International Journal of Applied Science and Engineering

Survey on software defect prediction techniques

Mahesh Kumar Thota[1*], **Francis H Shajin**[2], **P. Rajesh**[3]

[1] *Research Scholar, Department of Computer Science Engineering, KL University, Guntur, India*
[2] *Department of Electronics and Communication Engineering, Anna University, Chennai, India*
[3] *Department of Electrical and Electronics Engineering, Anna University, Chennai, India*

ABSTRACT

Recent advancements in technology have emerged the requirements of hardware and software applications. Along with this technical growth, software industries also have faced drastic growth in the demand of software for several applications. For any software industry, developing good quality software and maintaining its eminence for user end is considered as most important task for software industrial growth. In order to achieve this, software engineering plays an important role for software industries. Software applications are developed with the help of computer programming where codes are written for desired task. Generally, these codes contain some faulty instances which may lead to the buggy software development cause due to software defects. In the field of software engineering, software defect prediction is considered as most important task which can be used for maintaining the quality of software. Defect prediction results provide the list of defect-prone source code artefacts so that quality assurance team scan effectively allocate limited resources for validating software products by putting more effort on the defect-prone source code. As the size of software projects becomes larger, defect prediction techniques will play an important role to support developers as well as to speed up time to market with more reliable software products. One of the most exhaustive and pricey part of embedded software development is consider as the process of finding and fixing the defects. Due to complex infrastructure, magnitude, cost and time limitations, monitoring and fulfilling the quality is a big challenge, especially in automotive embedded systems. However, meeting the superior product quality and reliability is mandatory. Hence, higher importance is given to V&V (Verification & Validation). Software testing is an integral part of software V&V, which is focused on promising accurate functionality and long-term reliability of software systems. Simultaneously, software testing requires much effort, cost, infrastructure and expertise as the development. The costs and efforts elevate in safety critical software systems. Therefore, it is essential to have a good testing strategy for any industry with high software development costs. In this work, we are planning to develop an efficient approach for software defect prediction by using soft computing based machine learning techniques which helps to predict optimize the features and efficiently learn the features.

Keywords: Defect prediction, Soft computing, Verification, Validation.

Received: September 6, 2019
Revised: December 12, 2019
Accepted: July 3, 2020

Corresponding Author:
Mahesh Kumar Thota
tmakeshkumar0201@gmail.com

Publisher:
Chaoyang University of Technology
ISSN: 1727-2394 (Print)
ISSN: 1727-7841 (Online)

1. INTRODUCTION

One of the most exhaustive and pricey part of embedded software development is consider as the process of finding and fixing the defects (Ebert and Jones, 2009). Due to complex infrastructure, magnitude, cost and time limitations, monitoring and fulfilling the quality is a big challenge, especially in automotive embedded systems. However, meeting the superior product quality and reliability is mandatory. Hence, higher importance is given to V&V (Verification & Validation). Software testing is an integral part of software V&V, which is focused on promising accurate functionality and long-term reliability of software systems. Simultaneously, software testing requires much effort, cost, infrastructure and expertise as the development (Lemos et al., 2015). The costs and efforts elevate in safety critical software systems. Therefore, it is essential to have a good testing strategy for any industry with high software development costs.

Nowadays, the growth of the software industry is huge and more sophisticated. Therefore, anticipating the reliability of the software is an important task in software development process (Roy et al., 2014). A software bug is a defective behavior of the software system which arises due to definite and possible violation of security policies during the application runtime. It is mainly caused by improper development or erroneous specification of the software system (Ghaffarian and Shahriari, 2017). According to ref. of Abaei and Selamat (2014), analysis and prediction of defects are essential to serve three important requirements. First, it helps in assessing the progress of the project and assists in scheduling testing process by the project manager. Second, it helps in investigating the quality of the product. Lastly, it improves the reliability and functionality of the product. The fault-prone modules can be identified by distinct metrics, which have been reported by the previous fault prediction. Some of the crucial information, such as number of faults, location, severity, and distribution of defects are extracted to enhance the efficiency of testing process. It further helps in improving the software quality of the upcoming software release. The two main advantages of software fault prediction are, enhancement of the overall testing process by emphasizing on fault-prone modules, identification of the refactoring candidates which are rendered as most likely to undergo fault (Catal, 2014).

The models used for Software engineering cost and schedules, their estimation, etc., are implemented for several reasons which are,

- Budgeting: It is the first and foremost implication, but it is not the only purpose. The most important factor is "overall accuracy of the system".
- Project planning and control: It is yet another critical feature to offer cost and scheduling estimations with respect to modules, stage and process.
- Tradeoff and risk analysis: It involves the supplementary capability to focus on the project scheduling and costs involved in the project decisions (staffing, scoping, tools, reuse, etc.).
- Software improvement investment analysis: In involves the additional cost and efforts required for other strategies, such as recycling, tools, inventory, process maturity, etc.

In software programming, defect analysis and prediction can decisively determine potential bugs in the software and helps in discovering the modules which are more vulnerable to such problems. It can assist the engineers to allocate constrained resources to those modules of the software framework, which are more liable to be affected by bugs. Constructing a defect prediction models for a software framework is helpful for numerous of developmental or maintenance activities, for example, software quality assessment and monitoring quality assurance (QA).

The significance of defect prediction has propelled various scholars and engineers to characterize distinctive kinds of models or indicators that portray different parts of programming quality. Most research generally evaluates this issue as supervised learning problems and the results of those defect prediction models is dependent on the previous defect information. To be precise, a predictor model is built based on the training data obtained from the previous defects seen in past software releases. This defect predictor's can be used to defect bugs in upcoming software projects or to cross-validate on the same data set (He et al., 2012), also known as Within-Project Defect Prediction (WPDP). Zimmermann et al. (2009) expressed that the performance of defect prediction models can be better, if there are adequate quantity of information accessible to train the models. Nevertheless, this type of information is not available for freshly started projects. Therefore, high precision in defect prediction process in such projects become extremely difficult, sometimes implausible. However, there are little open-source information on defect datasets, such PROMISE (Wang and Li, 2010), Apache (Ghotra et al., 2015) and Eclipse (Ryu et al., 2016), which can be used to train the defect predictors.

To overcome such challenges, few engineers and scholars have made an attempt to apply the predictors from one project, on to a different one (Li et al., 2017; Lu et al., 2015). This process of using information between different projects to construct defect models is generally termed as Cross-Project Defect Prediction (CPDP). It involves the process of implementing a predictor model in a project, which was built for some other project.

The choice of training samples relies upon the distributional attributes of datasets. Few experimental examinations assessed the practical advantages of cross-project defect predictors with various programming metric, such as, process measurements, static code metrics, system metrics, etc., (Li et al., 2017; Lu et al., 2015), and how to uses such metrics in a complementary way (Zimmermann et al., 2009). Even though, several attempts are established for the implementation of CPDP, it is still not well developed, and suffers from poor performance in practice (Rahman et al., 2012). Besides, no definitive information is available on how the defect predictors amongst WPDP and CPDP are sanely selected, when there are no proper historical data on the project. In general, several type of software metrics, for example, history of code change, static code metrics, network metrics, process metrics (He et al., 2013), etc., are used for building defect predictors for various types of fault detection (Radjenović et al., 2013).

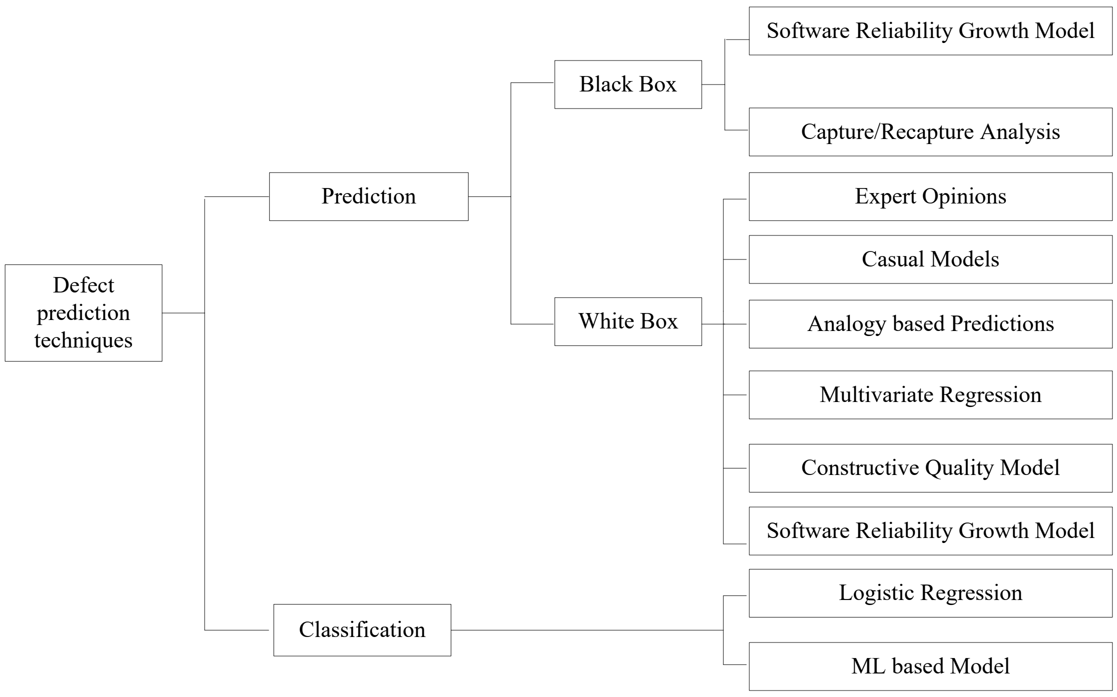
Defect prediction techniques
Prediction
Black Box
Software Reliability Growth Model
Capture/Recapture Analysis
White Box
Expert Opinions
Casual Models
Analogy based Predictions
Multivariate Regression
Constructive Quality Model
Software Reliability Growth Model
Classification
Logistic Regression
ML based Model

Fig. 1. Software defect prediction techniques

All current defect prediction models are constructed on the sophisticated amalgamation of programming metrics, with which a defect predictor can generally attain good level of precision. Nevertheless, few feature selection algorithms such as principal component analysis (PCA), can greatly lower the amount of data dimensions (He et al., 2015; Wu et al., 2017; Wang et al., 2013), they still result in a time-consuming process. Can a compromise solution be found, which can attain a tradeoff between cost and accuracy? As it were, would we be able to build a generic universal predictor with hand few of metrics, such as Lines of Code (LOC), which can attain accurate results in comparison to other complex prediction models? Apart from choosing a proper software metric, there are numerous classifiers available at disposal, such as Naive Bayes (Zhang et al., 2016), J48 (Song et al., 2006), Support Vector Machine (SVM) (Xia et al., 2016), Logistic Regression (Li et al., 2014), and Random Tree (Staron and Meding, 2008), etc. Apart from these, there are a few improved classifiers (Rana et al., 2013) and hybrid classifiers (Gondra, 2008) which are known to effectively improve classification results.

1.1 Software Defect Prediction Techniques

To foresee the quantity of flaws anticipated that would be found in a product module/venture or to group which modules are likely to be imperfect, Programming Defect Prediction (PDP) systems are utilized. Various distinctive strategies have been utilized for characterization /anticipating absconds; they can be extensively gathered into methods that are utilized to foresee if or not a given programming ancient rarity is probably going to contain a deformity (Classification) and procedures utilized for foreseeing anticipated that number of imperfections would be found in a given programming antique (Prediction) and Fig. 1 outlines normally utilized programming imperfection forecast methods clustered by the reason –fault check expectation or defect inclined arrangement.

In an investigation by Staron and Meding (Rajbahadur et al., 2017), professional views were utilized and their execution contrasted with other information based models. Author’s former works establishes the long term analytical power of SRGMs (Software Reliability Growth Models) within the automotive realm indicating their utility in analyzing or predicting fault and consistency. To categorize the software modules which are likely to be defective or to analyze the compactness of software defect, various software modules related to code features like complexity, size etc., has been utilized effectively.

Techniques that utilize code and modify measurements as sources of info and utilize machine learning strategies for categorizing and predicting have additionally been examined by Iker Gondra (Kim et al., 2011) and Xie et al. (2011). Pertinence of different strategies for programming imperfection forecasts over the life cycle periods of programming advancement and the attributes of every strategy are shown in Table 1.

1.2 Techniques for Defect Classification

Software defect classification is another important technique of defect prediction. These models strive to identify fault-prone software modules using variety of software project and product attributes. In general, defect classification models are implemented at lower granularity levels, more predominantly at file and class level. Hence,

Table 1. Different strategies for programming imperfection forecasts

Method	Input data required	Advantages and limitations
Causal Models	Inputs about estimated size, complexity, qualitative inputs on planned testing and quality requirements	Causal models biggest advantage is that they can be applied very early in the development process. Possible to analyse what-if scenarios to estimate output quality or level of testing needed to meet desired quality goals.
Expert Opinions	Domain experience (software development, testing and quality assessment).	This is the quickest and easiest way to get the predictions (if experts are available). Uncertainty of predictions is high and forecasts may be subjected to individual biases
Analogy Based Predictions	Project characteristics and observations from large number of historical projects.	Quick and easy to use, the current project is compared to previous project with most similar characteristics. Evolution of software process, development tool chain may lead to inapplicability or large prediction errors.
Constructive Quality Model	Software size estimates, product, personal and project attributes; defect removal level.	Can be used to predict cost, schedule or the residual defect density of the software under development. Needs large effort to calibrate the model.
Correlation Analysis	Number of defects found in given iteration; size and test effort estimates can also be used in extended models.	This method needs little data input which is available after each iteration. The method provides easy to use rules that can be quickly applied. The model can also be used to identify modules that show higher/lower levels of defect density and thus allow early interventions
Regression Models	Software code (or model) metrics as measure of different characteristics of software code/model; Another input can be the change metrics.	Uses actual code/models characteristic metrics which means estimates are made based on data from actual software under development. Can only be applied when code/models are already implemented and access to the source code/model is available. The regression model relationship between input characteristics and output can be difficult to interpret –do not map causal relationship
Machine Learning based models	Software code (or model) metrics as measure of different characteristics of software code/model; Another input can be the change metrics.	Similar to regression models, these can be used for either classification (defective/not defective) or to estimate defect count/densities. Over time as more data is made available, the models improvise on their predictive accuracy by adjusting their value of parameters (learning by experience). While some models as Decision Trees are easy to understand others may act like a black box (for example Artificial Neural Networks) where their internal working is not explicit
Software Reliability Growth Models	Defect inflow data of software under development (life cycle model) or software under testing.	Can use defect inflow data to make defect predictions or forecast the reliability of software based system. Reliability growth models are also useful to assess the maturity/release readiness of software close to its release. These models need substantial data points to make precise and stable predictions

the software products which are flagged as defect-prone can be prioritized according to their severity for more rigorous verification and validation activities.

1.2.1 Logistic Regression

A software module can be categorized as defect-prone or not, on the basis of logistic regression. Much like the multivariate regression, the classification of software modules is done by using several variety of process and product metrics are employed as predictor variables. Zimmermann, et al. (Köksal et al., 2011) worked on the principle of logistic regression to categorize file/packages in Eclipse project as defect prone.

1.2.2 Machine Learning Models

Some popular machine learning techniques uses statistical algorithms and data mining techniques, which is helpful for predicting and classifying defects. Such techniques are identical to regression approaches that use same type of independent variables. On the upside, the machine learning algorithms are dynamic in nature, and they progressively enhance the overall prediction and classification technique.

1.3 General Process of Software Defect Prediction

To build an efficient prediction model, we should have proper data on defects and metrics, which can be accumulated from software development efforts to use as the learning set. Thus, there is tradeoff between its prediction performance on additional data sets and how well this model fits in its learning set. Therefore, the performance of the model is assessed by the comparison of the predicted defects of the modules in a test, against the actual defects witnessed (Hewett, 2011).

1.4 General Defect Prediction Process

Labeling: An appropriate defect data must be collected for the purpose of training a prediction model. This step generally involves the extraction of instances and labeling the data items (True or False).

Extracting features and creating training sets: The extraction of features for prediction labels of instances is performed in this stage. Few common features for defect prediction are keywords, complexity metrics, deviations, and structural dependencies. By consolidating the labels and highlights of instances, a training set is generated which is used by the machine learning algorithms to develop a forecast model.

Building prediction models: The prediction models can be built with the help of a training set, implemented on the general machine learning algorithms, such as Bayesian Network or Support Vector Machines (SVM). Based on the learned data, the model can classify and label the test instances as TRUE or FALSE.

Assessment: The assessment of a prediction display is done on the basis of testing dataset collection and training set. The labels of the training dataset are used to build the prediction model, which is later evaluated by comparing the prediction and real labels. The training sets and testing sets are separated using 10-fold cross-validation technique.

2. RELATED WORKS

As indicated by Catal and Diri (2009), defect prediction models in software programming have become one of the significant research areas since 1990. In just two decades, the total amount of research papers in this area had increased two fold. A wide range of procedures and methodologies were utilized for defect prediction models, for example, decision trees (Selby and Porter, 1988) neural network system (Hu et al., 2007), Naïve Bayes (Menzies et al., 2004), case-based reasoning (Khoshgoftaar et al., 1997), fluffy logic (Yadav and Yadav, 2015) and the artificial immune recognition framework technique in Catal and Diri (2007).

Menzies et al. (2004) carried out an experiment derived from the open-source NASA datasets with the help of few data mining techniques. The results were later evaluated with the help of balance parameter, probability of false alarm and probability of detection. Prior to the implementation of the algorithm, the authors have used the log-transformation with Info-Gain filters. They further assured that performance of Naïve Bayes in terms of fault prediction was better than J48 algorithm. The authors have further contended that since a few models with low accuracy performed well, implementing such models as a dependable parameter for performance assessment was not suggested. Okutan and Yıldız (2014), estimated the probabilistic influential relationships between software metrics and probability of defect, with the help of Bayesian networks. Apart from the metric used in Promise data repository, two other metric were defined in this proposed research work, which were LOCQ for the source code quality and NOD for the number of developers. These metrics can be derived by examining the source code repositories of the targeted Promise data archives. Once the model was complete, the marginal probability of defect of the system can be understood, along with the set of influential metrics, and the correlation between defects and metrics.

Likewise, dictionary based learning algorithms were more popular in the field of software defect prediction. Jing et al. (2014) implemented software defect prediction models on the principle of machine learning techniques. The similarity between different software modules can be exploited to represent a small proportion of few modules with the help of some other modules. Furthermore, the coefficients of the pre-defined dictionary contains the historical software data, which are large inadequate. With the help of the qualities of the metrics extracted from the open-source programming modules, the researchers learn numerous dictionaries, including but not limited to, defective-free module, defective module, total dictionary, sub-dictionaries, etc. The researchers have also considered the problem of misclassification cost, as it usually imposes greater risk than other defective-free ones. Along these lines, we present a cost-sensitive discriminative dictionary learning (CDDL) technique for software defect classification and prediction.

The representative studies in software defect prediction are shown in Table 2. Over the past decade, several attempts were made to build efficient prediction models. Process metrics and source code (Rahman et al., 2012) are some of the widely studied metrics. Process metrics were derived from the software archives, for example, bug tracking systems, version control systems, etc., which keep track of all development histories. Process metrics evaluates numerous characteristics of software programming process such as, ownership of source code files, changes of source code, developer interactions, etc. The source code metric determines the intricate should the source code be. The fundamental basis about the source code metric was that

Table 2. Representative studies software defect prediction

Type	Categories	Representatives
Within/Cross	Metrics	Sorce code (Jing et al., 2014), Process (Churn (Kim et al., 2007), (Ghotra et al., 2015), Change (Kamei et al., 2010), Entropy (Zhang et al., 2016), Popularity (Roy et al., 2014), Authorship (He et al., 2015), Ownership (Catal, 2014), MIM (Hewett, 2011), Network measure (Okutan and Yıldız, 2014), (Peters et al., 2013), (Khoshgoftaar et al., 2010), Antipattern (Shin et al., 2010)
	Algorithm/Model	Classification, Regression, Active/Semi-supervised learning (Hu et al., 2007), (Bosu et al., 2014), BugCache (Xie et al., 2011)
	Finer prediction granularity	Change classification (Gondra, 2008), Method level-prediction (Song et al., 2006)
	Preprocessing	Feature selection/Extraction (Walden and Doyle, 2012), (Kim et al., 2007), Normalisation (Jing et al., 2014), (Catal and Diri, 2009), Noice handling (Rajbahadur et al., 2017), (Morrison et al., 2015)
Cross	Transfer learning	Metric compensation (Brereton et al., 2007), NN filter (Moshtari and Sami, 2016), TNB (Khoshgoftaar et al., 1997), TCA + (Catal and Diri, 2009)
	Feasibility	Decision Tree (Catal and Diri, 2009), (Xia et al., 2016)

more complex source code was more likely to be infected by bugs. Several studies have emphasized the significance of process metrics for defect prediction (Zhang et al., 2016; Fenton and Neil, 1999; Kamei et al., 2010).

The prediction models built by the machine learning algorithms have the ability to detect the probability of bugs or defects in the source code. Few research works have mplemented latest machine learning algorithms such as active/semi-supervised learning algorithms, which are known to enhance prediction performance (Li et al., 2012; Zhang et al., 2017). BugCache algorithm was suggested by Kim et al., which uses the locality information of previous defects and maintains a list of source code files or modules, which were more likely to be faulty (Kim et al., 2007).

BugCache algorithm uses machine learning techniques for building defect prediction models which uses non-statistical model. This entirely different from the other defect prediction models. It also fine tunes the prediction granularity. It attempts to find defects at various levels, such as, class, file, package, component, system. Few recent experiments have demonstrated that defects can be found at module level or method level, or change level (Koru and Liu, 2005). The developers can be benefited by the finer granularity model, as they can minimize the scope of source code, which must be inspected. Thus, preprocessing techniques are also an important part of defect prediction studies. Prior to the implementation of defect prediction model, few techniques are applied, such as normalization (Menzies et al., 2004), feature selection (Catal and Diri, 2009), noise handling (Khoshgoftaar and Rebours, 2007), etc.

Several authors have also emphasized on cross-project fault prediction. Majority of these experiments were portrayed and directed inside the prediction setting, which suggests that the forecast models were constructed and executed within the same project. In spite of this, it was challenging for few new projects, which did not contain any vital information about the historical data about the developmental process. Few of the popular representative approaches for cross defect prediction were Nearest Neighbour (NN) Filter (Zhang et al., 2017), metric compensation (Watanabe et al., 2008), Transfer Naive Bayes (TNB) (Ma et al., 2012), and TCA + (Nam et al., 2013).

2.1 Within-Project Defect Prediction

Catal and Diri (2009) has conducted an investigation on over 90 software defect prediction research works, which were published between the vicinity of 1990 and 2009. He reviewed these papers on the basis of the performance evaluation metrics, learning algorithms, experimental outcomes, datasets, etc. As indicated by this review, the author expressed that a large portion of these research works were based on utilizing the method-oriented metrics and prediction-models. Therefore, they were largely dependent on the machine learning procedures, and Naive Bayes techniques, which were regarded as a popular machine learning techniques for supervised prediction tasks.

Hall et al. (2011) carried out an investigation on the metrics, such as model contexts, modeling algorithms, independent variables, etc., and characterized their effects on the performance of defect prediction models, based on the 208 research works. Their outcomes demonstrated that rudimentary modeling systems, for example, Logistic Regression and Naive Bayes, portrayed better performance. Additionally, the performance was further enhanced by the combination of independent variables. The results are greatly improved by the application of feature selection on these combinations. The authors contend that there was considerable amount of defect prediction models, in which certainty was conceivable. However, more examinations which implemented a reliable technique have witnessed a comprehensive context, performance, and methodology. Most of these research works were reviewed with respect to two systemic literature surveys that were led with regards to WPDP. Nevertheless, they overlooked the fact that few of these research works were particularly new, and they normally have restricted or deficient information to train a proper prediction model for defect forecasting. Thus, a few researchers have started to work their way towards CPDP.

2.2 Cross-Project Defect Prediction

The primary research on CPDP was performed by Briand et al. (2000), who connected models based on an open-source venture (i.e., Xpose) to another (i.e., Jwriter). Despite the fact that the anticipated imperfection recognition probabilities were not reasonable, the defect-prone class positioning was precise. They additionally approved that such a model performed superior to anything the irregular model and beat it regarding class size. Zimmermann et al. (2009) led a large-scale investigation on information versus province versus process, and found that CPDP was not generally fruitful (21/622 expectations). They additionally detected that CPDP was not proportioned amongst internet explorer and Firefox. CPDP utilizing fixed code attributes in the view of 10 projects and even gathered from PROMISE archive was investigated by Turhan et al. They suggested a closest-neighbor sifting strategy to channel through the insignificances in cross-venture information. In addition, they examined the situation where models were developed from a combination of inside and cross-venture information, and checked for any enhancements to WPDP in the wake of including the information from different undertakings or projects. They presumed that when there was restricted venture chronicled information (e.g., 10% of recorded information), combined project estimates were reasonable, as they executed and additionally within-project forecast models.

Rahman et al. (2012) led a cost-delicate examination of the viability of CPDP on thirty eight arrivals of nine extensive Apache Software Foundation (ASF) ventures, by contrasting it with WPDP. Their detections uncovered that the cost-touchy cross-project estimation execution was not more regrettable than the inside-venture forecast execution, and was significantly superior to arbitrary expectation execution. To assist cross-company learning in contrast with the state of the art Peters et al. (2013) acquainted a new filter called Burak filter. The outcomes uncovered that their method could assemble sixty-four percent more valuable indicators than both cross-company and within-company approaches in view of Burak channels, and exhibited that cross-organization fault estimate could be connected ahead of schedule in a venture's lifecycle. He et al. (2015) directed three tests on similar informational indexes utilized as a part of this examination to approve training information from different projects can give worthy outcomes. They additionally suggested a way to deal with naturally choosing appropriate training information for ventures or projects without neighborhood information.

Herbold (2013) suggested a few methodologies in view of forty-four informational collections from fourteen open-source ventures regarding training data selection for CPDP. A few portions of their informational collections are utilized here in our paper. The outcomes exhibited that their choice procedures enhanced the realized progress rate essentially, though the nature of the outcomes was as yet unfit to contend with WPDP. The survey uncovers that earlier examinations have mostly explored the possibility of CPDP and the decision of preparing information from their tasks. Yet moderately little consideration was given to experimentally investigating the execution of a forecaster in light of a disentangled metric set from the viewpoints of exertion and-cost, precision and simplification. Besides, next to no was thought about whether the forecasters made with basic or least programming metric subsets acquired by wiping out some unimportant and repetitive highlights can accomplish adequate outcomes.

2.3 Software Metrics

A wide range of software models are regarded as features, which can be utilized for defect prediction, to enhance overall quality of the software programming. Simultaneously, various correlations are made among numerous software metrics to review which metric offers good level of performance. Shin and Williams (2013) examined whether source code and programming histories were discriminative and detect weak codes among sophisticated, code agitate, and parameters followed by the designer. It was discovered that 24 of the 28 metrics were discriminative for both Linux and Mozilla Firefox kernel. By utilizing all the three kinds of metrics, these models predicted more than 80% of the potential weaknesses in the files within under 25% false positives for the two activities. Marco et al. (2010) led three trials on five frameworks with process metrics, source code metrics, previous defect data, entropy of changes, and so forth. They found that the best performance can be obtained by the modest process metrics, which were marginally better than entropy and churn of source code metrics.

Zimmermann et al. (2009) utilized social network parameters extracted from dependency relation between software programming on Windows Server 2003 to predict which elements were more vulnerable to defects. With respect to predicting defects, the experimental results have shown that the performance of network metrics was better than source code metrics. Tosun et al. (2011) conducted experiments on five public datasets to replicate and verify their outcomes from two distinct levels of granularity. The outcomes have shown that network metrics were more appropriate for detecting defects for large-scale and complicated models, even though their performance in smaller models were not much impressive. Premraj and Herzig (2011) reproduced the Zimmermann and Nagappan's work to conduct further evaluation of the generality of these results. However, the results were found to be consistent

Table 3. Comparative analysis of classification based techniques for software defect prediction

Authors and publication year	Objective	Methodology	Key findings	Conclusion
Roy et al. (2014)	Software reliability prediction	Feed forward and recurrent neural network	Genetic algorithm is used for training the neural network	Better prediction of software defects
Ghaffarian and Shahriari (2017)	Survey of SDP techniques	Data mining and machine learning techniques for SDP	Data mining and ML techniques are good for early defect prediction and vulnerability	An extensive review which shows advantages and disadvantages of DM and ML techniques
Catal and Diri (2009)	Fault prediction	Artificial immune system and random forest approach for classification	Significant Feature selection can improve the performance	Comparative performance where it shows that Random forest achieves better accuracy
Wang and Li (2010)	Software defect prediction for improving the quality	Naïve Bayes classification model	Multi- variants Gauss Naive Bayes (MvGNB) used for reducing the complexity	MvGNB achieves better performance when compared with J48 classifier
Yadav and Yadav (2015)	Different artifact and defect prediction in software engineering	Fuzzy logic technique	Phase-wise computation along with fuzzy logic classification	It can be used for classifying the defect types
Okutan and Yıldız (2014)	Software defect prediction and level of defect	Feature extraction and Bayesian classification technique for SDP	Significant feature extraction and relationship between software metrics and defects.	It can be used for both supervised and unsupervised learning

with the original work. In any case, regarding the array of datasets, code metrics were more suitable for experimental investigations on open-source programming ventures.

Radjenovic' et al. (2013) grouped 106 papers on defect prediction with respect to context properties and metrics. hey discovered that the amount of process metrics, source code metrics, and object-oriented metrics, were 24%, 27%, and 49%, respectively. Chidamber and Kemerer's (CK) uite metrics were most frequently used. In comparison to complexity metrics and traditional size, the object-oriented and process metrics were more efficient. Thus, in comparison to static code metrics, the process metrics were more proficient in predicting post-release defects. On the basis of these research works, a comparative review was presented in Table 3, which gives details on techniques used, results, and strengths of individual works. The classification based techniques are presented first.

Zimmermann et al. (2011) examined the likelihood of detecting the presence of vulnerabilities and defects in binary modules of a popular software product (Microsoft Windows Vista). The researchers have used classical metrics which were implemented in past research works for defect prediction. Initially, correlations was computed which exists between the metrics and amount of defects per binary module. The Spearman's rank correlation was used for this purpose. The results revealed that there was a noteworthy connection among classical metrics and the number of vulnerabilities. Another research was led to determine the prediction capabilities of these metrics. For this purpose, a five groups of classical metrics (i.e., dependency, coverage, coverage, organizational, churn) were inspected using binary Logistic Regression.

Williams and Meneely (Meneely et al., 2008), examined the connection between software vulnerabilities and developer-activity metrics. The developer-activity metrics consists of number of commits made to a file, number of developers who have modified the codes in the source program, amount of geodesic paths which contains a file in the contribution network. The research was carried out on three open-source software projects. The assembled informational in a given research work contains a label which suggest if the source code file was patched or not. The version control logs would disclose the developer-activity metrics. With the help of statistical correlation analysis, the researchers have confirmed the existence of statistical correlation for every metric with the given quantity of vulnerabilities. However, the correlation was not very strong. The training and validation sets were generated by using Bayesian network with tenfold cross-validation, as the predictive model.

Walden and Doyle (2012) have led a study to inspect the correlation among software metric and vulnerabilities in 14 different popular open source web applications during 2006 and 2008, for example, Mediawiki, WordPress. The researchers implemented static analytical tools, such as, PHP CodeSniffer, Fortify Source Code Analyzer. With the help of this tool, they have estimated various metrics in source code repositories of these web applications, such as, source-code size, nesting complexity, static analysis vulnerability density (SAVD), Security Resources Indicator (SRI), etc. Williams and Shin (2008), conducted an experiment to check if the conventional defect prediction models, which are built on the principle of code-churn metrics and complexity, were any help in predicting the fault vulnerability. The experiment was carried out on Mozilla Firefox with a fault history metric, 5 code churn metrics, and 18 complexity metrics. With various classification techniques for the fault prediction, the researchers have concluded that the results were almost identical.

Shin et al. (2010) carried out an intense research to check if the vulnerability prediction was affected by the code-churn and developer-activity (CCD) and complexity. In regard to this, the researchers have conducted experiments on two open-source projects. The analysis was performed on over 28 CCD software metrics, which also consists of 3 code-churn metrics, 14 complexity metrics, and 11 developer-activity metrics.

The authors have used the Welch's t-test to assess the discriminative power of the metrics. For both the projects, the test hypotheses were supported by at least 24 of 28 metrics. For the purpose of evaluation of predictive power of the metrics, the authors have tested numerous classification techniques. They have discussed the result form just one technique, as the performance was similar. To verify the predictive capacity of the model, the authors have validation on next-release, where numerous releases were in progress. Apart from that, they have also performed cross-validation, where only a single release would be in progress.

Moshtari and Sami (2016) pointed out three important constraints of vulnerability prediction models of previous research works. Therefore, they presented a new technique to predict the potential location of the defects in the software. It is accomplished by complexity metrics by resolving the limitations of previous studies. For the purpose of detecting software vulnerabilities, the researchers have proposed a semi-automatic analysis framework. The output from this framework is used as vulnerability information, which was known to provide more comprehensive details about the vulnerabilities in software, as explained by the authors. This research had examined both cross-project and within-project fault prediction, with the help of accumulated information from over five different open-source projects.

Bosu et al. (2014) conducted a similar experiment, in which they investigated more than 260,000 code review requests from over 10 different open source projects. Subsequently, they were able to identify more than 400 vulnerable code changes, with the help of three-stage semi-automated process. The main objective was to detect the characteristics of vulnerable code changes, and developers who might cause such vulnerabilities. Some key discoveries of this study include:

1. The probability of fault elevates if the changes made in the codes are high.
2. Changes are made by less experienced developers increases the chances of defects.
3. The chances of defects are higher in new files, in comparison to modified files.

To recognize constraints which are responsible for vulnerabilities, Perl et al. (Brereton et al., 2007) examined the impacts of utilizing the meta-information enclosed in code sources close by code –metrics. The initiators declare the way that programming develops incrementally, and most open-source projects utilize adaptation control techniques, subsequently, constraints define normal units to check for vulnerabilities. With this intention, the creators accumulate a dataset containing 170,860 confers from sixty six C/C ++ GitHub projects, including 640 vulnerability contributing commits (VCCs) plotted to significant CVE IDs. The creators select an arrangement of code-beat and designer-interest metrics, and in addition GitHub meta-information from various extensions (creator, document, commit and project) and concentrate these highlights for the accumulated dataset. To distinguish VCCs from unbiased commits, the creators assess their recommended technique, named VCC Finder, which utilizes a Support Vector Machine (SVM) classifier based on this dataset.

To analyze the execution of foreseeing vulnerable programming mechanisms, in light of programming metrics against text-extracting procedures, Walden et al. (2014) played out an investigation. With this thought, the creators initially developed a manually-built dataset of vulnerabilities assembled from three vast and well known open-source PHP web applications (Moodle, PhpMyAdmin, Drupal), comprising of two hundred and twenty three vulnerabilities. As an endowment to the investigation group, this dataset is presented. An arrangement of twelve code unpredictability metrics was chosen for this examination in order to estimate vulnerability in the light of programming metrics. For content mining, every PHP source file was tokenized. Unwanted tokens are either transferred (punctuations, comments, string, numeric literals, etc.) or terminated. A count was kept on the frequencies of final tokens. The numerical feature-vectors are built from the textual tokens of each PHP source file, using the popular "bag-of-words" technique.

Morrison et al. (2015) explains that defect prediction models which are implemented by the Microsoft teams, are different from the vulnerability prediction models (VPMs). To clarify this disparity, for two fresh releases of the Microsoft Windows OS the researchers have made an attempt to reproduce a VPM technique, presented by Zimmermann et al. (2011).

Table 4. Uniqueness and few advantages of each work

Authors	Metrics	Granularity	Within/Cross-project	Vulnerability
Zimmerman et al. (2011)	Code churn, coverage, dependency, complexity , organizational	Binary modules	Within project	Public advisories
Meneely et al. (2008)	Developer activity	Source file	Within project	Public advisories
	Code complexity and security resources	Source file	Within project	Tool-based defection
Walden and Doyle (2012)	Complexity, fault-history, code churn	Source file	Within project	Public advisories
Yonghee et al. (Shin and Williams, 2011)	Code complexity, dependency network complexity and execution complexity	Source file	Within project	Public advisories
Shin and Williams (2008)	Complexity, code-churn, developer activity	Source file	Within project	Public advisories
Shin et al. (2010)	Unit complexity, coupling	Source file	both	Self-developed detection framework
Moshtari and Sami (2016)	Code churn, developer activity	Code commits	Within project	Public advisories
Bosu et al. (2014)	Developer activity	Code commits	Within project	Public advisories
Perl et al. (Brereton et al., 2007)	Code churn, developer activity, GitHub Metadata		Within project	Public advisories
Walden et al. (2014)	Code complexity	Source file	Both	Public advisories
Morrison et al. (2015)	Code churn, complexity, coverage, dependency, organizational	Binary module	Within project	Public advisories
Younis et al. (2016)	Code complexity, information flow, functions, Invocations.	Functions	Within project	Public advisories

Younis et al. (2016) made an attempt to detect the attributes of code, which contains defects that was more ikely to be susceptible. Since they commenced the study, they were able to recognize over 183 defects from the Linux kernel and Apache HTTPD web server projects. It must be noted that these projects contained 82 exploitable vulnerabilities. The researchers have chosen over 8 software metrics from 4 groups, in order to represent these ulnerabilities. They had used Welch's t-test to investigate the discriminative power of each metric. Furthermore, the researcher examined if there is a combination of these metrics which can be exploited as predictors for few defects, wherein, 3 diverse feature selection techniques and 4 classification techniques were verified.

In the previous section, a review was presented on various recent researches in the area of defect prediction models based on software metrics. Table 4 presents the summary of all the research works reviewed in this section and also tabulates the uniqueness and few advantages of each work.

3. APPLICATIONS OF DEFECT PREDICTION

One of significant objectives of defect prediction models is efficient utilization of available resources for assessing and testing programming modules. Nevertheless, there is only a hand few of contextual analyses which use defect prediction models (Lewis, 1999). Thus, Rahman et al. (2012) led most of their investigation on cost-viability. Lewis (1999) pioneered a recent contextual investigation directed by Google, which compares the BugCache and Rahman's technique, with respect to the amount of closed bugs (Peters et al., 2013). The outcomes have indicated that the designers favored Rahman's technique.

In any case, the defect prediction models do not give any advantages to the developers. In a recent survey, Rahman et al. (2012) demonstrated that defect prediction models could be useful to organize potential warnings discovered by the bug finders, for example, FindBug. It also helps in implementation of results from the defect prediction to organize or choose appropriate test cases. In regression testing, performing all the test cases are not financially feasible, and consumes large amount of time as well. Therefore, it is best to choose proper test cases, which investigates the potential faults in the system (Lessmann et al., 2008). The results of the defect prediction models can provide an idea on the potential bugs and their severity, which can be exploited to select and prioritize the test cases.

On the basis of previously reviewed works, it is obviously that the area of defect prediction has more to offer, and hence, it is in its early stages. It can be concluded with few of the future improvements and limitation, which can be extracted from past research works.

- A factual limitation in the area of defect prediction models is that the bugs and weaknesses are few in number in the given datasets. In data mining and machine learning algorithms, this limitation is termed as imbalance class data. This imbalance can create a greater drop in overall performance of the algorithms. However, there are few methods to overcome this issue (Khoshgoftaar et al., 2010). Furthermore, few research works were focused on achieving the same, with random under-sampling the majority class. This is regarded as a critical problem which should not be overlooked.
- Moshtari et al. (2016) has implemented a semi-automatic system for fault identification, rather than a data from public repositories and fault databases (example: NVD). Thus, in comparison to other techniques, this system resulted in better recall and precision values. This could pave the way for more intense research in the future.
- There are only few research works on the cross-project studies in the area of defect prediction. Therefore, it can be regarded as a field of future enhancement. The cross-project fault prediction models are not well researched in the context of defect prediction models. There are additional concerns in the Cross-project prediction models which are induced due to distribution of data in the training set, which can differ largely among themselves. Such variations can greatly degrade the performance of machine learning algorithms and statistical-analysis techniques. This limitation can be overcome by a descendant of the machine-learning algorithm, known as “inductive transfer” (or “transfer learning”) techniques. About the implementation of these techniques are well documents has in software defect prediction studies (Catal and Diri, 2009).
- Majority of the fault prediction techniques offered poor performance. This is mainly due to the use of traditional software metrics, which are not considered as the appropriate indicators of software defects. Morrison et al. (2015) has discussed about this situation. Later on, characterizing security-oriented metrics, for example, the Security Resources Indicator (SRI), which was proposed by Doyle and Walden (2012) this is another territory for future investigations.
- The use of deep-learning techniques for defect prediction is not well explored. It has emerged as a new area of machine-learning algorithm which is made impressive accomplishments in few application specific domains. Furthermore, it is increasing gain more popularity from scholars and professionals (Jiang et al., 2008). Yang et al. (2015) proposed a technique based on deep-learning methods for just-in-time software defect prediction. This laid the foundation for another area of research for future improvements.

4. CONCLUSION

This survey paper helps the researchers to study about software defects and software defect prediction techniques. To implement the data pre-processing technique; data cleaning, data normalization and data discretization will be performed in data mining. For feature extraction and selection to implement of new approach, to implement of evolutionary computation and optimization technique for best feature selection and to implement machine learning classification techniques for bug classification. An improved approach consists of data pre-processing low computation cost, complex model, software defect prediction comparative analysis and improved classification performance of the system.

REFERENCES

Abaei, G., Selamat, A. 2014. A survey on software fault detection based on different prediction approaches. Vietnam Journal of Computer Science, 1, 79–95.

Bosu, A., Carver, J.C., Hafiz, M., Hilley, P., Janni, D. 2014, November. Identifying the characteristics of vulnerable code changes: An empirical study. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 257–268. ACM.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M. 2007. Lessons from applying the systematic literature review process within the software engineering domain. Journal of systems and software, 80, 571–583.

Briand, L.C., Wüst, J., Daly, J.W., Porter, D.V. 2000. Exploring the relationships between design measures and software quality in object-oriented systems. Journal of systems and software, 51, 245–273.

Catal, C., Diri, B. 2007, February. Software defect prediction using artificial immune recognition system. In Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering 285–290. ACTA Press.

Catal, C., Diri, B. 2009. A systematic review of software fault prediction studies. Expert systems with applications, 36, 7346–7354.

Catal, C., Diri, B. 2009. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences, 179, 1040–1058.

Catal, C., Diri, B. 2009. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences, 179, 1040–1058.

Catal, C. 2014. A comparison of semi-supervised classification approaches for software defect prediction. Journal of Intelligent Systems, 23, 75–82.

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), 31–41. IEEE.

Ebert, C., Jones, C. 2009. Embedded software: Facts, figures, and future. Computer, 42, 42–52.

Fenton, N.E., Neil, M. 1999. A critique of software defect prediction models. IEEE Transactions on software engineering, 25, 675–689.

Ghaffarian, S.M., Shahriari, H.R. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM Computing Surveys (CSUR), 50, 56.

Ghotra, B., McIntosh, S., Hassan, A.E. 2015, May. Revisiting the impact of classification techniques on the performance of defect prediction models. In Proceedings of the 37th International Conference on Software Engineering-Volume 1, 789–800. IEEE Press.

Gondra, I. 2008. Applying machine learning to software fault-proneness prediction. Journal of Systems and Software, 8, 186–195.

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S. 2011. A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38, 1276–1304.

He, P., Li, B., Liu, X., Chen, J., Ma, Y. 2015. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 59, 170–190.

He, P., Li, B., Liu, X., Chen, J., Ma, Y. 2015. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 59, 170–190.

He, P., Li, B., Ma, Y., He, L. 2013. Using software dependency to bug prediction. Mathematical Problems in Engineering.

He, Z., Shu, F., Yang, Y., Li, M., Wang, Q. 2012. An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering, 19, 167–199.

Herbold, S. 2013, October. Training data selection for cross-project defect prediction. In Proceedings of the 9th International Conference on Predictive Models in Software Engineering 6. ACM.

Hewett, R. 2011. Mining software defect data to support software testing management. Applied Intelligence, 34, 245–257.

Hu, Q.P., Xie, M., Ng, S.H., Levitin, G. 2007. Robust recurrent neural network modeling for software fault detection and correction prediction. Reliability Engineering & System Safety, 92, 332–340.

Jiang, Y., Cukic, B., Ma, Y. 2008. Techniques for evaluating fault prediction models. Empirical Software Engineering, 13, 561–595.

Jing, X.Y., Ying, S., Zhang, Z.W., Wu, S.S., Liu, J. 2014, May. Dictionary learning based software defect prediction. In Proceedings of the 36th International Conference on Software Engineering, 414–423. ACM.

Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.I., Adams, B., Hassan, A.E. 2010, September. Revisiting common bug prediction findings using effort-aware models. In 2010 IEEE International Conference on Software Maintenance, 1–10. IEEE.

Khoshgoftaar, T.M., Rebours, P. 2007. Improving software quality prediction by noise filtering techniques. Journal of Computer Science and Technology, 22, 387–396.

Khoshgoftaar, T.M., Ganesan, K., Allen, E.B., Ross, F.D., Munikoti, R., Goel, N., Nandi, A. 1997, November. Predicting fault-prone modules with case-based reasoning. In Proceedings the eighth international symposium on software reliability engineering, 27–35. IEEE.

Khoshgoftaar, T.M., Gao, K., Seliya, N. 2010, October. Attribute selection and imbalanced data: Problems in software defect prediction. In 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, 1, 137–144. IEEE.

Kim, S., Zhang, H., Wu, R., Gong, L. 2011, May. Dealing with noise in defect prediction. In 2011 33rd International Conference on Software Engineering (ICSE). 481–490. IEEE.

Kim, S., Zimmermann, T., Whitehead Jr, E.J., Zeller, A. 2007, May. Predicting faults from cached history. In Proceedings of the 29th international conference on Software Engineering, 489–498. IEEE Computer Society.

Köksal, G., Batmaz, İ., Testik, M.C. 2011. A review of data mining applications for quality improvement in manufacturing industry. Expert systems with Applications, 38, 13448–13467.

Koru, A.G., Liu, H. 2005. Building effective defect-prediction models in practice. IEEE software, 22, 23–29.

Lemos, O.A.L., Ferrari, F.C., Silveira, F.F., Garcia, A. 2015. Experience report: Can software testing education lead to more reliable code?. In 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), 359–369.

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, 485–496.

Lewis, N.D. 1999. Assessing the evidence from the use of SPC in monitoring, predicting & improving software quality. Computers & Industrial Engineering, 37, 157–160.

Li, K., Chen, C., Liu, W., Fang, X., Lu, Q. 2014. Software defect prediction using fuzzy integral fusion based on GA-FM. Wuhan University Journal of Natural Sciences, 19, 405–408.

Li, M., Zhang, H., Wu, R., Zhou, Z.H. 2012. Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 19, 201–230.

Li, Z., Jing, X.Y., Zhu, X., Zhang, H., Xu, B., Ying, S. 2017. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. IEEE Transactions on Software Engineering.

Lu, J., Behbood, V., Hao, P., Zuo, H., Xue, S., Zhang, G. 2015. Transfer learning using computational intelligence: a survey. Knowledge-Based Systems, 80, 14–23.

Ma, Y., Luo, G., Zeng, X., Chen, A. 2012. Transfer learning for cross-company software defect prediction. Information and Software Technology, 54, 248–256.

Meneely, A., Williams, L., Snipes, W., Osborne, J. 2008, November. Predicting failures with developer networks and social network analysis. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering 13–23. ACM.

Menzies, T., DiStefano, J., Orrego, A., Chapman, R. 2004. Assessing predictors of software defects. In Proc. Workshop Predictive Software Models.

Mısırlı, A.T., Çağlayan, B., Miranskyy, A.V., Bener, A., Ruffolo, N. 2011, May. Different strokes for different folks: A case study on software metrics for different defect categories. In Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics, 45–51. ACM.

Morrison, P., Herzig, K., Murphy, B., Williams, L. 2015, April. Challenges with applying vulnerability prediction models. In Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, 4. ACM.

Moshtari, S., Sami, A. 2016, April. Evaluating and comparing complexity, coupling and a new proposed set of coupling metrics in cross-project vulnerability prediction. In Proceedings of the 31st Annual ACM Symposium on Applied Computing, 1415–1421. ACM.

Nam, J., Pan, S.J., Kim, S. 2013, May. Transfer defect learning. In 2013 35th International Conference on Software Engineering (ICSE), 382–391. IEEE.

Okutan, A., Yıldız, O.T. 2014. Software defect prediction using Bayesian networks. Empirical Software Engineering, 19, 154–181.

Peters, F., Menzies, T., Marcus, A. 2013, May. Better cross company defect prediction. In Proceedings of the 10th Working Conference on Mining Software Repositories, 409–418. IEEE Press.

Premraj, R., Herzig, K. 2011, September. Network versus code metrics to predict defects: A replication study. In 2011 International Symposium on Empirical Software Engineering and Measurement, 215–224. IEEE.

Radjenović, D., Heričko, M., Torkar, R., Živkovič, A. 2013. Software fault prediction metrics: A systematic literature review. Information and software technology, 55, 1397–1418.

Rahman, F., Posnett, D., Devanbu, P. 2012, November. Recalling the imprecision of cross-project defect prediction. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, 61. ACM.

Rajbahadur, G.K., Wang, S., Kamei, Y., Hassan, A.E. 2017, May. The impact of using regression models to build defect classifiers. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR) 135–145. IEEE.

Rana, R., Staron, M., Mellegård, N., Berger, C., Hansson, J., Nilsson, M., Törner, F. 2013, June. Evaluation of standard reliability growth models in the context of automotive software systems. In International Conference on Product Focused Software Process Improvement, 324–329. Springer, Berlin, Heidelberg.

Roy, P., Mahapatra, G.S., Rani, P., Pandey, S.K., Dey, K.N. 2014. Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction. Applied Soft Computing, 22, 629–637.

Ryu, D., Choi, O., Baik, J. 2016. Value-cognitive boosting with a support vector machine for cross-project defect prediction. Empirical Software Engineering, 21, 43–71.

Selby, R.W., Porter, A.A. 1988. Learning from examples: generation and evaluation of decision trees for software resource analysis. IEEE Transactions on Software Engineering, 14, 1743–1757.

Shin, Y., Williams, L. 2008, October. An empirical model to predict security vulnerabilities using code complexity metrics. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, 315–317. ACM.

Shin, Y., Williams, L. 2011, May. An initial study on the use of execution complexity metrics as indicators of software vulnerabilities. In Proceedings of the 7th International Workshop on Software Engineering for Secure Systems, 1–7. ACM.

Shin, Y., Williams, L. 2013. Can traditional fault prediction models be used for vulnerability prediction?. Empirical Software Engineering, 18, 25–59.

Shin, Y., Meneely, A., Williams, L., Osborne, J.A. 2010. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Transactions on Software Engineering, 37, 772–787.

Song, Q., Shepperd, M., Cartwright, M., Mair, C. 2006. Software defect association mining and defect correction effort prediction. IEEE Transactions on Software Engineering, 32, 69–82.

Staron, M., Meding, W. 2008. Predicting weekly defect inflow in large software projects based on project planning and test status. Information and Software Technology, 50, 782–796.

Walden, J., Doyle, M. 2012. SAVI: Static-analysis vulnerability indicator. IEEE Security & Privacy, 10, 32–39.

Walden, J., Stuckman, J., Scandariato, R. 2014, November. Predicting vulnerable components: Software metrics vs text mining. In 2014 IEEE 25th international symposium on software reliability engineering 23–33. IEEE.

Wang, H., Khoshgoftaar, T.M., Liang, Q. 2013. A study of software metric selection techniques: Stability analysis and defect prediction model performance. International journal on artificial intelligence tools, 22, 1360010.

Wang, T., Li, W.H. 2010, December. Naive bayes software defect prediction model. In 2010 International Conference on Computational Intelligence and Software Engineering, 1–4. Ieee.

Watanabe, S., Kaiya, H., Kaijiri, K. 2008, May. Adapting a fault prediction model to allow inter languagereuse. In Proceedings of the 4th international workshop on Predictor models in software engineering, 19–24. ACM.

Wu, F., Jing, X.Y., Dong, X., Cao, J., Xu, M., Zhang, H., Ying, S., Xu, B. 2017, May. Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 195–197. IEEE.

Xia, X., Lo, D., Pan, S.J., Nagappan, N., Wang, X. 2016. Hydra: Massively compositional model for cross-project defect prediction. IEEE Transactions on software Engineering, 42, 977–998.

Xie, X., Ho, J.W., Murphy, C., Kaiser, G., Xu, B., Chen, T.Y., 2011. Testing and validating machine learning classifiers by metamorphic testing. Journal of Systems and Software, 84, 544–558.

Yadav, H.B., Yadav, D.K. 2015. A fuzzy logic based approach for phase-wise software defects prediction using software metrics. Information and Software Technology, 63, 44–57.

Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J. 2015, August. Deep learning for just-in-time defect prediction. In 2015 IEEE International Conference on Software Quality, Reliability and Security, 17–26. IEEE.

Younis, A., Malaiya, Y., Anderson, C., Ray, I. 2016, March. To fear or not to fear that is the question: Code characteristics of a vulnerable functionwith an existing exploit. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, 97–104. ACM.

Zhang, F., Zheng, Q., Zou, Y., Hassan, A.E. 2016, May. Cross-project defect prediction using a connectivity-based unsupervised classifier. In Proceedings of the 38th International Conference on Software Engineering, 309–320. ACM.

Zhang, Z.W., Jing, X.Y., Wang, T.J. 2017. Label propagation based semi-supervised learning for software defect prediction. Automated Software Engineering, 24, 47–69.

Zimmerman, T., Nagappan, N., Herzig, K., Premraj, R., Williams, L. 2011, March. An empirical study on the relation between dependency neighborhoods and failures. In 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation 347–356. IEEE.

Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B. 2009, August. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 91–100. ACM.