International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

Analyze the Performance of Software by Machine Learning Methods for Fault Prediction Techniques

Nikita Gupta[1], Ripu Ranjan Sinha[2], Ankur Goyal[3], Neelam Sunda[4] and Divya Sharma[5]
[1]Department of Computer Science, RTU, Kota,
Rajasthan, India, nikitagupta.ssm@gmail.com
[2]Department of Computer Science, RTU, Kota,
Rajasthan, India, drsinhacs@gmail.com
[3]Department of Computer Science, Symbiosis Institute of Technology, Symbiosis International Deemed University,
Pune, Maharashtra, India, ankur_gg5781@yahoo.co.in
[4]Department of Computer Science, RTU, Kota, Rajasthan,
India, research.neelam@gmail.com
[5]Assistant Professor, Kanoria PG Mahila Mahavidyalaya, Jaipur, Rajasthan, India

Abstract— Trend of using the software in daily life is increasing day by day. Software system development is growing more difficult as these technologies are integrated into daily life. Therefore, creating highly effective software is a significant difficulty. The quality of any software system continues to be the most important element among all the required characteristics. Nearly one-third of the total cost of software development goes toward testing. Therefore, it is always advantageous to find a software bug early in the software development process because if it is not found early, it will drive up the cost of the software development. This type of issue is intended to be resolved via software fault prediction. There is always a need for a better and enhanced prediction model in order to forecast the fault before the real testing and so reduce the flaws in the time and expense of software projects. The various machine learning techniques for classifying software bugs are discussed in this paper.

Keywords- Fault, Machine Learning, Decision Tree, SVM, KNN, Ensemble Techniques,Software,Fault detection model.

I. INTRODUCTION

An error state that does not adhere to the software specifications or user expectations is referred to as a software defect in a software system. Unexpected and frequently inaccurate results produced by the programme are the result of a logic or coding error. During the programming the programmer or the software designer may make mistake, most of the errors are due to the such type of mistakes. The following are examples of flaws in software systems in reality: Arithmetic errors that arise from mistakes in certain arithmetic expressions; syntax errors brought on by the way the code was written. Logical errors are errors in the code's implementation, Performance flaws result in undesirable outcomes, Interaction between users and the software results in interface defects [1].

The availability and dependability of software systems are gravely threatened by software defects. Finding and fixing the system's flaws is very expensive once the flawed system has been put into place. By gaining crucial knowledge about the kind and location of defects, developers and programmers can profit from the prediction of unknown defects and increase the necessary level of confidence in the system. Prediction of software system flaws is currently one of the most researched topics by researchers [2].

Defect predictions, which assist programmers in locating bugs in malfunctioning code regions, allow programmers to prioritise their testing techniques according to the severity of the problematic code regions. Defect Prediction enables software testers and developers to evaluate the product's quality, determine whether or not quality standards are met, and determine whether the finished product satisfies users' needs and expectations. Additionally, it makes it easier to distribute resources for the system's formal verification as it is being developed [3].

Software has evolved into a crucial part of many systems as a result of the use of computer technology. The creation of these systems is becoming more difficult as software systems are integrated into daily life. Therefore, creating highly effective software is very difficult. The quality is still the most important feature of any software system out of all the desired attributes. By using out-dated methods, it is expensive to maintain product quality This could require a significant investment of time, money, and effort [4].

Prior to system testing, identifying fault-prone modules can help the software manager allocate resources to the right modules in order to cut costs and produce software that is almost error-free. The ultimate objective of these fault identification systems' design is to identify fault-prone modules as accurately as possible. In these approaches for prediction, software fault prediction is by far the most frequently researched topic, and numerous research centres have started brand-new initiatives in

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

this area. Models that are defective are predicted using software metrics and fault data [5].

The characteristics for delivering software quality through the application of metrics are well-established in the research community. The process has made extensive use of statistical methods. But over time, the trend has shifted away from conventional statistical methods and toward machine learning techniques. For fault prediction, various ML techniques have been employed. Software faults can occur at any stage of the software development process and are defined as programming errors that prevent the software from performing its essential function. They might originate from programming, design, or the outside environment. Some software flaws can result in anything from a straightforward calculation error to a complete system failure, depending on the type of fault. [6]

This type of issue is intended to be resolved by software fault prediction. There is always a need for better and better forecasts in order to pinpoint the defect before actual testing and thereby minimise delays and costs pertaining to software projects. Prediction of the fault can be done with respect to the severity, priority or for the developer classification. Not only on the bases of these categories only, classification may also be done depending on number of other ways also. Software modules are typically divided into classes that are defective or not. If modules contain a defect then it is represented as the faulty module, however the module having no defect is represented as the non-faulty module. Selecting a superior learning calculation seems to be just as important for developing a product fault forecast demonstration as selecting a programming measurement or other parameter.

Machine learning uses historical data to predict the likelihood of an event without explicit programming. The final result is predicted by machine learning, which focuses on the output variable and attempts to find patterns in the data. Machine learning algorithms come in two different varieties. Algorithms for supervised and unsupervised machine learning. In supervised algorithms, the output variable will be predetermined, and we will attempt to compare patterns between the dependent and independent variables. In unsupervised machine learning algorithms, the final outcome variable can be predicted by passing the data. The idea of a dependent variable does not exist in unsupervised machine learning algorithms. The algorithms generate the final outputs using all of the data [44].

The objectives of this paper is mentioned below:

- This study discusses the various fault prediction techniques in machine learning.
- This paper mention various datasets which is used in order to analyze the performance for the software fault prediction.
- This paper give thorough investigation of fault prediction using the integral approach (machine learning) and error probability methodology in order to foresee the error that occurred during the coding phases.

The remainder of the paper is organized as follows: Section 2 presents an overview of the Concept of Bug life Cycle; Section 3 describes in detail of Literature Review; Section 4 explained different Methods and Techniques; and finally Conclusions and future scope are described in Section 5.

II. CONCEPT OF BUG LIFE CYCLE

The importance of anticipating bugs earlier in the SDLC is demonstrated by the necessity of anticipating their occurrence and understanding the defect life cycle. Avoiding the time, expense, and effort required for defect detection and repair is beneficial. The quantity of bugs discovered during the development cycle is one of the major problems the modern software industry is dealing with. This causes the product's final delivery to be delayed and raises the overall cost of operation.

The average operation cost increase across projects is predicted to be 10% as a result of bugs that occur during software production and code development. We attempt to address the situation given the urgency of the hour. The model can be used as input to predict when a bug will occur before a tester notices it. As a result, the developer can review the code in the past and quickly address any problems. The model works well with the most well-known frameworks for software development, including waterfall, agile, V-shape, spiral, and others. We created this final set of machine learning models by keeping the various frameworks for software development modelling. This serves as a general representation for all defect prediction algorithms. One of the crucial elements is being aware of the defect management life cycle. The figure 1 shows the different phases of the bug life cycle [45].

- **New:** We refer to a defect as being new if it is reported by testers or anyone else for the first time during the "Software Development Life Cycle(SDLC)"
- **Assigned:** According to the seniority and priority of the bugs, each newly reported bug is assigned to one of the team members for immediate resolution. Then, the corresponding developer will carry out the necessary work.
- **Open:** If a bug is discovered after a developer's initial testing, it will be moved to the open state. Typically, a team of developers or product researchers can handle this. Fixed: Based on severity and priority, all bugs that are in the open state must be fixed. Within the software development life cycle, the stakeholders collectively determine the severity and priority of the bugs.
- **Fixed:** Based on severity and priority, all bugs that are in the open state must be fixed. Within the software development life cycle, the stakeholders collectively determine the severity and priority of the bugs.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

- **Retest:** If a bug is not tested or the description is incorrect, additional testing may be necessary to ensure that the reported bug is valid in all circumstances.
- **Closed:** If the bug is correctly fixed, this will be regarded as a bug that has been fixed.

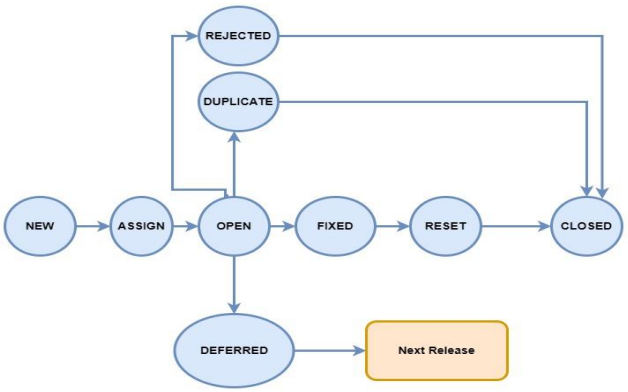

Figure 1: Different Phases of bug life cycle

- **Deferred:** Some of the bugs will be pushed to the upcoming release due to priorities. These insects have been identified as various
- **Duplicated:** The term "repeated bug" refers to a bug that has occurred more than once.
- **Rejected:** The tester raised the rejected bugs, which are not true bugs. These are primarily the result of a mismatch in skill sets and requirements.

As shown in the figure 2, a machine learning model may be used to pinpoint the software fault. The steps below are used to identify the software fault using the machine learning model.:

1. **Data Collection:** first of all, data need to be collected from various heterogonous resources of bug/fault repositories.
2. **Pre-processing:** Prior to applying the model, data need to be framed in order to deal with the missing values, duplicate values and noise etc.
3. **Feature Extraction:** After the prepressing, features are extracted from the data using the suitable technique like: 'Principal component analysis (PCA)', 'Linear discriminant analysis (LDA)' or 'Nonlinear dimensionality reduction via kernel principal component analysis (KPCA)' etc.
4. **Model Training:** Once the features extracted or decided, now the data is ready to build the model and to train the mode. So the model need to prepared using the appropriated algorithm like decision tree, SVM, KNN or ensemble approach. Almost 70% of the data is used for training of the model
5. **Model Optimization:** When it comes to algorithms, optimization is one of the most important factors. Learning optimization manages the cost function by greatly reducing the cost with the aid of the machine. The goal of many straightforward algorithms is to minimise the cost function by determining the parameters. Each parameter affecting the cost function is computed using the gradient descent approach [7-8].
6. **Model Evaluation:** A model's performance is assessed using a variety of performance measures, including recall, precision, accuracy, f-measure, and expectation time. If the performance of the model is not as per the expectation, then the model may be reconstructed or new feature may be selected or again optimized it depends on the requirement or situation. In other words, it can be said that the step 2-5, any step may be repeated [9-10].

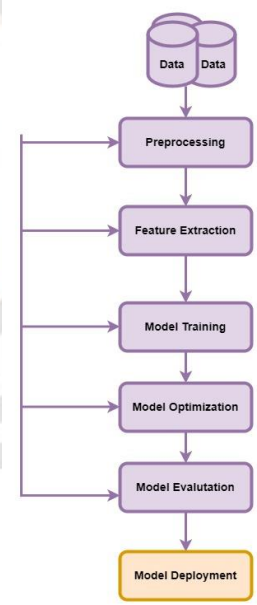

Figure 2: Fault Detection Model

III. LITERATURE REVIEW

Breu S et al., 2010 [11] studied how bug tracking systems, which are tools used for managing software projects, can be improved. Almost 600 defect reports from two projects, Mozilla and Eclipse, were viewed by the authors to examine the questions contained in the bug reports. They categorized the questions and measured how quickly and how often they were answered. The results showed that users play an important role in helping to solve the bugs they report, so they suggested four ways in which bug tracking systems can be improved. The main approaches discussed in this research work are qualitative and quantitative analysis. Qualitative analysis involves examining the data qualitatively, or in other words, looking at the data and making conclusions based on the patterns found. Quantitative analysis involves measuring the data in order to make more concrete and detailed conclusions. It was concluding that active

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

user participation is essential to being able to make progress on bugs reported through bug tracking systems.

Bhattacharya P and Neamtiu I, 2011[12] explained that predicting bug-fix time is helpful. The authors use two approaches in this paper— 'Regression Testing' and 'Machine Learning'—to assess the predictive power of existing models on 512,474 bug reports from different projects: Chrome, Eclipse and Mozilla. The testing results revealed that the explanatory accuracy of the existing models ranges from 30% to 49%, highlighting the importance of along with more predictor factors (attributes) when developing a prediction model. The authors also discovered that, unlike in commercial software, there is no connection between the likelihood of a bug being fixed, a bug-reputation, openers and as well as the time it takes to fix a bug in the projects that were examined. In this paper, a machine learning classification model is used, which is trained using the various input attribute values with respect to the expected output classes. After training, the model is presented with a set of input attributes and it predicts the most likely output class. It was concluded that existing models used for predicting bug-fix time have limited predictive power and need to be improved. The authors also discovered that, contrary to recent studies on commercial software, there is no connection between the likelihood of a bug being fixed, the reputation of the person who reported the bug, and the length of time it takes to fix a bug in the projects they looked at.

Zaineb, Ghazia, and IrfanAnjum Manarvi, 2011[13] quantifies the actual percentage of rejected bugs, and provides a list of causes that can lead to bug rejection. It also delves into the relation between severity levels and bug rejections, as well as discussing the effect of invalid bug reports on the efficiency of software testing. Eventually, the objective of this study is to take attention to the increasing rate of rejected bugs and how it can negatively impact software projects in terms of time and cost. Data collected from the bug tracking system and develops a cause-effect diagram to pinpoint the sources of invalid bug reports. It also looks into the defect rejection rate, reported bugs, time, effort, cost, and number of test cases executed in order to analyse the influence of rejected bugs. Finally, the paper provides recommendations on how to decrease the rejection rate in order to reduce the negative effects on software projects. Lastly, the authors provide correlation analysis between the number of reported bugs and rejected bugs, as well as recommendations on how to decrease the rejection rate.

An approach based on classification was put forth **by Kanwal J. and Maqbool O. in 2012[14]** to create a bug report recommender. The paper compares two classes of classifiers ('Naive Bayes' and 'Support Vector Machine') in order to assess the performance of the recommender. It also investigates the effects of various feature combinations on the accuracy of the results. The paper also introduces two new metrics to analyse the bug priority recommender's performance: 'Nearest False Negatives (NFN)' and 'Nearest False Positives (NFP)'. In short, this paper looks at how bug report management can be improved through the use of a classification-based recommender. When text features are used, it has been found that the 'Support Vector Machine' classifier outperforms the 'Nave Bayes' algorithm in terms of accuracy. For categorical features, the Naïve Bayes classifier outperforms SVM. It was observed that after combining the text and categorical features highest accuracy is witnessed. The paper also presents two new metrics, 'Nearest False Negatives (NFN)' and 'Nearest False Positives (NFP)', that add to our understanding of the bug priority recommender's output.

SJ Dommati et al., 2013 [15] explained the need for a structured mining algorithm that can be used to classify bugs. This algorithm would involve extracting features from bug reports, reducing noise in the data, and using probabilistic Naïve Bayes approach to classify network bugs. The performance of different algorithms using accuracy and recall parameters when given unseen bugs as input was compared. In other words, the paper looks at how a software program can identify different bugs and classify them correctly. In order to determine the class to which the bug should be assigned, a structured mining algorithm that uses the crash log as the input is presented by the author. This algorithm uses feature extraction, noise reduction, and classification of network bugs using the probabilistic Naïve Bayes approach.

MDM Suffian and S Ibrahim, 2014[16] creates a prediction model for defects in system testing. This model's primary objectives are to assist the testing team in managing and controlling the test execution process as well as to act as an early indicator of the quality of the software entering system testing. Metrics from earlier system testing phases are found and analysed to create this model. Regression analysis is then used to create mathematical equations using this data. The desired prediction model has p-value less than 0.05 and R-squared (adjusted) values greater than 90%. Finally, this model is proved with new projects to make sure it is suitable for actual implementation. The author came to the conclusion that it is feasible to develop a prediction model for system testing flaws that can act as a preliminary gauge of the calibre of the software being submitted for system testing. This model is based on metrics gathered from earlier system testing phases, and its suitability for actual implementation is confirmed with new projects.

Zhang, Yong, et al, 2014 [17] described how to use multiple individual classifiers (e.g., algorithms) together in order to achieve better performance than using just one single classifier alone. The idea behind this approach, called ensemble learning, is that different base classifiers can contribute differently towards the final classification result and so it makes sense to

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

assign greater weights or importance values for those with higher accuracy rates. To do this efficiently, the authors propose an optimization technique known as differential evolution which helps determine these weightings automatically based on their effectiveness during testing of various scenarios. Finally, once all the weights have been determined they are combined according to what's referred to as weighted voting combination rule - essentially meaning each vote counts more if it's from a model/classifier deemed more accurate by differential evolution earlier on in the process.

Prasad, M. C., Lilly Florence, and Arti Arya, et al., 2015 [18] proposed classification techniques for software defect prediction using software metrics. These techniques involve the use of data mining to identify defects based on existing software metrics, which can help improve overall quality and performance. The specific methods mentioned include decision tree algorithms, 'Support Vector Machines (SVMs)', 'Artificial Neural Networks (ANNs)' and logistic regression models. Each technique has advantages and disadvantages depending on the type of problem being solved or analysed. It was concluded that software defect prediction models can be used to recognise potential defects in a product and help developers improve the quality of their code.

Cross-project defect prediction was the subject of an exploratory study by **Satin, Ricardo FP et al. in 2015 [19].** The study examines the effects of applying various classification algorithms and a performance indicator when creating predictive models. The study applies various classification, feature selection, and clustering algorithms to 1270 projects in order to identify and group alike projects. The study then evaluates the performance of the different algorithms and measures the performance achieved through the application of the algorithms. The Naive Bayes algorithm performed the best, the study's results show, with 31.58% accurate predictions in 19 models that used it.

Jin, Kwanghue, et al., 2016 [20] proposed a method for improving bug severity prediction. This entails incorporating text and meta-fields from bug reports into our classifier model, as well as including 'normal' severity bugs, which account for a sizable portion of the total number of reported bugs. These two approaches have not been taken into consideration by other studies before this one, so it provides an innovative way for predicting bug severity more accurately than ever before.

Li, Jian, et al., 2017[21] proposed a new method for improving software reliability by predicting potential bugs in the code. The author proposed the framework using the Convolution Neural Network to predict the bug/defect. The proposed approach generates features from programmes' Abstract Syntax Trees using deep learning techniques (ASTs). The ASTs are first transformed into token vectors, which are then encoded as numerical vectors by using word embedding and mapping methods. These numerical vector representations of the program code can be fed into a convolutional neural network model that automatically learns semantic and structural information about the program's functionality. Finally, both the learned feature set generated by DP-CNN along with traditional hand crafted features will be used together for more effective software defect prediction results compared to existing methods. Studies reveal that this strategy can up to 12% more accurately predict defects than current methods.

Xuan, Jifeng, et al., 2017 [22] offered a new approach to bug triage, which is the process of sorting and categorizing bugs in software development. This proposed method combines both supervised (using labelled data) and unsupervised (using unlabelled data) techniques such as naive Bayes classifier and expectation maximization to take advantage of all available information from both types of reports. Additionally, the authors use weighted recommendation lists that assign weights according to multiple developers' opinions when training the model for better accuracy results on Eclipse's bug report dataset. In summary, proposed semi-supervised text classification approach improves upon existing methods by providing more accurate predictions with fewer labels required for training purposes.

Gomes, Heitor Murilo, et al., 2017[23] proposed a taxonomy (a system or structure) for understanding different types of ensemble learning algorithms used with data streams, such as combination techniques, diversity measures and dynamic updates like adding/removing classifiers from the model when needed. It also provides an overview on open source tools available related to these topics as well as discussing current challenges faced by researchers working with big datasets that evolve over time (concept evolution), feature drifts etc.,

Singh, V. B et al., 2017[24] used machine learning algorithms for automatic bug prediction. The authors used text mining techniques to extract summary terms from historical data which were then used as training candidates in order to develop models for making predictions on different levels of severity (e.g., low, medium or high). Author also employed two approaches called Vote and Bagging ensemble methods in order to deal with imbalanced datasets where some classes have more examples than others. Finally, they tested their approach by comparing results between Eclipse projects and Mozilla products using k-NN classifiers (which performed better) as well as SVM classifiers; showing that cross project predictions are possible even when dealing with different domains/products. In terms of various performance measures, it was found that k-NN classifiers outperformed SVM. Naive Bayes-measure was below 34.25%. All severity levels, including those with fewer bug reports, saw an improvement in accuracy and f-measure after developing training candidates by combining multiple datasets. The overall performance of the f-measure was

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

improved by 5% and 10% respectively by the two ensemble approaches (vote and bagging), when dealing with imbalanced datasets. Finally, cross project predictions between Eclipse projects and Mozilla products were successful using both k-NN as well as SVM classifiers; showing that reliable models can be built even across domains/products

Alia, SayedaShamma, et al., 2018[25] offered a newmethod called Class-Membership Information of a Term (CMT) which can be used to classify and predict the severity of software bugs. This approach does not require any prior knowledge or parameter tuning, making it computationally simple compared to current approaches. Studies done on three benchmark datasets reveal that CMT outperforms other cutting-edge techniques by up to 5% and 12.5%, respectively, in both within-project classification and cross-project classification.

By utilising text mining, **Katerina Goseva-Popstojanova and Jacob Tyo, 2018 [26]**, were able to categorise software bug reports into categories related to security and those that weren't. It employs three different types of feature vectors in both supervised (where labels are provided for the data) and unsupervised learning approaches. For supervised learning, multiple classifiers were tested on different sized training sets while a novel unsupervised anomaly detection method was proposed as well. The evaluation was done based on NASA datasets which showed that better performance can be achieved when more security information is available in the dataset used for classification tasks.

The process of identifying named entities that are specific to software bugs in a software bug repository was described by **Zhou, Cheng, et al., 2018 [27].** A software bug repository is a database that stores information about software bugs, such as the type of bug, the severity of the bug, and the date it was reported. Named entities are words or phrases that refer to specific people, places, or things. Knowing software bug-specific named entities in a “software bug repository” involves identifying these words or phrases in the bug reports and categorizing them according to their meaning. This process can help software developers better understand the bug reports and make it easier to fix the bugs.

Kumar, Raj, et al., 2019[28] discussed the issue of "software bugs", which are errors that occur during design or development, and cost a lot in terms of time and money. To detect these bugs more efficiently, data mining techniques may be applied on large repositories called bug repositories - this will help extract hidden information from them. Finally, numerous types of classification methods using data mining have been studied in order to compare their accuracy when detecting such issues with precision recall metrics like F-measures etc.

Using an algorithm, Otoom, **Ahmed Fawzi et al., 2019 [29]** divide incoming bug reports into corrective (defect fixing) and perfective categories (major maintenance). The classification model used keywords as features, meaning it looked for certain words in each report to determine its category. After testing their feature set on three different open source projects, they achieved high accuracy with SVM classification algorithms reporting an average accuracy rate of 93.1%. In other words, this system could accurately identify whether a given bug report should be classified as either corrective or perfective almost all the time!

Aindrila Sarkar et al., 2019[30] conducted research at Ericsson to reduce development costs by automating the process of bug triaging. Bug triaging involves correctly assigning bugs (errors or issues) to the right developer or team for resolution. The researchers applied existing approaches from literature on over 10,000 bug reports across 9 large products and found that using simple textual and categorical attributes in logistic regression classifier gave them highest precision (78%) and recall (79%). They also tried adding crash dumps/alarms information but it did not improve accuracy significantly so they developed an approach where only high confidence predictions were made which improved accuracy up to 90%, however this could be done for 62% of all reported bugs.

Cai, Xingjuan, et al. in 2020[31] presented an approach using SVM that selects the parameters for the SVM and deals with datasets where there is class imbalance**.** To accomplish this, they propose a hybrid multi-objective cuckoo search under-sampled software defect prediction model based on SVM (HMOCS-US-SVM) that synchronously solves both of these issues. The false positive rate (pf), probability of detection (pd), and G mean are three indicators that this method uses to gauge how well it performs when put to the test against eight distinct datasets from the Promise database. Results demonstrate that their approach works improved than other existing models for solving the problem of software defect prediction.

Kumar, Raj, and Sanjay Singla, 2020[32] exposed data mining algorithms can be used to classify software bugs according to their severity. It contrasts the accuracy, precision, recall, and execution time of four different algorithms (SVM, KNN, Decision Tree, and Naive Bayes). SVM has highest accuracy, according to the results, while decision trees and Naive Bayes perform well in terms of other factors like execution time.

Using the information from the bug report, **Kim, Sunghun, et al., 2021[33]** described how a fault detection model can be produced. However, this data can contain a lot of noise because the current practices for collecting defects rely on voluntary bug fix keywords or links to bug reports in change logs. Authors proposed approaches to deal with that noise and measure its impact on the accuracy of two well-known algorithms used for predicting defects. It also introduces an algorithm which detects

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

noisy instances and eliminates them so as to improve overall performance when making predictions about potential bugs in code.

Neighbourhood-based under-sampling (N-US), which **Goyal, Somya, and 2021 [34]** proposed, can be used to address class imbalance in software defect prediction (SDP). Class Imbalance means that the dataset has more data points belonging to one particular class than another. This algorithm under samples the dataset, meaning it reduces its size by removing some data points from majority classes while keeping all minority classes intact so as not to lose any information. Using five benchmark datasets from the NASA repository—KC1, KC2, JM1 and CM1—the proposed N-US approach was associated with three standards under sampling techniques and found to be superior in terms of accuracy, AUC score, and ROC curve position.

Koksal, Omer, and Bedir Tekinerdogan, 2022[36] proposed an automated bug classification for large-scale software projects. Defect reports are pieces of information sent by users to the developers when they encounter an issue with their software, and classifying them correctly helps identify what caused the problem so it can be fixed quickly. In this study, text mining, machine learning (ML) and natural language processing (NLP) approaches were used to classify bugs in both English and Turkish languages from commercial software systems - something which hasn't been done before. The results showed that automatic bug classification was more accurate than manual methods while also reducing time spent on manually categorizing each report.

IV. METHODS AND TECHNIQUES:

Bug dataset(s) may be taken from the various bug repositories like Mozilla, Firefox, Eclipse, Jira etc. for the study [37]. Fault detection/analysis can be done using the statistical as well as the machine learning algorithms as shown in the figure 3.

Statistical Methods for Fault Analysis: Static analysis is a type of automated testing that looks for errors or defects in the code before it's released to customers. It can help identify issues like incorrect data types, missing checks on user input, and other programming mistakes which could lead to security vulnerabilities if left unchecked. Software reviews involve manual inspection by developers or testers who look at the source code line-by-line looking for potential problems with logic flow, coding style violations, etc., as well as any bugs they may find while doing so. Both methods have their advantages; however, when used together they provide an effective way of detecting faults prior to release of a product ensuring its quality meets customer expectations [38].

Machine Learning Algorithms for Fault Detection: As shown in Figure 3, a dataset (or datasets) must be pre-processed in order to remove anomalies before applying any machine learning algorithm for fault detection [28]. Proceeding further features can be selected using the appropriate feature selection algorithm like Principal Component Analysis (PCA). The method used for the feature need to be selected on the basis of the dataset and the basis of the machine learning algorithm that we are going to use for the fault detection. The module will be classified as defective or not defective using a supervised machine learning approach. There are many supervised machine learning algorithms that can be used, including Decision Tree, Support Vector Machine (SVM), Naive Bayes, and K-Nearest Neighbour (KNN). Sometimes to achieve the high efficiency ensemble approaches may be used.

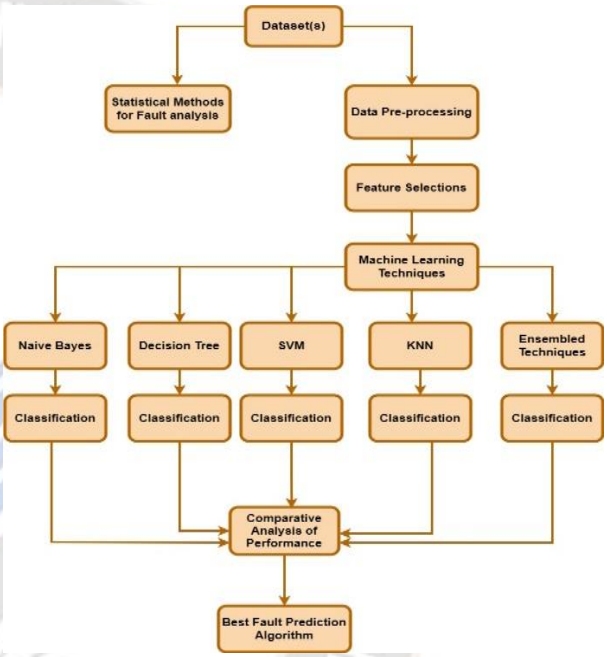

Figure 3: Methodology for Fault Detection.

- **Naïve Bayes:** Naive Bayes classifier is one of the most commonly used machine learning algorithm which uses Bayes theorem and is specifically designed to classify continuous variables. A naive Bayes classifier (sometimes called a hidden Markov model) is a statistical model that assumes that the probabilities of the parameters or features are independent of the data. The model uses these assumptions of independence to find a joint probability distribution over all possible parameter values. [39]. It is a probabilistic classifier and posterior probability is calculated using the Bayes theorem. The impact of a predictor's value (y) on a particular class (m) is thought to be independent of the value of other predictors [46]. Following equating used in Naïve Bayes algorithm.

$$P(m|y) = \frac{P(y|m)p(m)}{P(y)} \qquad (1)$$

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

P(m) denotes the class's prior probability, P(m|y) the class's posterior probability for a specific predictor (attribute), P(y|m) denotes the likelihood, or probability, of the predictor given a class, and p denotes the predictor's prior probability (y).

- **Decision Tree:** Decision tree is a powerful tool to analyse the data and generate values associated with the variables. It must be said that decision tree is not a linear algorithm that makes calculations by series of simple steps (if, then, else case), but rather an interactive system of decisions, based on the results of which a new decision can be made for each node in the tree. This approach allows us to analyse complex chains of events represented by numerous interrelated variables and their possible combinations. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In most of the cases DT is used for the classification, however it can also be used for the regression also. DT algorithms include the Iterative Dichotomies 3 (ID3) and ID3 Successor (C4.5) algorithms, the Classification and Regression Tree (CART) algorithm, the Chi-squared Automatic Interaction Detector (CHAID) algorithm, the Multivariate Adaptive Regression Splines (MARS) algorithm, the Generalized, Unbiased Interaction Detection and Estimation (GUIDE) algorithm, and the Conditional Inference Trees algorithm (CTREE) [42].
- **SVM:** Support vector machines (SVMs), models based on supervised learning and coupled with learning algorithms, offer a useful method for analysing data based on classification and regression analysis. One of the benefits of SVM is that it performs better than other model techniques with fewer characteristics, while another is that it has a more robust model. A third benefit is that SVM takes less time to compute than other methods like neural networks [40-41]. To differentiate the data points, the hyperplane in used in the SVM algorithm. Equation

  $$x+b=0 \qquad (2)$$

  represents the hyperplane
  where b is an offset and w is a vector's hyperplane normal.
  We can say that a point is positive if the value of w. x+b>0; otherwise, it is negative. So that the margin has a maximum distance, we now need (w,b).
  Support vector machines' benefits include [47]:
  - Efficient in environments with high-dimensions.
  - This method is still effective when the number of dimensions exceeds the number of samples.
  - Because the decision function only uses a portion of the training points (also referred to as support vectors), it is also memory-efficient.
  - The decision function is flexible in that different Kernel functions can be specified for it. Users can specify their own kernels in addition to the widely used ones that are available.

  The following are some drawbacks of support vector machines [47]:
  - If the number of features is significantly greater than the number of samples, it is imperative to avoid over-fitting when choosing kernel functions and regularisation terms.
  - Instead of directly providing probability estimates, SVMs use an expensive five-fold cross-validation procedure (see Scores and Probabilities, below).
- **KNN:** K nearest neighbour (KNN) is a method of machine learning to classify new data points based on their similarity to existing ones. KNN uses the distance (similarity) of two data points to each other and calculates a small number of neighbours that are more similar than the original data point, using them to calculate the classification prediction. K represents the number of the neighbouring data points. The accuracy of the KNN algorithm depends on the selection of the value of the 'K' and distance metrics [41]. Following equation represents, how distance can be calculated between two data points e.g. x and y.

$$D(x,y)=\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{1/p} \qquad (3)$$

  - Manhattan Distance (when value of p is1)
  - The Euclidean Distance is applied when p = 2
  - Chebychev Distance (when value of p = ∞)

Ensemble Techniques: An advanced meta machine learning algorithm that has gained favour recently is ensemble classification approaches. To improve prediction performance, these strategies aggregate predictions from various learning algorithms. In general, the inherent principles and sensitivity to training data of various machine learning algorithms varies. As a result, based on a set of data, various categorization approaches produce various predictions. These many outcomes are used by ensemble machine learning approaches to produce improved prediction outcomes. These methods aim to improve prediction accuracy from any one of the individual learning algorithms by reducing the bias and variance of prediction

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

models. Bagging, Boosting, Stacking and Voting are the some of the ensemble machine learning techniques [43].

V. CONCLUSION AND FUTURE SCOPE:

By foreseeing defects before the testing process, software fault prediction reduces the need for fault discovery activities. Additionally, it assists in expediting software quality assurance efforts to be used in the latter stages of software development and better utilising testing resources. We have found that measuring software fault proneness is exceedingly complicated, confusing, and multifaceted. At any stage of the software development process, a flaw can be found. During the testing phase, some problems go unnoticed and are sent. In this a complete methodology used for machine learning algorithms for the fault prediction is explained. Various datasets may be taken in order to analyse the performance for the software fault prediction. Python, R, Weka, Orange, Matlab etc. simulation tools may be used for the implementation. After doing the software bug classification using various machine learning algorithms, a comparative analysis will be done on the basis of the various performance measures like accuracy, recall, sensitivity, F-measure etc. in order to find the best prediction algorithmin.

REFERENCES:

[1] M. L. Hutcheson, Software testing fundamentals: Methods and metrics. John Wiley & Sons, 2003.

[2] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in Proceedings of the 30th international conference on Software engineering, 2008, pp. 181–190.

[3] R. Wahono, "A systematic literature review of software defect prediction," Journal of Software Engineering, vol. 1, no. 1, pp. 1–16, 2015.

[4] Ankur Goyal, Likhita Rathore, Sandeep Kumar,"A survey on solution of imbalanced data classification problem using smote and extreme learning machine",Pages 31-44, Communication and Intelligent Systems: Proceedings of ICCIS 2020.

[5] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study," Software process: Improvement and practice, vol. 14, no. 1, pp. 39–62, 2009.

[6] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," Software quality journal, vol. 18, no. 1, pp. 3–35, 2010.

[7] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in Proceedings of the sixteenth international conference on software engineering & knowledge engineering, 2004, pp. 1–6.

[8] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in Proceedings of the 28th international conference on Software engineering, 2006, pp. 361–370.

[9] Aishwary Kulshreshta, Ankur Goyal," Image Steganography Using Dynamic LSB with Blowfish Algorithm", International Journal of Computer & Organization Trends, Vol 3 ,2013.

[10] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in Proceedings of the 2006 international workshop on Mining software repositories, 2006, pp. 119–125.

[11] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," in Proceedings of the 2010 ACM conference on Computer supported cooperative work, 2010, pp. 301–310.

[12] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?" in Proceedings of the 8th Working Conference on Mining Software Repositories, 2011, pp. 207–210.

[13] G. Zaineb and I. Manarvi, "Identification and analysis of causes for software bug rejection with their impact over testing efficiency," International Journal of Software Engineering & Applications, vol. 2, no. 4, p. 71, 2011.

[14] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," Journal of Computer Science and Technology, vol. 27, no. 2, pp. 397–412, 2012.

[15] S. J. Dommati, R. Agrawal, S. S. Kamath, and others, "Bug Classification: Feature Extraction and Comparison of Event Model using Na\backslash" ive Bayes Approach," arXiv preprint arXiv:1304.1677, 2013.

[16] M. D. M. Suffian and S. Ibrahim, "A prediction model for system testing defects using regression analysis," arXiv preprint arXiv:1401.5830, 2014.

[17] Y. Zhang, H. Zhang, J. Cai, and B. Yang, "A weighted voting classifier based on differential evolution," in Abstract and Applied Analysis, 2014, vol. 2014.

[18] M. Prasad, L. Florence, and A. Arya, "A study on software metrics based software defect prediction using data mining and machine learning techniques," International Journal of Database Theory and Application, vol. 8, no. 3, pp. 179–190, 2015.

[19] R. F. Satin, I. S. Wiese, and R. Ré, "An exploratory study about the cross-project defect prediction: Impact of using different classification algorithms and a measure of performance in building predictive models," in 2015 Latin American Computing Conference (CLEI), 2015, pp. 1–12.

[20] K. Jin, A. Dashbalbar, G. Yang, J.-W. Lee, and B. Lee, "Bug severity prediction by classifying normal bugs with text and meta-field information," Adv. Sci. Technol. Lett, vol. 129, pp. 19–24, 2016.

[21] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in 2017 IEEE international conference on software quality, reliability and security (QRS), 2017, pp. 318–328.

[22] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," arXiv preprint arXiv:1704.04769, 2017.

[23] H. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," ACM Computing Surveys (CSUR), vol. 50, no. 2, pp. 1–36, 2017.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5s
DOI: https://doi.org/10.17762/ijritcc.v11i5s.6642
***Article Received**: 24 February 2023 **Revised**: 06 April 2023 **Accepted**: 23 April 2023*

[24] V. Singh, S. Misra, and M. Sharma, "Bug severity assessment in cross project context and identifying training candidates," Journal of Information & Knowledge Management, vol. 16, no. 1, p. 1750005, 2017.

[25] S. Alia, M. Haque, S. Sharmin, S. M. Khaled, and M. Shoyaib, "Bug severity classification based on class-membership information," in 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR), 2018, pp. 520–525.

[26] K. Goseva-Popstojanova and J. Tyo, "Identification of security related bug reports via text mining using supervised and unsupervised classification," in 2018 IEEE International conference on software quality, reliability and security (QRS), 2018, pp. 344–355.

[27] C. Zhou, B. Li, X. Sun, and H. Guo, "Recognizing software bug-specific named entity in software bug repository," in 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), 2018, pp. 108–10811.

[28] R. Kumar, S. Singla, R. K. Yadav, and D. Kumar, "An experimental analysis of various data mining techniques for software bug classification," International Journal of Innovative Technology and Exploring Engineering, vol. 8, no. 8, pp. 108–113, 2019.

[29] A. F. Otoom, S. Al-jdaeh, and M. Hammad, "Automated classification of software bug reports," in proceedings of the 9th international conference on information communication and management, 2019, pp. 17–21.

[30] A. Sarkar, P. C. Rigby, and B. Bartalos, "Improving bug triaging with high confidence predictions at ericsson," in 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 81–91.

[31] X. Cai et al., "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," Concurrency and Computation: Practice and Experience, vol. 32, no. 5, p. e5478, 2020.

[32] R. Kumar and S. Singla, "Multiclass Severity Classification for Software Bugs Using Support Vector Machine, K-Nearest Neighbor, Decision Tree and Naïve Bayes," Journal of Computational and Theoretical Nanoscience, vol. 17, no. 11, pp. 5109–5112, 2020.

[33] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in 2011 33rd international conference on software engineering (ICSE), 2011, pp. 481–490.

[34] S. Goyal, "Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction," Artificial Intelligence Review, vol. 55, no. 3, pp. 2023–2064, 2022.

[35] S. Sohrawardi, I. Azam, and S. Hosain, "A comparative study of text classification algorithms on user submitted bug reports," in Ninth International Conference on Digital Information Management (ICDIM 2014), 2014, pp. 242–247.

[36] Ö. Köksal and B. Tekinerdogan, "Automated classification of unstructured bilingual software bug reports: An industrial case study research," Applied Sciences, vol. 12, no. 1, p. 338, 2021.

[37] A. Hamdy and G. Ezzat, "Deep mining of open source software bug repositories," International Journal of Computers and Applications, pp. 1–9, 2020.

[38] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, "On the value of static analysis for fault detection in software," IEEE transactions on software engineering, vol. 32, no. 4, pp. 240–253, 2006.

[39] I. Rish and others, "An empirical study of the naive Bayes classifier," in IJCAI 2001 workshop on empirical methods in artificial intelligence, 2001, vol. 3, no. 22, pp. 41–46.

[40] R. Gholami and N. Fakhari, "Support vector machine: principles, parameters, and applications," in Handbook of neural computation, Elsevier, 2017, pp. 515–535.

[41] Ankur Goyal, Vivek Kumar Sharma," Modifying the MANET routing algorithm by GBR CNR-efficient neighbor selection algorithm", International Journal of Innovative Technology and Exploring Engineering,Vol 8,Issue 10,page no-912-917,2019.

[42] B. Charbuty and A. Abdulazeez, "Classification based on decision tree algorithm for machine learning," Journal of Applied Science and Technology Trends, vol. 2, no. 1, pp. 20–28, 2021.

[43] A. Goyal and N. Sardana, "Empirical analysis of ensemble machine learning techniques for bug triaging," in 2019 Twelfth International Conference on Contemporary Computing (IC3), 2019, pp. 1–6.

[44] S. D. Immaculate, M. F. Begam, and M. Floramary, "Software bug prediction using supervised machine learning algorithms," in 2019 International conference on data science and communication (IconDSC), 2019, pp. 1–7.

[45] A. Agrawal, A. Choudhary, and H. Sharma, "An Empirical Study on the Issues of Traditional Defect Life Cycle in Agile Model," 2019.

[46] K. Vembandasamy, R. Sasipriya, and E. Deepa, "Heart diseases detection using Naive Bayes algorithm," International Journal of Innovative Science, Engineering & Technology, vol. 2, no. 9, pp. 441–444, 2015.

[47] L. Cai and T. Hofmann, "Hierarchical document categorization with support vector machines," in Proceedings of the thirteenth ACM international conference on Information and knowledge management, 2004, pp. 78–87.