayes** |
| **ML** | **Machine Learning** |
| **DT** | **Decision Trees** |
| **TF-IDF** | **Term Frequency – Inverse Document Frequency** |
| **TP** | **True Positive** |
| **FN** | **False Negative** |
| **TN** | **True Negative** |
| **FP** | **False Positive** |
| **NLP** | **Natural Language Processing** |
| **GPU** | **Graphical Processing Unit** |

# CHAPTER-1

# INTRODUCTION

## 1.1. GENERAL INTRODUCTION

Bug fixing is a crucial daily activity in software development and maintenance that significantly impacts Software Quality Assurance (SQA). Bug tracking systems, such as Bugzilla, are frequently utilized to monitor and handle software bugs. Bug triaging is a crucial step in the defect assignment process of bug tracking systems, aiming to allocate bug reports to the most suitable developers. Ideally, a confirmed bug should be efficiently assigned to a suitable developer who can quickly fix it. However, bug reports are often inaccurately assigned to developers, leading to delays in bug resolution in open-source software projects, even with efforts to ensure accurate assignments. Bugs may be reassigned multiple times until eventually fixed by a developer, which can take a significant amount of time, as shown by a study in the Eclipse project. Therefore, precise developer recommendations for bug reports can improve the bug triaging process's efficiency and software maintenance. Researchers have proposed various automated methods in recent years to enhance the bug triaging process's precision and productivity while reducing human labour costs.

Here, we are developing a bug prioritization system so that an efficient programmer is assigned to the bug according to its severity. It helps software developers identify and prioritize bugs based on their potential impact and severity. One approach to developing such a system is to use meta-heuristic feature selection algorithms for optimization. These algorithms are designed to automatically identify the most important features of a dataset. Meta-heuristic algorithms, such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Genetic Algorithms (GA), are used to identify the most relevant features of a dataset by exploring a large search space and identifying the optimal solution. These algorithms are designed to be efficient, effective, and scalable, making them a powerful tool for building a bug prioritization system. By using meta-heuristic feature selection algorithms for optimization, a programmer recommendation system for bug triage can provide accurate and reliable recommendations to software developers, helping them to prioritize and address bugs more efficiently. This can lead to improved software quality, increased productivity, and reduced costs associated with software maintenance. Overall, a bug prioritization system using meta-heuristic feature selection algorithms is a promising approach to improving software development practices, and has the potential to become an essential tool for software development teams.

## 1.2. PROBLEM STATEMENT

Bug prioritization is a crucial task in software development, where it involves prioritizing the bugs based on their criticality to ensure that the most critical bugs are fixed first. However, manual bug prioritization is often time-consuming and may lead to biases, especially in large software projects. To address this issue, meta-heuristic algorithms have been proposed as a way to automate the bug prioritization process. The main motive here is to develop an automated approach that can accurately and efficiently prioritize the bugs based on their criticality using meta-heuristic algorithms such as Genetic Algorithm and Particle Swarm Optimization.

## 1.3. SIGNIFICANCE

A bug prioritization system is significant because it helps software developers identify and address bug based on their severity and potential impact. By prioritizing bugs, developers can focus their efforts on fixing the most critical issues first, which can improve software quality, increase productivity, and reduce costs associated with software maintenance. A well-designed bug prioritization system can also help allocate bugs to the most appropriate developers, reducing the time and effort required to assign a bug report. This leads to a more efficient bug triage process, as bugs are resolved more quickly and efficiently, reducing the number of reassignments and ensuring that bugs are fixed right the first time. Additionally, a bug prioritization system can also benefit stakeholders such as customers or end-users, as it ensures that critical bugs are addressed first, leading to a more reliable and stable software product. Overall, a bug prioritization system is a valuable tool for software development teams, as it can significantly improve software quality, increase productivity, reduce maintenance costs, and enhance customer satisfaction. Here are some of the key benefits of such a system:

**Improved Bug Triage:** A bug prioritization system can help developers to accurately identify the root cause of bugs based on their potential impact and severity. This can save developers time and effort, and help to ensure that bugs are addressed in a timely manner

**Increased Productivity:** By automating the bug triage process, a recommendation system can free up developers' time, allowing them to focus on more complex coding tasks. This can lead to increased productivity and faster time-to-market for software products.

**Improved Software Quality:** By identifying and addressing bugs more efficiently, a recommendation system can help to improve the overall quality of software products. This can lead to higher customer satisfaction, increased user engagement, and a stronger reputation for the software development team.

**Better Resource Allocation:** A prioritization system can help software development teams to allocate their resources more effectively, by focusing on the most critical bugs and issues. This can help to reduce costs associated with software maintenance, and ensure that resources are used in the most efficient manner possible.

## 1.4. EMPIRICAL STUDY

We collect a dataset of bugs and associated information from a software development team. The dataset will include information about the bug's severity, its root cause, and the time required to fix it. The bug prioritization system will be designed and implemented using meta-heuristic feature selection algorithms for optimization, and will be tested in a real-world software development environment. The effectiveness of the system will be evaluated using a range of metrics, including accuracy, precision, recall, and F1 score, as well as metrics related to intensity and severity of bug. The prioritization system will be compared to other bug triage approaches, such as manual bug triage or other automated systems. The comparison will involve evaluating the relative benefits of using meta-heuristic feature selection algorithms for optimization in bug triage. The study will also analyse the results to determine the strengths and weaknesses of the recommendation system, and identify areas for improvement.

The results of the study showed that the recommendation system was effective at accurately identifying and prioritizing bugs based on their potential impact and severity. The system's accuracy was higher than manual bug triage, and its precision and recall were also high. The recommendation system was able to provide effective recommendations to software developers, leading to improved productivity and code quality. However, the system struggled to identify some types of bugs, particularly those with complex root causes or those that were difficult to replicate.

## 1.5. SOLUTION APPROACH

This section presents the brief description of bug prioritization system for bug assignment. Before predicting the appropriate programmer for fixing the bugs, a meta-heuristic based feature weighting method is incorporated into the proposed prioritization system. The feature weighting method aims to identify the optimum features from the feature space. KNN-based ML methods estimate the severity of bugs on an online dataset taken from BugZilla. Additionally, it is presumable that a directory of programmers with a track record of effectively and speedily resolving defects is also present according to the seriousness of the Bug. It is also assumed that a directory of programmers with a proven track record of effectively and quickly resolving defect is available based on the seriousness of the bug.

The main steps of the proposed model are described below:

a) Data pre-processing

b) Feature extraction,

c) Feature weighting using the PSO, ACO, and GA algorithms

d) Bug prioritization based on severity.

# CHAPTER-2

# LITERATURE SURVEY

## 2.1. SUMMARY OF THE PAPERS STUDIED

Alazzam et al. (2020) proposed a novel approach for enhancing bug triaging systems using machine learning, which involves a graph-based feature augmentation approach. The proposed approach utilizes graph partitioning based on neighbourhood overlap to discover relationships among the terms of bug summaries. The terms of bug summaries are represented as nodes in a graph, and the graph is partitioned into clusters of terms. Terms in strong clusters are augmented to the original feature vectors of bug summaries based on their similarity to each other and to the bug summary. The authors also used other techniques such as term frequency, term correlation, and topic modelling to identify latent terms and augment them to the original feature vectors of bug summaries. Finally, they utilized frequency, correlation, and neighbourhood overlap techniques to create another feature augmentation approach that enriches the feature vectors of bug summaries for bug triaging. Overall, the proposed approach enhances the performance of bug triaging systems by improving the accuracy and effectiveness of the feature vectors used for machine learning.

Subbiah et al. (2019) proposed a model for predicting the presence or absence of bugs in code using machine learning classification models, which can be deployed on a cloud platform for software development companies to use as a web service worldwide. The model utilizes Microsoft Azure's machine learning platform and is referred to as Bug Prediction as a Service (BPaaS).To address three difficulties in bug prediction, including programmer rating, the evolution process, and tolerance, the researchers employ a social media network-driven technique. This technique prioritizes coders who actively engage in posting comment procedures for bug-solving activities. In addition, the researchers use NB and SVM classifiers to choose the correct coder. The findings indicate that the social network analysis strategy significantly improves the recognition rate of NB and SVM classifiers by 2% and 10%, respectively. Overall, the proposed model and technique provide a useful and efficient solution for bug prediction in software development.

Yadav et al. (2019) proposed a model to improve the outcomes of bug assessments by leveraging the text resemblance of bug reports and the tradition of programmers passing around bugs. The researchers also identify duplicate issues for all bug allocation methods used in their work. The proposed method

significantly reduces the bug tossing time by approximately 80 to 90%.

The process of bug triage is often viewed as an optimization challenge, as described in Mani et al. (2019). In their work, the researchers proposed an organizational filtering-based premium triage method to reduce the time required for issue fixes. The proposed models and techniques in various studies provide effective solutions for improving bug assessments and reducing the time required for issue fixes in software development.

In Lee and Seo (2019), a programmer recommendation engine that incorporates five ML techniques (Rules, DT, NB, C4.5, and SVM) is discussed. The proposed model operates in two stages: first, a group of bug reports is gathered from bug directories, and then ML algorithms are used to rate software programmers. The model achieved a 75% prediction performance for five projects (Myly, GCC, Bugzie, Eclipse, and Firefox). It is demonstrated that the proposed system is effective in selecting programmers who can fix related defects for a particular project.

In Hammouri et al. (2018), a social network-based rating strategy for selecting the coder for bug solving was proposed. The approach integrates competence rating with the KNN model and demonstrated that the social media platform's out-degree and basic frequency measurements outperform existing methods. The proposed model achieved more than a 60% recall rate for Mozilla and Firefox datasets.

# CHAPTER-3

# REQUIREMENT ANALYSIS AND SOLUTION APPROACH

## 3.1. REQUIREMENT ANALYSIS

### 3.1.1. INTRODUCTION

The Bug Prioritization System is a software system that utilizes meta-heuristic algorithms such as genetic algorithm (GA) and particle swarm optimization (PSO) to prioritize software bugs based on their severity and impact on the system. The system aims to assist software developers in identifying critical bugs that require immediate attention to prevent system failure and ensure optimal system performance.

### 3.1.2. FUNCTIONAL REQUIREMENTS

**Data Collection:** The system shall collect data on software bugs from BugZilla or similar bug tracking tools.

**Feature Extraction:** The system shall extract features from bug reports such as bug summary, description, severity, and bug assignee.

**Feature Weighting:** The system shall use meta-heuristic algorithms such as GA and PSO to weight the features based on their relevance to bug prioritization.

**Bug Prioritization:** The system shall prioritize bugs based on their severity and impact on the system using the weighted features.

**Visualization:** The system shall visualize the prioritized bugs in a graphical format such as a bar chart or scatter plot for easy understanding.

### 3.1.3. NON-FUNCTIONAL REQUIREMENTS

**Performance:** The system shall be able to process a large volume of bug reports and prioritize them within a reasonable time frame.

**Accuracy:** The system shall prioritize bugs accurately based on their severity and impact on the system.

**User Interface:** The system shall have an intuitive user interface that is easy to navigate and understand.

**Security:** The system shall ensure the confidentiality and integrity of the data collected and processed.

**Compatibility:** The system shall be compatible with different bug tracking tools and software development platforms.

### 3.1.4. SYSTEM ARCHITECTURE

The Bug Prioritization System shall consist of the following modules:

**Data Collection Module:** This module shall be responsible for collecting bug reports from BugZilla or similar bug tracking tools.

**Feature Extraction Module:** This module shall extract features from the bug reports such as bug summary, description, severity, and bug assignee.

**Feature Weighting Module:** This module shall use meta-heuristic algorithms such as GA and PSO to weight the features based on their relevance to bug prioritization.

**Bug Prioritization Module:** This module shall prioritize bugs based on their severity and impact on the system using the weighted features.

**Visualization Module:** This module shall visualize the prioritized bugs in a graphical format such as a bar chart or scatter plot for easy understanding.

### 3.1.5. SYSTEM CONSTRAINTS

**Hardware Requirements:** The system shall require a computer with sufficient processing power and memory to run the feature selection algorithms.

**Data Availability:** The system shall require a sufficient amount of bug report data for training and optimization.

### 3.1.6. ASSUMPTIONS AND DEPENDENCIES

**Bug Report Data:** The system assumes the availability of a sufficient amount of bug report data for training and optimization.

**Algorithm Availability:** The system depends on the availability of meta-heuristic feature selection algorithms, such as ACO, PSO, and GA.

## 3.2. SOLUTION APPROACH

Firstly, we have extracted various datasets i.e. Mozilla Core, Thunderbird, Eclipse and Firefox from Bugzilla. Then the whole workflow is divided into 4 phases as follows:

**PHASE-1:** The dataset we extracted was pre-processed by passing through various stages i.e. tokenization, Stop words Removal, punctuation Removal, Stemming and TF-IDF. Data pre-processing is a crucial step in natural language processing (NLP) tasks, including textual bug reports analysis. The primary goal of data pre-processing is to transform the unstructured and raw textual data into a structured format that can be used for analysis.

The following are the main steps involved in data pre-processing of textual bug reports using NLP:

Tokenization is the process of breaking down a text document into smaller units called tokens, which are essentially individual words. The goal of tokenization is to prepare the text for further analysis by separating it into its constituent parts. Tokenization is often used as the first step in NLP tasks such as sentiment analysis and topic modelling.

Stopwords are words that occur frequently in a text document but do not carry much meaning, such as "the", "a", and "an". Removing stopwords is a common technique used in text processing to improve the accuracy and efficiency of analysis. By removing stopwords, the remaining words in the text are more meaningful and provide better insights into the content of the document.

Punctuation marks are often irrelevant for text analysis, so removing them can also help to improve accuracy and efficiency. Punctuation removal involves eliminating all punctuation marks, such as commas, periods, and semicolons, from the text.

Stemming is the process of reducing words to their root form, or stem. This technique is often used to normalize words so that variations of the same word are treated as the same, e.g., "run", "running", and "ran" would all be reduced to "run". Stemming is useful for reducing the size of the vocabulary and improving the efficiency of text analysis.

TF-IDF stands for Term Frequency-Inverse Document Frequency, which is a commonly used technique in text processing to measure the importance of words in a document. It involves calculating

a score for each word based on its frequency in the document (term frequency) and its rarity across all documents (inverse document frequency). Words with higher TF-IDF scores are considered more important and are often used to represent the document in text analysis tasks such as clustering and classification.

**PHASE-2:** The normalized text we got from pre-processing was then used for Feature Extraction to make a Feature Set. Feature extraction involves selecting and extracting relevant information from textual bug reports, such as the severity of the bug, the description of the bug, the developer assigned to the bug, and other related metadata. The goal of feature extraction is to reduce the complexity of the data while retaining the most relevant and informative characteristics. The extracted features are then used to build a feature set, which is a structured representation of the data that can be used for further analysis. Once the features are extracted, they need to be organized into a feature set that can be used for analysis. The feature set typically includes a set of labeled examples, where each example consists of a set of features and a label indicating the label or category to which the example belongs. The feature set is then used to train machine learning models for bug triage, which can accurately classify new bug reports based on their features.

**PHASE-3:** Then the feature set was passed through the PSO, ACO and GA Algorithm to get a Reduced Feature Set. In this process, the feature set is passed through the PSO, ACO, or GA algorithm, which evaluates each feature subset based on a fitness function that measures the performance of the machine learning model. The algorithm then iteratively updates the feature subset based on the fitness function until it converges on the optimal subset of features.

The reduced feature set obtained through PSO, ACO, or GA is more efficient and effective than the original feature set for bug triage. It contains a smaller number of features that are most relevant and informative for classification, resulting in a more accurate and efficient machine learning model.

**PHASE-4:** Then this reduced feature set was used for prioritizing bugs for programmers.

Bug prioritization is the process of prioritizing bugs on the basis of severity so that when a programmer is assigned to the bug, then he/she first takes up the bug with a high severity and then low and so on. The goal of this framework is to improve the efficiency and effectiveness of the bug fixing process.

Bug prioritization typically involves analysing the characteristics of the bug, such as the intensity and severity. Machine learning algorithms are often used to analyse these characteristics and generate recommendations for the most suitable developer for the bug.

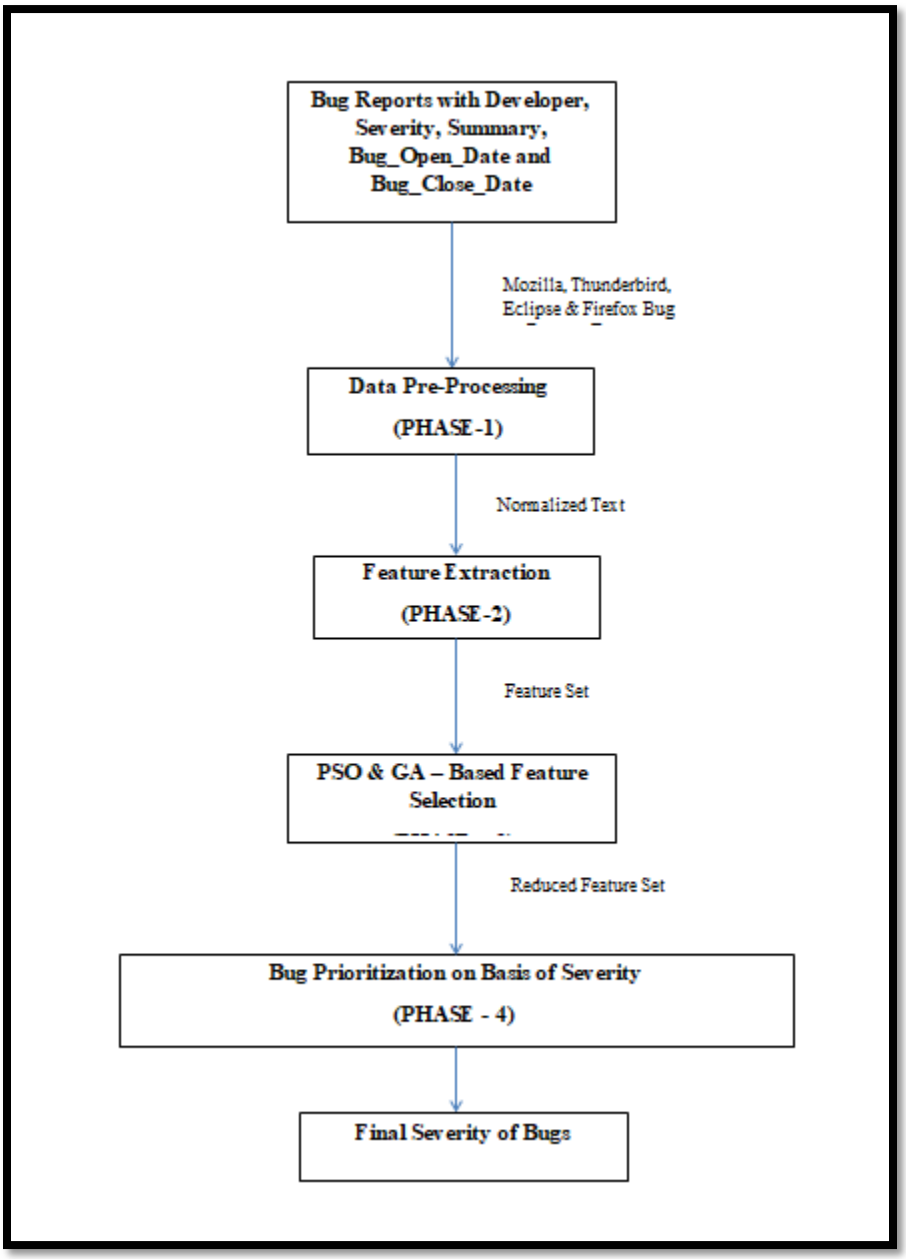



**Figure 1: Project Workflow**

# CHAPTER-4

# MODELLING AND IMPLEMENTATION DETAILS

## 4.1 DESIGN DIAGRAMS

### 4.1.1. USE CASE DIAGRAM

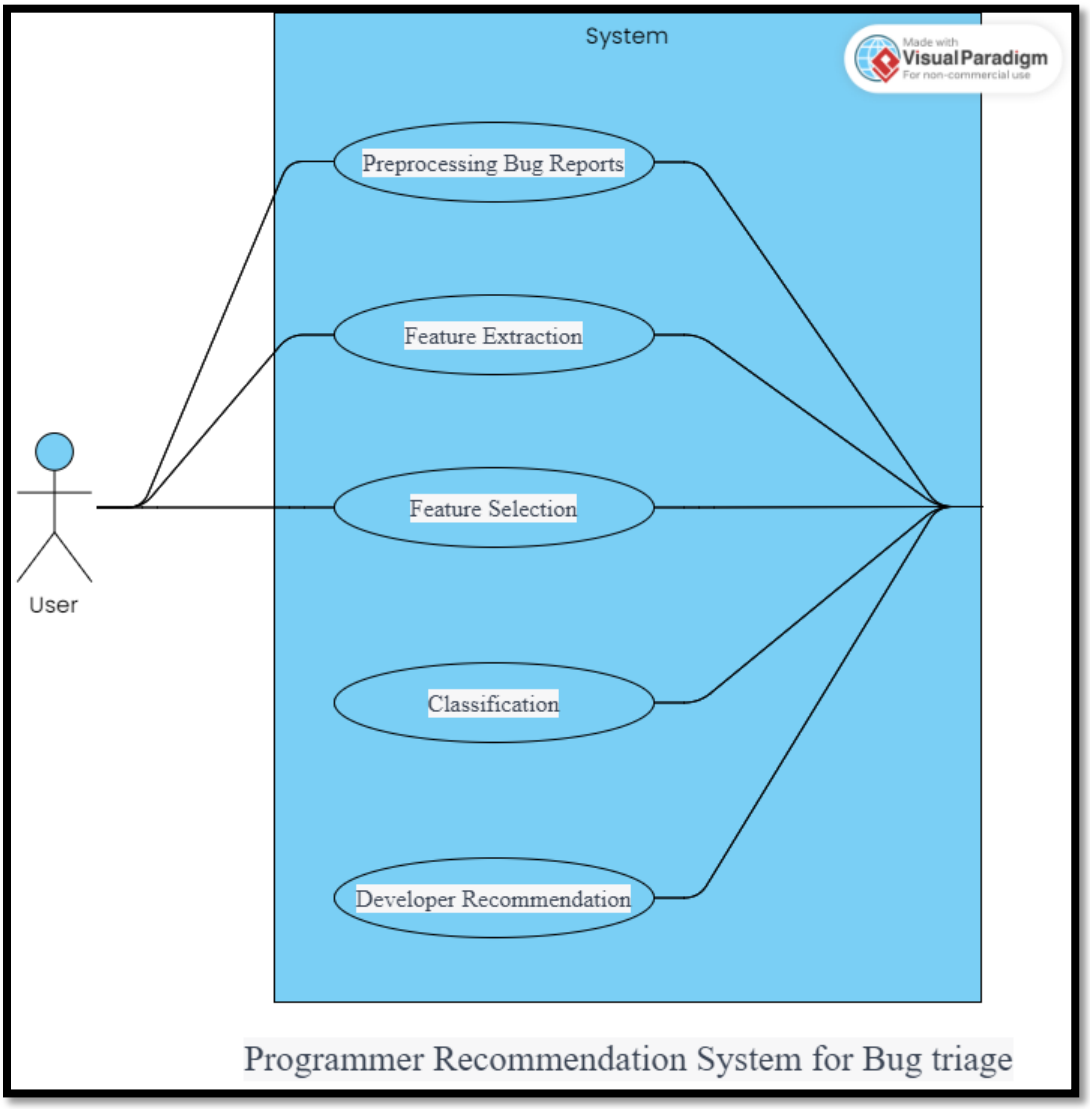



**Figure 2: Use Case Diagram**

### 4.1.2. CLASS DIAGRAM

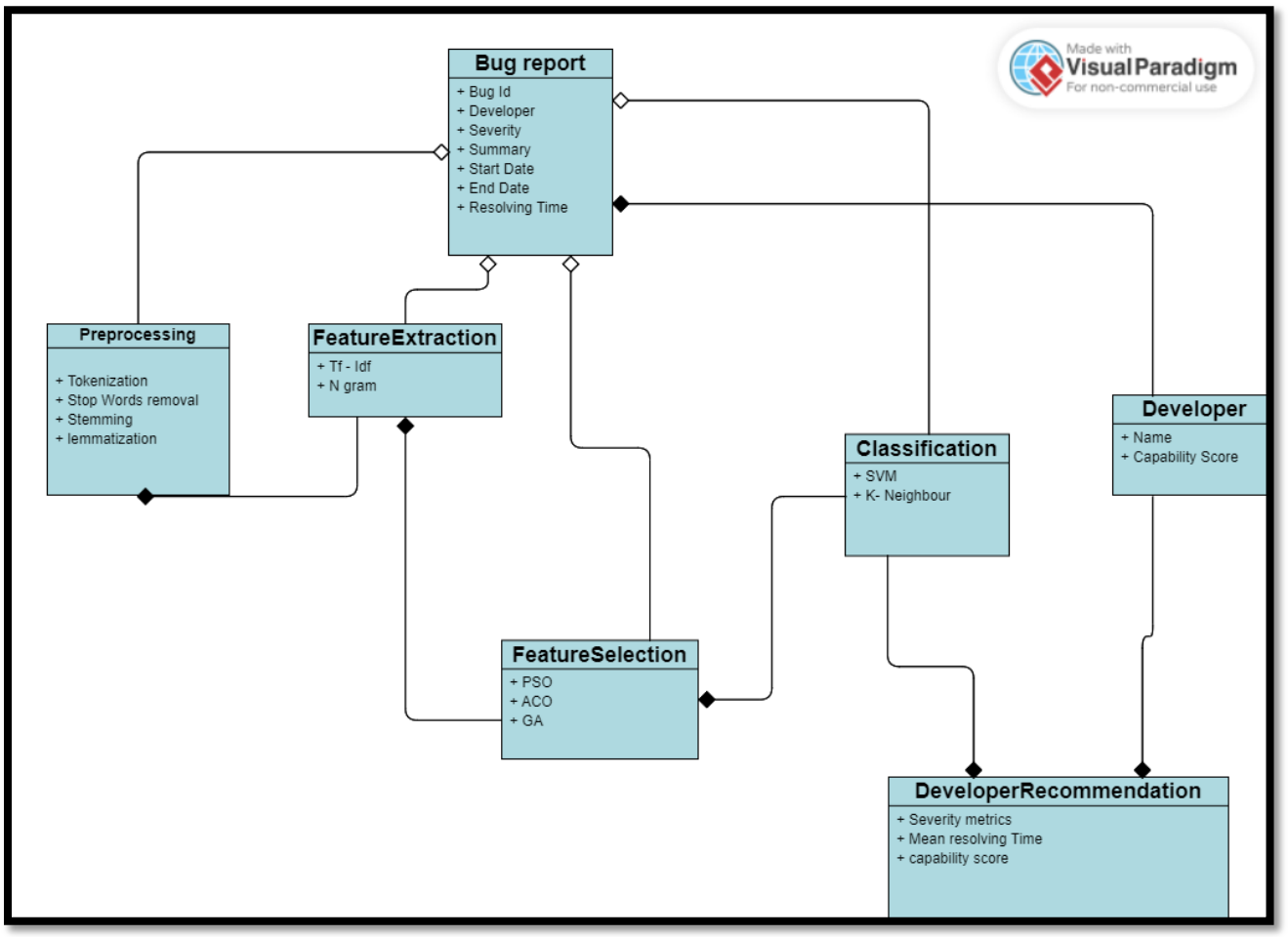



**Figure 3: Class Diagram**

### 4.1.3 ACTIVITY DIAGRAM

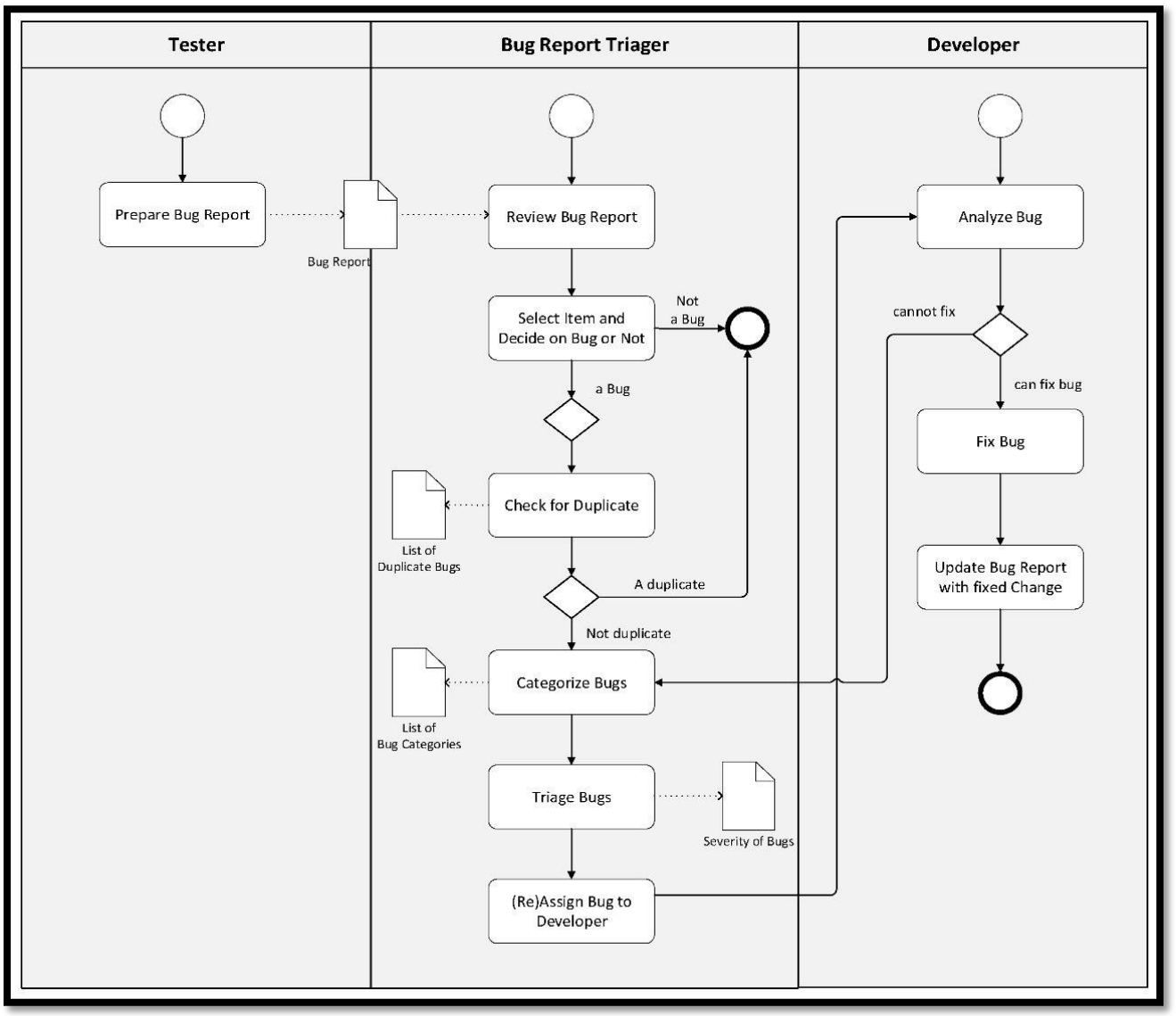
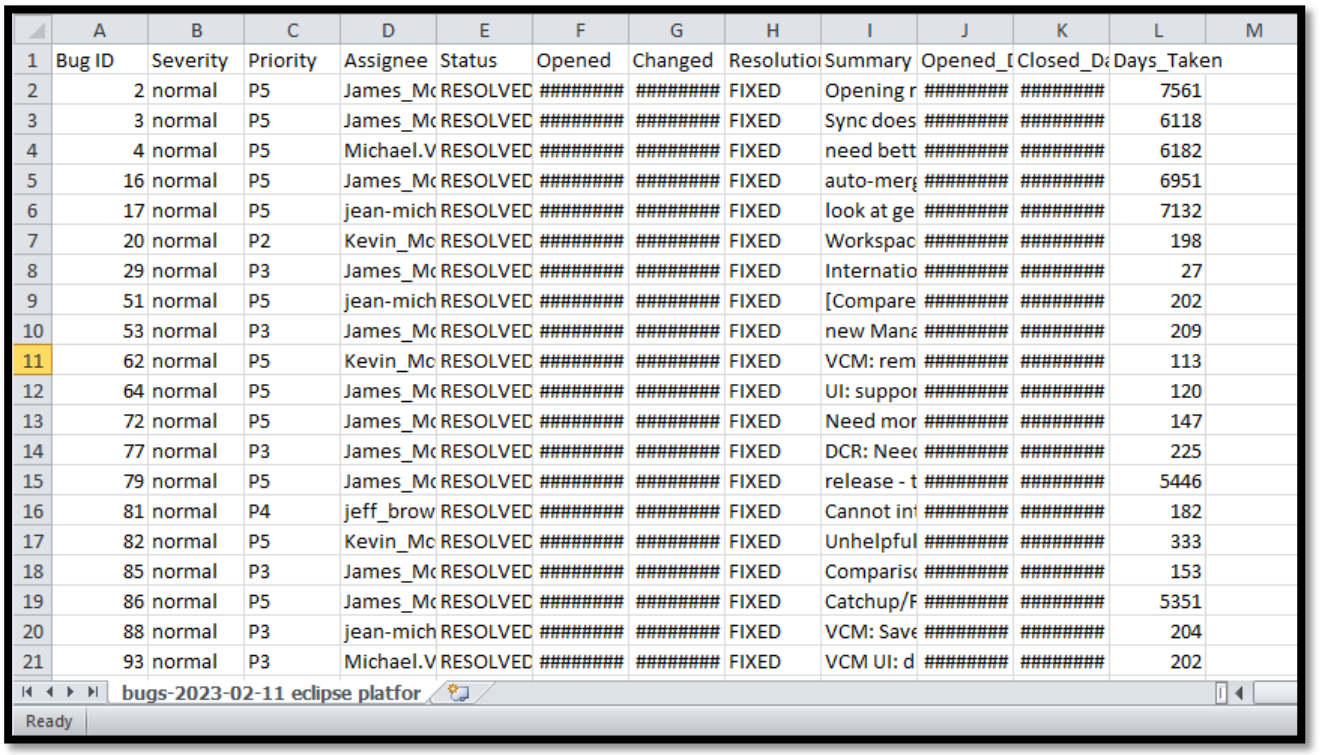



**Figure 4: Activity Diagram**

## 4.2. IMPLEMENTATION DETAILS

Firstly, we collected the Bug-Report datasets of four different platforms i.e. Mozilla Core, Firefox, Thunderbird and Eclipse from Bugzilla.

The Bug Reports consists of "BugID, Severity, Priority, Assignee, Status, Opened, Closed, Resolution, Summary and Total Days Taken"



| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Bug ID | Severity | Priority | Assignee | Status | Opened | Changed | Resolutio | Summary | Opened_[ | Closed_D | Days_Taken | |
| 2 | 2 | normal | P5 | James_Mc | RESOLVED | ######## | ######## | FIXED | Opening r | ######## | ######## | 7561 | |
| 3 | 3 | normal | P5 | James_Mc | RESOLVED | ######## | ######## | FIXED | Sync does | ######## | ######## | 6118 | |
| 4 | 4 | normal | P5 | Michael.V | RESOLVED | ######## | ######## | FIXED | need bett | ######## | ######## | 6182 | |
| 5 | 16 | normal | P5 | James_Mc | RESOLVED | ######## | ######## | FIXED | auto-merg | ######## | ######## | 6951 | |
| 6 | 17 | normal | P5 | jean-mich | RESOLVED | ######## | ######## | FIXED | look at ge | ######## | ######## | 7132 | |
| 7 | 20 | normal | P2 | Kevin_Mc | RESOLVED | ######## | ######## | FIXED | Workspac | ######## | ######## | 198 | |
| 8 | 29 | normal | P3 | James_Mc | RESOLVED | ######## | ######## | FIXED | Internatio | ######## | ######## | 27 | |
| 9 | 51 | normal | P5 | jean-mich | RESOLVED | ######## | ######## | FIXED | [Compare | ######## | ######## | 202 | |
| 10 | 53 | normal | P3 | James_Mc | RESOLVED | ######## | ######## | FIXED | new Mana | ######## | ######## | 209 | |
| 11 | 62 | normal | P5 | Kevin_Mc | RESOLVED | ######## | ######## | FIXED | VCM: rem | ######## | ######## | 113 | |
| 12 | 64 | normal | P5 | James_Mc | RESOLVED | ######## | ######## | FIXED | UI: suppor | ######## | ######## | 120 | |
| 13 | 72 | normal | P5 | James_Mc | RESOLVED | ######## | ######## | FIXED | Need mor | ######## | ######## | 147 | |
| 14 | 77 | normal | P3 | James_Mc | RESOLVED | ######## | ######## | FIXED | DCR: Neec | ######## | ######## | 225 | |
| 15 | 79 | normal | P5 | James_Mc | RESOLVED | ######## | ######## | FIXED | release - t | ######## | ######## | 5446 | |
| 16 | 81 | normal | P4 | jeff_brow | RESOLVED | ######## | ######## | FIXED | Cannot int | ######## | ######## | 182 | |
| 17 | 82 | normal | P5 | Kevin_Mc | RESOLVED | ######## | ######## | FIXED | Unhelpful | ######## | ######## | 333 | |
| 18 | 85 | normal | P3 | James_Mc | RESOLVED | ######## | ######## | FIXED | Comparisc | ######## | ######## | 153 | |
| 19 | 86 | normal | P5 | James_Mc | RESOLVED | ######## | ######## | FIXED | Catchup/F | ######## | ######## | 5351 | |
| 20 | 88 | normal | P3 | jean-mich | RESOLVED | ######## | ######## | FIXED | VCM: Save | ######## | ######## | 204 | |
| 21 | 93 | normal | P3 | Michael.V | RESOLVED | ######## | ######## | FIXED | VCM UI: d | ######## | ######## | 202 | |

bugs-2023-02-11 eclipse platfor

Ready

**Figure 5: Eclipse Platform Dataset**

We have imported the dataset first and then applied various Pre-Processing Techniques as follows:

Firstly, we dropped the un-necessary columns from each Bug-Report Dataset.

**Dropping the Un-necessary Columns**

```
In [4]: df = df.drop(['Opened'], axis = 1)
        df = df.drop(['Closed'], axis = 1)
        df = df.drop(['Priority'], axis = 1)
        df.head()
```

Out[4]:

| | Bug ID | Type | Assignee | Severity | Status | Resolution | Summary | Opened_Date | Closed_Date | Days_Taken |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 35689 | defect | jporterbugs | minor | RESOLVED | FIXED | Newsgroup postings don't linkify newsgroup hea... | 12-04-2000 | 17-09-2009 | 3445 |
| 1 | 36489 | enhancement | neil | normal | RESOLVED | FIXED | Combined To/From column in thread pane (aka "C... | 20-04-2000 | 24-03-2015 | 5451 |
| 2 | 58140 | enhancement | mkmelin+mozilla | normal | RESOLVED | FIXED | save multiple messages as individual files in ... | 26-10-2000 | 29-11-2009 | 3321 |
| 3 | 67219 | defect | mkmelin+mozilla | trivial | RESOLVED | FIXED | Disable filters menu item when no accounts are... | 31-01-2001 | 01-08-2009 | 3104 |
| 4 | 68784 | enhancement | acelists | normal | RESOLVED | FIXED | when sending mail, should first check "no reci... | 14-02-2001 | 16-04-2013 | 4444 |

**Figure 6: Dropping Unnecessary Columns**

Now, we have applied different pre-processing techniques to the “Summary” Column of each dataset as follows:

**1. Tokenization**

```
In [7]: def tokenize(column):

            tokens = nltk.word_tokenize(column)
            return [w for w in tokens if w.isalpha()]
```

```
In [8]: df['Tokenized'] = df.apply(lambda x: tokenize(x['Summary']), axis=1)
        df
```

Out[8]:

| | Bug ID | Type | Assignee | Severity | Status | Resolution | Summary | Opened_Date | Closed_Date | Days_Taken | Tokenized |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 35689 | defect | jporterbugs | minor | RESOLVED | FIXED | Newsgroup postings don't linkify newsgroup hea... | 12-04-2000 | 17-09-2009 | 3445 | [Newsgroup, postings, do, linkify, newsgroup, ... |
| 1 | 36489 | enhancement | neil | normal | RESOLVED | FIXED | Combined To/From column in thread pane (aka "C... | 20-04-2000 | 24-03-2015 | 5451 | [Combined, column, in, thread, pane, aka, Corr... |
| 2 | 58140 | enhancement | mkmelin+mozilla | normal | RESOLVED | FIXED | save multiple messages as individual files in ... | 26-10-2000 | 29-11-2009 | 3321 | [save, multiple, messages, as, individual, fil... |
| 3 | 67219 | defect | mkmelin+mozilla | trivial | RESOLVED | FIXED | Disable filters menu item when no accounts are... | 31-01-2001 | 01-08-2009 | 3104 | [Disable, filters, menu, item, when, no, accou... |

**Figure 7: Tokenization**

## 2. Converting the Summary into Lowercase

```
In [9]: df['String_Tokenized'] = [' '.join(map(str, l)) for l in df['Tokenized']]
        df['String_Tokenized'] = df['String_Tokenized'].apply(str.lower)
        df['String_Tokenized']

Out[9]: 0       newsgroup postings do linkify newsgroup header...
        1       combined column in thread pane aka corresponde...
        2       save multiple messages as individual files in ...
        3       disable filters menu item when no accounts are...
        4       when sending mail should first check no recipi...
                                      ...
        9995    when two or more messages are in the message p...
        9996    contains wrong filename if attachment was rena...
        9997    remove incoming mail default text encoding option
        9998                                remove preprocessing in
        9999                        could be used while undefined in
        Name: String_Tokenized, Length: 10000, dtype: object
```

**Figure 8: Converting into Lowercase**

## 3. Removing the Stopwords from the Summary

```
In [10]: stop_words = stopwords.words('english')
         df['String_Tokenized'] = df['String_Tokenized'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]
         df['String_Tokenized']

Out[10]: 0        newsgroup postings linkify newsgroup header like
         1       combined column thread pane aka correspondents...
         2       save multiple messages individual files directory
         3                  disable filters menu item accounts set
         4       sending mail first check recipient subject cur...
                                      ...
         9995    two messages message pane one messages highlig...
         9996    contains wrong filename attachment renamed for...
         9997    remove incoming mail default text encoding option
         9998                                 remove preprocessing
         9999                                 could used undefined
         Name: String_Tokenized, Length: 10000, dtype: object
```

**Figure 9: Removing Stopwords**

**4. Removing the Punctuations**

```
In [11]: df["Summary_wo_Punctuation"] = df['String_Tokenized'].str.replace('[^\w\s]','')
         df.head()
```

Out[11]:

| | Bug ID | Type | Assignee | Severity | Status | Resolution | Summary | Opened_Date | Closed_Date | Days_Taken | Tokenized | String_Tokenized | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 35689 | defect | jporterbugs | minor | RESOLVED | FIXED | Newsgroup postings don't linkify newsgroup hea... | 12-04-2000 | 17-09-2009 | 3445 | [Newsgroup, postings, do, linkify, newsgroup, ... | newsgroup postings linkify newsgroup header like | |
| 1 | 36489 | enhancement | neil | normal | RESOLVED | FIXED | Combined To/From column in thread pane (aka "C... | 20-04-2000 | 24-03-2015 | 5451 | [Combined, column, in, thread, pane, aka, Corr... | combined column thread pane aka correspondents... | |
| 2 | 58140 | enhancement | mkmelin+mozilla | normal | RESOLVED | FIXED | save multiple messages as individual files in ... | 26-10-2000 | 29-11-2009 | 3321 | [save, multiple, messages, as, individual, fil... | save multiple messages individual files directory | |
| 3 | 67219 | defect | mkmelin+mozilla | trivial | RESOLVED | FIXED | Disable filters menu item when no accounts are... | 31-01-2001 | 01-08-2009 | 3104 | [Disable, filters, menu, item, when, no, accou... | disable filters menu item accounts set | |

**Figure 10: Removing Punctuations**

**5. Removing ' " ' with blank space ' '**

```
In [12]: df['Summary_wo_Punctuation'] = df['Summary_wo_Punctuation'].apply(lambda x: x.replace('"', ''))
         df['Summary_wo_Punctuation']
```

```
Out[12]: 0        newsgroup postings linkify newsgroup header like
         1       combined column thread pane aka correspondents...
         2       save multiple messages individual files directory
         3                 disable filters menu item accounts set
         4       sending mail first check recipient subject cur...
                                    ...
         9995    two messages message pane one messages highlig...
         9996    contains wrong filename attachment renamed for...
         9997    remove incoming mail default text encoding option
         9998                                remove preprocessing
         9999                                could used undefined
         Name: Summary_wo_Punctuation, Length: 10000, dtype: object
```

**Figure 11: Changing ' " ' with blank space ' '**

### 6. Stemming the Summary Column

In [13]:
```
# Use English stemmer.
stemmer = SnowballStemmer("english")
df['Summary_wo_Punctuation'] = df['Summary_wo_Punctuation'].str.split()
df['Stemmed'] = df['Summary_wo_Punctuation'].apply(lambda x: [stemmer.stem(y) for y in x])
```

### 7. Converting the Stemmed column into string type

In [14]:
```
df['ListString'] = [' '.join(map(str, l)) for l in df['Stemmed']]
df
```

Out[14]:

| | Bug ID | Type | Assignee | Severity | Status | Resolution | Summary | Opened_Date | Closed_Date | Days_Taken | Tokenized |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 35689 | defect | jporterbugs | minor | RESOLVED | FIXED | Newsgroup postings don't linkify newsgroup hea... | 12-04-2000 | 17-09-2009 | 3445 | [Newsgroup, postings, do, linkify, newsgroup, ... |
| 1 | 36489 | enhancement | neil | normal | RESOLVED | FIXED | Combined To/From column in thread pane (aka "C... | 20-04-2000 | 24-03-2015 | 5451 | [Combined, column, in, thread, pane, aka, Corr... |
| 2 | 58140 | enhancement | mkmelin+mozilla | normal | RESOLVED | FIXED | save multiple messages as individual files in ... | 26-10-2000 | 29-11-2009 | 3321 | [save, multiple, messages, as, individual, fil... |
| | | | | | | | Disable filters menu | | | | [Disable, filters, menu |

**Figure 12: Stemming**

### 8. Applying TF-IDF using SkLearn

In [16]:
```
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(data)
```

In [17]:
```
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names(), tfidf.idf_):
    print(ele1, ':', ele2)
```

...

In [18]:
```
print('\nWord indexes:')
print(tfidf.vocabulary_)
```

...

In [19]:
```
# display tf-idf values
print('\ntf-idf value:')
print(result)
```

```
tf-idf value:
  (0, 1979)     0.3174764852739266
  (0, 1597)     0.2502890315467414
  (0, 1988)     0.5018475957064273
  (0, 2934)     0.38598777613745294
  (0, 2387)     0.660095833452727
  (1, 746)      0.4029965537432351
  (1, 102)      0.4029965537432351
  (1, 2800)     0.23264034233383535
```

**Figure 13: Applying TF-IDF**

Then, we applied PSO and Genetic Algorithm along with KNN (K-Nearest Neighbours) algorithm on each dataset i.e. Mozilla Core, Firefox, Thunderbird and Eclipse Platform.

**Particle Swarm Optimization (PSO):**

Particle Swarm Optimization (PSO) is a computational optimization technique inspired by the social behavior of bird flocks and fish schools. It is a population-based meta-heuristic algorithm that optimizes a problem by iteratively improving a population of candidate solutions.

In PSO, a group of particles, each representing a candidate solution, moves through the search space. The movement of each particle is guided by its own experience and the experience of its neighbors. The position and velocity of each particle are updated based on its current position, velocity, and the best solution found so far by itself and its neighbors.

PSO has been widely used in many different fields such as engineering, finance, image processing, data mining, and machine learning. It can be used to solve various optimization problems including function optimization, feature selection, parameter tuning, and clustering. PSO has proven to be effective in solving complex optimization problems, especially when the search space is high-dimensional and non-linear.

PSO can be used for feature reduction of textual features from bug reports. The main idea behind feature reduction is to identify a subset of features that are most relevant to the problem being solved while eliminating irrelevant or redundant features. This can improve the performance of a machine learning model by reducing over-fitting and increasing its generalization ability.

In the context of bug reports, textual features can be extracted from the description, title, comments, and other relevant fields of the report. These features can include words, phrases, or other patterns that are indicative of the type, severity, or other characteristics of the bug. However, the high dimensionality and sparsity of the textual features can make it challenging to analyze and model them effectively.

PSO can be used to select a subset of relevant features by optimizing an objective function that balances the classification performance of the model with the number of selected features. The objective function can be defined using various criteria such as information gain, mutual information, or correlation-based feature selection. PSO can search for the optimal subset of features by adjusting the position and velocity of particles in the feature space. Overall, PSO-based feature reduction can be

a useful technique for improving the performance and efficiency of bug report analysis and classification.

Particle swarm optimization (PSO) is a meta-heuristic algorithm inspired by the social behavior of birds flocking or fish schooling. The following steps can be used as a solution approach of PSO for bug prioritization:

1. Define the problem: The first step is to define the bug prioritization problem, including the objective function, constraints, and decision variables. The objective function should measure the criticality of bugs based on various factors such as severity, frequency, and impact on the system.

2. Represent the bugs as particles: Each bug can be represented as a particle, where the position of the particle represents the bug's criticality. The position of the particle can be represented using a vector of real numbers.

3. Initialize the particles: A population of particles is randomly generated, and each particle is evaluated using the objective function.

4. Set the velocity and acceleration: The velocity and acceleration of each particle are initialized to random values.

5. Update the particle position and velocity: The position and velocity of each particle are updated using the following equations:

    - new_velocity = w * old_velocity + c1 * r1 * (pbest_position - current_position) + c2 * r2 * (gbest_position - current_position)

    - new_position = current_position + new_velocity, where w is the inertia weight, c1 and c2 are the acceleration coefficients, r1 and r2 are random numbers, pbest_position is the personal best position of the particle, and gbest_position is the global best position of all particles in the swarm.

6. Evaluate the particle position: The fitness value of the particle is calculated using the objective function.

7. Update the personal best position: The personal best position of the particle is updated if the fitness value is better than the previous best position.

8. Update the global best position: The global best position of all particles in the swarm is updated if a particle's personal best position is better than the current global best position.

9. Repeat steps 5-8: Steps 5-8 are repeated until the stopping criteria are met. The stopping criteria can be a maximum number of iterations, a maximum fitness value, or a minimum improvement in the fitness value.

10. Select the best solution: The global best position of the swarm represents the optimal solution, which provides the most critical bugs that need to be fixed first.

In conclusion, the PSO approach for bug prioritization involves representing bugs as particles, initializing the particles, updating the position and velocity of the particles using the PSO equations, evaluating the particle position, updating the personal best and global best positions, and repeating the process until the optimal solution is found. The PSO approach can handle large datasets of bugs and provide an automated and efficient bug prioritization solution.

**Genetic Algorithm (GA):**

Genetic algorithm (GA) is a meta-heuristic optimization algorithm that is inspired by the process of natural selection and genetics. GA works by generating a population of candidate solutions and iteratively improving them using a combination of selection, crossover, and mutation operators. The selection operator chooses the best solutions from the current population to serve as parents for the next generation, while the crossover operator combines the genetic material of two parents to create new offspring. The mutation operator introduces random changes to the genetic material of the offspring to promote diversity in the population.

GA has been widely used in feature reduction, which involves selecting a subset of the most relevant features from a large dataset. Feature reduction can improve the performance and efficiency of machine learning models by reducing the dimensionality of the dataset and removing redundant or irrelevant features. GA can be used to search for the optimal subset of features by encoding each subset as a binary string and treating it as an individual in the population. The fitness function is evaluated using a machine learning model that measures the performance of the subset of features. The GA operators are then used to create new generations of feature subsets, and the process continues until the optimal subset is found. GA has been shown to be effective in feature reduction, and it can handle large datasets with a large number of features.

Genetic algorithm is a meta-heuristic algorithm that mimics the process of natural selection and evolution to solve optimization problems. The following steps can be used as a solution approach of genetic algorithm for bug prioritization:

1. Define the problem: The first step is to define the bug prioritization problem, including the objective function, constraints, and decision variables. The objective function should measure the criticality of bugs based on various factors such as severity, frequency, and impact on the system.

2. Represent the bugs as chromosomes: Each bug can be represented as a chromosome, where the genes represent the various factors that determine the bug's criticality. The chromosomes can be encoded using binary or real-valued representation.

3. Initialize the population: A population of chromosomes is randomly generated, and each chromosome is evaluated using the objective function.

4. Selection: The selection process involves selecting the fittest chromosomes from the population based on their fitness value. The fitness value is calculated based on the objective function.

5. Crossover: The crossover operator involves selecting two parent chromosomes and generating new offspring chromosomes by combining their genes. The crossover operator helps to create new solutions that may be better than the original ones.

6. Mutation: The mutation operator involves randomly changing a gene in a chromosome. The mutation operator helps to introduce new genetic diversity in the population.

7. Evaluate the offspring: The fitness value of the offspring chromosomes is calculated using the objective function.

8. Replace the population: The offspring chromosomes are inserted into the population, and the least fit chromosomes are removed. This process helps to maintain the diversity of the population and prevent premature convergence.

9. Repeat steps 4-8: Steps 4-8 are repeated until the stopping criteria are met. The stopping criteria can be a maximum number of iterations, a maximum fitness value, or a minimum improvement in the fitness value.

10. Select the best solution: The final population contains the fittest chromosomes, and the best

solution is selected based on the fitness value.

In conclusion, the genetic algorithm approach for bug prioritization involves representing bugs as chromosomes, initializing the population, applying selection, crossover, and mutation operators, evaluating the offspring, and repeating the process until the optimal solution is found. The genetic algorithm approach can handle large datasets of bugs and provide an automated and efficient bug prioritization solution

**K-Nearest Neighbour (KNN):**

K-nearest neighbours (KNN) is a simple and widely used classification and regression algorithm in machine learning. The algorithm works by assigning a new data point to the class or value of its nearest neighbours in the training set. In classification, the KNN algorithm works by calculating the distance between the new data point and each training example in the feature space and then only K nearest neighbours is selected based on the shortest distance, and the most common class label among these neighbours is assigned to the new data point. The value of K is a hyper-parameter that determines the number of neighbours to consider. In the case of regression, the KNN algorithm works by averaging the values of the K nearest neighbours in the training set to predict the value of the new data point.

KNN has several advantages, such as simplicity, non-parametric nature, and flexibility in handling different types of data. However, it also has some limitations, such as being sensitive to the choice of distance metric, the curse of dimensionality, and the lack of interpretability. KNN can be applied to various domains such as image recognition, text classification, and recommendation systems. Overall, KNN is a useful algorithm for many real-world applications and can be a good starting point for many machine learning tasks.

The K-nearest neighbors (KNN) algorithm is a machine learning algorithm that can be used for bug prioritization. The following steps can be used as a solution approach of KNN for bug prioritization:

1. Data Preparation: The first step is to prepare the bug data for training the KNN algorithm. The bug data should be labeled with their corresponding criticality values. The bug data should also be split into training and testing datasets.

2. Define the distance metric: The KNN algorithm uses the distance metric to measure the similarity between bugs. The distance metric can be Euclidean, Manhattan, or Minkowski distance, depending on the data type.

3. Choose the value of K: The K value represents the number of nearest neighbors that will be considered to predict the criticality of a new bug. The value of K should be chosen based on the dataset size and the complexity of the problem.

4. Train the KNN model: The KNN model is trained on the training dataset by calculating the distance between each bug in the training dataset and the new bug. The K nearest neighbors is selected based on the distance metric and the value of K. The criticality of the new bug is predicted based on the majority vote of the K nearest neighbors.

5. Evaluate the model: The KNN model is evaluated on the testing dataset to measure its accuracy and performance. The evaluation metrics can be precision, recall, F1 score, or accuracy.

6. Predict the criticality of new bugs: Once the KNN model is trained and evaluated, it can be used to predict the criticality of new bugs. The KNN model calculates the distance between the new bug and the bugs in the training dataset and selects the K nearest neighbors. The criticality of the new bug is predicted based on the majority vote of the K nearest neighbors.

In conclusion, the KNN approach for bug prioritization involves preparing the data, choosing the distance metric and the value of K, training and evaluating the KNN model, and predicting the criticality of new bugs. The KNN approach is a simple and interpretable algorithm that can be used for bug prioritization in small to medium-sized datasets. However, it may not be suitable for large and complex datasets due to its high computational complexity.

After applying the above mentioned meta-heuristics algorithms, we got the following reduced feature size for each dataset i.e. Eclipse, Firefox, Thunderbird and Mozilla Core:

**Table 1: Specific details of logs (Bug reports) taken into account for the experiment**

| Dataset | No. of bugs | BTS | No. Of features | Reduced Feature Size |
|---|---|---|---|---|
| Eclipse | 1000 | Bugzilla | 8463 | 4159 |
| Firefox | 1000 | Bugzilla | 4587 | 2254 |
| ThunderBird | 1000 | Bugzilla | 4358 | 2114 |
| Mozilla | 1000 | Bugzilla | 5573 | 2716 |

# CHAPTER-5

# TESTING (FOCUS ON QUALITY OF ROBUSTNESS AND TESTING)

## 5.1. TESTING PLAN

**Table 2: Testing Plan**

| Type of Test | Will Test Be Performed? | Comments/Explanations | Software Component |
|---|---|---|---|
| Requirements Testing | Yes | Requirement testing can be useful in a programmer recommendation system for bug triage because it can help ensure that the recommended programmers have the necessary skills and expertise to address the specific requirements of the bug being reported.<br>In bug triage, it's important to identify the severity of the bug, prioritize it, and assign it to the most suitable programmer for resolution. Requirement testing can assist in this process by evaluating the specific requirements and constraints of the bug, and then recommending programmers who have experience and expertise in those areas. | Database: Testing the database can help ensure that the system can store and retrieve data accurately and efficiently.<br><br>Recommendation algorithm: Testing the recommendation algorithm can help ensure that the system is making accurate and relevant recommendations to users.<br><br>Integration with bug tracking system: Testing the integration with the bug tracking system can help ensure that bugs are being identified and triaged correctly, and that recommended programmers are being assigned to the correct bugs. |

| | | | |
|---|---|---|---|
| | | | Performance: Testing the performance of the system can help ensure that it can handle a large number of requests and users without slowing down or crashing. |
| Unit | Yes | Unit testing can be an important tool for ensuring the quality and reliability of a programmer recommendation system for bug triage. By catching bugs earlier in the development process and facilitating continuous integration and deployment, unit testing can help ensure that the system is functioning as intended and delivering value to users. | Recommendation algorithm: Testing the recommendation algorithm can help ensure that the system is making accurate and relevant recommendations to users.<br><br>Database access layer: Testing the database access layer can help ensure that data is being stored and retrieved correctly. |
| Integration | Yes | Integration testing can help ensure that when a bug is reported, it is correctly identified, prioritized, and assigned to the most suitable programmer based on the recommendation algorithm. Integration testing can also help ensure that the programmer recommendation system is correctly integrated with the bug tracking system, and that data is being exchanged between the two systems correctly. | Bug tracking system integration: Testing the integration between the bug tracking system and the programmer recommendation system can help ensure that data is being exchanged correctly between the two systems and that bugs are being assigned to the correct programmers. |

| | | | |
|---|---|---|---|
| | | | Database integration: Testing the integration between the database and the other components of the system can help ensure that data is being stored and retrieved correctly, and that the system is able to scale as the amount of data grows. |
| Performance | Yes | Performance testing can be used to test the system's ability to handle a large number of bug reports and users simultaneously. Performance testing can help identify bottlenecks or areas of the system that are prone to slowdowns or crashes under heavy load, and can help ensure that the system is able to meet the performance requirements of users. | Recommendation algorithm: Testing the performance of the recommendation algorithm can help ensure that the algorithm is able to quickly process and analyze bug reports to provide recommendations to programmers.<br><br>Database: Testing the performance of the database can help ensure that it is able to handle the expected volume of data and that it can retrieve and update data quickly and efficiently. |
| Stress | No | Stress testing may not always be necessary or appropriate for a programmer recommendation system for bug triage, it is still important to ensure that the system is able to handle the expected workload and perform efficiently under normal usage conditions. This can be achieved | |

| | | | |
|---|---|---|---|
| | | through other types of performance testing, such as load testing and scalability testing, as well as through ongoing monitoring and maintenance of the system. | |
| Compliance | No | Compliance testing is not a one-time event, but an ongoing process. Compliance requirements can change over time, and it's essential to keep up with changes and ensure that the system remains compliant. | |
| Security | Yes | Security testing is important in a programmer recommendation system for bug triage because it can help identify potential vulnerabilities and ensure that the system is able to protect sensitive data from unauthorized access or attack. | Data storage and transmission: Testing the security of the system's data storage and transmission mechanisms can help ensure that user data is protected from unauthorized access or attack, both when it is stored and when it is transmitted between components of the system.<br><br>Third-party components: Testing the security of third-party components used in the system, such as libraries or APIs, can help ensure that these components do not introduce security vulnerabilities into the system. |
| Load | No | In our project, load testing can not be performed because the system only has a small number of users and requests. In such cases, the system may be able to | |

| | | | |
|---|---|---|---|
| | | handle the expected load without any performance issues, and load testing may not provide any meaningful insights. | |
| Volume | No | In a programmer recommendation system for bug triage, volume testing can help determine whether the system can handle large volumes of bug reports and recommendations. This is because our project has limited data and limited scope, and may not require the system to handle large volumes of data. | |

**Table 3: Software & Hardware Items**

| **TEST ENVIRONMENT-** |
|---|
| **SOFTWARE ITEMS** |
| ACO , PSO , GA algorithm implementation (e.g., Ant System, Ant Colony Optimization)<br>Programming language support (e.g., Python)<br>Test data generation tools (Machine learning algorithms and Bugzilla datasets)<br>Bug tracking and reporting system (e.g., Bugzilla, JIRA)<br>A text editor or integrated development environment (IDE), such as PyCharm, Visual Studio Code, or Jupyter Notebook.<br>An operating system, such as Windows, MacOS, or Linux. |
| **HARDWARE ITEMS** |
| A desktop or laptop computer with a modern CPU and at least 8 GB of RAM<br>A GPU with sufficient processing power to execute the PSO , GA algorithm (optional)<br>Sufficient disk space to store the algorithm implementation and test data |

## 5.2. COMPONENT DECOMPOSITION AND TYPE OF TESTING REQUIRED

**Table 4: Component Decomposition and Identification of Tests required**

| S.No | List of Various Components (modules) that require testing | Type of Testing Required | Technique for writing test cases |
|---|---|---|---|
| 1 | Data Collection module | Unit testing: Test individual components of the module to ensure they are working as expected.<br>Integration testing: Test the integration of the module with other modules to ensure it is collecting data correctly. | Test different types of input data to ensure the module can collect data from various sources and formats.<br><br>Test for edge cases, such as empty or invalid input, to ensure the module can handle such scenarios. |
| 2 | Data preprocessing module | Unit testing: Test individual components of the module to ensure they are working as expected.<br>Integration testing: Test the integration of the module with other modules to ensure it is preprocessing data correctly.<br>Performance testing: Test the efficiency of the module to ensure it is processing data quickly. | Black Box testing:<br><br>Equivalence class testing: Test the module with different types of valid and invalid input data, such as different data types and sizes.<br><br>Cause-effect testing: Test the module with inputs that exercise different causes, such as different types of data or missing values. |
| 3 | Feature extraction module | Unit testing: Test individual components of the module to ensure they are working as expected.<br>Integration testing: Test the integration of the module with other modules to ensure it is extracting features correctly.<br>Accuracy testing: Test the accuracy of the module to ensure it is extracting relevant features. | Black Box testing:<br><br>Equivalence class testing: Test the module with different types of valid and invalid input data, such as different data types and sizes.<br>Boundary value testing: Test the module with input data that is on the boundaries of the input domain, such as empty or large data sets. |

| | | | |
|---|---|---|---|
| 4 | Recommendation module | Recommendation module:<br>Unit testing: Test individual components of the module to ensure they are working as expected.<br>Integration testing: Test the integration of the module with other modules to ensure it is recommending appropriate programmers. | Test different types of input data to ensure the module can recommend appropriate programmers for various types of bugs.<br>Test for edge cases, such as empty or invalid input, to ensure the module can handle such scenarios. |
| 5 | Integration module | Unit testing: Test individual components of the module to ensure they are working as expected.<br>Integration testing: Test the integration of the module with other modules to ensure they are working together correctly. | Test the integration of different modules to ensure they are working together correctly.<br>Test for edge cases, such as empty or invalid input, to ensure the module can handle such scenarios.<br>Test the compatibility of the module with different systems and configurations |

## 5.3. ERROR AND EXCEPTION HANDLING

**Table 5: Debugging Techniques**

| Test case id | Test case for components | Debugging Technique |
|---|---|---|
| 2 | Data preprocessing module | Code review: This involves having other developers review the code for potential errors and suggest improvements. This technique can help identify errors and improve the quality of the code.<br>Backtracking: This involves stepping back through the code to identify the point at which the error first occurred. This technique can help identify the root cause of the error and fix it. |

| | | |
|---|---|---|
| 3 | Feature extraction module | Print (or tracing) debugging: This involves adding print statements in the code to trace the flow of execution and inspect the values of variables at various stages. This technique can help identify the cause of the error and the location of the bug.<br>Backtracking: This involves stepping back through the code to identify the point at which the error first occurred. This technique can help identify the root cause of the error and fix it. |
| 5 | Integration module | Post-mortem debugging: This involves examining the state of the system after a failure has occurred. This technique can help identify the cause of the error and provide insights for preventing similar errors in the future.<br>Delta Debugging: This involves systematically removing parts of the code to isolate the source of the error. This technique can help identify the exact line or lines of code that are causing the error. |

## 5.4. LIMITATIONS OF THE SOLUTION

There are several limitations to using meta-heuristic feature selection techniques such as ACO, PSO, and GA for programmer recommendation systems for bug triage. Here are a few:

**Computational Complexity:** Meta-heuristic feature selection techniques require a large number of iterations to converge to an optimal solution. This can be computationally expensive and may not be feasible for large software projects.

**Feature Selection Bias:** The effectiveness of meta-heuristic feature selection techniques depends on the quality and relevance of the features selected. If the initial set of features is not representative of the entire system, the algorithm may converge to a suboptimal solution.

**Lack of Interpretability:** Meta-heuristic feature selection techniques are often viewed as "black box" models, meaning that it can be difficult to understand how the algorithm is making decisions. This can be a problem when trying to explain recommendations to stakeholders or debugging the system.

**Over-fitting:** There is a risk of over-fitting when using meta-heuristic feature selection techniques, which can result in recommendations that are too specific to the training data and may not generalize well to new data.

**Ethical Concerns:** There may be ethical concerns around the use of a recommendation system in bug triage, such as the potential for bias or discrimination against certain groups of developers or users.

**Lack of Context:** The system may not have access to all of the contextual information needed to accurately triage a bug. For example, it may not have access to information about the user's environment or other software components that interact with the affected code.

# CHAPTER-6

# FINDINGS, CONCLUSION, AND FUTURE WORK

## 6.1. FINDINGS

The proposed bug prioritization framework was developed and implemented using python coding. To evaluate its efficiency, the framework was tested on four bug report datasets: Thunderbird, Firefox, Eclipse, and Mozilla. Severity of bugs was determined using various parameters including precision, recall, accuracy, and F-measure. An experiment was conducted to evaluate the impact and severity of the bugs, so that a programmer can prioritize the bug for fixing according to its severity.

**Experiment 1's Results and discussion**

**Accuracy:** The proportion of correct predictions among all the predictions made.

**Formula:** (True Positives + True Negatives) / (True Positives + False Positives + True Negatives + False Negatives)

The following key parameters are used to measure the performance of the proposed model:

**Precision:** The proportion of true positive predictions among all the positive predictions made. It measures how well the model predicts true positives without false positives.

**Formula:** True Positives / (True Positives + False Positives)

**Recall:** The proportion of true positive predictions among all actual positive samples. It measures how well the model predicts true positives without false negatives.

**Formula:** True Positives / (True Positives + False Negatives)

**F-measure:** The harmonic mean of precision and recall. It provides a single metric that balances both precision and recall.

**Formula:** 2 * (Precision * Recall) / (Precision + Recall)

The study conducted bug severity forecasting simulations on four popular bug datasets, namely Thunderbird, Mozilla, Eclipse, and Firefox. The results indicate that the GA-KNN method performs better than the PSO-KNN, KNN method in terms of average accuracy. It was observed that the KNN algorithm produces the worst outcomes for most datasets. On the other hand, PSO-KNN was found to provide more accurate bug severity predictions than KNN techniques. The proposed PSO-based and GA-KNN feature selection method significantly improves the results of KNN computations. The simulation results of the proposed bug assignment models, including KNN, PSO-KNN and GA-KNN approaches, using the four datasets are shown in table below:

**Table 6: Results**

| **Approaches** | **Dataset** | **Performance Measuring Parameters** | | | |
|---|---|---|---|---|---|
| | | **Accuracy** | **Precision** | **Recall** | **F-measure** |
| **K-neighbour** | Mozilla | 77.00 | 63.60 | 77.00 | 69.50 |
| | Firefox | 57.66 | 54.80 | 57.70 | 49.40 |
| | Thunderbird | 63.66 | 52.90 | 63.70 | 57.30 |
| | Eclipse | 72.66 | 65.30 | 72.70 | 68.60 |
| **Avg. results of K-Neighbour** | | 67.745 | 59.15 | 67.775 | 61.20 |
| **PSO K-neighbour** | Mozilla | 81.33 | 72.90 | 81.30 | 74.30 |
| | Firefox | 66.66 | 59.80 | 66.70 | 57.00 |
| | Thunderbird | 69.00 | 64.80 | 69.00 | 58.70 |
| | Eclipse | 79.66 | 72.10 | 79.70 | 74.00 |
| **Avg. results of PSO K-Neighbour** | | 74.1625 | 67.40 | 74.175 | 66.00 |

| | | | | | |
|---|---|---|---|---|---|
| **GA K-neighbour** | Mozilla | 80.33 | 80.40 | 80.30 | 73.20 |
| | Firefox | 67.66 | 69.80 | 67.70 | 58.70 |
| | Thunderbird | 70.33 | 62.30 | 70.30 | 60.50 |
| | Eclipse | 80.66 | 76.50 | 80.70 | 74.00 |
| **Avg. results of GA K-Neighbour** | | 74.745 | 72.25 | 74.75 | 66.6 |

## GRAPHICAL REPRESENTATION OF GA AND PSO

### ECLIPSE PLATFORM

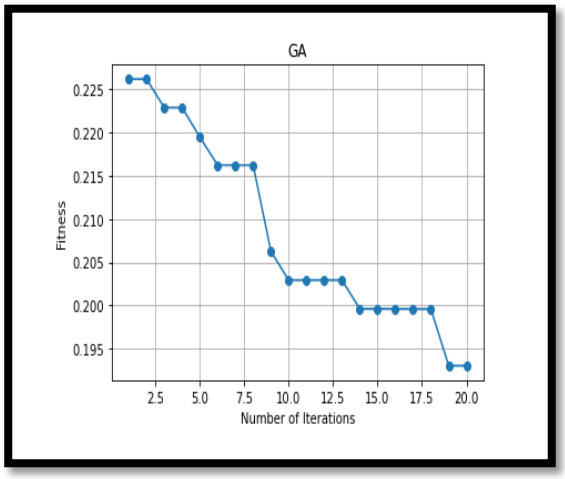



**Figure 14(a): Genetic algorithm**

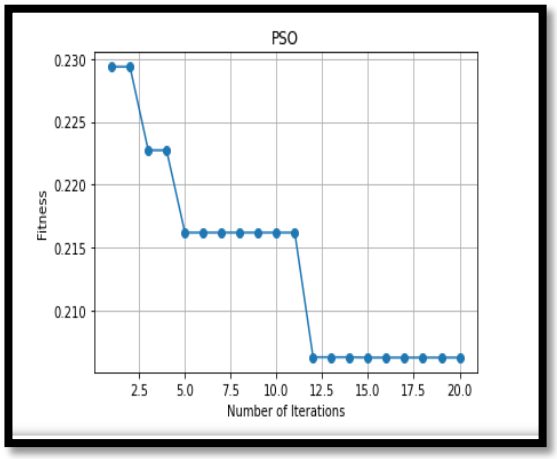



**Figure 14(b): PSO algorithm**

## FIREFOX PLATFORM

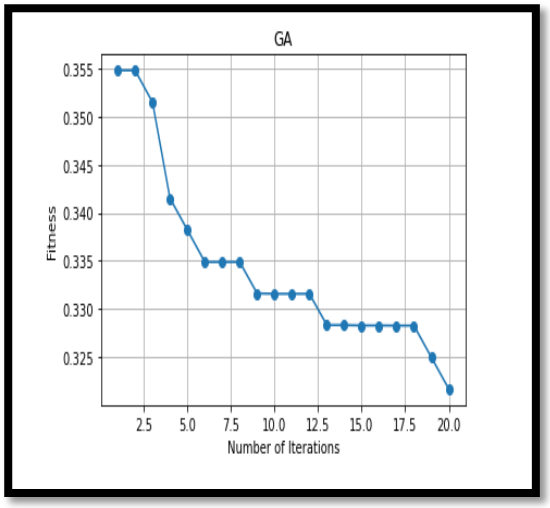



**Figure 15(a): Genetic algorithm**

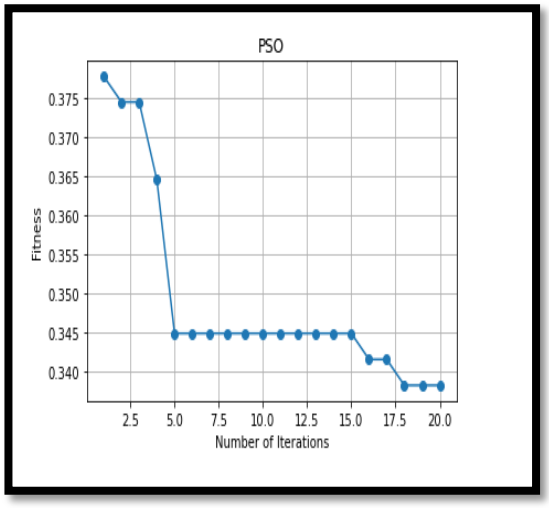



**Figure 15(b): PSO algorithm**

## MOZILLA PLATFORM

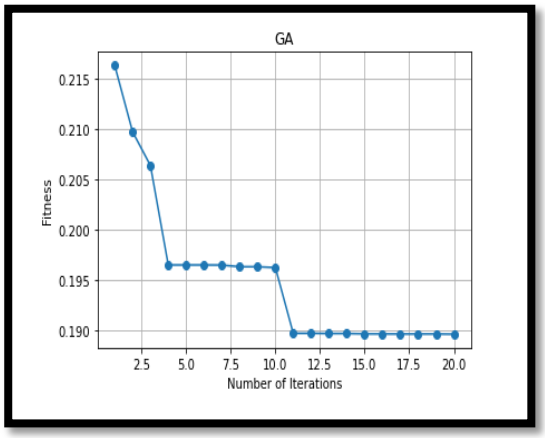



**Figure 16(a): Genetic algorithm**

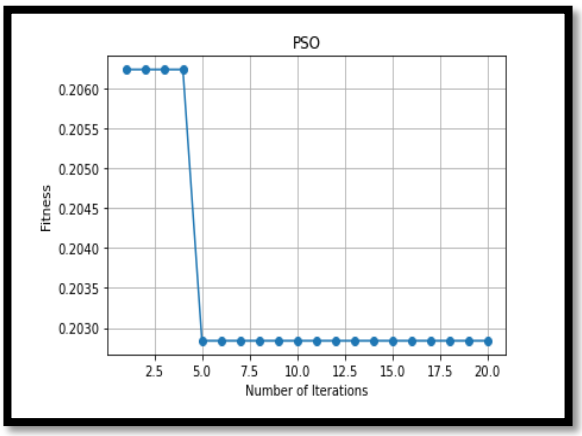



**Figure 16(b): PSO algorithm**

**THUNDERBIRD PLATFORM**

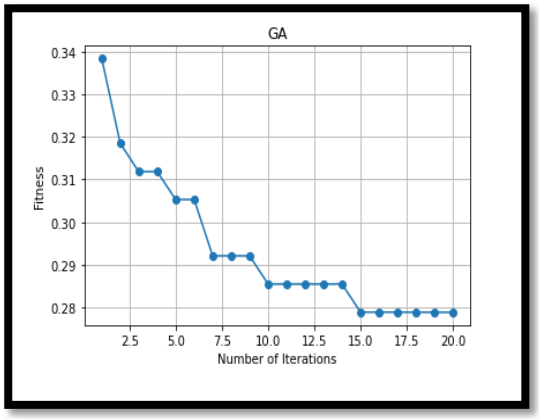



**Figure 17(a): Genetic algorithm**

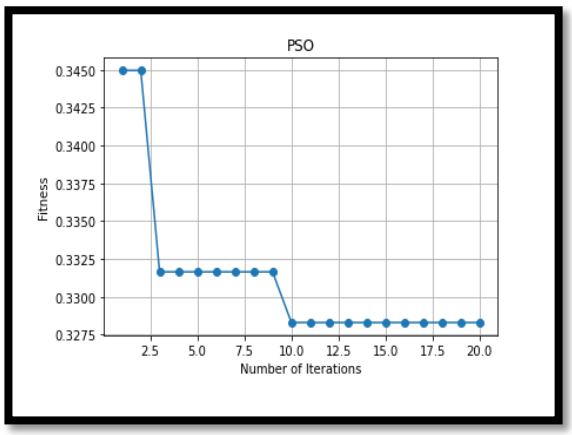



**Figure 17(b): PSO algorithm**

## 6.2. CONCLUSION

The bug prioritization system for bug triage using meta-heuristic feature selection algorithms for optimization is a powerful tool that can help organizations to manage their software development projects more efficiently. By automating the bug triage process and recommending the most suitable developers to fix the most severe bug, this system can significantly reduce the time and effort required for bug resolution.

The system incorporates various techniques, such as NLP-based data pre-processing, feature extraction, and feature selection using PSO and GA algorithms, to generate a reduced feature set that is more informative and relevant. The system then uses machine learning algorithms to classify the bug report into different categories based on its features and attributes.

The bug prioritization component of the system is another critical feature that recommends the most suitable developers for fixing the bug based on their severity. This component enables efficient allocation of resources, ensuring that the most skilled developers are assigned to each bug report.

Overall, the bug prioritization system for bug triage using meta-heuristic feature selection algorithms for optimization is a promising tool that can improve the efficiency and quality of software development projects. With further development and refinement, this system can become an essential tool for organizations looking to streamline their software development processes and deliver high-quality products to their customers.

Also, by looking at Table 7, we can conclude that Genetic algorithm works better than Particle Swarm Optimization as well as normal KNN.

## 6.3. FUTURE WORK

Incorporating more advanced natural language processing techniques: The current system uses basic NLP techniques for data pre-processing. Future work can explore more advanced NLP techniques, such as named entity recognition, sentiment analysis, and topic modelling, to improve the accuracy of bug classification and developer recommendation.

Evaluation and comparison of meta-heuristic algorithms: The current system uses PSO and GA algorithms for feature selection. Future work can evaluate and compare the performance of these algorithms and explore other meta-heuristic algorithms to identify the best algorithm for feature selection.

Integration with project management tools: The current system is standalone and does not integrate with project management tools. Future work can explore the integration of this system with project management tools, such as Jira or Trello, to provide real-time bug reporting and bug assignment.

Collaboration with developers: The current system recommends developers based on their past experience and expertise. Future work can explore the integration of developer feedback to improve the accuracy of developer recommendations. Developers can provide feedback on the recommended bug assignments and suggest changes to the feature set or algorithm to improve the recommendations.

Working with larger datasets is a critical area for future work in this project. While the current system can process and analyse bug reports efficiently, it may face challenges when dealing with large-scale datasets.

# REFERENCES


Alazzam, I., Aleroud, A., Al Latifah, Z., & Karabatis, G. (2020). Automatic bug triage in software systems using graph neighborhood relations for feature augmentation. *IEEE Transactions on Computational Social Systems*, *7*(5), 1288-1303.

Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. *International journal of advanced computer science and applications*, *9*(2).

Lee, D. G., & Seo, Y. S. (2019). Systematic review of bug report processing techniques to improve software management performance. *Journal of Information Processing Systems*, *15*(4), 967-985.

Mani, S., Sankaran, A., & Aralikatte, R. (2019, January). Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In *Proceedings of the ACM India joint international conference on data science and management of data* (pp. 171-179).

Subbiah, U., Ramachandran, M., & Mahmood, Z. (2019, January). Software engineering approach to bug prediction models using machine learning as a service (MLaaS). In *Icsoft 2018-proceedings of the 13th international conference on software technologies* (pp. 879-887).

Yadav, A., Singh, S. K., & Suri, J. S. (2019). Ranking of software developers based on expertise score for bug triaging. Information and Software Technology, 112, 1-17.