International Journal of Communication Networks and Information Security
2024, 16(4)
ISSN: 2073-607X, 2076-0930
https://https://ijcnis.org/ **Research Article**

A Literature Review on Software Defect Prediction: Trends, Methods, and Frameworks

Suresh Jat[1], Dr. Gurveen Vaseer[2]

[1]Research Scholar Oriental University, Indore.
[2]Associate Professor Symbiosis University of Applied Sciences Indore.

Email: [1]sureshjat.cs@gmail.com, [2]gurveenv@yahoo.com

ARTICLE INFO

Received: 28 Apr 2024
Accepted: 05 Sep 2024

ABSTRACT

Identifying possible problems at an early point in the development lifecycle is one of the most important things that software defect prediction can do to enhance software quality and minimize development costs. This is one of the most crucial roles that software defect prediction can play. Of all the functions that software can perform, this is one of the most crucial ones. This literature review aims to offer a thorough examination of the research trends, methodologies, and frameworks utilized in the field of software defect prediction. This study analyzes a broad range of scholarly publications. These publications cover a wide variety of topics related to defect prediction, including dataset features, prediction models, assessment measures, and prediction approaches. Within the context of minimizing the negative consequences of defects on software quality and project schedules, the review emphasizes the significance of software defect prediction. This investigation identifies significant research themes such as the use of machine learning algorithms, feature selection approaches, and ensemble methods in defect prediction. The paper also scrutinizes the challenges and limitations associated with the diverse defect prediction methodologies currently in use. These include the imbalance of the dataset, the bias in feature selection, and the overfitting of the model. Additionally, it highlights the development of research fields and the opportunities for future study, such as the incorporation of domain knowledge, the incorporation of varied data sources, and the development of advanced approaches to predictive modeling. Furthermore, it acknowledges the existence of these opportunities. In its entirety, this literature review provides researchers and practitioners working in the field of software engineering with critical insights into the present state of the art in software defect prediction.

Keywords: Software defect prediction Machine learning Model generalization Domain knowledge integration.

INTRODUCTION AND CONTRIBUTIONS

Software flaws can cause devastating effects, as demonstrated by the catastrophic crash of the Ariane 5 spacecraft on June 4, 1996. The software that was responsible for converting 64-bit floating-point numbers to 16-bit signed integers experienced a catastrophic overflow problem in this particular case. The software was responsible for converting the numbers. This situation ultimately led to a catastrophic failure of the computer system. The growing complexity and scale of modern software systems exacerbate their presence, making them a serious hazard. Despite greater efforts to combat software defects, both in academia and business, the prevalence of software defects continues to pose a substantial threat.

Code review, which is a process in which developers other than the original author methodically analyze the source code, continues to be an essential practice for identifying flaws and improving the quality of software. To ensure the safety and dependability of aircraft operations, it is critical to conduct a comprehensive analysis of airborne software, particularly in the context of airworthiness reviews. The detection of defects at an early stage in the software development lifecycle is of the utmost importance, given that their existence is unavoidable and that they have the potential to develop into severe problems.

It is essential to make a distinction between defects and bugs. Bugs, which are flaws in the source code. Defects, on the other hand, involve a wider abuse of variables, functions, and other components that can result in operational hazards or security vulnerabilities.

Static code analysis techniques play a crucial role in the process of assisting developers and reviewers in swiftly locating faults within the source code. These techniques make a significant contribution to improving the resilience and security of software projects. They do this by enabling early detection and mitigation of potential problems. A software flaw, according to Naik and Tripathy (2008), refers to an error, problem, or failure in software. Not only does it exhibit unintended behavior, but it also produces results that are either incorrect or unexpected. McDonald, Musson, and Smith (2007) assert that a specific flaw within the program can cause a software product to function in an unexpected manner. The IEEE (IEEE, 1990) provides standard definitions of mistakes, defects, and failure, providing the most accurate description of a defect. When an employee's actions lead to a bug, they commit an error. The difference between a failure and a flaw is that a failure occurs when the system performs something wrong while it is running, while a flaw occurs when a mistake in the code appears while doing so. Consider a mistake to be an error that the creator made. The process of reviewing and testing software is a very crucial element of the software development process, particularly when it comes to locating flaws and finding solutions to them. This is largely due to the ongoing expansion and complexity of software in today's world. It is with great regret that we must inform that the cost of fixing software faults or flaws is extremely high. The process of locating and fixing defects is reportedly one of the most expensive aspects of developing software, according to Jones and Bonsignour (2012). As a result of their research, Jones and Bonsignour (2012) discovered that this was one of the most expensive things to accomplish. As the development process progresses, the cost of a software defect will continue to rise. Detecting and correcting errors that occur throughout the writing process costs $977 per error. The price will increase to $7,136 for each error discovered during the testing phase. The cost of collecting and removing the data increased to $14,102 during the maintenance phase of the research project. When it comes to finding software problems, software defect forecasting methods are a far more cost-effective mode of investigation than software tests and reviews. This is because it is possible to anticipate potential software issues.

The development and evaluation of predictive models is one of the most important contributions that researchers have made to the field of software defect prediction. On the

basis of historical data and software metrics, academics and practitioners examine a wide range of machine learning techniques in order to develop models capable of accurately identifying probable modules that are prone to defects. Decision trees, support vector machines, logistic regression, and neural networks are some examples of the algorithms that fit into this category. When it comes to the performance of defect prediction models, feature selection and engineering are extremely important factors. Researchers look into a variety of software metrics, such as code complexity, size, coupling, and cohesiveness, as well as historical defect data, in order to determine which characteristics are the most important for defect prediction. In addition to this, they investigate methods for manipulating and combining features in order to improve the predictive performance of the models.

Major Factor of Software Failure

Most people believe that a software system can be defined as any software product or application that supports businesses. Numerous fields, including industry, banking, healthcare, insurance, aviation, social networking, e-commerce, and others, can implement these actions. To build and design software systems, one needs money, people with field knowledge, a lot of time, the right tools, and the right infrastructure. Additionally, having ample time is crucial. Table 1 shows that the number of software problems is steadily going up. This costs the company money and wastes time and energy. Even if a software company is very good at planning and carrying out projects, this is still the case. As a result of a flaw that was present throughout the entire software development cycle, the system may not work properly if the client does not understand IT projects or political or cultural issues. This is because the flaw was present throughout the whole process. Furthermore, it's possible that the customer will not give you the correct requirements.

We also asked the poll respondents to identify the factors that contribute to project difficulty. Many factors can lead to software failures, but the most prevalent ones include: users' lack of involvement, unclear goals and objectives, incomplete requirements, insufficient resources, poor project planning and scheduling, ineffective team communication, and incorrect testing. According to Table 1 below, the most important things that affect the success of projects are not allowing users to connect with them and not correctly identifying what they want. The following table shows some examples of the main things that can go wrong with software.

Table I Major factor for software failures

Factor	Description
Inadequate testing	bugs or problems that can't be found because testing methods are absent or not good enough.
Poor requirements gathering	Requirements that aren't clear, full, or consistent lead to bad software design.
Miscommunication	It's not clear how stakeholders, coders, and testers are talking to each other.
Tight deadlines	Project plans that are too ambitious cause development to be rushed and mistakes to happen more often..
Inexperienced team members	Team members who don't have enough experience or knowledge lead to answers that aren't as good as they could be.
Complexity of software	The program architecture is very complicated, which makes it hard to understand and keep up to date.

This study presents a comprehensive literature analysis on software defect prediction, focusing on the current trends, approaches, and frameworks in the field. The introduction begins with a discussion that emphasizes the value of software defect prediction in reducing

the most significant component that contributes to software failure. A synthesis of previous research, the identification of major trends, and a discussion of approaches and frameworks utilized in software defect prediction are the contributions that this study makes. It is clear that defect prediction is of the utmost significance in software engineering methods, as highlighted by the primary cause of software failure. This study investigates several facets of software defect prediction, particularly its role in enhancing software quality and dependability. Specifically, the paper is organized as follows:
Section II gives a summary of the research that is pertinent to the topic at hand, contextualizing the current state of the area within the context of the existing literature.
Section III provides an overview of the methodology that was utilized in the process of performing the literature review, as well as insights into the selection criteria and procedures for analysis.
Section IV examines trends, approaches, and frameworks that are utilized in the field related to software defect prediction. Explores research subjects that are relevant to software defect prediction. The purpose of this part is to provide an in-depth analysis of the present state of research concerning the prediction of software defects.
In the final section, Section V, conclusions and potential pathways for future research are offered. These sections summarize the most important findings and offer suggestions for the direction of additional research and development in its respective subject.

Machine Learning in Software Defect Prediction

In recent years, the field of machine learning has grown quickly. This has led to the creation of many learning algorithms that can be used in a wide range of cases. One of the most important things to think about when figuring out how useful these programs will be in the long run is how well they deal with problems that happen in the real world. Because of this, copying methods and using them to solve new problems is very important for the field's growth. This is the case because of what happened in the last point. A lot of academics who study machine learning, on the other hand, are writing papers right now about how to make models that can tell when software might fail. In the area of machine learning, these studies are being put out there. Three different methods—classification, grouping, and ensemble have been put into action, and now we are sorting good software fault models into three separate groups. The many different methods we are using are what led to the creation of these groups. At this point in time, there aren't many large and varied software defect prediction datasets, methods, and models. Because there are only a few of these things, this is the truth. Since this is the case, it is hard to give a full picture of where the study into defect prediction is at the moment. Between 2000 and 2024, many different study patterns, datasets, approaches, and ideas were used in the field of software failure prediction research. The goal of this study into the relevant literature is to find and evaluate these elements. The parts of this work are broken down below into how they are put together. The methods that were used in the study are talked about in more detail in the second section.

Problem Statement:

The dependability, functionality, and cost-effectiveness of software systems are all significantly impacted when faults in software are available. Locating and fixing these issues as early as possible in the development process is necessary to make sure that software is made that makes better products. This is because it is important to make sure that software is made that makes better goods. Software defect prediction tries to guess what problems might happen based on different program measurements and attributes so that developers can plan their tests better and make the best use of their resources. There are, however, a number of obstacles that continue to exist in spite of the substantial research that has been conducted in

this field. These obstacles include heterogeneity in the dataset, uneven class distribution, bias in feature selection, and problems with model generalization. The rapid development of software technologies and processes needs the ongoing adaption and improvement of defect prediction systems. This is because of the quick evolution of both. The issue statement of this study is therefore to conduct a complete analysis of the existing literature on software defect prediction, to identify research trends, methodologies, and frameworks, and to address the constraints and limits associated with the approaches that are currently being utilized. This project intends to progress the state-of-the-art in software defect prediction by tackling these issues. Additionally, it intends to promote the development of defect prediction models that are more accurate, dependable, and cost-effective.

REVIEW METHODOLOGY

The technique of conducting systematic literature reviews (SLRs) has swiftly become a well-established review procedure in the area of software engineering. A systematic literature review (SLR) is a process that involves finding, evaluating, and interpreting all of the study material available to provide answers to specific research questions, as stated by Kitchenham and Charters (2007). This method is characterized by a structured approach to discovering, evaluating, and interpreting these study materials. Using the principles initially established by Kitchenham and Charters (2007), this literature research was conducted in the form of a systematic literature review.

Search Strategy

A search involves several steps, such as choosing which digital libraries to use, defining the search string, running a pilot search, changing the search string, and getting a first set of primary studies from digital libraries that match the search phrase. The thorough search process incorporates each of these steps. Prior to initiating the search, select a collection of databases that enhances the likelihood of discovering articles particularly relevant to the topic under consideration. This crucial step must be taken before beginning the search, as it holds significant importance. To find the most complete collection of research that is not only possible but also possible, you need to search the most well-known literature sources in the field. In order to provide an accurate and thorough analysis of the books, it is crucial to examine them from various perspectives. Here is a list of the digital sources we examined:
Among the resources available to researchers are the following:
The ACM Digital Library (dl.acm.org),
IEEE eXplore (ieeexplore.ieee.org),
ScienceDirect (sciencedirect.com),
Springer (springerlink.com), and
Scopus (scopus.com).
The search string was constructed in the following way using the techniques listed below:
• The identification of search phrases that are included in titles, abstracts, and keywords that are relevant to the search
• The development of a complicated search string by making use of the search keywords that have been identified, ANDs and ORs in Boolean logic
• The discovery of probable synonyms, variant spellings, and antonyms for the words that are being searched for on the internet
Determined that using the following search string would be the most effective: AND (fault, defect, quality, or error-prone) this also applies to the terms "predict, prone, probability, evaluate, detect, estimate, or classify." AND (classify by evaluating, detecting, or estimating) The term encompasses both software and applications, as well as systems. Besides defects,

flaws, quality issues, or a tendency to make mistakes, THEN. We adjusted the search string, but the previous one remained in use. This was because altering the search phrase would significantly expand the already vast list of irrelevant studies. This was the reason for it. Subsequently, we modified the search phrase to align with the specific requirements set by each database in this instance. The databases utilized titles, keywords, and abstracts in their search operations. The years 2000–2024,only considered for review. Considered two categories of publications for inclusion in this collection: articles from journals and proceedings from conferences and only considered items written in English for this search.

Figure 1 Systematic Literature Review Steps

Study Selection: The criteria for inclusion and exclusion were utilized as selection criteria during the process of selecting the primary studies. Table 2 presents these requirements in their entirety.

Table 2 Inclusion and Exclusion Criteria

Inclusion Criteria	Research conducted in both academic and industrial settings, utilizing both big and small scale data sets
	Only the journal version will be included for research that have simultaneously been published in both the conference and journal versions.
	When there are many publications of the same study, only the one that is the most comprehensive and published most recently will be included.
Exclusion Criteria	Examples of software defect prediction studies that lack a robust validation or that do not include experimental results
	In the context of anything other than software defect prediction, studies that explore defect prediction datasets, methodologies, and frameworks
	Research that was not written in English

Study Quality Assessment and Data Synthesis

The study quality evaluation, the eighth phase, can provide a direction for the interpretation of the synthesis results. Additionally, we can use the evaluation to assess the validity of the extended conclusions. The process of data synthesis gathers evidence from selected studies to answer the research questions. While a single piece of evidence may have a low evidence force, the integration of numerous pieces of evidence can make a claim more compelling. We extracted both quantitative and qualitative data for this evaluation. We synthesised the retrieved data using a wide range of distinct research approaches, catering to various types of research inquiries. Most of the time, we employed a method commonly known as narrative synthesis. We tallied the collected information in a manner consistent with the asked questions. We utilized specialized visualization tools to enhance the data display of software defect prediction approaches and their accuracy. We used these tools to enhance the data display further. This compilation featured a variety of visualization tools, including bar charts, pie charts, and tables.

Threats to Validity

This work aims to analyze previous software fault prediction research. Machine learning and statistical techniques will form the foundation of the analysis. This review highlights the limited understanding of biases in the selection process of research findings. We did not carry out the search by individually reading the titles of every published article in a journal. This study may not have included all software defect prediction publications that appeared in specific journals or conference proceedings. The aforementioned fact is the reason behind this. Conference proceedings frequently publish experience reports, which means they include all studies presented during these sessions. This leads to the availability of a source of knowledge based on the industry's experience. Several systematic literature assessments, including one by Jorgensen and Shepperd in 2007, did not include conference proceedings. The reason for this is that incorporating conference proceedings would significantly increase the researcher's workload. On the other hand, Catal and Diri are in charge of doing a full literature review, which will consist of papers retrieved from conference proceedings as primary research.

Significant Journal Publications

There are ninety important studies that investigate the effectiveness of software flaw prediction methods, as indicated by this overview of the relevant literature. We provide the spread over the years to illustrate how people's interest in forecasting software defects has fluctuated over time at various points. We selected the most significant software flaw prediction publications based on the primary papers we studied, as illustrated in Figure 2. Please note that we have not displayed the conference materials here. This is a crucial issue.The following table breaks down the total number of articles written about software engineering across various domains, according to publishing. With seven scholarly articles, the Journal of Software came in first. Advanced Science Letters published a total of six articles, IEEE Transactions on Reliability and the Journal of Systems and Software published five papers, and Information Sciences published four papers. The MDPI came in second with ten articles, and the IEEE Transactions on Software Engineering came in third with five papers. This list includes only a handful of prestigious magazines. Five different journals, including Automated Software Engineering, Information and Software Technology, Empirical Software Engineering, and IET Software, published the article. In each issue, there were three studies. The IEEE Transactions on Systems, Man, and Cybernetics, the IEEE Transactions on Knowledge and Data Engineering, the Software Quality Journal, and the International Journal of Software Engineering and Its Applications all published two studies.

The Software Quality Journal also published a pair of papers. It is crucial to acknowledge that this distribution underscores the numerous locations where collaborative research in software engineering can take place. These establishments cover a wide range of subjects, from academic to real-world research, in their writings.

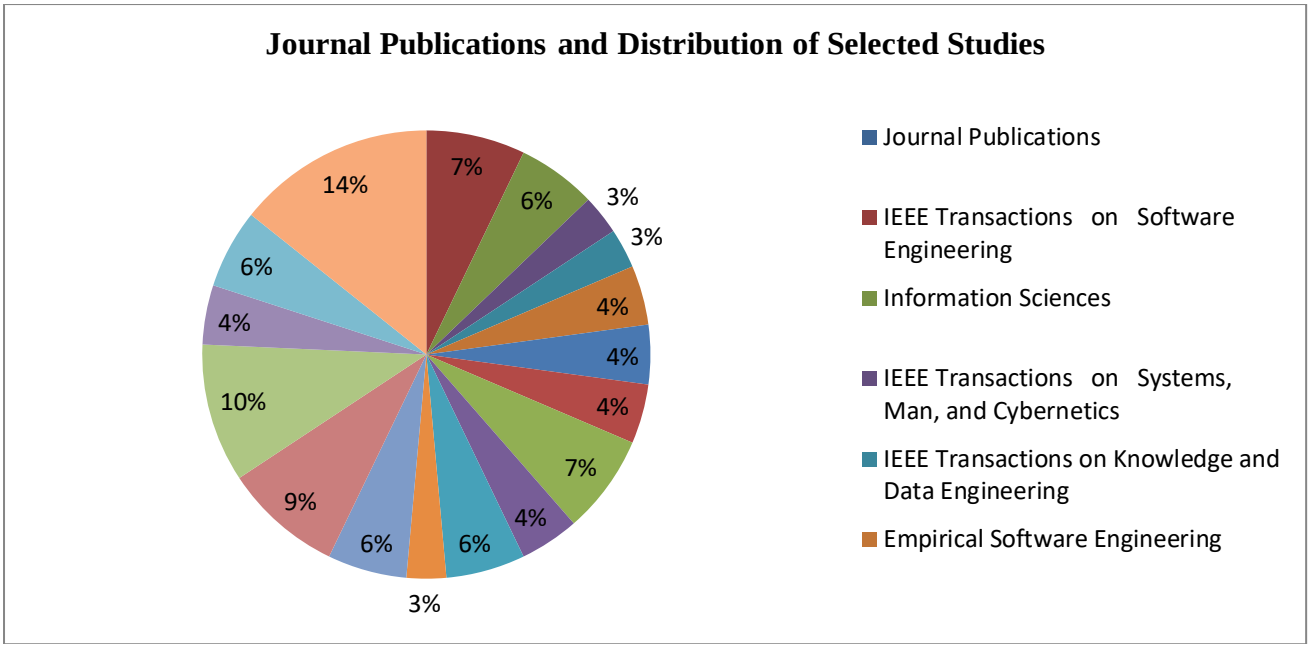

Figure 2 Journal Publications and Distribution of Selected Studies

Research Topics in the Software Defect Prediction Field

The prediction of software faults is considered to be one of the most significant study areas that are pertinent to the field of software engineering, as stated by Song et al. (2011). After doing an analysis of the important articles that were selected, it was found that the current research on software defect prediction concentrated on the five issues that are listed below:

- Utilising the technique of estimation (estimating) in order to arrive at an estimate of the number of defects that are still present in software systems
- By utilizing the association rule algorithm (Association), the identification of defect links can be accomplished.
- The utilization of the association rule algorithm (Association) for the purpose of problem connection identification
- Using the clustering approach in conjunction with the clustering methodology to cluster the software faults depending on the item
- Carrying out an analysis and preparatory processing on the datasets, which includes the identification of software flaws (Dataset Process)

Within the first category of work, known as estimation, statistical methods and neural networks (Benaddy and Wakrim 2012) (Zhang and Chang 2012) are used to estimate the number of defects that are still present in software. It is known as estimation for this type of task. This is achieved through the utilization of inspection data as well as data regarding the quality of the process. For the purpose of providing software engineers with support, the forecast result refers to an important tool that can be utilized. Furthermore, the conclusion of the prediction can be utilized to manage the software process and to evaluate the quality of a software system that is going through the process of being constructed at the present time.

In this kind of research, association rule mining methods are used, which come from the

group of data miners. The goal is to find out how software bugs are connected. It is specifically said that this second type of work can be used for three different purposes by Song et al. After the first step, which is to find as many problems as possible that are linked to the recorded faults and how they affect each other, is finished, the computer program will be changed to work better. Something that could be useful is that this makes it possible for testing to be more targeted and makes better use of the limited testing resources that are accessible. The second step in the process is to look at the results of the software reviewers who were there for the inspection. So, to be sure that the job is done, it needs to be looked over again. The third goal of this project is to help software development managers improve the software development process by looking into what causes certain mistakes to happen one after the other. It is possible to reach this goal with the help of software development managers. In the event that the research shows that there is a problem with the process, however, the management can come up with ways to fix the issue.

Categorization is the third type of work. It sorts software units into two groups: those that are likely to have bugs and those that are not likely to have bugs. This classification was finished with the help of metric-based classification.

According to Lessmann et al. (2008), the classification approach is a well-known machine learning method that is used to expect software errors. This method is used to predict mistakes in software. The use of this method makes it possible to predict software bugs. Gayatri, Reddy, and Nickolas (2010) say that it describes the features of the software code as either broken or not broken. Based on the features of the software code, this description was made. The method used to group these things into these categories is a classification model made from data about software metrics and the successful finishing of previous development projects. There is a chance that the classification system can figure out which parts are more likely to have problems. This makes it possible to use testing tools in a more targeted way, which is good. If a problem is found during field or system tests, it is recorded as a 1 in the fault data for that module. Any other module's fault data is recorded as a 0 in the fault data. This is because the module that is causing the fault has fault data with a number of 1. Catal says that the measures made by the software are used as separate factors to improve prediction models. For this method, on the other hand, the fault data is the dependent variable. For the calculation process that happens when the prediction model's parameters are being set, fault data and software readings from the past are used. A number of classification methods, such as logical regression, decision trees, neural networks, and naive bayes, have been used over the years to help predict software bugs.

The fourth type of work is clustering, which is the process of finding software fault groups by using clustering methods created by the community of data miners. Clustering is an example of an unsupervised learning method that could be used to reach the goal of predicting bugs in software modules. In cases where fault labels are hard to find, the above comment is very helpful to remember. Bishnu and Bhattacherjee (2012) came up with the K-Means method to predict when bugs would happen in software modules. Their plan was to make the guess. Most of the time, Quad Trees are used by the K-Means Algorithm to find the first cluster centers that will be fed into the algorithm. A concept called "clustering gain" was used to make it easier to figure out the quality of clusters so that the Quad Tree-based initialization method could be tested. It was discovered that the Quad Tree-based methods produced groups with the highest possible gain values. This was found out through the finding process. Figure 3 showing the Distributions of Research Topics used in this literature study

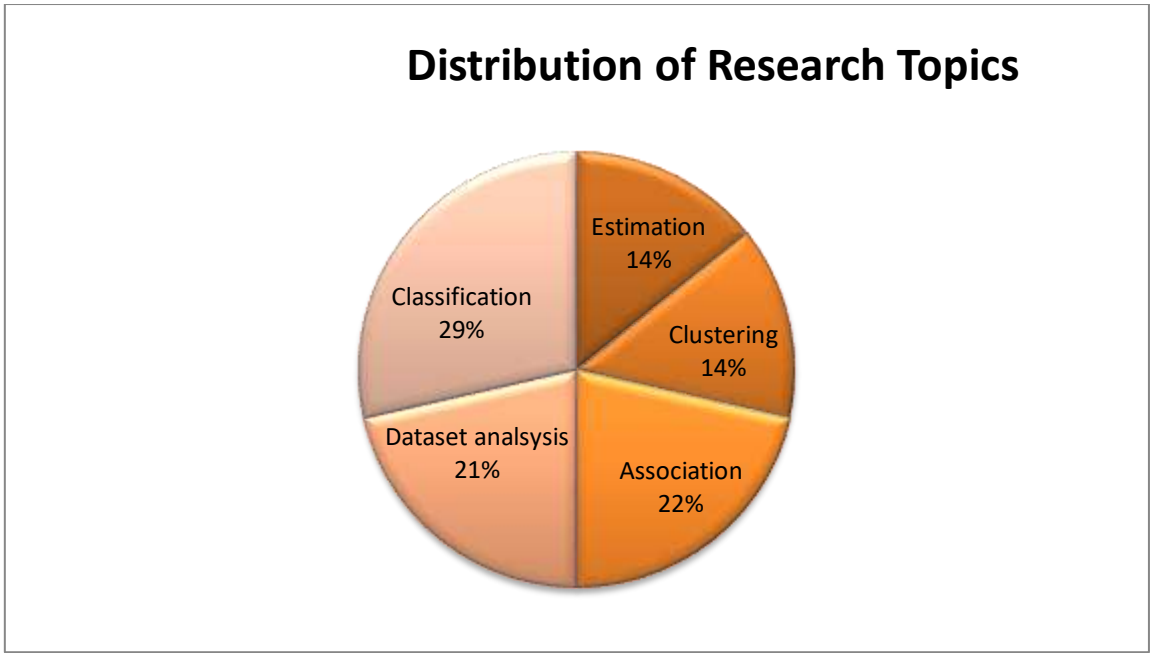

Figure 3 Distributions of Research Topics

Research Topics in the Software Defect Prediction Field

Within the realm of software engineering, the prediction of software faults is considered to be one of the most fascinating topics of research, as stated by Song et al. (2011). The present research on software defect prediction focuses on the following five areas, as determined by the findings of the analysis of the primary papers that were selected for evaluation:

- Using the estimating algorithm (estimating) to make an estimate of the amount of software systems that still have flaws.
- Using the association rule algorithm (Association) to figure out the associations between defects
- Utilizing the classification technique (Classification) to divide the types of software modules that are prone to defects into two distinct categories, namely defect-prone and non-defect-prone software modules
- Using the clustering algorithm to group the software defects into clusters according to the objects they affect (Clustering)
- Performing an analysis and preliminary processing on the datasets including software defects (Dataset Analysis)

Datasets Used for Software Defect Prediction A dataset, as defined by Sammut and Webb (2011), is an assemblage of data obtained with the intention of facilitating machine learning. A training set, on the other hand, is a collection of data that is used as input by a learning system, which uses the data analysis to build a model. Test sets and evaluation sets are collections of data that are used to assess the model that a learning system has learned. Two further subsets of a training set are a growth set and a pruning set. However, because a test set includes data sets that are unrelated to the training set, it is also known as a holdout set.

Catal and Diri (2009a) state that one of the biggest obstacles to a software defect prediction study's success is using non-public datasets. Many companies created defect prediction models using their own data, which they subsequently showcased at conferences. Nevertheless, it is not feasible to compare the results of such study with the results of the suggested models because their datasets cannot be analyzed. Because they faced similar difficulties, machine learning researchers founded a repository known as the University of

California Irvine (UCI) in the 1990s. The PROMISE repository was established in 2005 by software engineering researchers and houses a multitude of public datasets. The work at the University of California, Irvine served as the model for this repository. The datasets that NASA uses to predict software flaws are available in PROMISE. The ARFF format is used as the default format file when these datasets are used straight out of the open source machine learning applications WEKA or Rapid Miner. This enables the use of these datasets.

This literature review contains information from various previous studies. The effectiveness of software defect prediction is examined in this research. The PROMISE and NASA MDP (metrics data program) archives include the great majority of public datasets, which are available for free distribution. "Public datasets" and "private datasets" are not synonymous concepts. Private enterprises are the owners of private datasets. Graph 8: The Whole Dataset Distribution The way the distribution has been presented over the years shows how interest in various dataset types has changed over time. It's bad that 35.21 percent of the studies used private datasets. Because of this, the results of only one of the three investigations can be compared, and the results are repeatable.

It is not possible to compare the results of this research with the results of the models that have been provided because the datasets from these investigations are not publicly available. (Duri and Catal 2009a) Because standard datasets are used, the research may be replicated, challenged, and confirmed. The distribution of the primary studies throughout time and according to the sources is shown in Figure 9. Both the number of published publications and the quantity of publicly available datasets used in software defect prediction research have increased since 2005. The PROMISE repository was established in 2005, as previously mentioned. Furthermore, scholars are beginning to agree more and more on the usage of publicly accessible datasets. 14.08 percent 1.41% is equal to 77.46%. 1.41%, 5.63%, and A guess Relationship Classification Cluster-Based Dataset Analysis 34.21 percent

To illustrate the evolution of interest in various dataset types over time, the distribution throughout the years is presented. 35.21 percent of the studies used private datasets in total. As a result, the results of one study out of three studies can only be compared, and the study can be repeated. However, it is not possible to compare the results of this research with the results of the models that have been presented or proposed because their datasets are not publicly available. Because the study was carried out with standard datasets, it may be replicated, contested, and verified (Catal and Diri 2009a). The number of major studies that were carried out over time and in accordance with the relevant sources is depicted in Figure 9. Since 2005, there has been an increase in the quantity of articles published and the number of public datasets used in software defect prediction research. As previously mentioned, the PROMISE repository was established in 2005. Another encouraging trend is that academics are starting to appreciate the value of using public datasets more often.

Proposed Method Improvements for Software Defect Prediction

In an attempt to increase machine learning classifier accuracy for software defect prediction, researchers proposed several different strategies or techniques. Improving the prediction accuracy of a created model is the goal of recently developed technologies. Some instances of these methods are as follows: Various strategies have been employed in the field of machine learning:

- Adapting and grouping particular machine learning techniques (Mısırlı, Bener, & Turhan, 2011) (Tosun, Turhan, & Bener, 2008)
- Applying a boosting algorithm (Zheng, 2010) (Jiang, Li, Zhou, & Member, 2011)
- Including feature selection (Gayatri et al. 2010) (Khoshgoftaar and Gao, 2009) (Catal and Diri 2009b) (Song et al., 2011)
- Applying parameter optimization for particular classifiers (Peng and Wang 2010).

While many defect prediction techniques have been developed, none of them have consistently proven to be reliable (Challagulla et al., 2005, Lessmann et al., 2008). This is the case even if a lot of techniques have been developed. In the realm of software defect prediction, the subject of how to create a more reliable and accurate classification method to build a better prediction model is still open. It is essential to have frameworks that can accurately predict defects, and these frameworks also need to be more resilient to noise and other issues related to datasets.

Feature Selection

Feature selection in the context of improving machine learning efficacy refers to the investigation of techniques that account for the dimensionality of the data. The goal of feature selection in the context of a dataset with N characteristics and M dimensions (or features, attributes) is to determine a means of reducing M to M', where M' is either equal to or fewer than M (Sammut and Webb 2011). One technique that is frequently employed and that many people believe to be very important is dimensionality reduction. Another method that effectively fulfills its aim is the extraction of features. The results of the two ways are one of the most essential factors that distinguish them from one another and give them their own unique characteristics. The two features that were chosen are a subset of the four features that were initially selected (for instance, F1 and F3), but the two features that were extracted are a combination of the four features that were first selected. Given that we have four characteristics, which are represented by the letters F1, F2, F3, and F4, this is the assumption that we are making.

Applications that need to preserve the original attributes while keeping their integrity frequently use feature selection. There are numerous examples, such as document classification, medical condition diagnosis and prognosis, and gene expression profiling. It advances machine learning in terms of predicted accuracy, comprehensibility, learning efficiency, compact models, and effective data collection. There are several benefits associated with the feature selection process, and these benefits are multifaceted. Maimon and Rokach (2010) state that the goal of feature selection is to eliminate characteristics that are superfluous or unimportant, leaving only those that are pertinent to the current circumstance. The results of a study by Khoshgoftaar and Van Hulse (2009) show that some academics have labeled certain features as redundant and worthless because they include noisy attributes. There is no need to worry about the learning performance being negatively affected by the elimination of any parts that are judged irrelevant. Redundant features are a type of feature that falls under the category of characteristics that are not relevant. It is essential to bring to your attention the fact that a redundant feature is one that presumes the existence of another feature present. The learning performance of the system will not be affected in any way by the removal of either of the features themselves, despite the fact that each feature is significant in its own right.

Filtering, wrappering, and embedding are the three methods of feature selection that are considered to be the older and more conventional approaches. According to the findings of study conducted on the subject, feature selection can be advantageous to the learning performance of a classifier that has incorporated feature selection capacity. When it comes to the operation of a filter model, metrics that pertain to the naturally occurring qualities of the data are absolutely necessary. The evaluation of the quality of data can be accomplished by the utilization of a variety of measurements, including mutual information and data consistency, to name just two examples. A learning algorithm, which is sometimes referred to as a classifier, is utilized by a wrapper model at the beginning stages of the process of determining the quality of the features. If, for example, the removal of a feature does not have

an effect on the accuracy of the classifier, then it is acceptable to do so. It should come as no surprise that the selection of features is tweaked in this manner in order to enhance a particular classification algorithm. In order to decide whether a feature should be kept or eliminated, a classifier must be created for each feature that is considered. To decide which feature should be chosen, this is done. As such, the wrapper model's implementation may need a large financial outlay. Feature selection can be incorporated into the classifier's learning process through the use of an embedded model. The requirement that a feature be chosen first at each branching point serves as the best example of decision tree induction. This holds true for the duration of the procedure. Filler and wrapper models are commonly employed in feature selection processes aimed at facilitating data preparation. The filter model is most frequently used when the goal of feature selection is something other than improving learning performance (such increased classification accuracy). This is so because the most suitable model is the filter model.

Ensemble Machine Learning

According to Sammut and Webb (2011), the term "ensemble learning" describes methods for training a large number of learning machines and combining their outputs, treating them as a "committee" of stakeholders. When individual predictions are integrated appropriately, it is anticipated that the committee's decision will have, on average, a higher overall accuracy than any one member's. This is the main concept. A substantial body of empirical and theoretical research has shown that ensemble models often achieve greater accuracy than single models.

The ensemble members can forecast a great deal of different things, such as real-valued values, class labels, posterior probabilities, ranks, clusterings, and more. Therefore, it is possible to include their decisions using a variety of techniques, such as voting, averaging, and probabilistic methods. It is true that most ensemble learning techniques are general, meaning they can be used to a wide range of model types and learning situations. It is a reality that this exists.

To do this, a multitude of machine learning techniques can be applied. These methods involve picking up a variety of models and using them in tandem with each other. According to Witten, Frank, and Hall (2011), bagging, boosting, and stacking are the most widely used strategies of these. These are the tactics that yield the greatest results. When compared to a single model, it is conceivable for all of them to improve prediction performance most of the time. It is truly feasible to accomplish this. These generic approaches can be applied to numerical prediction challenges as well as classification jobs, yielding answers for both kinds of issues. The techniques of bagging, boosting, and stacking have been developed during the previous few decades, and these processes often work quite well. It has been challenging for machine learning researchers to comprehend the reasons underlying this phenomena. In the course of this conflict, new tactics have been developed, some of which have proven to be even more effective than those that were previously employed. Although human committees usually do not benefit from loud interruptions, adding random classifiers to the bagging process can improve performance. This is true even if ineffective noise diversion rarely benefits human committees.

Hrishikesh Kumar et al. (2023) this study focuses on software fault prediction, addressing critical challenges like class imbalance and the need for robust, generalizable predictive models. It involves cross-project analysis to explore how well models trained on one project can predict faults in another. Key research questions include handling class imbalance, the effectiveness of cross-project prediction, and the impact of diverse training samples on prediction accuracy. The authors highlight the importance of correcting class imbalance and using datasets with similar features to the target project for reliable predictions. Comprehensive experiments compare various classifiers, considering metrics such as

accuracy, precision, recall, and F1 score, to improve model resilience and dependability.

Sagnik Mondal et al. (2023) this paper systematically reviews literature on machine learning algorithms for software fault prediction, analyzing 52 articles published from 2009 to 2022. It categorizes the algorithms based on their application methods, revealing that supervised learning algorithms like Support Vector Machine (SVM), Random Forest, and Naive Bayes are most commonly used. The study finds that the most effective prediction models often employ a combination of different algorithms, underscoring the potential of hybrid approaches in enhancing prediction accuracy.

S. Kaliraj et al. (2024) the authors propose a Genetic Algorithm-based feature selection method to improve software fault prediction by identifying the most useful subset of features. They integrate this method with classifiers like KNN, Decision Tree, and Naive Bayes. Experimental results show that using Genetic Algorithms for feature selection significantly enhances prediction accuracy and reduces the number of features needed, demonstrating its efficacy in handling large datasets.

Khoa Phung et al. (2023) this research introduces a supervised feature selection method based on ant colony optimization for software fault prediction. The method aims to reduce dataset dimensionality and remove redundant features. Using classifiers such as KNN, Naive Bayes, and Decision Tree, the study demonstrates that the ant colony optimization algorithm effectively improves prediction accuracy. Fitness plots and accuracy tables across 12 datasets highlight the method's success.

Gabriel Omar Navarro Cedeño et al.(2023) The paper presents a novel set of software metrics called Error-type Software Metrics (ESM), focusing on predicting Java runtime errors like Index Out Of Bounds Exception, Null Pointer Exception, and Class Cast Exception. By modeling and extracting error patterns using Stream X-Machine and machine learning techniques, the study shows that ESM significantly enhances the performance of fault-proneness prediction models, providing valuable insights into software fault prediction.

Raghuraj Singh et al. (2024) this systematic review examines machine learning algorithms used for software fault prediction, analyzing 52 articles from 2009 to 2022. It finds that supervised learning algorithms, including SVM, Random Forest, and Naive Bayes, are predominant. However, combining different algorithms yields more effective prediction models. The review highlights the complementary nature of these techniques and suggests ensemble methods for improved performance.

Mashiat Zahan (2023) the study explores deep learning techniques for software fault prediction, emphasizing their ability to extract complex patterns from large datasets. It reviews various deep learning models, feature engineering approaches, evaluation metrics, and datasets. The findings suggest that hybrid deep learning and nature-inspired techniques outperform traditional machine learning and deep learning methods, offering better prediction accuracy and robustness.

Sushant Kumar Pandey et.al. (2023)The paper evaluates seven machine learning algorithms, including AdaBoost, CatBoost, LightGBM, Random Forest, XGBoost, and ensemble methods, for predicting faults in embedded software. Using six datasets and eight performance metrics, Random Forest and ensemble stacking methods show the best results in accuracy, precision, and specificity. The study demonstrates the effectiveness of these models in software fault prediction.

Roshan Samantaray et.al.(2023)This research focuses on Just-In-Time Software Fault Prediction (JIT-SFP), proposing a method using a deep belief network and long short-term memory (JITCP-Predictor) to address the challenge of insufficient training data in early software development stages. Cross-Project (CP) data is utilized to enhance predictive performance, and the proposed model significantly outperforms benchmark methods, proving effective for large and moderate-sized projects by avoiding class imbalance and overfitting

issues.

Fatuhu Habib Abba et.al. (2023), the study investigates the impact of the bagging ensemble technique on software fault prediction. Using classifiers like Decision Tree, Logistic Regression, KNN, Gaussian Naive Bayes, and SVM, the research finds that Random Forest, which employs bagging, consistently delivers the best predictive performance. The results highlight the effectiveness of ensemble methods in improving fault prediction accuracy.

Yoginee Surendra Pethe et.al. (2023), this paper explores the use of the Cat Swarm optimization algorithm for predicting functional bugs during the software testing stage. The study shows that this algorithm achieves high accuracy, precision, recall, and F1 scores, demonstrating its effectiveness in reducing rework costs by identifying defects early in the testing process. The results highlight the potential of optimization algorithms in enhancing software fault prediction.

Aman Omer et al. (2024), The authors present a Differential Evolutionary Algorithm (DE) for feature selection in software fault prediction, contrasting it with traditional evolutionary algorithms. The study combines DE with classifiers like Naive Bayes, KNN, and Decision Tree, showing that DE improves prediction accuracy across all classifiers. The findings emphasize the importance of feature selection in handling large datasets for fault prediction.

Baraah Alsangari et al. (2023) The paper introduces a Mixture-of-Experts (MoE) approach for software fault prediction, combining decision trees and multilayer perceptrons with a Gaussian mixture model for improved performance. Experiments on 35 datasets reveal that the MoE-based models outperform individual and ensemble methods, offering better fault prediction accuracy and robustness. The study underscores the potential of combining different learning techniques for enhanced predictive capability.

Xhulja Shahini et al. (2023) this study investigates various machine learning techniques for software fault prediction using the NASA JM1 dataset. Techniques like Logistic Regression, Random Forest, Naive Bayes, SVM, and KNN are evaluated with different normalization methods. The results indicate that Random Forest achieves the highest accuracy, highlighting the effectiveness of this technique in software fault prediction.

Nanfei Yang et al. (2023) The research analyzes the variance caused by no determinism-introducing (NI) factors in machine learning algorithms for software fault prediction. Experimental results show significant variance in prediction accuracy due to stochastic elements, posing challenges for reproducing research results and practical application. The study discusses strategies to mitigate such variance, emphasizing the need for stable and reliable fault prediction models.

Proposed Frameworks for Software Defect Prediction

Recent advances in software fault prediction and reliability modeling have shown significant improvements in addressing the challenges associated with predicting software faults in various contexts. Pradeep Kumar Rath et al. (2023) highlighted the need for enhanced prediction models within agile frameworks, particularly the Scaled Agile Framework (SAFe). Their proposed model, which utilizes a Bi-Directional Gated Recurrent Unit (BiGRU) and a self-attention mechanism, offers a comprehensive approach by considering cumulative sprint effects. This methodology contrasts with traditional models that treat each sprint independently, thereby providing a more accurate and reliable prediction of software faults in large-scale agile projects.Complementing this, Hrishikesh Kumar et al. (2023) tackled the challenges of class imbalance and cross-project prediction in software fault prediction. Their study emphasizes the importance of addressing class imbalance and using datasets with similar features to the target project for reliable predictions. Through comprehensive experiments, they demonstrated the effectiveness of various classifiers in improving model resilience and dependability, highlighting the significance of diverse training samples.

Additionally, Sagnik Mondal et al. (2023) conducted a systematic review of machine learning algorithms for software fault prediction, analyzing 52 articles published between 2009 and 2022. They found that supervised learning algorithms such as Support Vector Machine (SVM), Random Forest, and Naive Bayes are most commonly used. The study revealed that hybrid approaches, combining different algorithms, enhance prediction accuracy, underscoring the potential of such methods.

S. Kaliraj et al. (2024) proposed a Genetic Algorithm-based feature selection method to improve software fault prediction by identifying the most useful subset of features. Their integration of this method with classifiers like KNN, Decision Tree, and Naive Bayes demonstrated significant enhancements in prediction accuracy and reduction in the number of features needed, showcasing its efficacy in handling large datasets.

Khoa Phung et al. (2023) introduced a supervised feature selection method based on ant colony optimization for software fault prediction. Their research demonstrated that this method effectively reduces dataset dimensionality and removes redundant features, resulting in improved prediction accuracy when used with classifiers such as KNN, Naive Bayes,

Gabriel Omar Navarro Cedeño et al. (2023) presented a novel set of software metrics called Error-type Software Metrics (ESM), focusing on predicting Java runtime errors. By modeling and extracting error patterns using Stream X-Machine and machine learning techniques, they showed that ESM significantly enhances the performance of fault-proneness prediction models, providing valuable insights into software fault prediction.

V Conclusion and Future Works

This literature review presents a comprehensive analysis of software defect prediction research from 2000 to 2024. This review aims to illuminate the key trends, approaches, and frameworks employed in this field. By identifying probable problems at an earlier stage in the development lifecycle, the review highlights the significance of defect prediction in terms of improving software quality and lowering development costs. It draws attention to the development of significant trends in defect prediction research, including the utilization of machine learning algorithms, feature selection techniques, and ensemble methods. The paper identifies several obstacles to enhance the efficiency and dependability of defect prediction models. These challenges include heterogeneity in the dataset, imbalanced class distribution, and model generalization issues. In spite of these obstacles, the review serves to illustrate that defect prediction has the potential to have a considerable impact on software development processes. Furthermore, it highlights the necessity of continuing study and innovation in this particular field. Considering this literature evaluation, we can identify several potential directions for future research. First, we need to construct reliable defect prediction models that can efficiently manage the heterogeneity of the dataset, the imbalanced distribution of classes, and other inherent issues in real-world software datasets. In order to accomplish this, it may be necessary to investigate novel feature selection strategies, ensemble methods, and hybrid approaches that incorporate a variety of data sources and modeling models. In addition, there is a growing interest in infusing defect prediction models with domain knowledge and relevant context information in order to improve the accuracy of these models and their application in a variety of software development scenarios. The review highlights the importance of benchmarking and assessment procedures to objectively compare the performance of various defect prediction models. In order to promote fair comparisons and the reproducibility of outcomes, future research should concentrate on standardized evaluation methods and datasets. Future work in software defect prediction has the potential to greatly enhance the state-of-the-art and improve the quality and reliability of software systems. This is due to its ability to tackle current research challenges and explore emerging opportunities.

REFERENCES

[1] Aedah Abd Rahman, Nurdatillah Hasim (2015) Defect Management Life Cycle Process for Software Quality Improvement 2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS) 2015 IEEE

[2] Arvinder Kaur; Kamaldeep Kaur (2016) Value and Applicability of Academic Projects Defect Datasets in Cross-Project Software Defect Prediction 2016 2nd International Conference on Computational Intelligence and Networks (CINE) 2016, IEEE

[3] Azar, D., & Vybihal, J. (2011). An ant colony optimization algorithm to improve software quality prediction models: Case of class stability. Information and Software Technology, 53(4), 388– 393. http://doi.org/10.1016/j.infsof.2010.11.013

[4] Buzan, T., & Griffiths, C. (2013). Mind Maps for Business: Using the ultimate thinking tool to revolutionize how you work (2nd Edition). FT Press.

[5] Cao, H., Qin, Z., & Feng, T. (2012). A Novel PCA-BP Fuzzy Neural Network Model for Software Defect Prediction. Advanced Science Letters, 9(1), 423–428.

[6] Catal, C., Sevim, U., & Diri, B. (2011). Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. Expert Systems with Applications, 38(3), 2347– 2353. http://doi.org/10.1016/j.eswa.2010.08.022

[7] Challagulla, V., Bastani, F., & Yen, I. (2006). A Unified Framework for Defect Data Analysis Using the MBR Technique. 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), 39–46. http://doi.org/10.1109/ICTAI.2006.23

[8] Chang, R. H., Mu, X. D., & Zhang, L. (2011). Software Defect Prediction Using Non-Negative Matrix Factorization. Journal of Software, 6(11), 2114–2120. http://doi.org/10.4304/jsw.6.11.2114-2120

[9] Dejaeger, K.,Verbraken, T.,& Baesens, B.(2013). Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers. IEEE Transactions on Software Engineering, 39(2), 237–257. http://doi.org/10.1109/TSE.2012.20

[10] Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, 81(5), 649–660. http://doi.org/10.1016/j.jss.2007.07.040

[11] Faheem Ahmed; Hasan Mahmood; Adeel Aslam Green computing and Software Defects in open source software: An Empirical study 2014 International Conference on Open Source Systems & Technologies 2014, IEEE

[12] Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., & Mishra, R. (2007). Predicting software defects in varying development lifecycles using Bayesian nets. Information and Software Technology, 49(1), 32–43. http://doi.org/10.1016/j.infsof.2006.09.001

[13] Gondra, I. (2008). Applying machine learning to software fault- proneness prediction. Journal of Systems and Software, 81(2), 186–195.

[14] Gray, D., Bowes, D., Davey, N., & Christianson, B. (2011). The misuse of the NASA Metrics Data Program data sets for automated software defect prediction. 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 96–103.

[15] Güneş Koru, a., & Liu, H. (2007). Identifying and characterizing change-prone classes in two large-scale open-source products. Journal of Systems and Software, 80(1), 63–73. http://doi.org/10.1016/j.jss.2006.05.017

[16] Hamdi A. Al-Jamimi Toward comprehensible software defect prediction models using fuzzy logic 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS) 2016,: IEEE

[17] J. Pai, G., & Bechta Dugan, J. (2007). Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods. IEEE Transactions on Software Engineering, 33(10), 675–686. http://doi.org/10.1109/TSE.2007.70722
[18] Jie Zhang; Jiajing Wu; Chuan Chen; Zibin Zheng; Michael R. Lyu CDS: A Cross–Version Software Defect Prediction Model With Data Selection IEEE Access Year: 2020 Volume: 8 Journal IEEE
[19] Jin, C., Jin, S.-W., & Ye, J.-M. (2012). Artificial neural network- based metric selection for software fault-prone prediction model. IET Software, 6(6), 479.
[20] K. Punitha; S. Chitra 2013 Software defect prediction using software metrics - A survey International Conference on Information Communication and Embedded Systems (ICICES) 2013: IEEE
[21] Khoshgoftaar, T. M., Seliya, N., & Sundaresh, N. (2006). An empirical study of predicting software faults with case-based reasoning. Software Quality Journal, 14(2), 85–111. http://doi.org/10.1007/s11219-006-7597-z
[22] Khoshgoftaar, T. M., Van Hulse, J., & Napolitano, A. (2011). Comparing Boosting and Bagging Techniques With Noisy and Imbalanced Data. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 41(3), 552–568.
[23] Koru, A. G., & Liu, H. (2005). An investigation of the effect of module size on defect prediction using static measures. In Proceedings of the 2005 workshop on Predictor models in software engineering - PROMISE '05 (Vol. 30, pp. 1–5). New York, New York, USA: ACM Press. http://doi.org/10.1145/1082983.1083172
[24] Liu, Y., Khoshgoftaar, T. M., & Seliya, N. (2010). Evolutionary Optimization of Software Quality Modeling with Multiple Repositories. IEEE Transactions on Software Engineering, 36(6), 852–864.
[25] Lyu, M. R. (2000). Software quality prediction using mixture models with EM algorithm. In Proceedings First Asia-Pacific Conference on Quality Software (pp. 69–78). IEEE Comput. Soc. http://doi.org/10.1109/APAQ.2000.883780
[26] Ma, Y., Guo, L., & Cukic, B. (2007). A Statistical Framework for the Prediction of Fault-Proneness. In Advances in Machine Learning Applications in Software Engineering (pp. 1–26).
[27] Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. Information and Software Technology, 54(3),248–256. http://doi.org/10.1016/j.infsof.2011.09.007
[28] Martin Shepperd; Tracy Hall; David Bowes Authors' Reply to "Comments on 'Researcher Bias: The Use of Machine Learning in Software Defect Prediction'" IEEE Transactions on Software Engineering 2018 | Volume: 44, Issue: 11 Journal IEEE
[29] Mehmet Söylemez; Ayça Tarhan Using Process Enactment Data Analysis to Support Orthogonal Defect Classification for Software Process Improvement 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement IEEE
[30] Mende, T., & Koschke, R. (2009). Revisiting the evaluation of defect prediction models. Proceedings of the 5th International Conference on Predictor Models in Software Engineering - PROMISE '09, 1. http://doi.org/10.1145/1540438.1540448
[31] Menzies, T., DiStefano, J., Orrego, A. S., & Chapman, R. (2004). Assessing predictors of software defects. In Proceedings of the Workshop on Predictive Software Models.
[32] Myrtveit, I., Stensrud, E., & Shepperd, M. (2005). Reliability and validity in comparative studies of software prediction models. IEEE Transactions on Software Engineering, 31(5), 380–391. http://doi.org/10.1109/TSE.2005.58
[33] Naik, K., & Tripathy, P. (2008). Software Testing and Quality Assurance. John Wiley & Sons, Inc.

[34] Nur Hafizah Haron; Sharifah Mashita Syed-Mohamad Test and Defect Coverage Analytics Model for the assessment of software test adequacy 2015 9th Malaysian Software Engineering Conference (MySEC) 2015, IEEE
[35] Patrick Rempel; Parick Mäder Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality IEEE Transactions on Software Engineering Year: 2017 | Volume: 43, Issue: 8 | Journal Article | Publisher: IEEE
[36] Pelayo, L., & Dick, S. (2012). Evaluating Stratification Alternatives to Improve Software Defect Prediction. IEEE Transactions on Reliability, 61(2), 516–525. http://doi.org/10.1109/TR.2012.2183912
[37] Peng Xiao; Bin Liu; Xiaobo Yan; Fuqun Huang How Domain Knowledge Accumulation Influences Software Defects: An Empirical Analysis 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C) 2017 ,IEEE
[38] Peng, Y., Wang, G., & Wang, H. (2012). User preferences based software defect detection algorithms selection using MCDM. Information Sciences,191,3–13.
[39] Peters, F., Menzies, T., Gong, L., & Zhang, H. (2013). Balancing Privacy and Utility in Cross-Company Defect Prediction. IEEE Transactions on Software Engineering, 39(8), 1054–1068. http://doi.org/10.1109/TSE.2013.6
[40] Pizzi, N. J., Summers, A. R., & Pedrycz, W. (2002). Software quality prediction using median-adjusted class labels. Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290), (1), 2405–2409.http://doi.org/10.1109/IJCNN.2002.1007518
[41] Qinbao Song; Yuchen Guo; Martin Shepperd A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction IEEE Transactions on Software Engineering ,2019 Volume: 45, Issue: 12 Journal Publisher: IEEE
[42] Rakesh Rana; Miroslaw Staron; Jörgen Hansson; Martin Nilsson Defect prediction over software life cycle in automotive domain state of the art and road map for future 2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA) Year: 2014, IEEE
[43] Seiffert, C., Khoshgoftaar, T. M., & Van Hulse, J. (2009). Improving Software-Quality Predictions with Data Sampling and Boosting. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 39(6), 1283–1294.
[44] Seliya, N., & Khoshgoftaar, T. M. (2007). Software Quality Analysis of Unlabeled Program Modules with Semi supervised Clustering. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 37(2), 201–211.http://doi.org/10.1109/TSMCA.2006.889473
[45] Shamsul Huda; Sultan Alyahya; Md Mohsin Ali; Shafiq Ahmad; Jemal Abawajy; Hmood Al-Dossari; John Yearwood A Framework for Software Defect Prediction and Metric Selection IEEE Access 2018, Volume: 6, Journal IEEE
[46] Shepperd, M., & Kadoda, G. (2001). Comparing software prediction techniques using simulation. IEEE Transactions on Software Engineering, 27(11),1014–1022. http://doi.org/10.1109/32.965341
[47] Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data Quality: Some Comments on the NASA Software Defect Datasets. IEEE Transactions on Software Engineering, 39(9), 1208– 1215. http://doi.org/10.1109/TSE.2013.11
[48] Song Wang; Taiyue Liu; Jaechang Nam; Lin Tan Deep Semantic Feature Learning for Software Defect Prediction IEEE Transactions on Software Engineering Year: 2020 | Volume: 46, Issue: 12 ,Journal Article IEEE
[49] Sun, Z., Song, Q., & Zhu, X. (2012). Using Coding-Based Ensemble Learning to Improve Software Defect Prediction. IEEE Transactions on Systems, Man, and

Cybernetics, Part C (Applications and Reviews), 42(6), 1806–1817. http://doi.org/10.1109/TSMCC.2012.2226152

[50] Syed Rashid Aziz; Tamim Khan; Aamer Nadeem Experimental Validation of Inheritance Metrics' Impact on Software Fault Prediction IEEE Access ,2019 Volume: 7 , Journal IEEE

[51] Turhan, B., Kocak, G., & Bener, A. (2009). Data mining source code for locating software bugs: A case study in telecommunication industry. Expert Systems with Applications, 36(6), 9986–9990. http://doi.org/10.1016/j.eswa.2008.12.028

[52] Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 14(5), 540– 578. http://doi.org/10.1007/s10664-008-9103-7

[53] Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., & Haesen, R. (2008). Mining software repositories for comprehensible software fault prediction models. Journal of Systems and Software, 81(5), 823–839.

[54] Wang, H., Khoshgoftaar, T. M., & Napolitano, A. (2010). A Comparative Study of Ensemble Feature Selection Techniques for Software Defect Prediction. 2010 Ninth International Conference on Machine Learning and Applications, 135–140.

[55] Wang, S., & Yao, X. (2013). Using Class Imbalance Learning for Software Defect Prediction. IEEE Transactions on Reliability, 62(2), 434–443.

[56] Wong, W. E., Debroy, V., Golden, R., Xu, X., & Thuraisingham, B. (2012). Effective Software Fault Localization Using an RBF Neural Network. IEEE Transactions on Reliability, 61(1), 149– 169. http://doi.org/10.1109/TR.2011.2172031

[57] Xing, F., Guo, P., & Lyu, M. R. (2005). A Novel Method for Early Software Quality Prediction Based on Support Vector Machine. 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), 213–222. http://doi.org/10.1109/ISSRE.2005.6

[58] Y. I. Peng, G. Kou, G. Wang, W. Wu, and Y. Shi, "Ensemble of Software Defect Predictors: An Ahp-Based Evaluation Method," International Journal of Information Technology & Decision Making, vol. 10, pp. 187- 206, 2011.

[59] Zhou, Y., & Leung, H. (2006). Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. IEEE Transactions on Software Engineering, 32(10), 771–789. http://doi.org/10.1109/TSE.2006.102

[60] Song,Q.,Jia,Z.,Shepperd,M.,Ying,S.,&Liu,J.(2011).A General Software Defect-Proneness Prediction Framework. IEEE Transactions on Software Engineering, 37(3), 356–370.

[61] El Emam, K., & Laitenberger, O. (2001). Evaluating capture- recapture models with two inspectors. IEEE Transactions on Software Engineering, 27(9), 851–864. http://doi.org/10.1109/32.950319

[62] Karthik, R., & Manikandan, N. (2010). Defect association and complexity prediction by mining association and clustering rules. 2010 2nd International Conference on Computer Engineering and Technology, V7–569–V7–573.

[63] Cukic, B., & Singh, H. (2004). Robust Prediction ofFault-Proneness by RandomForests.15thInternationalSymposiumonSoftware Reliability Engineering, 417–428. http://doi.org/10.1109/ISSRE.2004.35

[64] Catal, C., Sevim, U., & Diri, B. (2011). Practical development of an Eclipse-based software fault prediction to lousing Naive Bayes algorithm. Expert Systems with Applications, 38(3), 2347– 2353. http://doi.org/10.1016/j.eswa.2010.08.022

[65] Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P., & Aud, S. J. (1997). Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks / a Publication of the IEEE Neural

Networks Council, 8(4), 902–9. http://doi.org/10.1109/72.595888
[66] Bishnu, P. S., & Bhattacherjee, V. (2012). Software Fault Prediction Using Quad Tree-BasedK-Means Clustering Algorithm. IEEE Transactions on Knowledge and Data Engineering, 24(6), 1146–1150.
[67] Sun, Z., Song, Q., &Zhu, X. (2012). Using Coding-Based Ensemble Learning to Improve Software Defect Prediction. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),42(6), 1806–1817
[68] Mısırlı, A. T., Bener, A. B., & Turhan, B. (2011). An industrial case study of classifier ensembles for locating software defects. Software Quality Journal, 19(3), 515–536. http://doi.org/10.1007/s11219-010-9128-1
[69] Peng, J., & Wang, S. (2010). Parameter Selection of Support Vector Machine based on Chaotic Particle Swarm Optimization Algorithm. Electrical Engineering, 3271–3274
[70] Challagulla, V. U. B., Bastani, F.B., & Paul, R.A. (2004).Empirical Assessment of Machine Learning based Software Defect Prediction Techniques. In 10th IEEE International
[71] Maimon, O., & Rokach, L. (2010). Data Mining and Knowledge Discovery Handbook Second Edition. Springer
[72] Khoshgoftaar, T.M.,& Van Hulse,J.(2009).Empirical Case Studies in Attribute Noise Detection. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 39(4),379–388.
[73] Sammut, C., & Webb, G. I. (2011). Encyclopedia of Machine Learning. Springer. Sandhu, P.S., Kumar, S.,&Singh, H.(2007).Intelligence Systemfor Software Maintenance Severity Prediction. Journal of Computer Science, 3(5), 281–288. http://doi.org/10.3844/jcssp.2007.281.288
[74] Menzies,T.,Milton,Z.,Turhan,B.,Cukic,B.,Jiang,Y.,&Bener, A. (2010). Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering, 17(4), 375–407.
[75] Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, 34(4), 485–496.
[76] Hrishikesh Kumar; Himansu Das(2023) "Software Fault Prediction using Wrapper based Feature Selection Approach employing Genetic Algorithm" 2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON) 2023
[77] Sagnik Mondal; Adarsh Kumar Sahu; Hrishikesh Kumar; Radha Mohan Pattanayak; Mahendra Kumar Gourisaria; Himansu Das (2023)"Software Fault Prediction using Wrapper based Ant Colony Optimization Algorithm for Feature Selection" 2023 6th International Conference on Information Systems and Computer Networks (ISCON), 2023
[78] S. Kaliraj;A. M. Kishoore;V. Sivakumar(2024) "Software Fault Prediction Using Cross-Project Analysis: A Study on Class Imbalance and Model Generalization" IEEE Access,2024
[79] Khoa Phung; Emmanuel Ogunshile; Mehmet Aydin "Error-Type—A Novel Set of Software Metrics for Software Fault Prediction" IEEE Access ,2023
[80] Gabriel Omar Navarro Cedeño; Katherine Cortés Moya; Ahmed Somarribas Dormond; Antonio González- Torres; Yenory Rojas-Hernández(2023) "Systematic Literature Review: Machine Learning for Software Fault Prediction" ,2023
[81] Raghuraj Singh; Kuldeep Kumar(2024) "Software Fault Prediction in Service-Oriented Based Systems" 2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT) ,2024

[82] Mashiat Zahan (2023) "Prediction of Faults in Embedded Software Using Machine Learning Approaches" 2023 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD), 2023
[83] Sushant Kumar Pandey; Anil Kumar Tripathi (2023) "Cross-Project setting using Deep learning Architectures in Just-In-Time Software Fault Prediction: An Investigation" 2023 IEEE/ACM International Conference on Automation of Software Test (AST) ,2023
[84] Roshan Samantaray; Himansu Das(2023) Performance Analysis of Machine Learning Algorithms Using Bagging Ensemble Technique for Software Fault Prediction2023 6th International Conference on Information Systems and Computer Networks (ISCON), 2023
[85] Fatuhu Habib Abba; Kabir Umar; Umar Adam Ibrahim; Abubakar Ibrahim Dalhatu(2023)"Search-Based Prediction of Software Functional Fault Slip-Through"2023 2nd International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS) ,2023
[86] Yoginee Surendra Pethe; Himansu Das(2023)"Software fault prediction using a differential evolution-based wrapper approach for feature selection" 2023 International Conference on Communication, Circuits, and Systems (IC3S) ,2023
[87] Aman Omer; Santosh Singh Rathore; Sandeep Kumar(2024) "ME-SFP: A Mixture-of-Experts-Based Approach for Software Fault Prediction "IEEE Transactions on Reliability 2024
[88] Baraah Alsangari; Göksel Bircik (2023) "Performance Evaluation of various ML techniques for Software Fault Prediction using NASA dataset" 2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA),2023
[89] Xhulja Shahini; Domenic Bubel; Andreas Metzger(2023) "Variance of ML based software fault predictors: are we really improving fault prediction" 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) ,2023
[90] Nanfei Yang; Yuliang Shi; Zhiyuan Su; Xinjun Wang; Zhongmin Yan; Fanyu Kong(2023) "FSFP: A Fine-Grained Online Service System Performance Fault Prediction Method Based on Cross-attention" 2023 30th Asia-Pacific Software Engineering Conference (APSEC),2023