Journal of Computer Science

Review

Machine Learning Techniques for Software Bug Prediction: A Systematic Review

[1]Syahana Nur'Ain Saharudin, [1]Koh Tieng Wei and [2]Kew Si Na

[1]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), 43400 Serdang, Malaysia
[2]Faculty of Social Sciences and Humanities, Universiti Teknologi Malaysia (UTM), 81310 Skudai, Malaysia

Article history
Received: 29-09-2020
Revised: 12-11-2020
Accepted: 14-11-2020

Corresponding Author:
Koh Tieng Wei
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), 43400 Serdang, Malaysia
Email: twkoh@upm.edu.my

Abstract: The goal of software bug prediction is to identify the software modules that will have the likelihood to get bugs by using some fundamental project resources before the real testing starts. Due to high cost in correcting the detected bugs, it is advisable to start predicting bugs at the early stage of development instead of at the testing phase. There are many techniques and approaches that can be used to build the prediction models, such as machine learning. This technique is widely used nowadays because it can give accurate results and analysis. Therefore, we decided to perform a review of past literature on software bug prediction and machine learning so that we can understand better about the process of constructing the prediction model. Not only we want to see the machine learning techniques that past researchers used, we also assess the datasets, metrics and performance measures that are used during the development of the models. In this study, we have narrowed down to 31 main studies and six types of machine learning techniques have been identified. Two public datasets are found to be frequently used and object-oriented metrics are the highly chosen metrics for the prediction model. As for the performance measure, both graphical and numerical measures are often used to evaluate the performance of the models. From the results, we conclude that the machine learning technique can predict the bug, but there are not many applications in this area that exist nowadays. There are a few challenges in constructing the prediction model. Thus, more studies need to be carried out so that a well-formed result is obtained. We also provide a recommendation for future research based on the results we got from this study.

Keywords: Software Bug Prediction, Machine Learning Techniques, Literature Review

Introduction

Software quality modelling is an important part in the software development process and this concept is well-known in the software engineering field (Al-Jamimi, 2016). Also, testing is considered as the most essential stage in the development process because this stage is strongly linked to the software quality. If the bugs are detected earlier through prediction, then the quality of software can be improved. With the earlier detection of bug, testers can be assisted in defining the delivery of resources wisely so that the bug can be successfully detected (Xia *et al*., 2014). When bugs are found before the release of the software, they can be removed before the deployment of the software. The goals of software bug prediction, especially when being applied to the early stage (Hassan *et al*., 2018), are to increase the value of the software and lessen the cost, which eventually offer a well-panned software management.

Currently, it is a new era of technology and because of this the complexity and magnitude of a software has grown rapidly. Therefore, testing plays an important part during the development process. Menzies *et al*. (2010; Wahono, 2015) stated that the chance of detection using this approach might be higher than the chance of current reviews that is used in the industry. Due to this, software bug prediction is a popular research area in the field of software engineering today. This research has attracted many researchers from different domains, making them propose a variety of

frameworks, models and techniques for bug prediction. There are also researchers that focused on improving the existing techniques and models.

Despite many efforts have been performed, the research area of software bug prediction still has many ambiguities. Even though there are many models and frameworks have been proposed, not a single technique has its own limitations. Among all the domains, the widely used approach is machine learning. Different machine learning algorithms are used to detect bugs, such as neural network, support vector machine and bayesian network. There are also different datasets that are available publicly so that the practitioners can easily conduct their experiment without having any worry on data, such as PROMISE and NASA MDP repositories. These datasets have various metrics, which said to be related to defective or non-defective modules, such as Halstead metrics and McCabe metrics. In order to check the performance of the proposed model, different type of performance measures are used for evaluation such as Area Under Curve (AUC) or F-Measure.

To enable the practice of machine leaning techniques in the context of bug prediction, it is required to review the experimental evidence gained on these techniques through the existing studies. Kamei and Shihab (2016) discussed on software bug prediction in their recent work. However, this study only give a summary on bug prediction, its component and laid down some achievements that have been made in the area. Wahono (2015) also conducted a review on software bug prediction, but the study focused on the datasets used for the prediction model, its methods and frameworks that have been proposed by past studies. Also, the study included the past literature from 2000 to 2013. Jayanthi and Florence (2017) presented a review on defect prediction techniques using product metrics. The study analyzed various software metrics and summarized the techniques used for defect prediction. Not only that, the study also discussed on the constraints and limitation of building software defect prediction model. However, the study did not include the datasets used for the model and the performance measures to evaluate the models. Prasad and Sasikala (2019) also presented a review on software defect prediction techniques, but did not mentioned the software metrics used, the datasets and the performance measures.

Our study will be focusing on several scopes of software bug prediction. The objective of this study is to summarize, analyze and evaluate the experimental evidence on the machine learning techniques that have been used in software bug prediction. We will also be evaluating the datasets used for the model, frequently used software metrics and the performance measures for model's assessment. Therefore, we can obtained the desirable techniques and methods that can be used in the future experiment.

The rest of the paper is structured as follows: Section 2 discussed the method in discovering the related studies and presented how research questions are defined. Section 3 discussed the results to the research questions. Section 4 described an overview of the bug prediction models, along with some challenges based on the past studies. Section 5 presented the limitation of this study. Finally, in section 6, we concluded the paper and provide recommendations for future work.

Methodology

The methodology for this study is Systematic Literature Review (SLR). This approach has been chosen to review the studies on software bug prediction and SLR is a well-known review method, which consist of identifying, evaluating and understanding the available research evidence with the goal of answering the defined research questions (Kitchenham and Charters, 2007).

Research Question

In order to guide us for the reviewing and assessment of the past studies, research questions are defined. These questions were designed according to Population, Intervention, Comparison, Outcomes and Context (PICOC) criteria (Kitchenham and Charters, 2007). Table 1 describes the criteria of PICOC.

The purpose of this review is to provide and evaluate the experimental evidence gained from the past studies regarding the usage of machine learning techniques for bug prediction model. The research questions that will be answered in this SLR are listed down as below:

- RQ1 - Which datasets are frequently used for software bug prediction?
- RQ2 - What kind of machine learning techniques that have been selected for prediction model?
- RQ3 - Which metrics are frequently used for software bug prediction?
- RQ4 - Which performance measures are used for software bug prediction?

Table 1: PICOC criteria

Population	Software, system, application, information system
Intervention	Software bug prediction, software defect prediction, software fault prediction, error-prone, bug-prone, techniques, methods
Comparison	Not available
Outcomes	Positive bug prediction techniques
Context	Small and large datasets, studies in academy and industry

Review Protocol

The process of searching the studies include choosing digital repositories, constructing the search string, performing an initial search and getting the first list of main studies from the digital repositories that matched the search string. Appropriate digital repositories were selected and the digital databases that are used to do the searching are listed as follow:

- ScienceDirect
- Google Scholar
- SpringerLink
- IEEE Xplore

After choosing the repositories, we need search string to perform an exhaustive search in order to select the main studies. We chose exhaustive search because the number of main studies is not very large, along with a smaller number of studies that focused on empirical research. The combination of words and characters that have been entered by the user are known as a search string and this is used to find the desired results. The results given by the digital databases can be affected by the information provided to the search engine. If we want to guarantee that all the main studies have been covered, we need to be wary when selecting the keywords and in placing the keywords into the search string. Therefore, we defined a few steps to construct the search string and the steps are listed as below:

- Identify the search terms by analyzing the research questions using PICOC
- Identify the search terms in significant titles, abstracts and keywords
- Identify the alternative words of search terms
- Use Boolean and/or when defining search string

Using the steps that have been defined above, we eventually used the following search string:

Software and (Bug or Fault or Defect) and (Proneness or Prediction) and (Machine Learning or Neural Network or Bayesian Network or Decision Tree or Support Vector Machine or Random Forest)

The four digital databases that were listed above have been used as the platform for the defined search string. We restricted the search from 2014 to 2020 in order to identify the machine learning techniques that are used in the current research. In order to select the main studies from the initial list, the inclusion and exclusion criteria were designed. These criteria are listed below:

a. Inclusion criteria

- Studies that discuss software bug prediction model using machine learning
- Studies that discuss and compare the performance of bug prediction models
- Studies that are empirical in nature
- Studies that have been presented at Q1 and Q2 journal
- Studies that are written in English

b. Exclusion criteria

- Studies that do not discuss about software bug prediction model using machine learning
- Studies that do not discuss on the performance of bug prediction models
- Studies that are not empirical
- Studies that do not presented at Q1 and Q2 journal
- Studies that do not written in English

Based on the search string that had been designed, we managed to collect a total of 1452 initial list of studies from four digital repositories. Then, we excluded the main studies based on the title and abstract, which lead us to 213 main studies. We continued to examine these main studies thoroughly and applied the inclusion and exclusion criteria and finally narrowed down to 31 studies. Table 2 presents the number of studies from their respective digital repositories.

Data Extraction

The main studies are taken from the repositories so that the gathered data can contribute to the research questions concerned in this SLR. The form of data extraction was designed to gather data from the main studies that are necessary to answer the research questions. The characteristics that are used to answer the research questions are shown in Table 3, whereas Table 4 shows the relationship between the main studies and research questions, whether the studies answered the questions or not.

Table 2: Summary of search results

Repository	Initial list	Second list	Final list
ScienceDirect	226	82	16
Google Scholar	143	22	4
SpringerLink	319	51	8
IEEE Xplore	764	58	3
Total	1452	213	31

Table 3: Data extraction characteristics linked to research questions

Characteristic	Research question
Researchers, publications, titles	General
Software bug datasets	RQ1
Software bug prediction machine Learning techniques	RQ2
Software metrics	RQ3
Performance measures for software bug prediction model	RQ4

Table 4: Result of data extraction

Study ID	Reference	RQ1	RQ2	RQ3	RQ4
S1	Erturk and Sezer (2015)	√	√	√	√
S2	Kumar (2018)	√	√	√	√
S3	Pan *et al*. (2019)	√	√	√	√
S4	Zhou *et al*. (2019)	√	√		√
S5	Jin and Jin (2015)	√	√	√	√
S6	Abaei and Selamat (2014)	√	√		√
S7	Okutan and Yildiz (2014)	√	√	√	√
S8	Arar and Ayan (2015)	√	√	√	√
S9	Laradji *et al*. (2015)	√	√	√	√
S10	Rhmann *et al*. (2020)	√	√	√	√
S11	Majd *et al*. (2020)	√	√	√	√
S12	Boucher and Badri (2018)	√	√	√	√
S13	Park and Hong (2014)	√	√	√	√
S14	Jakhar and Rajnish (2018)	√	√	√	√
S15	Ma *et al*. (2014)	√	√	√	√
S16	Ni *et al*. (2017)	√	√	√	√
S17	Kalsoom *et al*. (2018)	√	√	√	√
S18	Miholca *et al*. (2018)	√	√	√	√
S19	Wu *et al*. (2018)	√	√	√	√
S20	Mori and Uchihira (2019)	√	√	√	√
S21	Geng (2018)	√	√	√	√
S22	Dong *et al*. (2018)	√	√		√
S23	Abaei *et al*. (2015)	√	√	√	√
S24	Ryu *et al*. (2015)	√	√	√	√
S25	Rathore and Kumar (2017)	√	√	√	√
S26	Rana *et al*. (2015)	√	√	√	√
S27	Ji *et al*. (2019)	√	√	√	√
S28	Hua *et al*. (2019)	√	√	√	√
S29	Zhao *et al*. (2018)	√	√	√	√
S30	Wei *et al*. (2018)	√	√	√	√
S31	Yang *et al*. (2014)	√	√	√	√

Result

Datasets

Dataset is known as a collection of information that is used in the specific domain in order to solve the problem under consideration. There are various datasets that are available publicly for the researchers to use in order to construct the bug prediction model. It is not easy to find a standard dataset, especially from organization, because organization mostly reluctant to display their datasets to the public (Kamei and Shihab, 2016). However, public datasets have issue with the quality. Pan *et al*. (2019) simplified the existing dataset, such as PROMISE dataset, to solve this issue and constructed Simplified PROMISE Source Code (SPSC) dataset. The authors simplified the dataset by enlarging the original datasets for their research. Many researchers came out with different frameworks using different datasets and it is not easy to assess the proposed frameworks because of their different nature in datasets. Figure 1 shows the percentage of datasets that are frequently used in the main studies.

Based from the gathered results, we can conclude that both PROMISE repository (Sayyad, 2005) and NASA Metrics Data Program (MDP) (Jacob and Raju, 2017) repository are mostly used by past researchers as datasets for software bug prediction. Both repositories were used in 13 studies respectively. PROMISE repository is a library for software engineering research and offers free and long-term storage for research datasets. This repository consist dataset such as SOFTLAB and NASA datasets, which mostly about the industrial software projects and can help researchers in the development of predictive models. NASA MDP datasets is a library that stores problem, product and metrics data. The datasets consist of 13 original NASA datasets and metrics were generated from these datasets and then reports were generated and made available to the public freely.

AEEM datasets, which had been used in three studies, were collected by (D'Ambros *et al*., 2010) and the datasets include Eclipse and Apache. The purpose of this dataset is to compare the performance of different feature space. Relink datasets was collected by (Wu *et al*., 2011) and it has been used in two studies. Other datasets that are used in the remaining studies are open source Java projects, Git repository, Code4Bench and Android projects. All of the datasets that are used in the main

studies are public datasets and because of this, the datasets have attracted many researchers to perform their studies.

Machine Learning Techniques

Many techniques for software bug prediction are presented in the literature and based from the 31 studies, we classified the six most used techniques in software bug prediction. The methods and distribution of the studies are shown in Fig. 2. Despite many studies reported on the comparison regarding the techniques' performance in modelling the bug prediction, there is no solid agreement on the best technique when we looked at the studies individually. The six techniques that have been identified are Bayesian Network (BN), Neural Network (NN), Support Vector Machine (SVM), Clustering, Feature Selection (FS) and Ensemble Learning (EL).

Among these techniques, the most widely used is NN, such as Artificial Neural Network (ANN), Deep Neural Network (DNN) and Convolutional Neural Network (CNN). Arar and Ayan (2015) pointed out that the feasibility of NN is restricted because of the trouble in choosing the right parameters for network architecture even though NN has a good accurateness as a classifier when it comes to predicting bugs. Therefore, the authors proposed to combine ANN with novel Artificial Bee Colony (ABC) algorithm in order to find the optimal weights of the bugs as the parameter. Miholca *et al*. (2018) also proposed a new framework, where they combined ANN with gradual relational association rule to separate between defective and non-defective software entities.

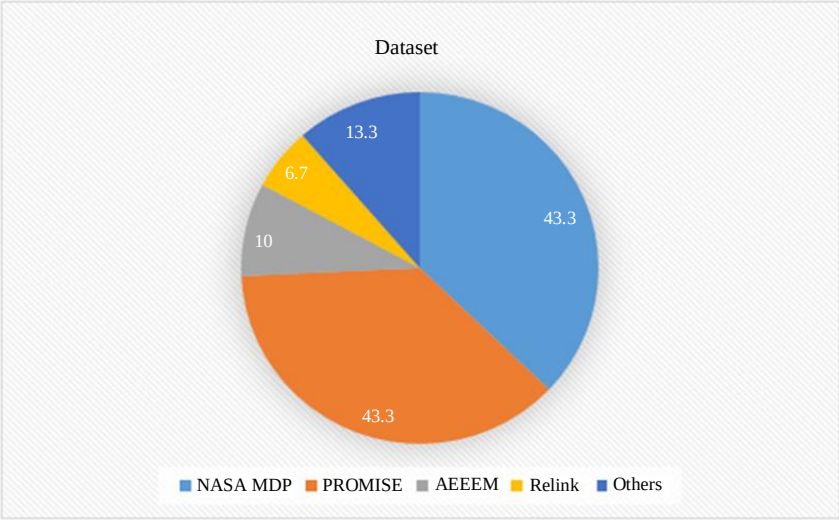

Fig. 1: Distribution on Software Bug Datasets

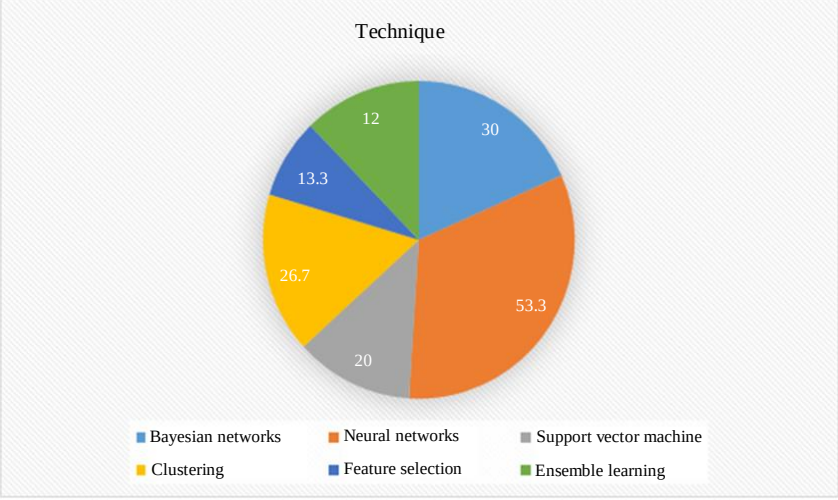

Fig. 2: Distribution on machine learning techniques

There is also a Study on combining ANN with Self-Organizing Map (SOM) (Abaei *et al*., 2015), where the goal is to predict the label of the modules. SOM is one of NN based algorithm that creates a similarity map of input data and the concept is it compresses information while preserving the most important relationships of main data (Li *et al*., 2010). The combination of ANN and SOM proposed by (Abaei *et al*., 2015) found that the hybrid model can be used as an automated tool to assist the testing effort by prioritizing the module's defects, leading to increasing quality of development.

Bayesian algorithms, specifically Naïve Bayes (NB), are second widely used in modelling the bug prediction. NB has better performance because of its easiness for the certain dataset. Despite its simplicity, there is still room for improvement. Wu *et al*. (2018) proposed a novel classifier called diffused Bayes to increase the performance of traditional NB classifier. The new classifier obtained better result compared to the traditional classifier through a diffusion function that is built on the vibration of string. The new classifier is proposed as a solution to the short supply of cross-project training data and non-normal distributed attributes (Mori and Uchihira, 2019).

Clustering techniques is known as unsupervised learning methods and it is more suitable to use in the case where the label of the bugs is not presented. Ryu *et al*. (2015) used K-nearest neighbor, which is one of the clustering algorithms, to predict bug. They implemented the algorithm with NB to solve the class imbalance problem, where the ratio of bug class to clean class is far low. Therefore, the authors proposed a hybrid framework using K-nearest neighbor and NB, where the K-nearest neighbor is used to select the learning local knowledge and NB is used to select global knowledge. Their experimental results display high performance of bug prediction.

SVM is quite a popular algorithm to be used as classifier of machine learning. However, in recent studies, the algorithm is not widely used because it is said to perform less well in software bug prediction. SVM might perform below expectation since they required parameter optimization to get great performance. Because of this problem, (Wei *et al*., 2018) integrated traditional SVM with NPE algorithm to improve SVM performance. NPE algorithm can holds the vital problems of bug measurement in high-dimensional and small case.

EL techniques, which possess the same percentage as SVM, have a positive impact in handling small-sized and imbalanced datasets. EL models have been shown to provide better performance compared to single weak learners, especially when it comes to dealing with high dimensional, classification problems and complex regression (Kazienko *et al*., 2013). The most popular EL algorithm is Random Forest, where it consists of several regression trees. The concept of Random Forest is they built trees that make random choices on which variables to exclude at each node, but this kind of concept can lead to high-dimensional spaces problem. Therefore, (Zhou *et al*., 2019) used cascade strategy on traditional Random Forest to help choose suitable bug features and representation learning based on the layer-by-layer structure.

FS is the study of algorithms to reduce data's dimension so that the performance of the technique can be improved. However, most of the studies used FS as a method to select the best metrics and classifiers to be used for software bug prediction (Kumar, 2018; Laradji *et al*., 2015; Jakhar and Rajnish, 2018; Ni *et al*., 2017).

Software Metrics

Software metrics are used as independent variables when predicting bug proneness in most of the studies. In the domain of software engineering, there exist several metrics to measure the value of the software. We describe the type of metrics used in the main studies as independent variable in Fig. 4 and 3 shows the percentage of metrics used in the main studies.

The frequently used software metric in the main studies is McCabe metrics, such as Cyclomatic Complexity, Essential Complexity and Design Complexity, which was introduced by Thomas McCabe in 1976. Line Of Code (LOC) metrics have been used in half of the main studies related to software bug prediction, by measuring the number of lines in a code, number of comment, number of code and comment and so forth. LOC is the most useful in bug prediction if we integrated it with other software metrics. Halstead metrics, which was introduced by (Halstead, 1977), also widely used in the main studies. The goal of the metrics is to identify the measurable attributes of software and the relations between them.

CK Metrics Suite was proposed by (Chidamber and Kemerer, 1994) and it is widely used to measure the characteristics of object-oriented systems such as inheritance, classes and encapsulation (Michura *et al*., 2013). QMOOD metrics, which stands for Quality Model for Object-Oriented Design metrics, was proposed by (Bansiya and Davis, 2002) and it measures the relationship between quality attributes and design property that have been defined (Couto *et al*., 2014). The widely used Martin's metrics was presented by (Andresen *et al*., 1994) and the purpose is to measure the quality of object-oriented design by looking at the interdependence between the classes (Kaur and Sharma, 2015).

Miscellaneous referred to other metrics that have been used in the main studies besides the one mentioned above. Other metrics that are used are branch count, requirements metrics, decision count, edge count, code churn, metric and Henderson-Sellers metric. The metric proposed by (Tang *et al.* 1999) is a quality-oriented metrics that extended the original CK Metrics Suite. Henderson-Sellers metric, which was proposed in 1996, is also an extension of CK Metrics Suite. The extension was done based on the Lack of Cohesion in Methods (LCOM). Code Churn metrics measure the amount of changes in code that take place within a software unit over time. There are two types of churn metrics (Yang *et al.*, 2014) which are LOC-ADDED and LOC-DELETED.

Certain studies reported that the object-oriented metrics, such as CK Metrics Suite and QMOOD metrics are strongly connected to bug proneness. Coupling Between Objects (CBO) and Response For a Class (RFC), which are CK Metrics and LOC are the best metrics for software bug prediction based on feature selection methods (Okutan and Yildiz, 2014; Boucher and Badri, 2018). Kumar. (2018) added that Measure Of Aggregation (MOA), Cohesion Among Methods of class (CAM), Coupling between Methods (CBM) and Average Method Complexity (AMC) as the best metrics. CBM and AMC are metrics proposed by the authors had used two types of feature selection methods, such as feature ranking method and feature subset selection, to determine which metrics are useful for software bug prediction. On the other hand, the results obtained from the main studies specified that Number Of Children (NOC) and Depth of Inheritance Tree (DIT) as not the best metrics for software bug prediction (Okutan and Yildiz, 2014). However, none of the main studies give any result for procedural metrics, such as Halstead and McCabe metrics, that are not useful for software bug prediction.

Performance Evaluation

It is essential to evaluate the proposed approach because it is to check its efficiency and effectiveness. Different evaluation strategies are used by different researchers to evaluate the performance of their proposed approach. Figure 5 shows the percentage of performance measures for evaluation. There are two types of measurement, such as graphical measure and numerical measure. Graphical measure consists of precision-recall curve, cost curve and ROC curve, whereas numerical measure consist of accuracy, F-Measure, precision and others more.

Based on the results that we have obtained, Area Under Curve (AUC) has been frequently used in the studies. The success of the prediction model is depended on the calculation of area under the ROC curve and this measurement are used to test the usefulness of the models. Recall is the second most widely used performance measure for bug prediction model. This measurement is regarding the quantity of bug-prone classes that have been predicted correctly among the actual bug-prone classes. F-Measure is the next performance measure that the researchers used and this measure provides the trade-off between the classifier's performance. Precision is where we measure the correctness of the model, whereas accuracy can be defined as the amount of correctly identified bugs divided by the total number of bugs.

Other metrics that are not frequently used in the main studies are MCC (6.7%), TER (13.3%), Specificity (6.7%), Probability of false alarm (10.0%), False positive rate (16.7%), False negative (13.3%), G-Mean (0.3%), Balance (0.3%) and normalized expected cost (0.3%).

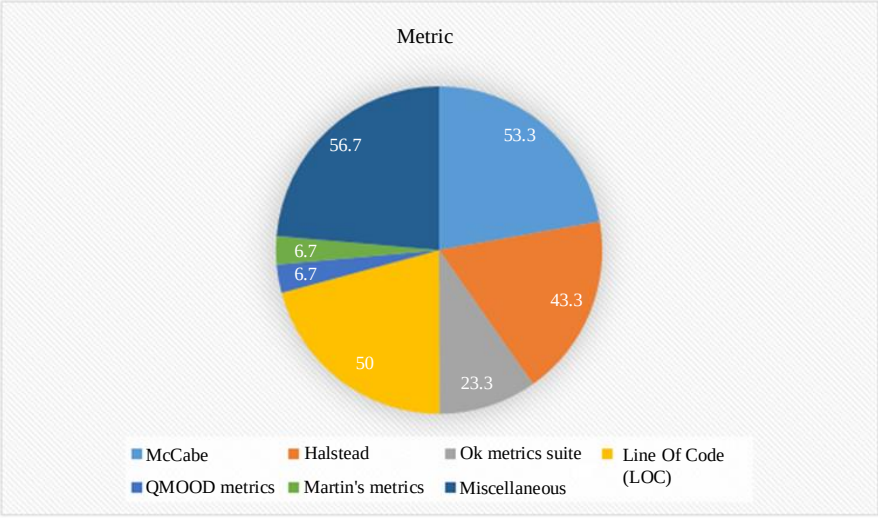

Fig. 3: Distribution of software metrics

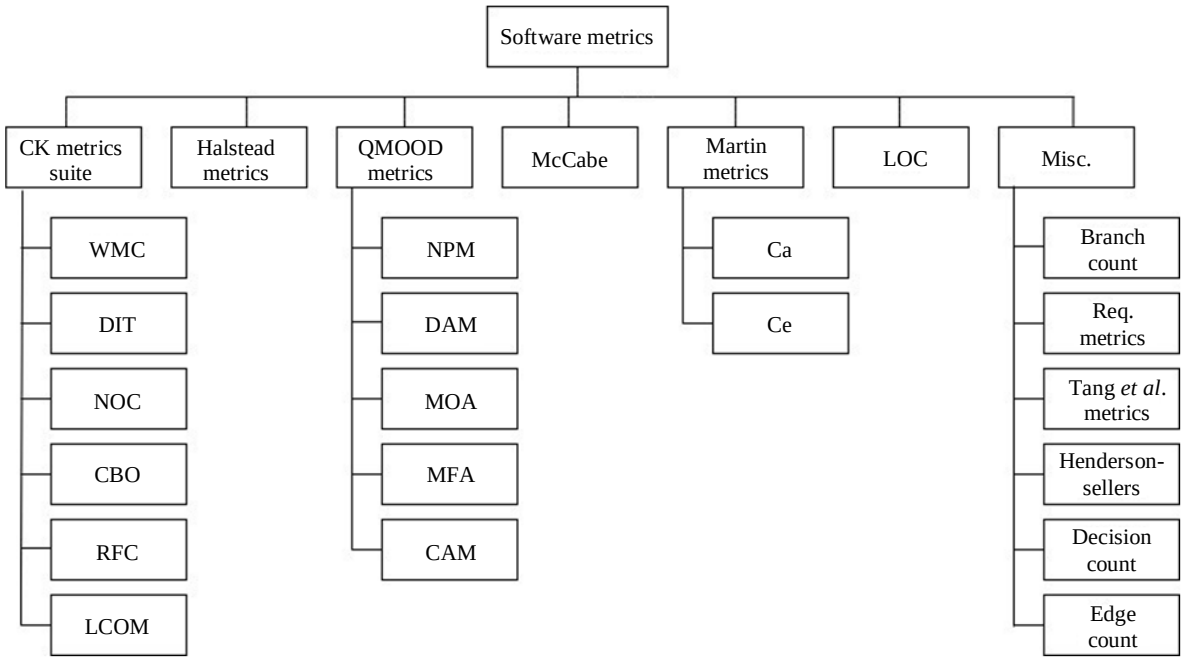

Fig. 4: Type of metrics

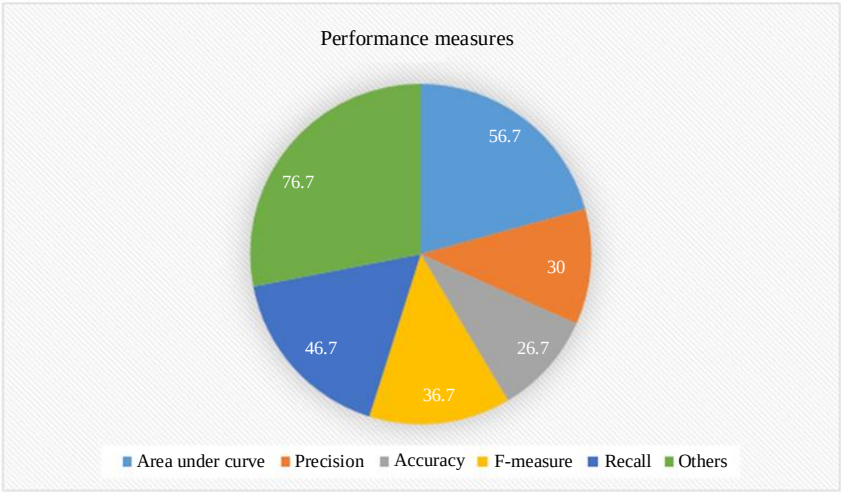

Fig. 5: Distribution of performance measures

Discussion

In this study, we have reviewed 31 journal papers on software bug prediction that were published from 2014 to 2020. The goal of this study is to provide a summary of software bug prediction model and find the scopes on developing the model. We have conducting a search in various digital repositories so that we can extract the studies that have been published between our desired time-range.

Based on our observation, the prediction of bug can be measured in several ways despite its complexity and ambiguity. A bug can be discovered in any stage of the development process and some bugs can remain hidden during the testing and making their appearance during the deployment to the real-world. Binary class classification is widely used in the early prediction and it has been chosen as the basis for prediction model. There is a downside to this method because classifying the bug into defective and non-defective does not give a clear picture of the prediction. There might be some modules that are sensitive to bugs that we had missed. Instead of doing classification, it is better to focus on the bug's level of seriousness and predicting the number of bugs existed. This kind of practice can help us focus more on the severe

modules, prioritizing them for correction and eventually it will lead to the development of robust system.

If we followed the research questions that we have defined previously, the first question is regarding the type of datasets that are frequently used by the researchers for constructing the prediction model. It has been found that NASA MDP and PROMISE repositories are the popular datasets among the researchers because of their availability. But, as mentioned before, public datasets can have inheriting issues, especially when it comes to quality, which can lead to poor prediction results. To solve this problem, we can consider applying some proper data cleaning and data pre-processing techniques (Pan *et al*., 2019).

The second question is regarding the machine learning techniques that are mostly used for building the model and NN has been chosen as the frequently used technique. Traditional SVM has been chosen as the least technique to be used for bug prediction model, unless we did some integrations with other algorithms (Wei *et al*., 2018). FS and EL has been chosen as the best methods for choosing the appropriate classifiers or metrics because of their tree-like structure (Kumar, 2018; Laradji *et al*., 2015; Jakhar and Rajnish, 2018; Ni *et al*., 2017; Kalsoom *et al*., 2018). In the future, we can improve the selection of bug prediction techniques using machine learning because merely using better technique than before does not guarantee the improvement of performance. Still, there are some researchers focused on proposing hybrid frameworks (Erturk and Sezer, 2015; Arar and Ayan, 2015; Rhmann *et al*., 2020; Miholca *et al*., 2018; Abaei *et al*., 2015; Ryu *et al*., 2015), or improving the existing techniques (Pan *et al*., 2019; Zhou *et al*., 2019; Rathore and Kumar, 2017; Wei *et al*., 2018).

The third research question is about the software metrics that are used as independent variables in software bug prediction. It is found that object-oriented metrics, such as CK Metrics Suite and QMOOD metrics, have the likelihood to be chosen in the prediction model. There is also research by (Okutan and Yildiz, 2014) where they proposed a new metric to measure the quality of the code, LOCQ. This metric can be used to predict faultiness and it is as effective as the famous object-oriented metrics.

The last question is about the performance measures, where AUC is the most widely used in the main studies. AUC is popular because the ranking of this approach is they place positive prediction higher than the negative prediction. AUC depends on the area of ROC curve and ROC is independent of the change in proportion of responders. That is why AUC is the most preferable performance measure to evaluate the prediction model.

In this study, we also present some challenges when it comes to bug prediction and provide some description on the works that had been done to solve these challenges. The first challenge that we had discovered is about the implementation of bug prediction on agile development. Nowadays, agile approaches have been widely used for software development because of their iteratively manner and less documentation. When developing a prediction model, we need to depend on the past data that has been gathered from previous software project. This is quite difficult for agile development because their release cycle is very fast and sometimes there is insufficient amount of data for early releases of software project. Erturk and Sezer (2015) presented a new framework for agile development, where they combined fuzzy interference systems and expert knowledge to predict the bugs in the early releases of software development. When the sufficient past data is presented, then they used the conventional bug prediction process.

Another challenge is regarding the approach to build bug prediction model. Based on the review of past studies, we can used various machine learning techniques to perform prediction. However, we can consider to try other approach, such as using ensemble learning algorithms and other classifiers to predict the bugs (Rathore and Kumar, 2017). It is found that this kind of approach has better performance compared to individual approach.

It is also a challenge when we want to make the prediction models more informative. Most of the researchers construct the models by classifying them, for example, whether they are defective or non-defective. There are not many researchers focused on the seriousness of bugs and their numbers. It is better to have the information on the modules that have a large number of bugs instead of having defective or non-defective information. Yang *et al*. (2014) reported their study on this approach, where they focused on predicting the rank of software modules and number of bug prediction.

Threat to Validity

The purpose of this study is to analyze the past studies on software bug prediction using the machine learning techniques. Most of the studies have a huge range of datasets, but we cannot be sure whether these datasets represent the bug prediction scenarios or not. For this study, we did not resort to manual reading of titles of all published papers in journal during the searching stage. In fact, we used the search string that we had constructed earlier to find the relevant studies on bug prediction. We have search as many studies as we can in accordance to inclusion and exclusion criteria. However, there is a likelihood that we had overlooked other proper studies. Also, this review did not include the studies from conference proceedings since we only focused on papers from the primary journals.

Therefore, it had limited other machine learning techniques for our review. The final concern is about the researcher bias, where they have the tendency to confirm that the written information was true.

Conclusion

In this study, we conducted a review so that we can analyze and evaluate the performance of software bug prediction model using machine learning techniques. After a detailed investigation followed by an orderly step, we identified 31 main studies within the period of 2014 to 2020. We summarized the studies based on the datasets, machine learning techniques, software metrics and performance evaluation measurements. The main findings that we have gotten from the main studies are summarized as below:

- NASA MDP and PROMISE repositories were the most frequently used dataset in the past literature
- BN, EL, FS, NN, Clustering and SVM were the machine learning techniques that we have identified and the most widely used technique for bug prediction model were NN and BN
- CK Metrics Suite was found to be the most widely chosen as independent variables in the past literature. CBO, RFC and LOC were found to be the most useful metrics in bug prediction domain
- AUC, precision, recall, F-Measure and accuracy are the most frequently used performance measures in the main studies

The following are the recommendations for future research on software bug prediction using machine learning techniques:

- There are a few studies that adopt the software bug prediction for agile development
- There are a few studies that improve the performance of bug prediction models through integration with other algorithms
- There are few studies that proposed an approach to make the models more informative

Acknowledgement

This study has been funded by the Ministry Of Education (MOE) Malaysia under Fundamental Research Grant (FRGS) project no. 05-01-19-2199FR (5540324). Authors would like to thank editor and all anonymous reviewers for valuable comments.

Author's Contributions

Syahana Nur'Ain Saharudin: Collecting, reviewing, synthesizing relevant literature and drafting manuscript contents.

Koh Tieng Wei: Supervising, revising manuscript contents and editing manuscript.

Kew Si Na: Reviewing and editing manuscript.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Andresen, B. H., Casasanta, J. A., Keeney, S. C., Martin, R. C., & Satoh, Y. (1994). U.S. Patent No. 5,355,037. Washington, DC: U.S. Patent and Trademark Office.

Abaei, G., & Selamat, A. (2014). Increasing the accuracy of software fault prediction using majority ranking fuzzy clustering. International Journal of Software Innovation (IJSI), 2(4), 60-71.

Abaei, G., Selamat, A., & Fujita, H. (2015). An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. Knowledge-Based Systems, 74, 28-39.

Al-Jamimi, H. A. (2016, August). Toward comprehensible software defect prediction models using fuzzy logic. In 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS) (pp. 127-130). IEEE.

Arar, Ö. F., & Ayan, K. (2015). Software defect prediction using cost-sensitive neural network. Applied Soft Computing, 33, 263-277.

Bansiya, J. & Davis, C. (2002). A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 28(1), 4-17.

Boucher, A., & Badri, M. (2018). Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. Information and Software Technology, 96, 38-67.

Chidamber, S. & Kemerer, C. (1994). A metrics suite for object-oriented design. IEEE Transactions of Software Engineering, 20(6), 476-493.

Couto, C., Pires, P., Valente, M. T., Bigonha, R. S. & Anquetil, N. (2014). Predicting software defects with causality tests. Journal of Systems and Software, 93, 24-41.

D'Ambros, M., Lanza, M., & Robbes, R. (2010, May). An extensive comparison of bug prediction approaches. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010) (pp. 31-41). IEEE.

Dong, F., Wang, J., Li, Q., Xu, G., & Zhang, S. (2018). Defect prediction in android binary executables using deep neural network. Wireless Personal Communications, 102(3), 2261-2285.

Erturk, E., & Sezer, E. A. (2015). A comparison of some soft computing methods for software fault prediction. Expert systems with applications, 42(4), 1872-1879.

Geng, W. (2018). Cognitive Deep Neural Networks prediction method for software fault tendency module based on Bound Particle Swarm Optimization. Cognitive Systems Research, 52, 12-20.

Halstead, M. H. (1977). Elements of software science (Vol. 7, p. 127). New York: Elsevier.

Hassan, F., Farhan, S., Fahiem, M. A., & Tauseef, H. (2018). A Review on Machine Learning Techniques for Software Defect Prediction. Technical Journal, 23(02), 63-71.

Hua, W. E. I., Chun, S. H. A. N., Changzhen, H. U., ZHANG, Y., & Xiao, Y. U. (2019). Software Defect Prediction via Deep Belief Network. Chinese Journal of Electronics, 28(5), 925-932.

Jacob, S. G., & Raju, G. (2017). Software defect prediction in large space systems through hybrid feature selection and classification. Int. Arab J. Inf. Technol., 14(2), 208-214.

Jakhar, A. K., & Rajnish, K. (2018). Software fault prediction with data mining techniques by using feature selection based models. International Journal on Electrical Engineering and Informatics, 10(3), 447-465.

Jayanthi, R. F., & Florence, L. (2017). A review on software defect prediction techniques using product metrics. International Journal of Database Theory and Application, 10(1), 163-174.

Ji, H., Huang, S., Wu, Y., Hui, Z., & Zheng, C. (2019). A new weighted naive Bayes method based on information diffusion for software defect prediction. Software Quality Journal, 27(3), 923-968.

Jin, C., & Jin, S. W. (2015). Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization. Applied Soft Computing, 35, 717-725.

Kalsoom, A., Maqsood, M., Ghazanfar, M. A., Aadil, F., & Rho, S. (2018). A dimensionality reduction-based efficient software fault prediction using Fisher linear discriminant analysis (FLDA). The Journal of Supercomputing, 74(9), 4568-4602.

Kamei, Y., & Shihab, E. (2016, March). Defect prediction: Accomplishments and future challenges. In 2016 IEEE 23rd international conference on software analysis, evolution and reengineering (SANER) (Vol. 5, pp. 33-45). IEEE.

Kaur, G. & Sharma, D. (2015). A study on Robert C. Martin's metrics for packet categorization using fuzzy logic. International Journal of Hybrid Information Technology, 8(12), 215-224.

Kazienko, P., Lughofer, E., & Trawiński, B. (2013). Hybrid and ensemble methods in machine learning J. UCS special issue. J Univers Comput Sci, 19(4), 457-461.

Kitchenham, B., & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering.

Kumar, L. S. (2018). Effective fault prediction model developed using least square support vector machine (LSSVM). Journal of Systems and Software, 137, 686-712.

Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. Information and Software Technology, 58, 388-402.

Li, L., Vaishnavi, V. K., & Vandenberg, A. (2010). SOM Clustering to Promote Interoperability of Directory Metadata: A Grid-Enabled Genetic Algorithm Approach. J. UCS, 16(5), 800-820.

Ma, Y., Zhu, S., Qin, K., & Luo, G. (2014). Combining the requirement information for software defect estimation in design time. Information Processing Letters, 114(9), 469-474.

Majd, A., Vahidi-Asl, M., Khalilian, A., Poorsarvi-Tehrani, P., & Haghighi, H. (2020). SLDeep: Statement-level software defect prediction using deep-learning model on static code features. Expert Systems with Applications, 147, 113156.

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.

Michura, J., Capretz, M. A., & Wang, S. (2013). Extension of Object-Oriented Metrics Suite for Software Maintenance. ISRN Software Engineering, 2013.

Miholca, D. L., Czibula, G., & Czibula, I. G. (2018). A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. Information Sciences, 441, 152-170.

Mori, T., & Uchihira, N. (2019). Balancing the trade-off between accuracy and interpretability in software defect prediction. Empirical Software Engineering, 24(2), 779-825.

Ni, C., Liu, W. S., Chen, X., Gu, Q., Chen, D. X., & Huang, Q. G. (2017). A cluster based feature selection method for cross-project software defect prediction. Journal of Computer Science and Technology, 32(6), 1090-1107.

Okutan, A., & Yıldız, O. T. (2014). Software defect prediction using Bayesian networks. Empirical Software Engineering, 19(1), 154-181.

Pan, C., Lu, M., Xu, B., & Gao, H. (2019). An Improved CNN Model for Within-Project Software Defect Prediction. Applied Sciences, 9(10), 2138.

Park, M., & Hong, E. (2014). Software fault prediction model using clustering algorithms determining the number of clusters automatically. International Journal of Software Engineering and Its Applications, 8(7), 199-204.

Prasad, V. S. & Sasikala, K. (2019). Software defect prediction techniques - A review. Journal of Information and Computational Science, 9(9), 619-638.

Rana, Z. A., Mian, M. A., & Shamail, S. (2015). Improving Recall of software defect prediction models using association mining. Knowledge-Based Systems, 90, 1-13.

Rathore, S. S., & Kumar, S. (2017). Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems. Knowledge-Based Systems, 119, 232-256.

Rhmann, W., Pandey, B., Ansari, G., & Pandey, D. K. (2020). Software fault prediction based on change metrics using hybrid algorithms: An empirical study. Journal of King Saud University-Computer and Information Sciences, 32(4), 419-424.

Ryu, D., Jang, J. I., & Baik, J. (2015). A hybrid instance selection using nearest-neighbor for cross-project defect prediction. Journal of Computer Science and Technology, 30(5), 969-980.

Sayyad, S. J., & Menzies, T. J. (2005). *PROMISE* Software Engineering Repository. Univeristy of Ottawa. School of Information Technology and Engineering. http://promise.site.uottawa.ca/SERepository

Tang, M., Kao, M. H., & Chen, M. H. (1999). An empirical study on object oriented Metrics. In Proceedings of the International Symposium on Software Metrics (Cat. No. PR00403) (pp. 242-249). IEEE.

Wahono, R. S. (2015). A systematic literature review of software defect prediction. Journal of Software Engineering, 1(1), 1-16.

Wei, H., Shan, C., Hu, C., Sun, H., & Lei, M. (2018). Software defect distribution prediction model based on NPE-SVM. China Communications, 15(5), 173-182.

Wu, R., Zhang, H., Kim, S., & Cheung, S. C. (2011, September). Relink: Recovering links between bugs and changes. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (pp. 15-25).

Wu, Y., Huang, S., Ji, H., Zheng, C., & Bai, C. (2018). A novel Bayes defect predictor based on information diffusion function. Knowledge-Based Systems, 144, 1-8.

Xia, Y., Yan, G., Jiang, X., & Yang, Y. (2014, May). A new metrics selection method for software defect prediction. In 2014 IEEE International Conference on Progress in Informatics and Computing (pp. 433-436). IEEE.

Yang, X., Tang, K., & Yao, X. (2014). A learning-to-rank approach to software defect prediction. IEEE Transactions on Reliability, 64(1), 234-246.

Zhao, L., Shang, Z., Zhao, L., Qin, A., & Tang, Y. Y. (2018). Siamese dense neural network for software defect prediction with small data. IEEE Access, 7, 7663-7677.

Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep forest. Information and Software Technology, 114, 204-216.