AN EFFICIENT MACHINE LEARNING TOOL FOR AUTOMATIC BUG TRIAGING

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE AWARD OF DEGREE

OF

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

Submitted By

RISHABH SIROHI
(2K21/SWE/19)

Under the supervision of

Ms. PRIYA SINGH
ASSISTANT PROFESSOR

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi - 110042

MAY 2023

M.Tech (Software Engineering) Rishabh Sirohi 2023

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CANDIDATE'S DECLARATION

I, **Rishabh Sirohi**, Roll No **2K21/SWE/19** a student of M. TECH (Software Engineering) declare that the project Dissertation titled "**An Efficient Machine Learning Tool for Automatic Bug Triaging**" which is submitted by me to Department of Software Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Fellowship or other similar title or recognition.

[illegible]

Place: DTU, Delhi
Date: 30/05/2023

RISHABH SIROHI
(2K21/SWE/19)

DECLARATION

We/I hereby certify that the work which is presented in the Major Project-II/Research Work entitled Efficient Machine Learning tool for Automatic Bug Triaging in fulfilment of the requirement for the award of the Degree of Bachelor/Master of Technology in Software Engineering and submitted to the Department of Software Engineering, Delhi Technological University, Delhi is an authentic record of my/our own, carried out during a period from 2021-2023, under the supervision of Ms. Priya Singh.

The matter presented in this report/thesis has not been submitted by us/me for the award of any other degree of this or any other Institute/University. The work has been published/accepted/communicated in SCI/ SCI expanded/SSCI/Scopus indexed journal OR peer reviewed Scopus indexed conference with the following details:

Title of the Paper: Automatic Bug Triaging Analysis using Machine Learning Techniques : A review
Author names (in sequence as per research paper): Rishabh Sirohi, Priya Singh
Name of Conference/Journal: 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)
Conference Dates with venue (if applicable): 11 November 2022
Have you registered for the conference (Yes/No)?: Yes
Status of paper (Accepted/Published/Communicated): Published
Date of paper communication: 30 September 2022
Date of paper acceptance: 21 October 2022
Date of paper publication: 20 March 2023

[illegible]
Rishabh Sirohi
2K21/SWE/19
Student(s) Roll No., Name and Signature

SUPERVISOR CERTIFICATE

To the best of my knowledge, the above work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere. I, further certify that the publication and indexing information given by the students is correct.

[illegible] 29/5/23

Place: Delhi

Date: 29/05/2023

Supervisor Name and Signature
Ms. PRIYA SINGH
ASSISTANT PROFESSOR

NOTE: PLEASE ENCLOSE RESEARCH PAPER ACCEPTANCE/ PUBLICATION/COMMUNICATION PROOF ALONG WITH SCOPUS INDEXING PROOF (Conference Website OR Science Direct in case of Journal Publication).

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CERTIFICATE

I, hereby certify that the Project titled "**An Efficient Machine Learning Tool for Automatic Bug Triaging**" which is submitted by Rishabh Sirohi, Roll No: 2K21/SWE/19, Department of Software Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of project work carried out by the student under my supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

[illegible]

Place: DTU, Delhi
Date: 30/05/2023

Ms. PRIYA SINGH
SUPERVISOR
(Assistant Professor, SE, DTU)

ACKNOWLEDGEMENT

I am very thankful to Ms. Priya Singh (Assistant Professor, Software Eng. Dept.) and all the faculty members of the Software Engineering Department of DTU. They all provided immense support and guidance for the completion of the project undertaken by me. It is with their supervision that this work came into existence.

I would also like to express my gratitude to the university for providing the laboratories, infrastructure, test facilities, and environment which allowed me to work without any obstructions.

I might likewise want to see the value in the help gave to us by our lab partners, seniors and our companion bunch who supported us with all the information they had in regards to different subjects

Sirohi

RISHABH SIROHI
M.TECH (SWE)
2K21/SWE/19

ABSTRACT

As technology advances at an exponential rate every day, the development and testing teams do their utmost to address problems as soon as they arise in order to meet customer deadlines. Finding the appropriate developer to address a specific bug is typically simple and quick in small organisations, but it can be challenging for large organisations to find the developer who will be able to address the bug quickly, which is one of the main tasks of bug triaging. In this report, we will examine numerous methods for automatically triaging bugs and attempt to identify the optimal method based on a series of research questions that will enable us to understand the statistical analysis of these methods.

CONTENTS

LIST OF TABLES

LIST OF FIGURE

LIST OF ABBREVIATIONS

CNN	**Convolutional Neural Network**
RNN	**Recurrent Neural Network**
SVM	**Support Vector Machine**
DBRNN	**Deep Bidirectional Recurrent Neural Network with Attention**
LDA	**Latent Dirichlet Allocation**

Chapter 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

In order to deliver the customer's product on schedule and to ensure that it has the fewest possible flaws that won't ruin the customer experience, bug fixing has become a crucial component for large organisations. Nowadays, practically all small and large businesses use bug repositories (like Bugzilla), which provide all the specific details about the problems that are open, fixed, need to be rechecked, etc.Although bug repositories offer a wide range of services such as bug status, bug description, bug summary, etc., there are still two significant limitations that limit their use: big scale bug data, and bad quality bug data.

According to [2], approximately 34,917 developers and users reported 333,371 problems to Eclipse between 2001 and 2010. Therefore, handling such a vast amount of data manually becomes extremely difficult. Low quality bug data, on the other hand, is made up of noisy and redundant data. These two factors are extremely important, and developers should make sure they are taken out of the dataset during the pre-processing step itself to avoid noisy data misleading them and redundant data wasting valuable development time when managing bugs. So, these factors made us realise that there is a need to work upon this field and research on different techniques to automate this process.

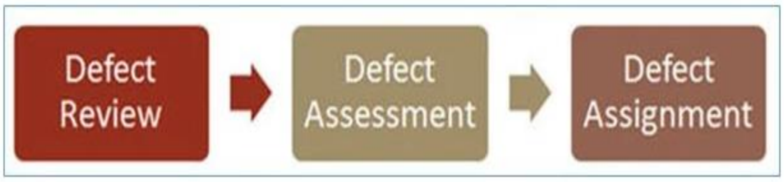


Figure 1.1[5] Bug Triage flow

1.1 BUG TRIAGING

A human triager selects the recipient of each bug after it has been reported on the bug repository. If the designated developer is able to solve the bug, it is listed as resolved; however, if the developer was unsuccessful or left the firm during that period, then a new or another developer is assigned to resolve it. Bug triaging is the full process of selecting a developer to address a bug. Figure 1.1 depicts an overview of it. Manually assigning the bug is a difficult task. It is challenging to examine the numerous descriptions of bugs before appointing the appropriate developer to work on them as you can see in Figure 1.2.Additionally, because developers' profiles frequently change from one project to another, it is unknown if they are still working on that particular project.

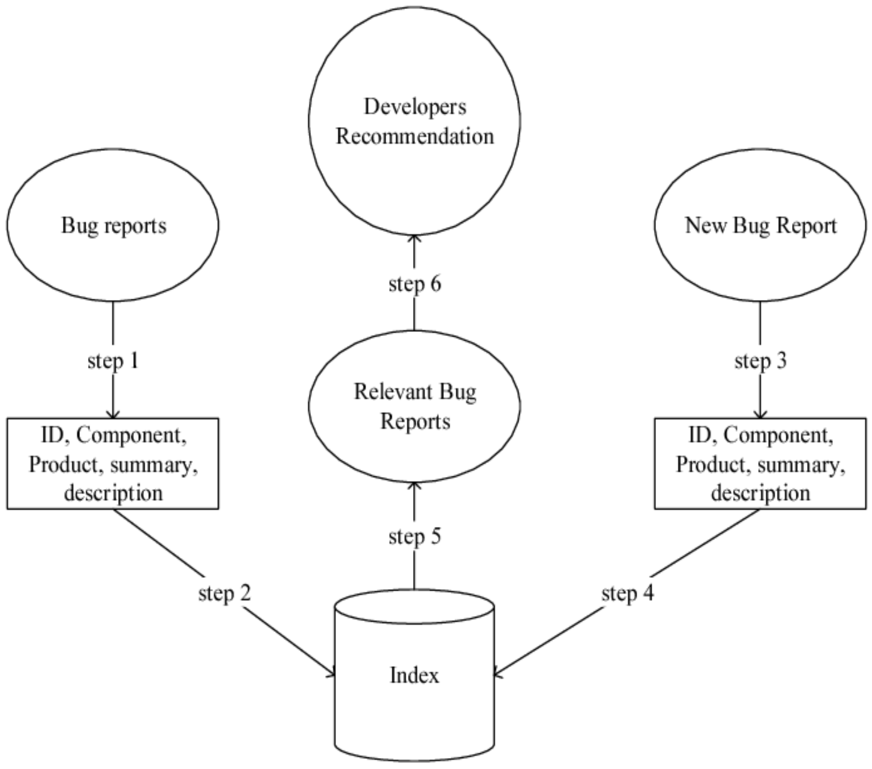

Developers Recommendation
Bug reports
New Bug Report
step 6
step 1
step 3
Relevant Bug Reports
ID, Component, Product, summary, description
ID, Component, Product, summary, description
step 5
step 2
step 4
Index

Figure 1.2[3] Bug triaging framework

Over time, it became apparent that Automatic Bug triaging was necessary to address all of these issues. Automatic bug triaging will do away with the need for human triagers to actively assign bugs to the appropriate developers. In this review study, we reviewed a number of bug triaging strategies that have been used by different academics. Our objective is to find the best approach that will work quickly and with minimal error.

1.2 RESEARCH OBJECTIVE

This research project's objective is to provide a thorough literature review to address queries about automatic bug triaging using machine learning algorithms. Through this review effort, we have looked at a number of strategies that have been put out by different scholars and have statistically determined which one is the best and least time- and error-consuming.

1.3 ORGANIZATION OF THESIS

The remaining chapters of the thesis are as follows:

Chapter 2 consists of a discussion of Literature review of our own work in the field of bug triaging represents **Chapter 3** discusses the methodology for the review where we mention the research questions formed, and the search strategy we have adopted following that the exclusion and inclusion criteria. **Chapter 4** shows the results of our implementations **Chapter 5** talks about the limitations of bug triaging which are still a major drawback **Chapter 6** concludes the paper and mentions the future scope.

Chapter 2

LITERATURE SURVEY

2.1. RESEARCH QUESTION FORMULATION

The major goal of the review is to look at several strategies that have been put out by different scholars and statistically determine which technique is the best among them and will be the least time-consuming and error-prone. As indicated in table 2.1, we have developed the following research questions to help us examine and comprehend automatic bug triaging technique. Firstly, we identify different type of techniques used for bug triaging (RQ1). Similar to this in RQ2, we identified various types of datasets used for bug triaging. In RQ3, we concentrated on understanding the benefits and drawbacks of the various strategies employed thus far.

Table 2.1. Research Questions

RQ. No	Research Questions	Objective
RQ1)	What are the different categories of bug triaging approaches proposed?	To list the many approaches currently being utilised for bug triaging.
RQ2)	Which Datasets are most commonly used?	For the purpose of identifying the various dataset types used for bug triaging.
RQ2.1)	What is the effect on results based upon the dataset used?	To identify whether a particular type of dataset gives better results and performance than other for bug triaging.

RQ3)	What are the strengths and weaknesses of ML models used for Bug triaging?	To identify the advantages and disadvantages of the techniques used.
RQ4)	What are the mostly used statistical test to evaluate Bug triaging using ML models?	To identify studies in which results of automated bug triaging using various techniques are validated using statistical tests.

2.2. SEARCH STRATEGY AND STUDY SELECTION

While conducting our review we focused mostly on the latest and all the advanced techniques. After conducting a thorough search and analysing the papers that were found after the initial retrieval, we chose the most promising research to be examined.

Triage AND (Bug OR Defect) AND Machine Learning (classification OR regression) AND Information Retrieval AND Developer Recommendation AND Severity Prediction AND Topic modelling AND Graph model AND Tossing (support vector machine OR decision tree OR neural network OR linear regression OR multiple regression OR multivariate regression OR genetic algorithm OR search-based techniques)

We chose five databases to search based on the access that was accessible to the databases:

ACM Digital Library

- IEEE Xplore
- Springer
- Science Direct
- John Wiley Inc.

For insect triaging in the SLR, we incorporated empirical investigations employing machine learning, tossing, information retrieval, topping, and graph approaches. We therefore used our own inclusion and exclusion criteria to effectively choose these research, which enabled us to choose the papers in line with the standards we used to evaluate the investigations.

Inclusion Criteria

1. Research on automatic bug triaging utilising a mix of two or more strategies.
2. Research projects that evaluate the effectiveness of various machine learning methods.
3. Research projects that involved developer involvement in bug repositories.
4. Research that describes the technique and experimental testing of proposed algorithms.

Exclusion Criteria

1. Research based on estimation of bug severity.
2. Research that are without any experimental results or empirical analysis.
3. Research that are focusing on bug classification rather than severity prediction.
4. Studies without the right performance metrics and dataset

2.3. DATA EXTRACTION AND SYNTHESIS

In order to answer the RQs, after selecting the primary studies, we made a data extraction sheet to store all the important information related to these studies. Prepared different fields in the sheet were we mentioned following details related to studies: Study title; year of publication, name of journal or conference, dataset type, techniques used for bug triaging, statistical test used, performance measure and merits/demerits of that study.

While analysing the data we focused on both quantitative and qualitative viewpoints. We considered this as an important perspective because through qualitative analysis one get to know about statistical measures takes, validation methods adopted, type of dataset used and usage of metrics whereas quantitative analysis consists of performance measures adopted for a type of data set. Summary of all the RQs are showcased in further sections with the help of charts, graphs, plots and tables.

2.4. DESCRIPTION OF PRIMARY STUDIES AND QUALITY ANALYSIS

Following the application of inclusion and exclusion criteria on 48 studies, we were able to consider thirty-six studies as mentioned in Sect II. Now scoring of these studies was done where 14 being assigned as highest score and 0 as the lowest. In order to select good quality papers, studies having score equal to or greater than 8.5 were selected for answering our RQs while less than 8.5 score studies were rejected. Studies which were not having an ample citation count and didn't considered good amount of references, where rejected (N. Sreenivas [35], A. Goyal[36] and R. Jaiswal[37]) after the quality assessment phase. The accepted studies were sorted yearly wise and arranged in ascending order as listed in Table 2.2. Out of the 33 primary studies selected, 27 % of the selected primary studies are published in journals, and 73 % are published in conference proceedings.

2.5. YEAR-WISE DISTRIBUTION OF SELECTED PRIMARY STUDIES

From January 2003 to January 2022, the year-wise distribution of primary studies in shown in figure 2.1. We have only considered last two decades' studies to track the change in technology. [4] used the technique of text categorization for bug triaging. Text categorization is one of the most traditional and popular technique which is still used where the combination of feature selection and instance selection is used to reduce the dataset for bug triaging. Till 2010 studies mainly focused using machine learning solely or in combination with other technique. [17] proposed a new approach of recommendation system for developers.

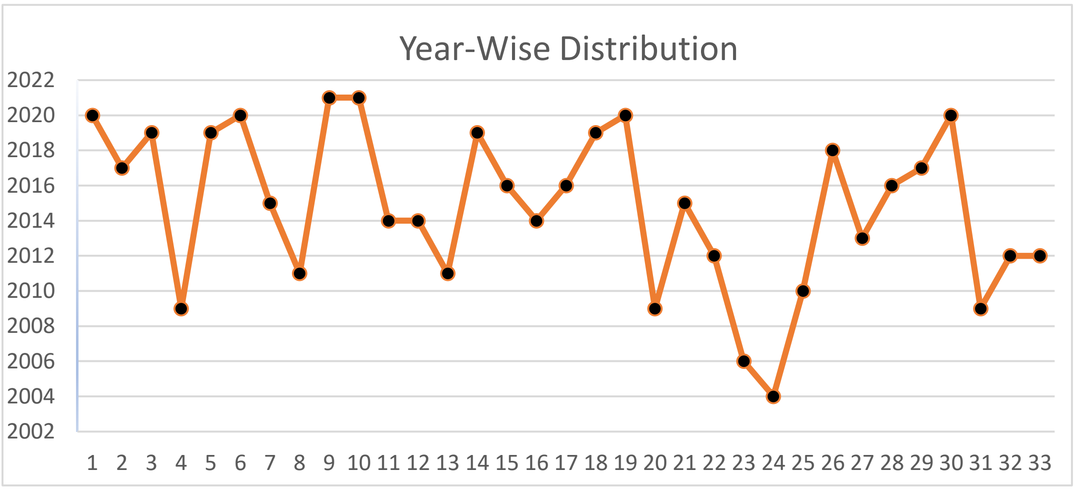


Figure 2.1. Year Wise Distribution of Primary Studies

Table 2.2 Description of Primary Studies

Primary Study	Author	References	Primary Study	Author	References
PS1	Davor Cubranic (2004)	[4]	PS18	Xin Xia (2016)	[5]
PS2	John Anvik (2006)	[6]	PS19	V Govindasamy (2016)	[7]
PS3	Gaeul Jeong (2009)	[8]	PS20	Sun-Ro Lee (2017)	[9]
PS4	Syed Nadeem Ahsan (2009)	[10]	PS21	Snehal Chopade (2017)	[11]
PS5	Shivkumar Shivaji (2009)	[12]	PS22	Ying Yin (2018)	[13]
PS6	Jifeng Xuan (2010)	[14]	PS23	Senthil Mani (2019)	[15]
PS7	John Anvik (2011)	[16]	PS24	Sheng-Qu Xi (2019)	[17]
PS8	Weiqin Zou (2011)	[18]	PS25	Cícero Augusto De Lara Pahins (2019)	[19]
PS9	Huzefa Kagdi (2012)	[20]	PS26	Aindrila Sarkar (2019)	[21]
PS10	Jifeng Xuan (2012)	[2]	PS27	Shikai Guo (2020)	[22]
PS11	Xin Xie (2012)	[23]	PS28	Wei Zhang (2020)	[24]
PS12	Tao Zhang (2014)	[25]	PS29	Iyad Alazzam (2020)	[26]
PS13	Hao Hu (2014)	[27]	PS30	Raf Almhana(2020)	[28]
PS14	Geunseok Yang (2014)	[29]	PS31	Syed Farhan Alam Zaidi (2020)	[30]
PS15	Tao Zhang (2014)	[31]	PS32	Syed Farhan Alam Zaidi (2021a)	[32]
PS16	Ali Sajedi Badashian (2015)	[33]	PS33	Syed Farhan Alam Zaidi (2021b)	[34]

PS17	Jifeng Xuan (2015)	[1]			

Various surveys that are closely related to bug triaging approaches have been conducted in the past [38 - 42] and have classified the various techniques based on their own criteria for classifying bug reports and carrying out automatic bug triaging.. [38] classified the bug triaging techniques into three categories namely; Metadata approach, Profile based and Machine Learning based approach where Metadata includes time stamp of bug and bug history which is used to allot appropriate developer to bug report, profile based approach recommend the bug report to developer on the basis of Developers profile history where on the other hand Machine learning algorithm uses already existing bug report to train a classifier and then use this classifier to assign new bug reports to developer.

Another survey on bug triaging is provided by [39], who classified different bug triaging techniques into categories like Text categorization, Tossing Graph, -Recommendation, Role-Based, and Text Mining. In contrast, [40] focused primarily on the application of different Machine learning algorithms (Nave Bayes, Decision Tree, K-Nearest-Neighbor , Neural Network) for bug triaging.

According to [41], there are only two categories that may be used to classify bug assignment and evaluation: machine learning and information retrieval. While [42] concentrated on conducting surveys based on bug prioritisation rather than bug triaging, they conducted a comparative analysis of both techniques and came to the conclusion that information retrieval methods are more accurate than machine learning algorithms. Bug prioritising, according to Jamal Uddin et al. [42], is a difficult and error-prone process because any poor choice here would result in inefficient resource use and additional time loss.

[43 - 46] worked upon predicting severity of bug using Machine Learning algorithms where [43] built and designed a tool named SEVERIS that effectively predict the severity of bug based on rule-learning technique. On the hand [44] make the use of Naïve Bayes classifier to predict whenever a new bug arrives whether it belongs to a “severe” or “non-severe” category. [45] used BM25 textual similarity function to measure the similarity between the bugs in bug repository and used KNN to

determine the severity of the bugs. [46] focused on using Graph approach to determine the severity of bug reports based on graph metrics.

The following studies [47], [48], [49], and [50] have been eliminated from our review based on our inclusion-and-exclusion criteria since they do not meet our strict research standards. Despite the fact that we discovered a sizable number of articles on bug triaging and severity prediction, we found most of them were related to survey papers only. To the best of our knowledge regarding searching paper we didn't find any paper that provide the systematic review related to automatic bug triaging. So, we decided to write a systematic literature review related to automatic bug triaging so that we can get the detailed analysis regarding the technique used till now and can contribute further to improve it.

Now we summarises the methods employed, presents the findings of the primary studies as the research questions we have developed, compares them in light of several criteria, and identifies any research gaps.

RQ1: What are the different state-of-the-art proposed for bug triaging?

To adopt the Automatic Bug triaging, wide range of techniques have been used by practitioners and researchers over the past 20 years. The purpose of these methods is to do away with manual bug triaging, which is a time-consuming and error-prone way to assign a suitable developer to a bug report. After analysing 33 papers, we categorise the techniques into the following groups;

- Text Classification and Machine Learning(TC+ML)
- Tossing Graph (TG)
- Topic Modelling (TM)
- Machine Learning (ML)
- Recommendation System and Machine Learning (RS+ML)
- Recommendation System and Information Retrieval (RS+IR)
- Graph and Machine Learning (Graph + ML)
- Ensemble Learning Machine

The figure 2.2 clearly shows that machine learning algorithms account for around 33% of the techniques utilised for automatic bug triaging. A subject modelling approach has been selected over the deployment of a recommendation system together with a machine learning algorithm. We also discovered that the accuracy of the output we get from the ML method, Graph, and Tossing Graph is nearly identical, despite a small variance in percentage.

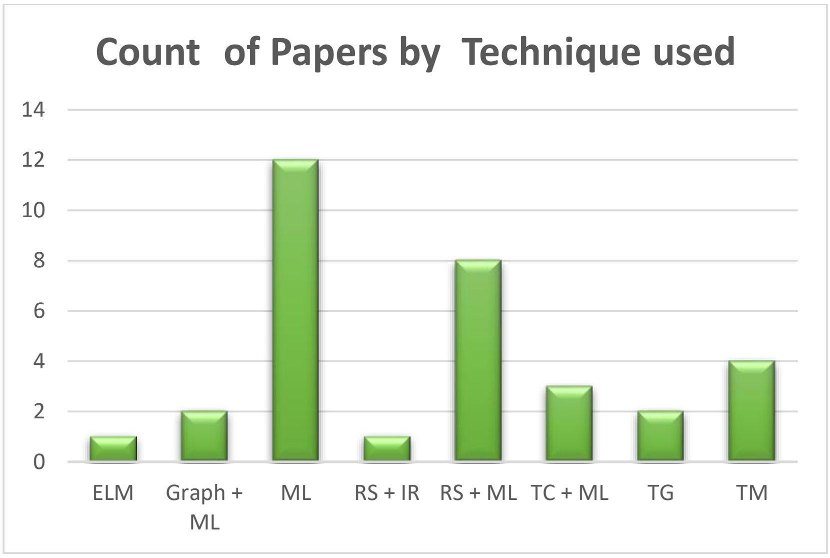


Figure 2.2. Different bug triaging approaches

[1 - 3], [7], [8], [11], [12], [16], [22 - 24], [29 - 30], [38], [39], [41] used Machine Learning Algorithms as shown in figure 2.3 to assign a bug to an appropriate Developer where [1], [7], [16], [22] models are based on Text categorisation technique where they have used the combination of feature selection and instance selection to reduce the dataset for bug triaging whereas [39] only focused on feature selection to reduce the dataset and predict bug performance using Naıve Bayes and Support Vector Machine.

[30] suggested a method for automatically triaging bugs that is similarly based on text categorization. Text categorization, sometimes referred to as text classification, is a method for automatically classifying a group of documents into groups based on a preset set of categories. Using the bug's description, developers will be anticipated in this paper. In order to forecast the correct developer, this work used supervised machine learning technique with a Nave Bayes classifier.

[3] used DBRNN-A that focused on syntactic and semantic features using unsupervised Machine learning.

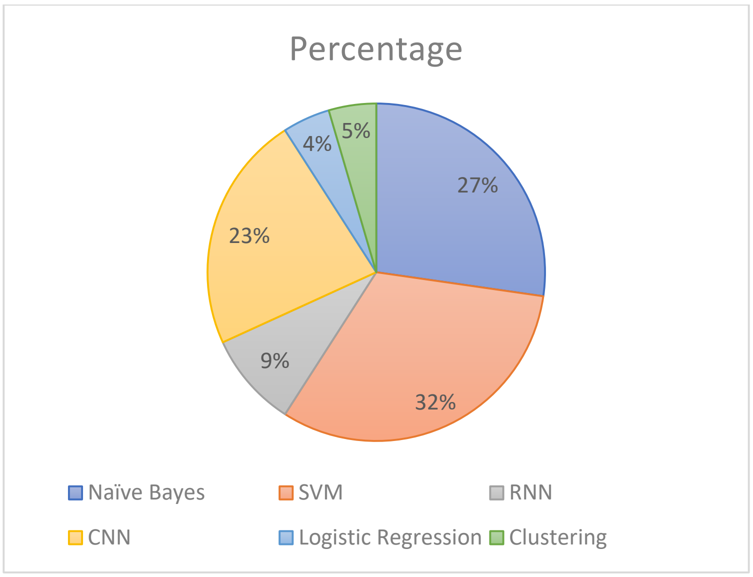


Figure 2.3. Different Machine Learning techniques

[2] focused on using Prim's method, [8] created ML based recommender system to deal bug whereas [11] used SVM that trains a classifier for each new Developer to resolve the bug. [12] proposed a ML based recommender system called as BugFixer which recommends bugs to new developer on the basis of historical bug information.

[16] focused on dealing noisy data by using feature selection which improved the performance by 5%. [22] introduces a model that focus on reducing the data scale and improving the quality. To avoid random grouping of data they used clustering in this approach to group the similar bug reports which make it easy to assign the bug to the appropriate developer whereas [23] used valid time split evaluation where sequentially train and test on a large industrial data set is done.

[24] and [38] proposed deep learning based bug triaging technique using CNN where [24] focused on finding technique best for word representation by comparing three embedding techniques: two context-insensitive; Word2Vec (Word to Vector), GloVe (Global Vector) and one context-sensitive; ELMo (Embedding's from Language Models) and [38] used word2vec method in combination with CNN (CNNDA) and One-Hot word vector methods in conjunction CNN for the same.

[29] proposed a semi-automated approach using machine learning. It is a type of text categorisation technique where bug reports are called as text documents and document label are the developers names appropriate to resolve the bug reports.

A supervised machine learning algorithm takes a set of instances as input with known labels and generates a classifier and then suggests the future bug reports to an appropriate developer based on the classifier used.

[41] focused to train a quality Developer recommender system by choosing four developer features, namely network centrality, developer workspace, developer expertise, and transmissibility of developers to know among them which is crucial for the bug reassignment and then predict the potential developer by applying that feature to six Machine learning algorithms.

[4] and [5] introduced new techniques to reduce the bug tossing time where [4] used Markov chain based graph model that detect the tossing history of bug and then find the potential developer whereas [5] focused on improving the bug triaging efficiency by integrating three important aspects, the textual content in the bug reports, the metadata in the bug reports, and the tossing sequence of the bug reports.

[6] used Graph based RFSH Algorithm and [28] used Information retrieval techniques to solve the bugs.

[15] proposed an approach that combined topic modelling and multi-feature (i.e. component, product, severity and priority) where topic modelling is used to extract topic from bug repository using LDA and multi-feature is used to identify corresponding reports that have same multi-feature with the new bug reports. On the basis of this user is able to recommend the appropriate developer to fix the bug and predict its severity also.

RQ2: Which datasets are most commonly used for Bug Triaging?

Over the years, scientists have carried out a wide variety of experiments. Typically, open source to commercial data sets are used for these experiments. 33 original papers were examined, and it was discovered that just two employed industrial data sets while the others conducted experiments using open-source datasets. The proportion of primary research using various open bug repositories is shown in Figure 2.4.

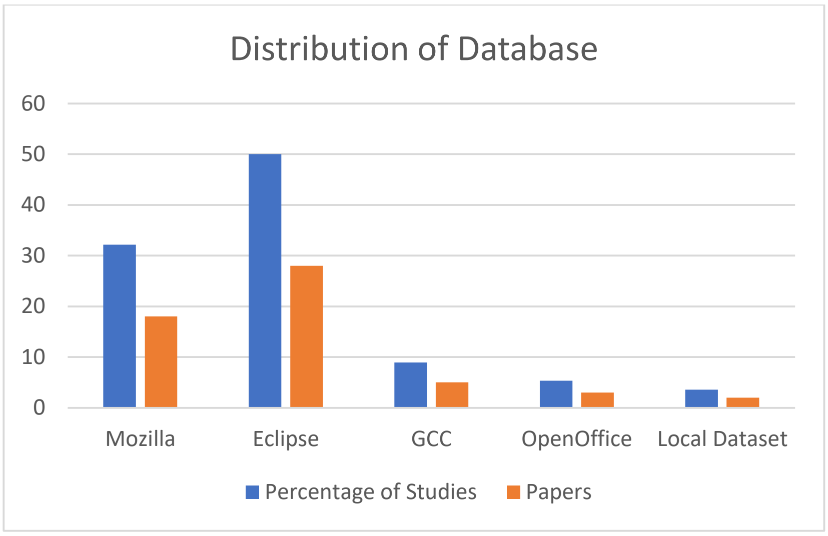


Figure 2.4. Different types of Dataset used

RQ2.1: What is the effect of different types of Dataset on the performance of Bug Triaging?

Only a few experiments [9], [27] that employed industry data sets yielded findings that were superior to those of the open source dataset. Few factors are mostly determined to be in charge of its performance. First off, compared to open source, the quality of bug reports for commercial applications is carefully maintained. Since bug reports are created by users, contributors , and project participants for open-source projects. Because there are so many different types of open source system issue reports, it is more likely that they will be of lesser quality and contain more duplicates. On the other side, the Quality Assurance team writes and properly structures the bug reports for industrial projects. Second, since the open source development community is very broad and unstable. In contrast to industrial projects, which are more structured and solid, open source projects allow for any developer to join or leave at any time. Last but not least, compared to a balanced dataset of industry projects, the performance accuracy of open source is hampered by more imbalanced data..

RQ3: What are the strengths and weaknesses of frequently used ML models for Bug triaging?

Different Machine learning algorithms vary in their advantages and drawbacks. Knowing them is essential since it aids in choosing the right algorithm for the bug report. After knowing the strength and weaknesses of the algorithm new researchers and software developers would be able to select the algorithm suitably to perform automatic bug triaging easily, efficiently and effectively. Strengths and weaknesses of various machine learning techniques used are depicted in table 2.3.

Table 2.3. Strengths and Weaknesses of frequently used ML techniques

ML Algorithm	**Strength**	**Weaknesses**
Logistic Regression	In terms of output, logistic regression provides high probabilistic interpretation capabilities, and overfitting can be avoided. Updates to any new data often use stochastic gradient descent.	Logistic regression typically performs poorly for non-linear decision boundaries. It is inappropriate for really complex partnerships.
Support Vector Machines (SVM)	The SVM algorithm, a subset of supervised machine learning, may also simulate non-linear decision boundaries. In n-dimensional space in particular, it is resistant to overfitting. Compared to the majority of the other machine learning algorithms, provides more accurate results.	Scalability of huge datasets is a major issue with SVM, and it is also more memory-intensive, making it difficult to tune by choosing the right Kernel.
Naive Bayes (NB)	Another sort of supervised machine learning method that is simple to use, predicts results accurately, and is easy to execute is naive Bayes.	When the dataset is huge, it occasionally fails to forecast the outcomes accurately. Because it assumes that all features tend to be independent, it is not

		appropriate for situations found in real life.
Clustering	Due of its ease of use and simplicity, K-means is frequently employed as a clustering algorithm for bug reporting.	Occasionally, it becomes challenging to mention the clusters' number. K-means can only handle numerical data, as well. Therefore, converting bug reports into numerical format becomes a time-consuming process for analysis.

RQ4: What are the mostly used statistical test to evaluate Bug triaging using ML models?

The use of statistical tests in research for model prediction serves as a technique to reinforce the model's predictions by statistically analysing them. Only 27% of the 33 primary studies [4, [9], [11–13], [17], [18], and [20–22] that we chose for review used statistical testing. Figure 2.5 illustrates the application of statistical test distribution. The Friedman test, t-test, and Chi-squared test were each employed by two investigations, whilst five studies used the Wilcoxon signed-rank test for comparative statistical analysis. It is notable that more studies have concluded that nonparametric tests, such as the Wilcoxon signed-rank test, are appropriate for statistical testing than parametric t-tests. Bonferroni-Dunn test and Nemenyi test are both only applied in one study each.

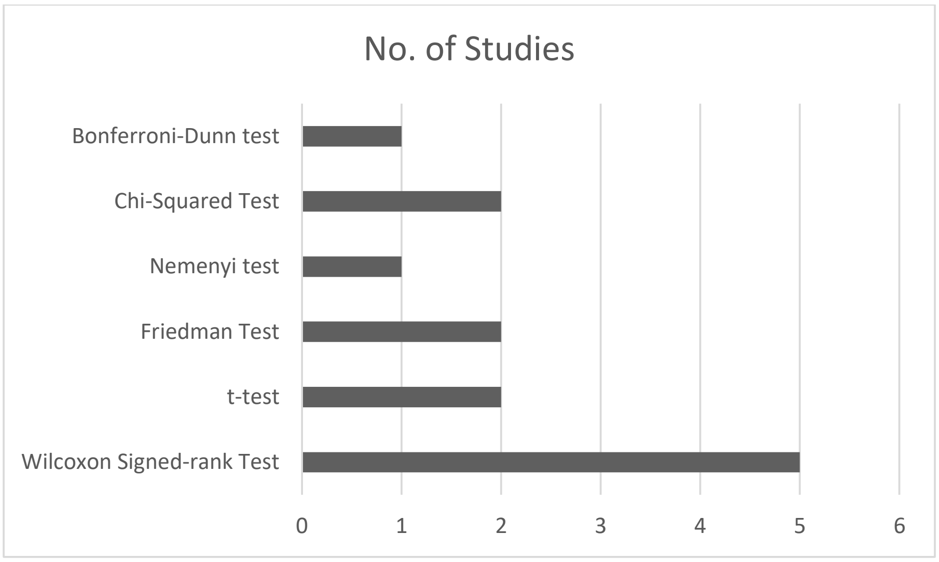


Figure 2.5. Different Statistical test used

Chapter 3

RESEARCH METHODOLOGY

3.1 EXPERIMENTAL SETUP

In this section, we will provide a concise overview of the hardware and software tools employed in our research. Additionally, we will delve into the performance measures, framework, and methodology utilized throughout the study.

3.1.1 HARDWARE USED

The proposed project is based on ML approaches, which revolve around classification; the only hardware instrument required for implementation is a computer system. The model is created and run on a laptop with the given minimum hardware requirements.

• System Type	Windows 10, Macintosh
• Processor	Core i3 processor
• RAM	4GB
• Hard disk	500GB

3.1.2 SOFTWARE USED

The Jupyter Notebook and Anaconda Navigator tools were used to implement each ML model. 3.6 Python Python is a general-purpose, interpreted object-oriented programming language. It is a language that is open-source and free grown in popularity as a result of its condensed, straightforward, and extensive library support.

3.2 SAMPLE DATASET

This section describes the dataset and ML methods that we employed to complete the task at hand. The dataset will be covered in detail in the first section of this chapter, and all the machine learning and optimisation approaches that we employed in this study will be covered in the second half.

Input: A bug report in natural language prose that the reporter submits after describing the issue.
Output: The component in which the bug might possibly exist, together with the developer (or list of developers) who could be given responsibility for fixing it.

A problem is reported by a user through the software's bug tracker. Since the description of the bug provided by the user is a natural language text, natural language processing is utilised to extract pertinent keywords from the bug report that would explain the bug the user has experienced. Stop-word elimination and stemming are used during processing to extract pertinent keywords from the bug report's description. Based on the previously learned dependencies, these extracted keywords are utilized to pinpoint the component that is most likely to be problematic. A list of developers will then be informed of this bug's resolution based on their faulty Component and Tossing History. It is important to choose the developers on the list such that there is the least possible chance that the bug will be reassigned. The developer and the component involved in the bug are noted/labelled in the bug report after it has been fixed. For the purpose of supervised learning from the repaired bugs, a dependency structure gradually develops.

An open source software bug tracker tool's dataset is a collection of bug reports that have been repaired and contain the relevant details about the components, developers, and reassignments. This data is classed, categorised, and partially organised. The bug tracker programme stores a user-submitted bug report, which is often a natural language text, in XML format. Detailed information in the dataset

• Severity: How serious the bug is determined how quickly it needs to be corrected.
• Product: The specific software program to which the bug relates.
• Component: The product's pertinent subsystem for the bug that was reported.

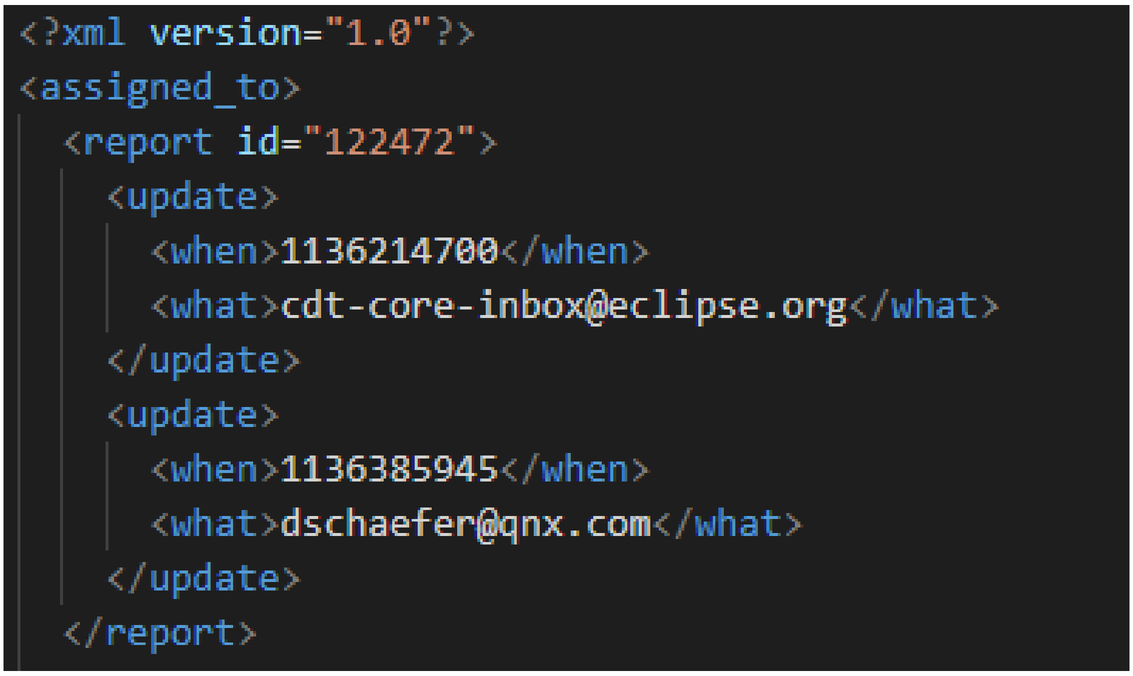
• Assigned to: The name of the developer who was given responsibility for fixing the bug.

• Brief description: Incorporates user-embedded natural language text.

• Bug status: A bug's condition as of each update. REOPENED, NEW, ASSIGNED, RESOLVED, VERIFIED.

• Fix: Marking the bug report as needing maintenance. WORKSFORME, REMIND, INVALID, FIXED.

These details enable the establishment of dependencies between the developers, components, and reassignment.



```xml
<?xml version="1.0"?>
<assigned_to>
  <report id="122472">
    <update>
      <when>1136214700</when>
      <what>cdt-core-inbox@eclipse.org</what>
    </update>
    <update>
      <when>1136385945</when>
      <what>dschaefer@qnx.com</what>
    </update>
  </report>
```

Figure 3.1 Sample Dataset

An illustration of a report in an assignedto.xml data set. Along with the date and time of each update, it identifies the developer to whom it was allocated.

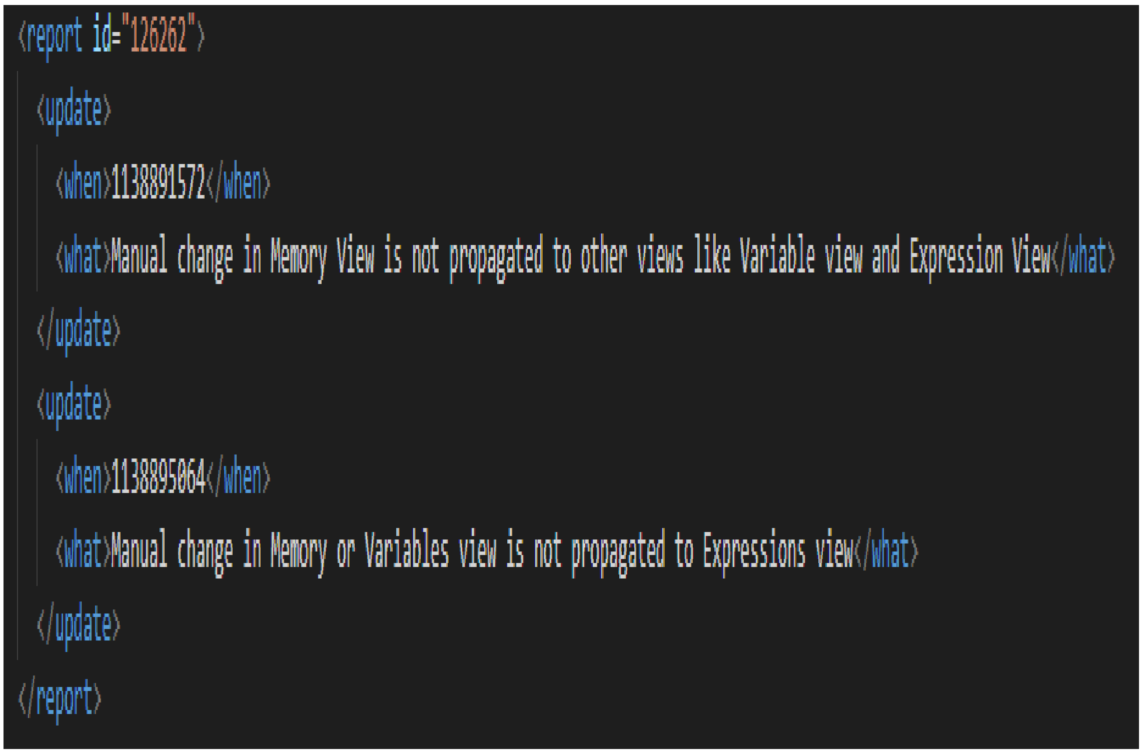
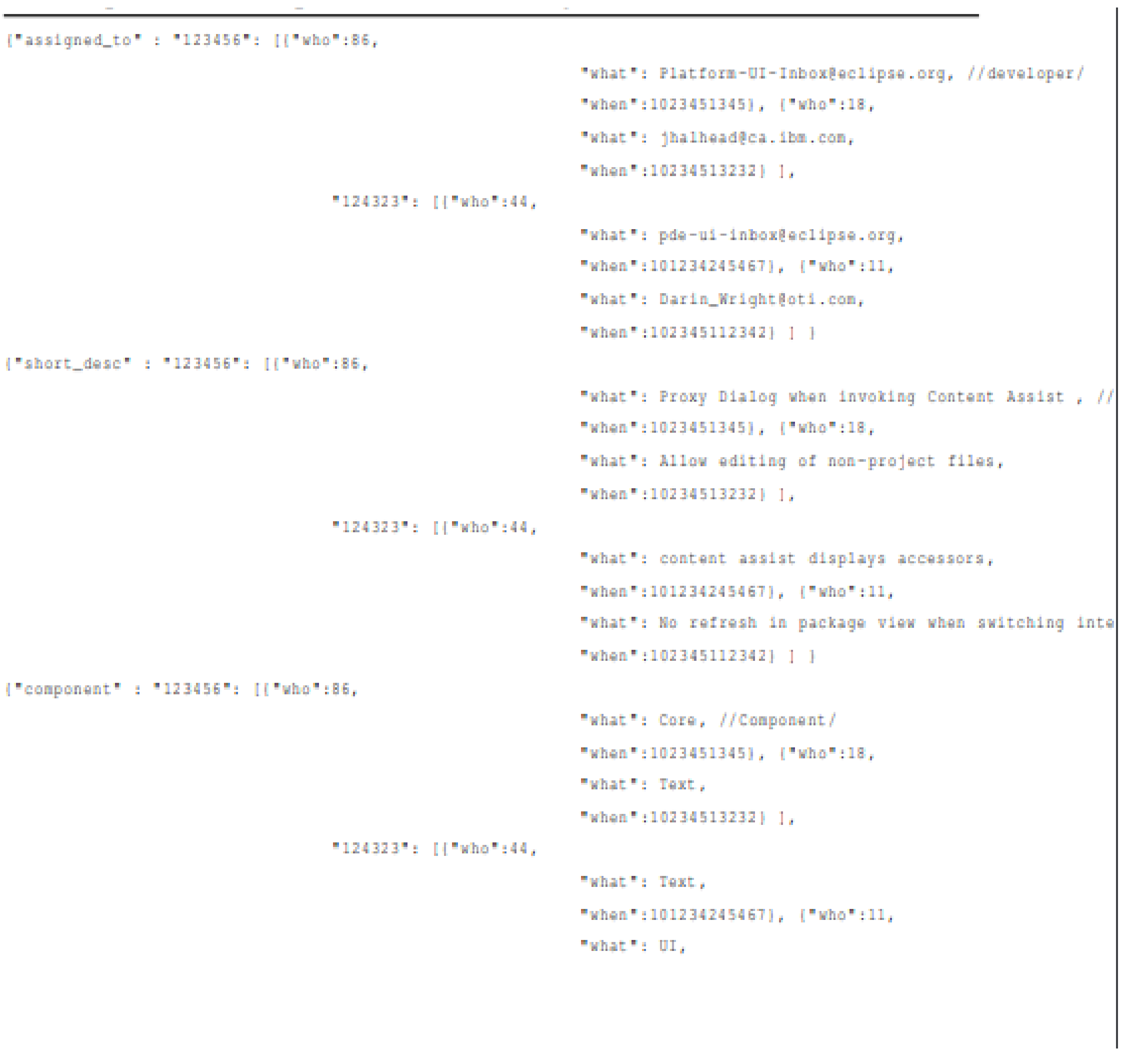

```
<report id="126262">
  <update>
    <when>1138891572</when>
    <what>Manual change in Memory View is not propagated to other views like Variable view and Expression View</what>
  </update>
  <update>
    <when>1138895064</when>
    <what>Manual change in Memory or Variables view is not propagated to Expressions view</what>
  </update>
</report>
```

Figure 3.2 Short description of the bug report submitted by the reporter.

We used a dataset in JSON format with roughly 1,60,000 well-structured reports in order to increase efficiency. The training data set in JSON format compares each report-id's update("when") in the relevant files, together with the "what" content present in the brief description (to get the bug report), component (to get the component), and assigned to (the developer). The pre-processed file is converted into a feature-vector pair, with the developer serving as the vector and the bug report and the component in question as the feature. When receiving problem reports, the classifier uses the feature-vector pair as input to choose the appropriate developer. With the aid of an additional feature-vector pair (component and developer) collected by the classifier, graphs are generated.. The next likely developer who can repair the bug is identified by combining the probabilities of a developer fixing a bug in a specific component and passing responsibility to another developer.



```
{"assigned_to" : "123456": [{"who":86,
                                                "what": Platform-UI-Inbox@eclipse.org, //developer/
                                                "when":1023451345}, {"who":18,
                                                "what": jhalhead@ca.ibm.com,
                                                "when":10234513232} ],
                          "124323": [{"who":44,
                                                "what": pde-ui-inbox@eclipse.org,
                                                "when":101234245467}, {"who":11,
                                                "what": Darin_Wright@oti.com,
                                                "when":102345112342} ] }
{"short_desc" : "123456": [{"who":86,
                                                "what": Proxy Dialog when invoking Content Assist , //
                                                "when":1023451345}, {"who":18,
                                                "what": Allow editing of non-project files,
                                                "when":10234513232} ],
                          "124323": [{"who":44,
                                                "what": content assist displays accessors,
                                                "when":101234245467}, {"who":11,
                                                "what": No refresh in package view when switching inte
                                                "when":102345112342} ] }
{"component" : "123456": [{"who":86,
                                                "what": Core, //Component/
                                                "when":1023451345}, {"who":18,
                                                "what": Text,
                                                "when":10234513232} ],
                          "124323": [{"who":44,
                                                "what": Text,
                                                "when":101234245467}, {"who":11,
                                                "what": UI,
```

Figure 3.3 Short description of the assigned bug to developer

3.3 TECHNIQUES USED

The dataset being used affects the different Automatic Bug Triaging techniques. After parsing, the training dataset was processed through stop-word removal and stemming. Stop-word elimination and stemming are accomplished using the Snowball Stemming algorithm. The data set is transformed into feature vectors using a multinomial NB classifier.

Using an XSLT parser, the product, component, and brief description of each bug from the Training data set are parsed to create a uniform text file. Each bug's report ID is extracted from the assigned-to.xml file, together with the "when" attribute of each update, and compared to the corresponding entries in the short-desc.xml, product.xml, and component.xml files. The extracted data is then exported to a text file in a text processing-friendly format.

3.3.1 TEXT CLASSIFICATION NAÏVE BAYES ALGORITHM

```
Algorithm: NAIVE BAYES CLASSIFIER - Text classification

Input: T - Training corpus

B - New Bug Report

Output: dj - The developer with the highest probability to whom the bug will be
assigned

  1. From Training corpus T, extract Vocabulary V (collect unique words from all
     bug reports)
  2. Initialize P(dj) as an empty dictionary for each developer dj in D
  3. for each developer dj in D do
          reportsj ← all bug reports in developer dj from T
          P(dj) ← |reportsj| / |total no. reports| (calculate the prior probability P(dj)
          for developer dj) end for
  4. Textj ← concatenate all bug reports in T for each developer dj
  5. Initialize P(wk|dj) as an empty dictionary for each word wk in Vocabulary V
  6. for each word wk in Vocabulary V do
           nk ← count the number of occurrences of word wk in Textj
          P(wk|dj) ← (nk + const) / (n + const * |Vocabulary|) (calculate the
          conditional probability P(wk|dj) with Laplacian Smoothing) end for
  7. Calculate the probability P(B|dj) for each developer dj using the words in the
     New Bug Report B:
        • Initialize P(B|dj) as 1
        • for each word in B do
              • if the word exists in Vocabulary V then
                    • P(B|dj) *= P(w|dj) (multiply the conditional probability of
                      each word in B) end for
  8. Select the developer dj with the highest probability P(B|dj) and assign it to dj
  9. return dj as the developer with the highest probability to whom the bug will be
     assigned
```

3.3.2 MULTINOMIAL NAIVES BAYES ALGORITHM

```
Algorithm: TRAIN MULTINOMIALNB - Text classification

Input: R - Training Corpus (List of bug reports)
```

```
C - List of developers
Output: V - Vocabulary prior - Prior probabilities
            condprob - Conditional probabilities
    1. V ← extract Vocabulary from R (collect unique words from all bug reports)
    2. N ← count the total number of bug Reports in R
    3. Initialize prior as an empty dictionary
    4. Initialize condprob as an empty dictionary
    5. for each developer d in C do
    6. Nc ← count the number of bug reports in developer d from R
            prior(d) ← Nc / N (prior probability of developer d)
            wordsc ← collect all words from all bug reports in developer d
            Initialize T0_c as the count of words in wordsc
            Initialize condprob[d] as an empty dictionary
            for each word w in V do 12. Tc ← count the occurrences of word w in
            wordsc
                T0_c ← count the total number of words in wordsc
                condprob[d][w] ← (Tc + 1) / (T0_c + |V|) (apply Laplacian
                Smoothing) end for end for
    7. return V, prior, and condprob
```

3.3.3 LINEAR SUPPORT VECTOR MACHINE

```
Algorithm: Linear SVM
Input: X - Training data (features)
 y - Training labels
C - Regularization parameter
Output: w - Weight vector b - Bias term
    1. Initialize the Lagrange multipliers α to zeros for each training example.
    2. Define the learning rate η and the number of iterations.
    3. for t = 1 to number of iterations do
        4. Select a training example (xi, yi) randomly from X and y.
        5. Compute the decision function: f(xi) = Σ(α * yi * xi) + b.
        6. Compute the margin: $margin = yi * f(xi)$.
        7. if $margin < 1$ then 8. Update the weight vector: $w = w + \eta * (yi * xi)$.
              9. Update the bias term: $b = b + \eta * yi$.
              10. Update the Lagrange multiplier: $\alpha = \alpha + \eta * C$.
        8. else 12. Update the Lagrange multiplier: $\alpha = \alpha + \eta * C$.
        9. end if end for
4. return w, b as the learned weight vector and bias term for the linear SVM.
```

Chapter 4

RESULT AND DISCUSSION

To evaluate the effectiveness of our proposed algorithm, we utilized bug data from Mozilla and Eclipse. We performed a comprehensive analysis of the entire lifespan of both applications. To ensure thorough coverage, we divided the bug data into 10 folds and conducted 9 iterations. During the data collection process, we gathered four types of information from the bug reports:

1. Keywords: We extracted keywords from the bug descriptions and comments provided in the bug reports. These keywords were useful for classification and analysis purposes.
2. Bug source: We retrieved details about the product and component to which each bug was assigned, as mentioned in the bug reports. This information aided in categorizing the bugs based on their respective areas within the software.
3. Temporal information: We recorded the dates when the bugs were reported and when they were fixed. This temporal data helped us analyze the timeline and duration of bug resolution.
4. Developers assigned: We compiled a list of developer IDs assigned to each bug. This information allowed us to examine developer involvement and assignment patterns throughout the bug resolution process.

In our studies, we changed the amount of the vocabulary and test set. Figure 4.1 shows that when 90% of the document corpus is utilised as training and 10% as the test set, the algorithm accurately assigns just under 75% of the bugs. As the test corpus size is increased to 50, the accuracy gradually decreases to 65%.

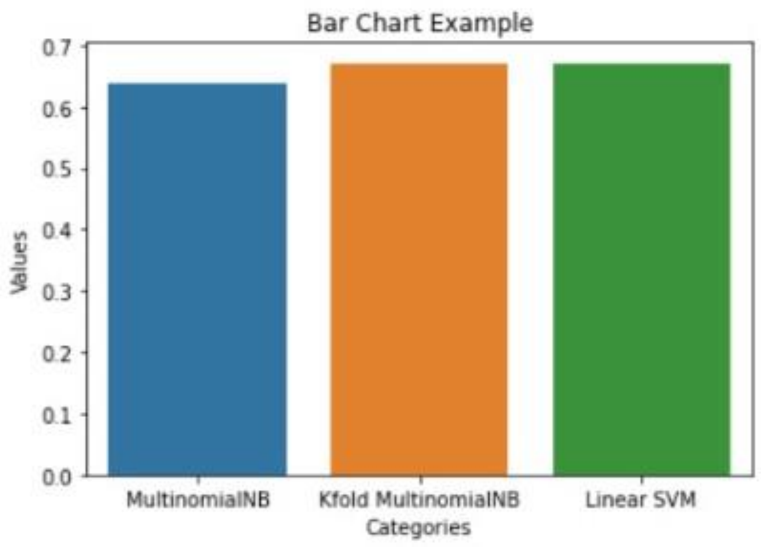
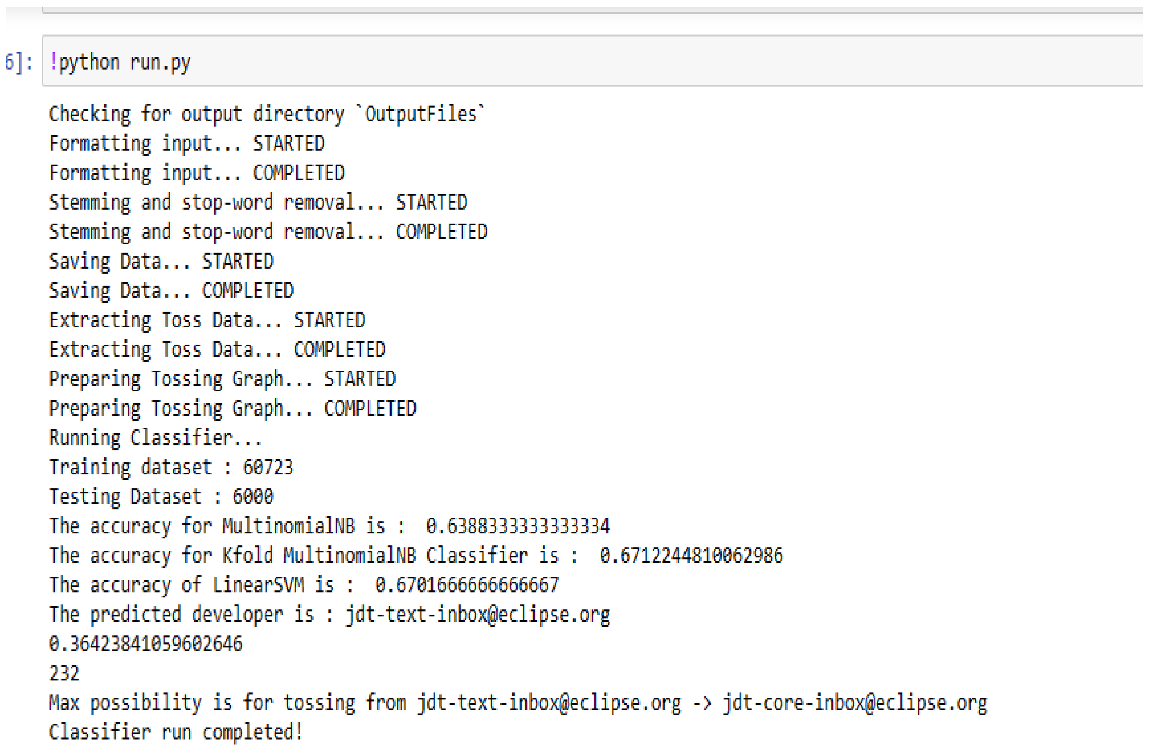


Figure 4.1 Accuracy bar graph of different techniques used



```
6]: !python run.py

    Checking for output directory `OutputFiles`
    Formatting input... STARTED
    Formatting input... COMPLETED
    Stemming and stop-word removal... STARTED
    Stemming and stop-word removal... COMPLETED
    Saving Data... STARTED
    Saving Data... COMPLETED
    Extracting Toss Data... STARTED
    Extracting Toss Data... COMPLETED
    Preparing Tossing Graph... STARTED
    Preparing Tossing Graph... COMPLETED
    Running Classifier...
    Training dataset : 60723
    Testing Dataset : 6000
    The accuracy for MultinomialNB is :  0.6388333333333334
    The accuracy for Kfold MultinomialNB Classifier is :  0.6712244810062986
    The accuracy of LinearSVM is :  0.6701666666666667
    The predicted developer is : jdt-text-inbox@eclipse.org
    0.36423841059602646
    232
    Max possibility is for tossing from jdt-text-inbox@eclipse.org -> jdt-core-inbox@eclipse.org
    Classifier run completed!
```

Figure 4.2 Experiment Output

Finally, accuracy of around 67.12% is reached with 10 fold cross validation, while accuracy of 67.00% is attained with linear SVM as shown in Figure 4.2.

Chapter 5

LIMITATIONS

After doing comparative study of Machine learning techniques used for automatic bug triaging, we found that there are some limitations in the current Machine learning technique which are as follows;

- Natural Language is used to write bug reports. Consequently, if bug reports are of poor quality, a user will not be able to obtain enough information.
- Because trials are typically conducted on open source projects, it is questionable whether or not identical results will also be obtained through industrial projects.
- Manual selection of the dataset and application of the proper algorithm is used to test the outcomes. Therefore, there is a potential that the experiments will be biassed or mistaken.
- When assigning developers, there is a possibility to eliminate a candidate by looking at their bug-resolving history. If they haven't fixed more than five issues, they may be able to in the future.
- Only a select few algorithms are tested for bug severity predictions. Therefore, it should be conducted and evaluated against other algorithms in order to gain higher accuracy.

Chapter 6

CONCLUSION AND FUTURE SCOPE

The main goal of automatic bug triaging is to use a method that takes less time and is less prone to mistakes. Over the past 20 years, academics have developed a variety of machine learning techniques to solve these issues and have attempted to remove the reliance on human triage to assign bugs to the right developer. Accordingly, the analysis of these methods showed us how manual issue triaging may be substituted by automatic bug triaging using a suitable ML system that can precisely predict developers' responses to incoming bug reports. Automatic bug triaging will do away with the need for human triagers to actively assign bugs to the appropriate developers. Our objective is to identify the optimum method that will take less time and be less prone to error.

After reviewing these methods, we think that Automatic Bug Triaging in macro organisation utilising Machine learning algorithm will be necessary in the near future. Implementing these techniques doesn't call for major hardware modifications; instead, it opens up the possibility of handling bug reports independently of other software bug repositories like Bugzilla, Jira, etc., and helps the organisation to manage bugs more quickly. Although the majority of organisations have access to and utilise bug tracking repositories, the cost of using these repositories will likely rise in the near future as data volumes rise. The goal of organisations is to maximise profits while maintaining tight control over costs, allowing for the best possible resource utilisation. By using this automated bug triaging technique, it will be possible to meet the deadline for the client's project and save enough money to make the business profitable and both employers and employees satisfied with the compensation received. According to us there is still some possibility of performance enhancement and we will be comparing most frequently used Machine Learning techniques on some specific performance metrics.

REFERENCES

[1] J. Xuan *et al.*, “Towards effective bug triage with software data reduction techniques,” *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 264–280, 2015, doi: 10.1109/TKDE.2014.2324590.

[2] J. Xuan, H. Jiang, Z. Ren, and W. Zou, “Developer prioritization in bug repositories,” *Proc. - Int. Conf. Softw. Eng.*, pp. 25–35, 2012, doi: 10.1109/ICSE.2012.6227209.

[3] Barbara Kitchenham and S. Charters, “Методи за автоматично управление на подемни устройства при Jack-up системите,” 2007, doi: 10.1145/1134285.1134500.

[4] D. Cubranic and G. C. Murphy, “Automatic bug triage using text categorization,” *16th Int. Conf. Softw. Eng. Knowl. Eng.*, pp. 92–97, 2004.

[5] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, “Improving Automated Bug Triaging with Specialized Topic Model,” *IEEE Trans. Softw. Eng.*, vol. 43, no. 3, pp. 272–297, 2017, doi: 10.1109/TSE.2016.2576454.

[6] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?,” *Proc. - Int. Conf. Softw. Eng.*, vol. 2006, pp. 361–370, 2006, doi: 10.1145/1134285.1134336.

[7] V. Govindasamy, V. Akila, G. Anjanadevi, H. Deepika, and G. Sivasankari, “Data reduction for bug triage using effective prediction of reduction order techniques,” *2016 Int. Conf. Comput. Power, Energy, Inf. Commun. ICCPEIC 2016*, pp. 85–90, 2016, doi: 10.1109/ICCPEIC.2016.7557229.

[8] G. Jeong, S. Kim, and T. Zimmermann, “Improving bug triage with bug tossing graphs,” *ESEC-FSE’09 - Proc. Jt. 12th Eur. Softw. Eng. Conf. 17th ACM SIGSOFT Symp. Found. Softw. Eng.*, pp. 111–120, 2009, doi: 10.1145/1595696.1595715.

[9] S. R. Lee, M. J. Heo, C. G. Lee, M. Kim, and G. Jeong, “Applying deep learning based automatic bug triager to industrial projects,” *Proc. ACM SIGSOFT Symp. Found. Softw. Eng.*, vol. Part F1301, pp. 926–931, 2017, doi: 10.1145/3106237.3117776.

[10] S. N. Ahsan, J. Ferzund, and F. Wotawa, “Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine,” *4th Int. Conf. Softw. Eng. Adv. ICSEA 2009, Incl. SEDES 2009 Simp. para Estud. Doutor. em Eng. Softw.*, pp. 216–221, 2009, doi: 10.1109/ICSEA.2009.92.

[11] S. Chopade and P. More, “Effective bug triage with Prim’s algorithm for feature selection,” *Proc. IEEE Int. Conf. Signal Process. Commun. ICSPC 2017*, vol. 2018-Janua, no. July, pp. 217–220, 2018, doi: 10.1109/CSPC.2017.8305842.

[12] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, “Reducing features to improve bug prediction,” *ASE2009 - 24th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, no. Section II, pp. 600–604, 2009, doi: 10.1109/ASE.2009.76.

[13] Y. Yin, X. Dong, and T. Xu, “Rapid and Efficient Bug Assignment Using ELM for IOT Software,” *IEEE Access*, vol. 6, no. c, pp. 52713–52724, 2018, doi: 10.1109/ACCESS.2018.2869306.

[14] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, “Automatic bug triage using semi-supervised text classification,” *SEKE 2010 - Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng.*, no. 60805024, pp. 209–214, 2010.

[15] S. Mani, A. Sankaran, and R. Aralikatte, “Deeptriage: Exploring the effectiveness of deep learning for bug triaging,” *ACM Int. Conf. Proceeding Ser.*, pp. 171–179, 2019, doi: 10.1145/3297001.3297023.

[16] J. Anvik and G. C. Murphy, “Reducing the effort of bug report triage: Recommenders for development-oriented decisions,” *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, 2011, doi: 10.1145/2000791.2000794.

[17] S. Q. Xi, Y. Yao, X. S. Xiao, F. Xu, and J. Lv, “Bug Triaging Based on Tossing Sequence Modeling,” *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 942–956, 2019, doi: 10.1007/s11390-019-1953-5.

[18] W. Zou, Y. Hu, J. Xuan, and H. Jiang, “Towards training set reduction for bug triage,” *Proc. - Int. Comput. Softw. Appl. Conf.*, pp. 576–581, 2011, doi: 10.1109/COMPSAC.2011.80.

[19] C. A. De Lara Pahins, F. D’Morison, T. M. Rocha, L. M. Almeida, A. F. Batista, and D. F. Souza, “T-REC: Towards accurate bug triage for technical groups,” *Proc. - 18th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2019*, pp. 889–895, 2019, doi: 10.1109/ICMLA.2019.00154.

[20] M. Zanoni, F. Perin, F. A. Fontana, and G. Viscusi, “Pattern detection for conceptual schema recovery in data-intensive systems,” *J. Softw. Evol. Process*, vol. 26, no. 12, pp. 1172–1192, 2014, doi: 10.1002/smr.

[21] A. Sarkar, P. C. Rigby, and B. Bartalos, “Improving Bug Triaging with High Confidence Predictions at Ericsson,” *Proc. - 2019 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2019*, pp. 81–91, 2019, doi: 10.1109/ICSME.2019.00018.

[22] S. Guo *et al.*, “Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network,” *Neural Process. Lett.*, vol. 51, no. 3, pp. 2589–2606, 2020, doi: 10.1007/s11063-020-10213-y.

[23] X. Xie, W. Zhang, Y. Yang, and Q. Wang, “DRETOM: Developer recommendation based on topic models for bug resolution,” *ACM Int. Conf. Proceeding Ser.*, pp. 19–28, 2012, doi: 10.1145/2365324.2365329.

[24] W. Zhang, “Efficient Bug Triage for Industrial Environments,” *Proc. - 2020 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2020*, pp. 727–735, 2020, doi: 10.1109/ICSME46990.2020.00082.

[25] T. Zhang and B. Lee, “A hybrid bug triage algorithm for developer recommendation,” *Proc. ACM Symp. Appl. Comput.*, pp. 1088–1094, 2013, doi: 10.1145/2480362.2480568.

[26] I. Alazzam, A. Aleroud, Z. Al Latifah, and G. Karabatis, “Automatic Bug Triage in Software Systems Using Graph Neighborhood Relations for Feature Augmentation,” *IEEE Trans. Comput. Soc. Syst.*, vol. 7, no. 5, pp. 1288–1303, 2020, doi: 10.1109/TCSS.2020.3017501.

[27] H. Hu, H. Zhang, J. Xuan, and W. Sun, “Effective bug triage based on historical bug-fix information,” *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 122–132, 2014, doi: 10.1109/ISSRE.2014.17.

[28] R. Almhana and M. Kessentini, “Considering dependencies between bug reports to improve bugs triage,” *Autom. Softw. Eng.*, vol. 28, no. 1, pp. 1–26, 2021, doi: 10.1007/s10515-020-00279-2.

[29] G. Yang, T. Zhang, and B. Lee, “Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports,” *Proc. - Int. Comput. Softw. Appl. Conf.*, pp. 97–106, 2014, doi: 10.1109/COMPSAC.2014.16.

[30] S. F. A. Zaidi, F. M. Awan, M. Lee, H. Woo, and C. G. Lee, “Applying Convolutional Neural Networks with Different Word Representation Techniques to Recommend Bug Fixers,” *IEEE Access*, vol. 8, pp. 213729–213747, 2020, doi: 10.1109/ACCESS.2020.3040065.

[31] T. Zhang, G. Yang, B. Lee, and E. K. Lua, “A novel developer ranking algorithm for automatic bug triage using topic model and developer relations,” *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 1, pp. 223–230, 2014, doi: 10.1109/APSEC.2014.43.

[32] S. F. A. Zaidi and C. G. Lee, “Learning Graph Representation of Bug Reports to Triage Bugs using Graph Convolution Network,” *Int. Conf. Inf. Netw.*, vol. 2021-Janua, pp. 504–507, 2021, doi: 10.1109/ICOIN50884.2021.9333902.

[33] K. Somasundaram and G. C. Murphy, “Automatic categorization of bug reports using latent Dirichlet allocation,” *Proc. 5th India Softw. Eng. Conf. ISEC’12*, pp. 125–130, 2012, doi: 10.1145/2134254.2134276.

[34] S. F. A. Zaidi and C. G. Lee, “One-Class Classification Based Bug Triage System to Assign a Newly Added Developer,” *Int. Conf. Inf. Netw.*, vol. 2021-Janua, pp. 738–741, 2021, doi: 10.1109/ICOIN50884.2021.9334002.

[35] N. Sreenivas and S. J. Saritha, “Enhancement towards efficient bug triage with software data reducing methods,” *2017 Int. Conf. Energy, Commun. Data Anal. Soft Comput. ICECDS 2017*, pp. 3649–3652, 2018, doi: 10.1109/ICECDS.2017.8390144.

[36] A. Goyal, “Effective bug triage for non-reproducible bugs,” *Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion, ICSE-C 2017*, pp. 487–488, 2017, doi: 10.1109/ICSE-C.2017.41.

[37] R. Jaiswal, M. Sahare, and U. Lilhore, “Genetic Approach based Bug Triage for Sequencing the Instance and Features,” *2018 Int. Conf. Comput. Commun.*

Informatics, ICCCI 2018, pp. 1–7, 2018, doi: 10.1109/ICCCI.2018.8441350.

[38] A. Yadav and S. K. Singh, "Survey Based Classification of Bug Triage Approaches," *APTIKOM J. Comput. Sci. Inf. Technol.*, vol. 1, no. 1, pp. 1–11, 2016, doi: 10.34306/csit.v1i1.37.

[39] V. B. Sawant and N. V. Alone, "A Survey on various Techniques for Bug Triage," *Int. Res. J. Eng. Technol.*, vol. 02, no. 09, pp. 917–920, 2015, [Online]. Available: http://www.academia.edu/download/54836058/IRJET-V2I9157.pdf

[40] N. Bhardwaj and A. . Bhattacharya, "Survey on General Classification Techniques for Effective Bug Triage," *Int. J. Comput. Sci. Eng.*, vol. 2, no. 11, pp. 6–10, 2015, doi: 10.14445/23488387/ijcse-v2i11p102.

[41] A. Goyal and N. Sardana, "Machine learning or information retrieval techniques for bug triaging: Which is better?," *E-Informatica Softw. Eng. J.*, vol. 11, no. 1, pp. 117–141, 2017, doi: 10.5277/e-Inf170106.

[42] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artif. Intell. Rev.*, vol. 47, no. 2, pp. 145–180, 2017, doi: 10.1007/s10462-016-9478-6.

[43] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," *IEEE Int. Conf. Softw. Maintenance, ICSM*, pp. 346–355, 2008, doi: 10.1109/ICSM.2008.4658083.

[44] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," *Proc. - Int. Conf. Softw. Eng.*, pp. 1–10, 2010, doi: 10.1109/MSR.2010.5463284.

[45] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 215–224, 2012, doi: 10.1109/WCRE.2012.31.

[46] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," *Proc. - Int. Conf. Softw. Eng.*, pp. 419–429, 2012, doi: 10.1109/ICSE.2012.6227173.

[47] A. Yadav and S. Kumar Singh, "An Information-Theoretic Approach for Bug Triaging," *Proc. 8th Int. Conf. Conflu. 2018 Cloud Comput. Data Sci. Eng. Conflu. 2018*, pp. 7–13, 2018, doi: 10.1109/CONFLUENCE.2018.8442506.

[48] T. S. Mian, "Automation of Bug-Report Allocation to Developer using a Deep Learning Algorithm," *2021 Int. Congr. Adv. Technol. Eng. ICOTEN 2021*, 2021, doi: 10.1109/ICOTEN52080.2021.9493515.

[49] K. Inkpen, ACM Digital Library., and ACM Special Interest Group on Computer-Human Interaction., "Proceedings of the 2010 ACM conference on Computer supported cooperative work.," p. 450, 2010.

[50] C. Sun, D. Lo, S. C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," *2011 26th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2011, Proc.*, pp. 253–262, 2011, doi: 10.1109/ASE.2011.6100061.

turnitin

Similarity Report ID: oid:27535:36490817

PAPER NAME

r_final.pdf

WORD COUNT	CHARACTER COUNT
7777 Words	**43108 Characters**
PAGE COUNT	FILE SIZE
35 Pages	**1.1MB**
SUBMISSION DATE	REPORT DATE
May 30, 2023 11:42 AM GMT+5:30	**May 30, 2023 11:43 AM GMT+5:30**

[illegible] 30/5/23

● 10% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 2% Internet database
- 7% Publications database
- Crossref database
- Crossref Posted Content database
- 2% Submitted Works database

● Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 8 words)

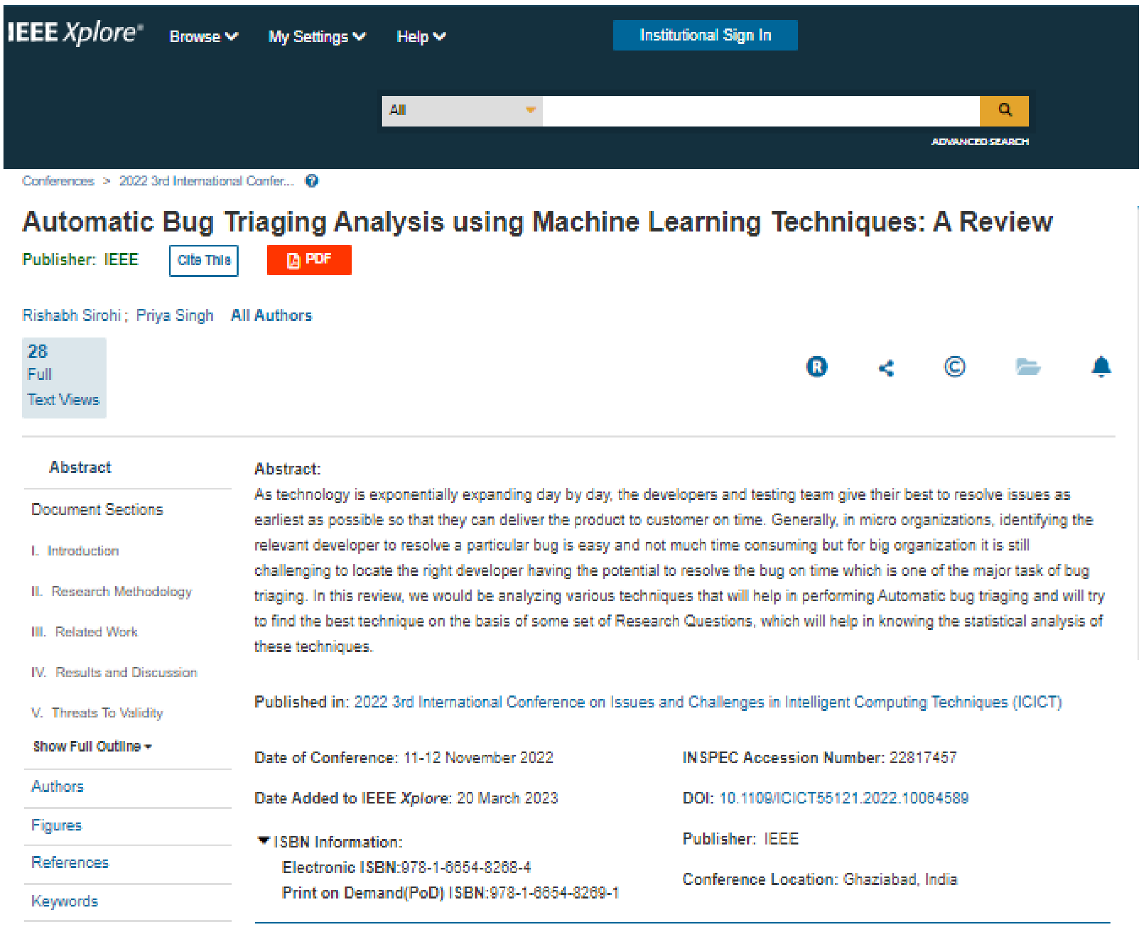
LIST OF PUBLICATIONS

Rishabh Sirohi, Priya Singh, "Automatic Bug Triaging Analysis using Machine Learning Techniques: A Review"

Name of Conference: 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)

Date of Conference: 11-12 November 2022

Date of paper publication: 20 March 2023



Automatic Bug Triaging Analysis using Machine Learning Techniques: A Review

Publisher: IEEE

Cite This

PDF

Rishabh Sirohi ; Priya Singh All Authors

28 Full Text Views

Abstract

Document Sections

I. Introduction

II. Research Methodology

III. Related Work

IV. Results and Discussion

V. Threats To Validity

Show Full Outline

Authors

Figures

References

Keywords

Abstract:
As technology is exponentially expanding day by day, the developers and testing team give their best to resolve issues as earliest as possible so that they can deliver the product to customer on time. Generally, in micro organizations, identifying the relevant developer to resolve a particular bug is easy and not much time consuming but for big organization it is still challenging to locate the right developer having the potential to resolve the bug on time which is one of the major task of bug triaging. In this review, we would be analyzing various techniques that will help in performing Automatic bug triaging and will try to find the best technique on the basis of some set of Research Questions, which will help in knowing the statistical analysis of these techniques.

Published in: 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)

Date of Conference: 11-12 November 2022

Date Added to IEEE Xplore: 20 March 2023

ISBN Information:
Electronic ISBN:978-1-6654-8268-4
Print on Demand(PoD) ISBN:978-1-6654-8269-1

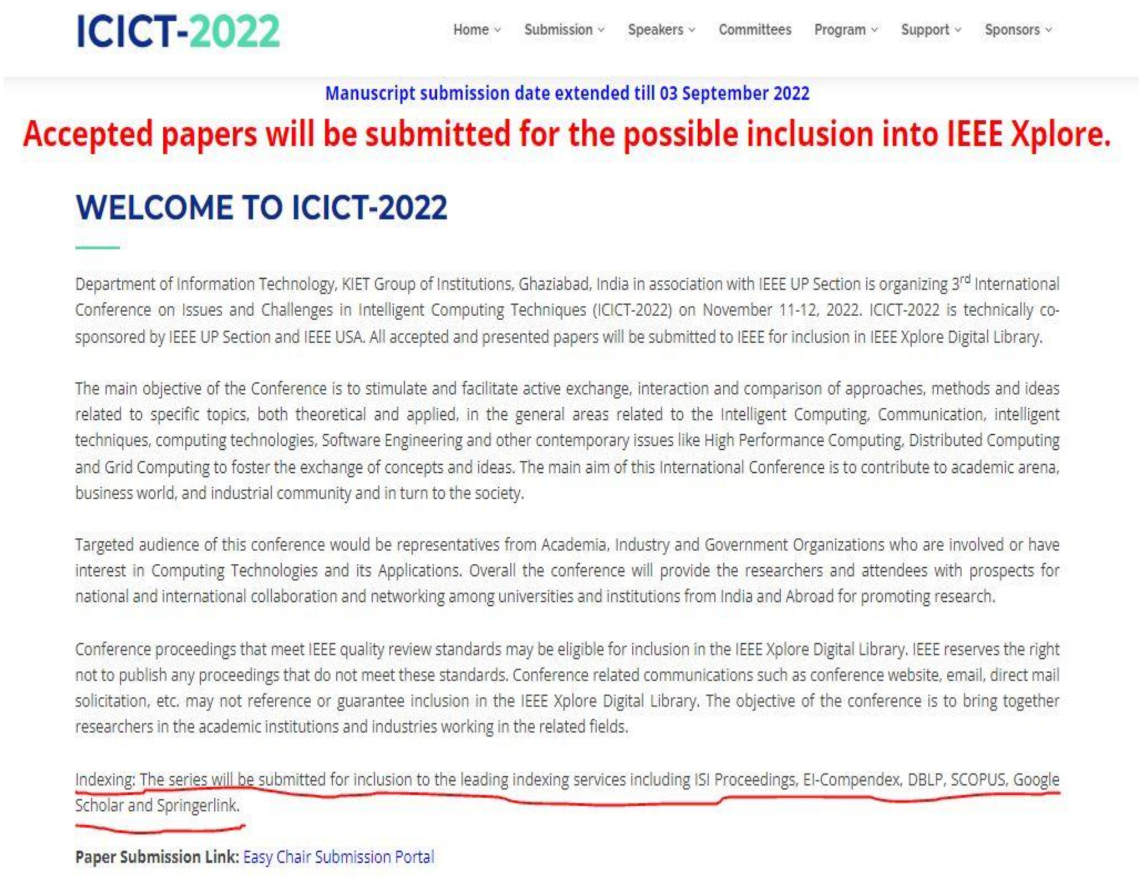
INSPEC Accession Number: 22817457

DOI: 10.1109/ICICT55121.2022.10064589

Publisher: IEEE

Conference Location: Ghaziabad, India

PROOF OF SCOPUS INDEXING



Manuscript submission date extended till 03 September 2022

Accepted papers will be submitted for the possible inclusion into IEEE Xplore.

WELCOME TO ICICT-2022

Department of Information Technology, KIET Group of Institutions, Ghaziabad, India in association with IEEE UP Section is organizing 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT-2022) on November 11-12, 2022. ICICT-2022 is technically co-sponsored by IEEE UP Section and IEEE USA. All accepted and presented papers will be submitted to IEEE for inclusion in IEEE Xplore Digital Library.

The main objective of the Conference is to stimulate and facilitate active exchange, interaction and comparison of approaches, methods and ideas related to specific topics, both theoretical and applied, in the general areas related to the Intelligent Computing, Communication, intelligent techniques, computing technologies, Software Engineering and other contemporary issues like High Performance Computing, Distributed Computing and Grid Computing to foster the exchange of concepts and ideas. The main aim of this International Conference is to contribute to academic arena, business world, and industrial community and in turn to the society.

Targeted audience of this conference would be representatives from Academia, Industry and Government Organizations who are involved or have interest in Computing Technologies and its Applications. Overall the conference will provide the researchers and attendees with prospects for national and international collaboration and networking among universities and institutions from India and Abroad for promoting research.

Conference proceedings that meet IEEE quality review standards may be eligible for inclusion in the IEEE Xplore Digital Library. IEEE reserves the right not to publish any proceedings that do not meet these standards. Conference related communications such as conference website, email, direct mail solicitation, etc. may not reference or guarantee inclusion in the IEEE Xplore Digital Library. The objective of the conference is to bring together researchers in the academic institutions and industries working in the related fields.

Indexing: The series will be submitted for inclusion to the leading indexing services including ISI Proceedings, EI-Compendex, DBLP, SCOPUS, Google Scholar and Springerlink.

Paper Submission Link: Easy Chair Submission Portal