An Effective Software Defect Estimation Framework using Isolation Forest Algorithm: (IFDE-Framework)

Tehseen Fatma[1], Dr. Najeeb Ahmad Khan[2], Dr. Saoud Sarwar[3]

1&3: Department of Computer Science and Engineering, Al-Falah University, Faridabad, Haryana.
2: Department of Computer Science, Arunachal University of Studies, Namsai, Arunachal Pradesh

Abstract: Software metrics are affected by many factors, including reliability, usability, integrity, and maintainability. The metrics are important for measuring software performance, planning work items, measuring productivity, debugging, and estimating cost. In order to estimate the expected delivered quality and maintenance effort, several industries seek to investigate the number of bugs in software modules before they are delivered. Hence, automated defect estimation has been a crucial and fundamental task in the field of software development. In order to assist with this endeavor, a huge number of software metrics and statistical methods have been proposed by researchers, and an equally extensive body of written material has been produced. Current software frameworks are generally huge and complicated; they have a large number of associated metrics that capture various elements of the software modules. This study reviewed all the prediction techniques and discussed various projects that have been studied in recent years. Furthermore, using the results of this study, we have proposed an Isolation Forest Defect Estimation (IFDE-Framework) for identifying defects in software as it can detect previously unknown and abnormal behaviors. Moreover, we can provide difficult challenges for the successive stage in the process of software defect prediction.

Keywords: defect estimation, software metrics, software defect prediction,

1. INTRODUCTION

In the field of software development, defect prediction has become a challenging task for researchers. A vast number of studies have been proposed in this domain [1]–[5]. Defect prediction techniques are useful for identifying bug-prone code. Since quality assurance teams have not much amount of resources to investigate the software, they can easily utilize these bug-prone codes for software testing [6].

According to the report of Gartner, $3.8 trillion was spent on enterprise software globally in 2014 [7], with 23% of the budget going toward quality control and testing [8]. The enormous budget that is allotted for testing purposes demonstrates the significance and importance of software testing in the software development life cycle. On the other hand, existing software systems are particularly huge and complicated, which makes them very vulnerable to errors. Software packages are regularly modified, including changes to various components, including processes, products, and resources[9]. When programmers switch between projects or acquire new tools, they frequently change the available resources. This change makes software systems broad, complicated, and growing. Therefore, constructing measurable objectives and assuring the quality of such complicated and rapidly growing software systems is extremely difficult.

In order to construct an estimation framework, the first thing that needs to be done is to produce instances from software archives. These archives may include VCS (version control systems), email archives, issue tracking systems, and so on. Depending on the granularity of the prediction, an instance may refer to a software package, a file of source code, and a method. Several features or metrics are retrieved from software archives which have a label that indicates whether this instance is buggy or clean or how many bugs it has. In Figure 1, B stands for buggy; C stands for clean and/or number of bugs.

Once all the instances have been generated, we can prepare our data by using well know preprocessing machine-learning techniques. Preprocessing includes feature extraction, normalization of data, and reducing any unwanted behavior that is meaningless (noise). [10] Although, for software defect prediction, preprocessing of instances is an optional step, we can also avoid this step.[11]

Finally, the instances can be trained on the estimation framework, as demonstrated in Figure 1. Now this framework is able to determine whether a newly created instance contains a bug or not. A regression framework can be used to estimate the total number of bugs in an instance, whereas a binary classifier can be used to predict whether or not an instance will include any flaws.

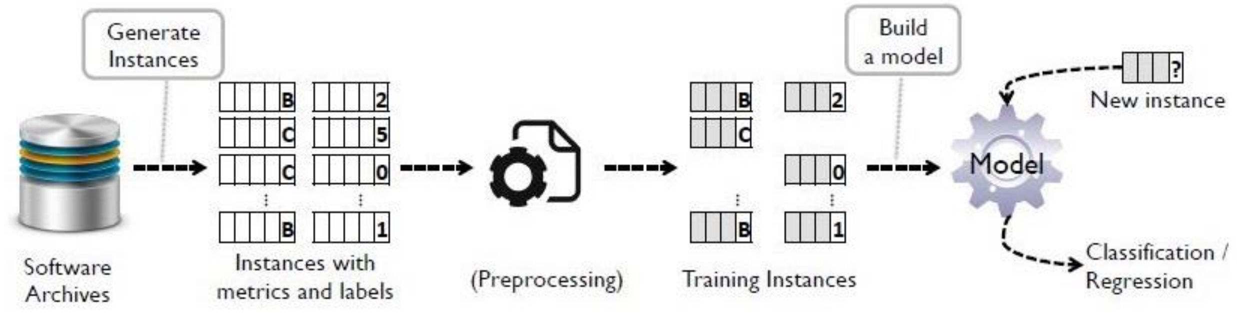

Figure 1. Common software defect prediction process as discussed in [5]

Numerous papers propose statistical frameworks and metrics that, taken together, claim to be able to resolve the quality dilemma. The quality of a software system can be defined in many ways. In the same way, defects can also be described in a wide variety of ways; nevertheless, the most frequent description of a defect is a deviation from a set of expectations or standards that has the potential to result in operational failure.

In general, efforts concentrated on the three core problems discussed in [12]–[14]

1. Total number of defects in the system
2. Calculation of system reliability (time to failure)
3. Finding out how the development and testing phases affect the frequency and severity of defects

Numerous different kinds of prediction models have been suggested by the researchers. Many authors have attempted to predict how many bugs a system will reveal during the execution or testing by using complexity and size measurements. To predict the failure rates, a reliability framework has been proposed. Furthermore, Defects have been predicted using data gathered during testing and the defect detection procedure.

There are a number of different works of literature have been proposed that contain descriptions of each metric. Nevertheless, in order to maintain the integrity of our study, we provide a summary of the most relevant software metrics.

Source line of Code (SLOC) or Line of Code (LOC) Metric

It is used to determine the size of a computer program that includes the total number of executable lines. Blank lines, comments, and non-executable instructions do not consider in LOC. A large number of research have demonstrated a rough correlation between LOC metrics and software defects [15][42].

The McCabe Cyclomatic Complexity (MCC) Metric

To test the complexity of a source code, the MCC metric is used. It determines how many decision logic are present in the source code of the program. MCC can also be applied to a particular function, a particular class and/or a particular module within the program. Yu, et, al., [16] prove that MCC is an important metric for software defects prediction[43].

Object Oriented (OO) Metric

Various object-oriented metrics have been proposed for software defect prediction. Some of them are; Coupling between objects (CBO) [17], Response for a class (RFC) [17] , Message passing coupling (MPC) [18], and Data abstraction coupling (DAC) [18]. This study presents an analytical review for the purpose of pointing researchers in the right direction of defect prediction.

Table 1: Defects Prediction Strategies

Metric types	Description	Ref.
Past defects	Past defects in the system are used to predict future defects.	[19]
Process metric	Modifications in source code are the main cause of these bugs.	[20]
Changes of Entropy	There is a higher chance of making an error with a complex modification than a simple one.	[21]
Source code metric	It's more difficult and error-prone to make changes too complicated modules.	[22]
Entropy (metrics of source code)	Metrics based on the source code are more accurate in describing the entropy of modifications.	Novel

2. LITERATURE SURVEY

In this article, we will discuss a number of different techniques to defect prediction, as well as the types of data that are necessary for each method and the different data sets on which it was verified. Although each method needs a defective archive in order to be validated, carrying out the analysis it does not always require access to the archive. When this happens, we clarify it.

Finding and fixing bugs in a software system is considered to be the most expensive and time-consuming task [23], [24]. Continuous monitoring and ensuring the quality of a software system is still a big challenge because of its magnitude, limitations of time, and its cost due to complex infrastructure. Further, it is also mandatory to fulfill the major product quality and reliability as well. Software testing process is an essential part of ensuring that software systems are correct in their operation and reliable over the long term. At the same time, testing software needs a significant amount of effort, financial investment, physical infrastructure, and technical know-how. [25]

In most cases, both internal and external metrics are used to designate the software reliability level. The source code of a software system is measured by internal metrics, whereas the functionality of a software system is measured by external metrics.[8] In the early stages of product development, software engineers require access to the key internal metrics that are producing defects in order to accurately predict key metrics of entities that will be used later in the life cycle. Various software metrics have been used for defect prediction. A systematic literature survey with the objective of software fault prediction has been presented [26] after analyzing the 64 machine-learning techniques from 1991 to 2013. The author claimed that machine learning models perform much better than traditional statistical models in classifying software systems as defective or not. In addition, authors have also proved that traditional source code metrics are also helpful for software defect prediction [40]. [27] They analyzed 106 relevant studies on software defect prediction to determine which metric is useful.

The idea of entropy changes was first proposed by Ahmad E & Hassan [21], which is a measurement of the complexity of modifications. The possibility that modifying already-existing software can result in unanticipated issues aims that aren't achieved, or both is referred to as software entropy. Authors have compared entropy with the number of modifications and number of previously reported defects, it was found to be preferable better in most cases. OpenBSD, FreeBSD, and PostgreSQL are the well know open-source systems on which the entropy metric was calculated. Similarly, Moser et al. [20] used a variety of metrics such as code churn, previous defects, refactoring, number of contributors ..etc. to identify the bug presence or absence in eclipse files. The top ten list method developed by Hassan and Holt verifies heuristics on the bug proneness of the files that have been updated or bug-fixed the most recently by making use of the data from the defects repositories.[28]

Erturk et al, [29] discussed soft computing methods that have been used to predict the fault in a software system. Based on the study, they came to a decision that the Adaptive Neuro-fuzzy inference system (ANFIS) concept was a successful attempt to apply for software defect prediction, which incorporates domain information in the form of several fuzzy sets and kinds of membership classifiers throughout the training and learning phased. To gain an understanding of how automated predictions were achieved, a rule extraction method was utilized in the process of developing the decision trees.[30] [31] For this purpose, a strategy that is based on association rule mining was proposed. This approach makes use of the software fault patterns in order to discover the activity patterns that are accountable for the software faults.

Further, when comparing software failure predictors across different sizes of datasets and different sets of metrics, machine learning-based models such as random forest (RF) and naive bayes (NB) approaches performed best throughout. [32] Conversely, the prediction of bugs using the class-level metrics demonstrates that NB is the best fault predictor, regardless of whether covariance metric sampling was given or not to it.

In addition, a comprehensive review was given by Zhao et al, [33] focusing on code metrics and the design of software systems to evaluate how they are effective in predicting the bugs in software modules. They demonstrated that focusing just on the design metric at the start of the software development cycle may be an effective strategy for bug prediction. Since this study was carried out only evaluated using a single dataset, the outcomes of the finalized bug prediction model may be subject to some degree of bias as a result, which is considered to be one of the drawbacks of this study.

2.1.Research Questions

In order to keep the work focused, research questions were addressed. B.A Kitchenham & Charters [34] PICOC criteria were designed, which include population, intervention, comparison, outcome, and context Table 2.

Table 2: A summary of PICOC

Criteria	Descriptions
Population	In software engineering, the population can be defined as the minimum number of primary studies required. It includes testers, managers, system engineers…etc.
Intervention	it is a software tool or technique used to deal with a particular problem, such as system testing or cost estimation
Comparison	To compare the intervention in software engineering, this methodology is being used. The term "control" technique is frequently used when the comparator method is the conventional or widely-used technology.
Outcome	Results should be related to characteristics that users care about, such as increased dependability, decreased manufacturing costs, and shortened time to market.
Context	This refers to the context where comparisons are made either in academia or industry. Practitioners take part and small as well as large tasks performed.

The literature addressed the following question.

RQ1: Which research areas are seeing the most publications in high-quality journals in software fault prediction?
RQ2: What kinds of research topics are chosen by professionals working in the field of software fault prediction?
RQ3: Who are among the most engaged and well-respected researchers working in the field of software defect prediction?
RQ4: Which datasets are the most commonly employed for software fault prediction?
RQ5: What distinguishes the performance of supervised from unsupervised software fault prediction?
RQ6: What types of techniques are used to identify software defects?
RQ7: Which approach delivers the best results when employed in the prediction of software defects?
RQ8: Which approach works the best for predicting software defects?
RQ9: Which types of method innovations for software fault prediction are suggested?
RQ10: Which types of frameworks are recommended for predicting software defects?

Software engineer researchers, as well as practitioners, are principally interested in all these research questions. The questionnaire that the researcher is interested in software engineering to find the helpful contents (RQ1, RQ2 & RQ3). It provides us with an overview and description of a certain domain of study within the area of software defect prediction. To answer the techniques used for software defect prediction, the dataset used and framework being utilized (RQ4, RQ8, and RQ9). For the purpose of supervised and unsupervised performance (RQ5).

2.2. Search Strategy

The procedure for conducting a search includes a number of phases, such as choosing online databases, and digital libraries, specifying the search string, revising the search string, and obtaining the primary list of key studies that are matching with the search phrase. While conducting the search, it is necessary to select an optimal collection of databases in order to maximize the possibility of discovering articles that are of a high level of relevance to the topic in mind. In order to gather the most comprehensive set of research available, a search was conducted through the literature databases that were the most widely used in the area. It is important to have a broad viewpoint in order to provide an extensive and comprehensive review of the literature.

The following online resources were searched through for relevant information:

- ACM digital library
- Springer
- IEEE eXplore
- Scopus
- ScienceDirect

Following are the steps to create the search string

1) PICOC terms that have been identified as relevant to the search
2) Keywords from research questions
3) Searching related titles, abstracts, and keywords
4) A complex search string was constructed by employing recognized search phrases, Boolean ANDs, and ORs.

The modification of the search query was carried out, but the initial one was maintained due to the fact that the modification of the search string would significantly expand the already large list of studies that were not relevant. After that, the search phrase was modified so that it better matched the parameters of each individual database. Journal articles and proceedings from conferences were both types of publishing that were taken into consideration. Only items that were originally published in English were considered in the search.

2.3. Study Selection

The primary studies were chosen based on whether or not they met the inclusion and exclusion criteria.

Following are the inclusion criteria on which we included the studies

- ✓ Research carried out in both academia and business, utilizing both big and small-volume datasets.
- ✓ Research analyzing and contrasting the performance of various modeling approaches in the field of software fault prediction.
- ✓ In the case of studies that are available in both conference and journal formats, only the journal version is considered for inclusion.
- ✓ When there are many publications relating to the same work, only the most recent and comprehensive one is considered for inclusion.

Similarly, the exclusion criteria are as follows:

- Studies that do not have a good validation or that do not include scientific results of predicting software faults
- Papers that examine software defect datasets, methodologies, and frameworks but not in the context of software defects
- Research that has not been written in the English language

The results of the search were saved and managed with the help of the Mendeley software application. Title and abstract screening, as well as full-text screening, might lead to the removal of original research. Studies based only on a review of the relevant literature and similar works are not included. Studies that are related to software defect prediction in some way are also included.

2.4. Challenges to the Validity

The purpose of this study is to investigate the research that has been conducted on the topic of predicting software defects using statistical and machine-learning approaches. In this review, we do not consider the possibility of bias in the selection of research. Not every journal article's title was read by a person to compile the database. Therefore, our study may have excluded certain software defect prediction publications from conference proceedings or journals.

Due to the fact that experience findings are typically published in conference proceedings, our study did not exclude research from conference proceedings. As a result, we have incorporated a source of knowledge regarding the industry's previous experiences. However, there are studies published that include conference proceeding works.

3. TECHNIQUES FOR SOFTWARE DEFECT ESTIMATION

Most of the currently available frameworks for defect estimation are built on a strange combination of programming measures. With these metrics, a fault predictor can typically achieve a reasonable level of precision. However, there are only a few feature selection techniques available that can significantly reduce the number of dataset dimensions; principal component analysis (PCA) is one of them. [35] Is it possible to find a solution that achieves a balance between the two competing demands of cost and accuracy? In addition to choosing an appropriate software metric for defect prediction, there are a number of classifiers available, which we have shown in Figure2

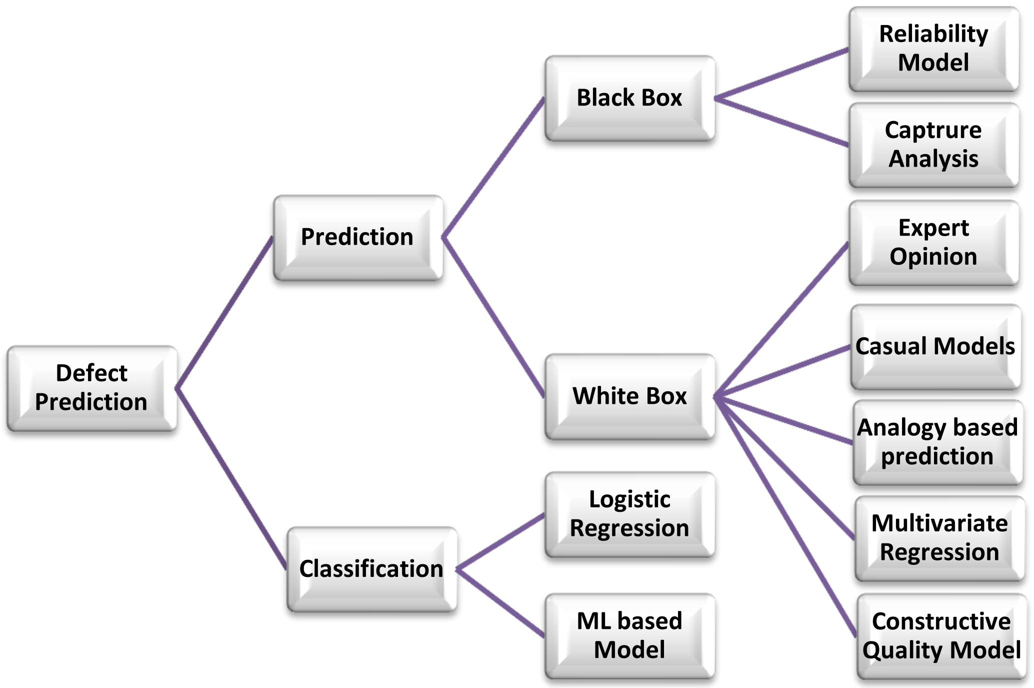

Figure 2. Techniques for software defect estimation

3.1. Defect Estimation Techniques

PDP, known as programming defect prediction, is applied to estimate the expected number of defects detected in a particular component or to classify which components are probably defective. In the process of characterization and anticipating absconds, a variety of distinct strategies have been proposed. These strategies can be broadly categorized into methods that are utilized to predict whether or not a given method is going to contain abnormality Figure 2.

Staron and Meding [36] conducted an investigation into the matter and concluded that the perspectives of experts are an important feature to consider, and their applications were compared to those of other knowledge models. A previous study by the author demonstrates the long-term analytical capabilities of software reliability growth models in the automotive domain, showing their usefulness in the evaluation of defect and consistency. Numerous software components associated with coding, such as complexity, size, and so on, have been effectively utilized in order to classify the software components that have a chance of being defective. Kim et at., [10] have also investigated methods that use coding and modifying measurements as sources of data and that make use of machine learning methodologies for classifying and making predictions.

3.2. Defect Classification Techniques

To evaluate the bugs in software modules, researchers have introduced classification techniques for software defect prediction Figure 1. Such methods make use of a wide number of software product and/or project parameters in an effort to determine which software modules are prone to errors. Generally, these models are implemented at a lesser level of granularity and most frequently at the file and class levels, where the data is more easily accessible. As a result, modules with known bugs can be ranked by fault severity and subjected to more thorough testing.

Furthermore, In order to construct an estimation model that is both accurate and useful, we need accurate data on metrics and defects to serve as the learning set. This information can be gathered during the process of software development projects. Therefore, there must be a trade-off to be made between the accuracy of its predictions on further data sets and the degree to which this model fits within its training set. As a result, the overall quality of the model is evaluated based on a comparison between the predicted faults of the components in a test and the actual defects that were seen in the modules [37].

3.2.1. Logistic Regression

Based on logistic regression, a software component may be characterized as defect-prone or not. To classify a software module, several product and process metrics are used as predictors. This process is exactly the same as multi-linear regression. In order to classify files and packages in the Eclipse project as defect-prone, logistic regression was used [41].

3.2.2. Machine learning-based model

The application of statistical algorithms and data mining techniques in several well-known machine learning approaches helps in the prediction and classification of defects. These methods are the same as regression methods that employ the same kinds of independent variables. The machine learning techniques are dynamic by nature, which helps them improve the overall prediction and classification method over time.

To create an effective prediction model, we need accurate metrics and defect data, which may be gathered via software development activities and used as the set of training data. Therefore, there is a balance between how effectively this method works in its training set and how well it predicts other data sets. As a result, the effectiveness of the model is evaluated by contrasting the expected and observed defects of the components during a test.

The majority of defect prediction methods rely on machine learning. As we discussed, this method can be categorized as classification and regression depending on what to predict about bugs. Moreover, semi and active supervised learning have also been developed for better prediction. In addition to machine learning models, other methods have also been proposed, such as Bug Cache.

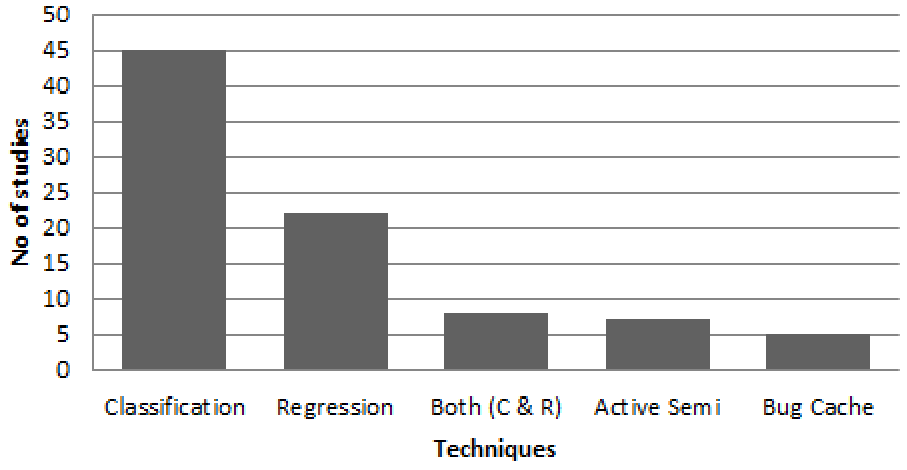

Figure 3. Defect prediction models used in prediction paper.

Since both regression and classification are based on a machine learning approach, their prediction processes are similar. The only differences between these two techniques are classification predicts whether a model is bug-proneness or not, whereas regression predicts the total number of bugs present in a model. The quality assurance team determines which model should be used based on the purpose of the model users.

3.2.3. Isolation Forest

The Isolation Forest algorithm (IF), which is based on the principles of the Decision Tree algorithm, proves to be well-suited for defect estimation. By randomly selecting features and splitting values within the dataset, the algorithm effectively isolates outliers and defects. Anomalies or defects are typically characterized by shorter path lengths in the constructed trees, distinguishing them from normal data points.

Mathematically,

Let X be the input dataset consisting of n data points, where each data point x_i is represented by a d-dimensional feature vector $(x\_i1, x\_i2, \dots, x\_id)$.

Tree Construction: Randomly select a feature f and a split value C within the range of feature f.

Partition the data points based on the feature f and split value C, resulting in two subsets, $X_left\ and\ X_righ$t.

Recursively repeat the partitioning process until a termination condition is met (e.g., the maximum tree depth is reached, or all data points in a partition are the same).

Isolation Tree Ensemble: Repeat the tree construction process for a specified number of trees (T).

Each tree is constructed using a different random subset of the input dataset X.

Anomaly Score Calculation: For a given data point x_i, calculate the average path length $E(h(x\_i))$ in the ensemble of isolation trees.

$E(h(x\_i))$ represents the average number of edges traversed to isolate x_i across all trees.

Defect Score Normalization: Calculate the anomaly score $S(x_i)$ for data point x_i by normalizing the average path length $E\left(h(x_i)\right)$ with a scaling factor $c(n)$: $S(x_i) = 2^{-\frac{E\left(h(x_i)\right)}{c(n)}}$, where c(n) is the average path length of an unsuccessful search in a binary tree ($c(n) \approx 2 * \left(\log(n - 1\right) + 0.5772156649) - 2 * \frac{(n-1)}{n}$).

Thresholding: Determine a threshold value T, above which a data point is considered an anomaly. The threshold can be set based on domain knowledge or by analyzing the distribution of anomaly scores.

Defect Detection: Flag data points with anomaly scores $S(x\_i)$ above the threshold T as defects or outliers.

4. PROPOSED FRAMEWORK

Our study represents that the IFDE-Framework is an unsupervised machine learning technique that is used to estimate defects in software modules. The framework uses the IF which is based on isolating anomalies rather than the most common patterns in the data. This algorithm works by randomly selecting a feature and a random split value between the maximum and minimum values of the selected feature. The algorithm then separates the observations that are less frequent in the dataset and isolates the most abnormal observations. The IFDE-Framework is a powerful tool for identifying defects in software as it can detect previously unknown and abnormal behaviors. The framework has been widely used in the field of software engineering, and it is known for its high performance and accuracy in identifying defects in software modules. Figure 4 shows the overall description of the framework.

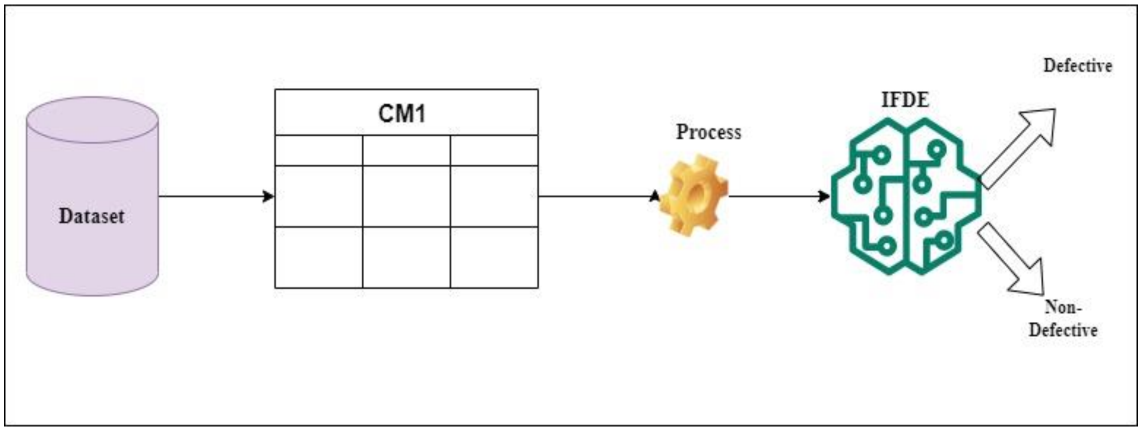

Figure 4. IFDE-Framework for Defect Estimation

5. EVALUATION MEASURES

Various measures have been used by researchers to evaluate the performance of the defect estimation framework. They are:

5.1.Classification measure

In order to evaluate the performance of classification methods of an estimation of the framework, we should first take into account the following prediction outcomes:

$$\mathit{False\ Positive\ Rate}\ (FPR) = \frac{FP}{TN+FP} \quad (1)$$

Table 3: Confusion matrix to identify buggy and clean instances in the defect prediction framework

	Positive	**Negative**
Positive	True positive (TP) • All the buggy occurrences predicted as buggy	False Negative (FN) • All the buggy occurrences were predicted as clean
Negative	False positive (FP) • All the clean occurrences are predicted as buggy.	True Negative (TN) • All the clean occurrences are predicted as clean.

5.1.1. Accuracy

There must be a balance between true positives and true negatives across all occurrences for accuracy to be achieved. Accuracy, thus, is the proportion of events that have been appropriately labeled. The class imbalance of defect prediction datasets makes accuracy an inappropriate metric, especially in defect prediction.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (2)$$

5.1.2. Precision

$$\text{Precision} = \frac{TP}{TP+FP} \quad (3)$$

5.1.3. Recall

$$\text{Recall} = \frac{TP}{TP+FN} \quad (4)$$

5.1.4. F-Measure

$$\text{F} - \text{Measure} = \frac{2*(precision*recall)}{precision+recall} \quad (5)$$

The term recall is also known as Probability Detection (PD). The recall percentage indicates the percentage of incorrectly predicted buggy instances among the total number of buggy instances.

Since there is a trade-off between precision and recall, f-measure has been applied in many studies. [4], [10], [38], and [39].

6. Dataset for SDE

There are many datasets available for defect estimation; some of the most commonly used ones are:

- NASA Metrics Data Program (MDP) dataset: This dataset contains data collected from multiple NASA projects, including metrics such as lines of code, number of defects, and complexity measures. It is widely used in research studies and can be found on the NASA website.
- PROMISE Repository: This repository contains a large collection of datasets for software engineering research, including datasets for defect prediction, software metrics, and other related tasks.
- Defects4J: This dataset contains a set of defective and fixed versions of Java projects, along with the corresponding git commits. It is widely used for research in the field of automatic fault localization.
- Jureczko dataset: This dataset contains information about software metrics and defects for a number of Java projects. It is widely used for research in the field of software defect prediction.

- SEAGrid dataset: This dataset contains information about software metrics and defects for a number of open-source projects. It is widely used for research in the field of software defect prediction.

Table 4. Dataset description

Dataset Name	Total Element	Attributes	Non-defective	Defective	Language
CM1	1988	21	1942	46	C^{++}
PC1	705	21	644	61	C
JM1	7782	21	6110	1672	C
KC1	2109	21	1783	326	C^{++}

7. Experimental Study

An experimental study of defect estimation involves using a CM1 dataset of software metrics and defects to train and evaluate a framework for predicting defects in software modules. The study typically involves selecting a dataset from the PROMISE repository of NASA, preprocessing the data, and training and evaluating a framework using various metrics such as precision, recall, f1-score, and accuracy. The goal of the study is to understand the performance of the framework and identify any factors that may impact its ability to accurately predict defects. The study should be conducted using a rigorous experimental design, with appropriate control groups and multiple runs, to ensure the validity and reliability of the results. The results of the study can be used to understand the strengths and weaknesses of the framework, as well as to identify areas for improvement. The results of the IFDE-Framework show that the framework was able to estimate the presence of defects in the CM1 dataset with a high level of accuracy, specifically with 98.8% accuracy. A confusion matrix was used to evaluate the results, with the true positives being the number of defects that were correctly identified and the false positives being the number of non-defects that were incorrectly identified as defects.

8. Result and Discussion

Table 5. Comparative Analysis

Frameworks	**Precision**	**Recall**	**Accuracy**	**F-1**
Support Vector Machine	43.74	50.00	87.48	81.64
Random Forest	54.44	54.40	87.05	82.94
K-Nearest Neighbors	58.40	54.42	86.50	83.08
Decision Tree	69.08	69.86	85.75	85.84
Bayes	36.27	49.94	17.27	12.22
IFDE-Framework	**70.09**	**71.01**	**98.83**	**82.61**

Table-5 shows the results of different frameworks applied to software defect prediction. The techniques used are Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors

(KNN), Decision Tree, Bayes, and IFDE-Framework. The results are presented in terms of precision, recall, accuracy, and F-1 score.

Precision refers to the proportion of true positive predictions among all positive predictions made by the framework. High precision means that the framework is good at identifying true positives and minimizing false positives.

Recall refers to the proportion of true positives among all actual positives in the dataset. High recall means that the framework is good at identifying all real positives and minimizing false negatives.

Accuracy is the proportion of correct predictions among all estimations made by the framework.

F1 score is the harmonic mean of precision and recall, and it's a measure of a framework's balance between precision and recall.

From Table 5, it's clear that the Decision Tree and IFDE-Framework have the highest F-1 score, which is 85.84, and 82.61, respectively, which indicate that they are the best performers in terms of precision and recall. The KNN, Random Forest, and SVM have relatively lower F-1 scores, which indicate that they are not performing as well as the other techniques in terms of precision and recall. The Bayes has the lowest F-1 score, which is 12.22. This means that the Bayes algorithm is not performing well for this dataset.

The IFDE-Framework has a precision of 70.09 and recall of 71.01, the accuracy is 98.83, and the F1 score is 82.61

8.1. Measure for Regression

Many defect prediction studies employ measurements based on correlation calculations between the number of real bugs and predicted bugs of instances to evaluate the accuracy of defect prediction outcomes using a regression framework. Pearson correlation, Spearman's correlation, R2, and other techniques are used to represent the measure. Such measurements have also been used to examine the relationship between metrics and the number of bugs. [75]

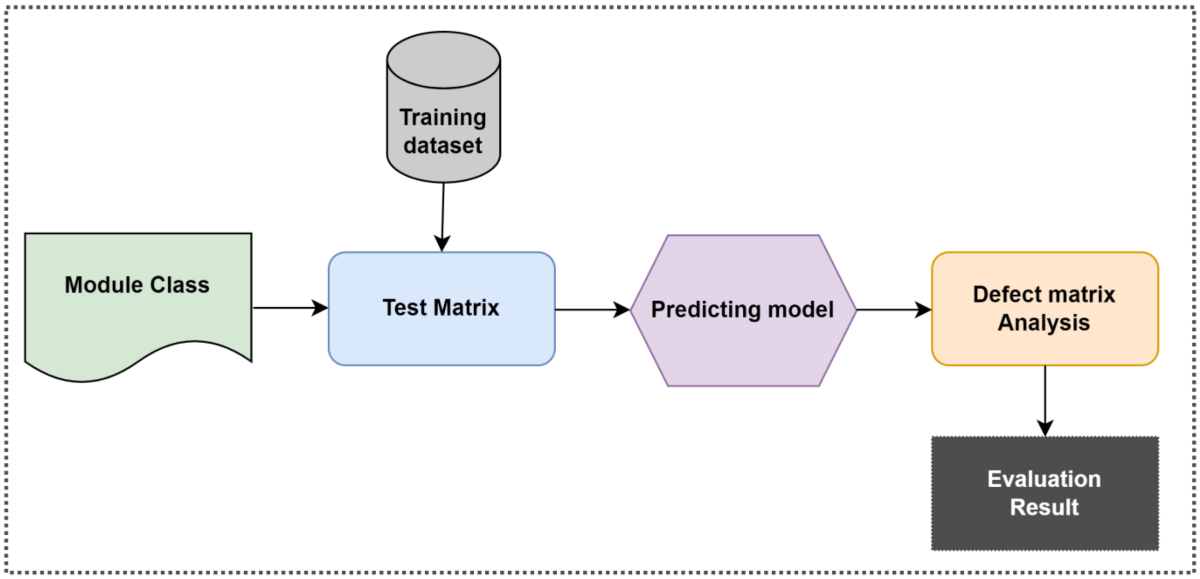

Figure 5. Demonstration of software defect prediction procedure

9. ISSUES AND CHALLENGES

A significant amount of studies have been proposed to predict the defect in software systems. Researchers are constantly trying to overcome the issues present in the defect estimation framework. Yet there are many difficulties are there that need to be addressed.

In spite of the fact that there are a great number of excellent studies available, it is not easy to put those methodologies into reality because of the following reasons:

- A vast number of studies were conducted and validated within the open-source project. Hence there is a possibility that the existing prediction model will not work properly for any other product particularly commercial software. On the other hand, confidential datasets are not made available to the public due to privacy concerns.
- It is necessary to do research into the privacy concerns associated with cross-project defect prediction since the evaluations of prediction methods will be more accurate if we have access to a greater number of confidential datasets.
- Almost all the proposed metrics and models for defect prediction do not provide a guarantee for good prediction performance. Software repositories are growing; we will be able to fetch new kinds of information pertaining to the development process that we have never used.
- With the increasing complexities of software systems, relying solely on file-level defect prediction may no longer be cost-effective. The field needs more research into fine-grained defect prediction techniques, including change categorization and line-level defect prediction.

10. CONCLUSION

Software defect prediction is a crucial task in software development that involves analyzing various factors to identify potential issues in code. Machine learning techniques can be used to automate the process and improve the accuracy of predictions. However, the success of the predictions depends on the quality of the data used for training and the effectiveness of the algorithm used. To reduce costs, it is essential for software industries to predict both software faults and the amount of effort needed to fix them. In this context, the predictive technique employed must be precise and easily understood. Despite the challenges, software defect prediction can provide significant benefits, such as reducing the cost and time required to identify and fix bugs and improving the overall quality of software.

In this work, an IFDE-Framework has been proposed that is an effective framework for identifying defects in software modules. The framework uses the Isolation Forest algorithm which is based on isolating anomalies rather than the most common patterns in the data. The results of the framework using the CM1 dataset showed a high level of accuracy, with 98.8%. These results indicate that the IFDE-Framework is a powerful tool for identifying defects in software, and it can be used to improve the quality of software development and maintenance. However, it's important to note that the results will depend on the quality of the data and the specific requirements of the project, it's always good to evaluate the results with different approaches and compare the results to get a more accurate prediction. Furthermore, this paper also reviewed various other defect prediction techniques and their limitations, providing a comprehensive understanding of the field. Overall, the IFDE-Framework is a promising approach for software defect prediction, and further research can be done to improve its performance and applicability to real-world scenarios.

References

[1] N. Fenton, N. E. Fenton, I. C. Society, M. Neil, and I. C. Society, “A Critique of So ware Defect Prediction Models”.

[2] T. Menzies, J. Greenwald, and A. Frank, “to Learn Defect Predictors,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–14, 2007.

[3] J. Nam, S. J. Pan, and S. Kim, “Transfer defect learning,” *Proc. - Int. Conf. Softw. Eng.*, pp. 382–391, 2013, doi: 10.1109/ICSE.2013.6606584.

[4] T. Lee, D. G. Han, S. Kim, and H. P. In, “Micro interaction metrics for defect prediction,” *SIGSOFT/FSE 2011 - Proc. 19th ACM SIGSOFT Symp. Found. Softw. Eng.*, pp. 311–321, 2011, doi: 10.1145/2025113.2025156.

[5] J. Nam, *Survey on Software Defect Prediction*. 2014. [Online]. Available:

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.722.3147

[6] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," *ESEC-FSE'09 - Proc. Jt. 12th Eur. Softw. Eng. Conf. 17th ACM SIGSOFT Symp. Found. Softw. Eng.*, pp. 91–100, 2009, doi: 10.1145/1595696.1595713.

[7] Gartner, "Gartner Says Worldwide IT Spending on Pace to Reach $3.8 Trillion in 2014," *Gartner*, 2014, [Online]. Available: http://www.gartner.com/newsroom/id/2643919

[8] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Appl. Soft Comput.*, vol. 33, pp. 263–277, 2015, doi: https://doi.org/10.1016/j.asoc.2015.04.045.

[9] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *Int. J. Appl. Sci. Eng.*, vol. 17, no. 4, pp. 331–344, 2020, doi: 10.6703/IJASE.202012_17(4).331.

[10] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," *Proc. - Int. Conf. Softw. Eng.*, pp. 481–490, 2011, doi: 10.1145/1985793.1985859.

[11] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empir. Softw. Eng.*, vol. 17, no. 4–5, pp. 531–577, 2012, doi: 10.1007/s10664-011-9173-9.

[12] J. K. Blundell, M. Lou Hines, and J. Stach, "The measurement of software design quality," *Ann. Softw. Eng.*, vol. 4, no. 1, pp. 235–255, 1997, doi: 10.1023/A:1018914711050.

[13] D. Potier, J. L. Albin, R. Ferreol, and A. Bilodeau, "Experiments with computer software complexity and reliability," *Proc. - Int. Conf. Softw. Eng.*, pp. 94–103, 1982.

[14] T. Nakajo and H. Kume, "A case history analysis of software error cause-effect relationships," *IEEE Trans. Softw. Eng.*, vol. 17, no. 8, pp. 830–838, 1991, doi: 10.1109/32.83917.

[15] H. Zhang, *An Investigation of the Relationships between Lines of Code and Defects*. 2009. doi: 10.1109/ICSM.2009.5306304.

[16] L. Yu and A. Mishra, "Experience in Predicting Fault-Prone Software Modules Using Complexity Metrics," *Qual. Technol. Quant. Manag.*, vol. 9, no. 4, pp. 421–434, 2012, doi: 10.1080/16843703.2012.11673302.

[17] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994, doi: 10.1109/32.295895.

[18] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *J. Syst. Softw.*, vol. 23, no. 2, pp. 111–122, 1993, doi: https://doi.org/10.1016/0164-1212(93)90077-B.

[19] S. Kim, T. Zimmermann, E. J. Whitehead, and A. Zeller, "Predicting faults from cached

history," *Proc. - Int. Conf. Softw. Eng.*, pp. 489–498, 2007, doi: 10.1109/ICSE.2007.66.

[20] R. Moser, W. Pedrycz, and G. Succi, "A Comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," *Proc. - Int. Conf. Softw. Eng.*, pp. 181–190, 2008, doi: 10.1145/1368088.1368114.

[21] A. E. Hassan, "Predicting faults using the complexity of code changes," *Proc. - Int. Conf. Softw. Eng.*, pp. 78–88, 2009, doi: 10.1109/ICSE.2009.5070510.

[22] L. B. and W. L. M. Victor R. Basili, "A VALIDATION OF OBJECT-ORIENTED DESIGN METRICS AS QUALITY INDICATORS," 1995.

[23] C. Ebert, "EMBEDDED SOFTWARE : Facts, Future," pp. 42–52, 2009.

[24] M. Mustaqeem and M. Saqib, "Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection," *Cluster Comput.*, vol. 24, no. 3, pp. 2581–2595, 2021, doi: 10.1007/s10586-021-03282-8.

[25] O. A. L. Lemos, F. C. Ferrari, F. F. Silveira, and A. Garcia, "Experience report: Can software testing education lead to more reliable code?," *2015 IEEE 26th Int. Symp. Softw. Reliab. Eng. ISSRE 2015*, pp. 359–369, 2016, doi: 10.1109/ISSRE.2015.7381829.

[26] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, 2015, doi: https://doi.org/10.1016/j.asoc.2014.11.023.

[27] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1397–1418, 2013, doi: https://doi.org/10.1016/j.infsof.2013.02.009.

[28] A. E. Hassan and R. C. Holt, "The Top Ten List: Dynamic fault prediction," *IEEE Int. Conf. Softw. Maintenance, ICSM*, vol. 2005, pp. 263–272, 2005, doi: 10.1109/ICSM.2005.91.

[29] E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Syst. Appl.*, vol. 42, no. 4, pp. 1872–1879, 2015, doi: 10.1016/j.eswa.2014.10.025.

[30] C.-P. Chang, C.-P. Chu, and Y.-F. Yeh, "Integrating in-process software defect prediction with association mining to discover defect pattern," *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 375–384, 2009, doi: https://doi.org/10.1016/j.infsof.2008.04.008.

[31] K. Anwar, J. Siddiqui, and S. S. Sohail, "Machine learning-based book recommender system: A survey and new perspectives," *Int. J. Intell. Inf. Database Syst.*, vol. 13, no. 2–4, pp. 231–248, 2020, doi: 10.1504/IJIIDS.2020.109457.

[32] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Inf. Sci. (Ny).*, vol. 179, pp. 1040–1058, 2009, doi: 10.1016/j.ins.2008.12.001.

[33] M. Zhao, C. Wohlin, N. Ohlsson, and M. Xie, “A comparison between software design and code metrics for the prediction of software fault content,” *Inf. Softw. Technol.*, vol. 40, no. 14, pp. 801–809, 1998, doi: https://doi.org/10.1016/S0950-5849(98)00098-6.

[34] B.A Kitchenham & Charters, “Guidelines for performing systematic literature reviews in software engineering,” *Tech. report, Ver. 2.3 EBSE Tech. Report. EBSE*, vol. 1, pp. 1–54, 2007.

[35] F. Wu *et al.*, “Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution,” in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 195–197. doi: 10.1109/ICSE-C.2017.72.

[36] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, “The Impact of Using Regression Models to Build Defect Classifiers,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 135–145. doi: 10.1109/MSR.2017.4.

[37] R. Hewett, “Mining software defect data to support software testing management,” *Appl. Intell.*, vol. 34, no. 2, pp. 245–257, 2011, doi: 10.1007/s10489-009-0193-8.

[38] K. Anwar, J. Siddiqui, and S. Saquib Sohail, “Machine Learning Techniques for Book Recommendation: An Overview,” *SSRN Electron. J.*, pp. 1291–1297, 2019, doi: 10.2139/ssrn.3356349.

[39] T. Siddiqui, M. Mutaqeem, S. Athar, and N. A. Khan, “Impact Analysis of Machine Learning Techniques in Software Engineering Department of Computer Science , Aligarh Muslim University , Aligarh (UP) Department of Computer Science and Engineering , Al-Falah University , Institute of Technology and Management (A . K . T . U .), Aligarh (UP) Abstract :,” no. February, 2021.

[40] Mustaqeem, Mohd, and Tamanna Siddiqui. "Original Research Article A hybrid software defects prediction model for imbalance datasets us-ing machine learning techniques:(S-SVM model)." Journal of Autonomous Intelligence 6.1 (2023).

[41] Bashir, Kamal, Tianrui Li, and Mahama Yahaya. "A novel feature selection method based on maximum likelihood logistic regression for imbalanced learning in software defect prediction." Int. Arab J. Inf. Technol. 17.5 (2020): 721-730.

[42] Tamanna Siddiqui, Ausaf Ahmad; "Complexity Clarification through Code Metrics"; 5th International Conference on Computing for Sustainable Global Development; © INDIACom-2018; IEEE sponsored, ISSN 0973-7529; ISBN 978-93-80544-28-1; Pages 3746-3749, Publisher: IEEE Explorer 2018;

[43] Tamanna Siddiqui, Ausaf Ahmad; "Mining Software Repositories for Software Metrics (MSR-SM): Conceptual Framework"; International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8 Issue-10, August 2019; pp 4173- 4177.