Informatics Bulletin, Faculty of Computers and Artificial Intelligence, Helwan University
Published Online *Vol 6 Issue 1, January 2024*
(https://fcihib.journals.ekb.eg)

Users Review's on Software Defect Prediction Utilizing Machine Learning methods

O. E. Emam [1], M. A. Elsabagh[2,*], M. G. Gafar[2], T.Medhat[3]

[1]*Department of Information Systems, Faculty of Computers and Artificial Intelligence, Helwan University, Egypt.*
[2]*Department of Machine Learning and Information Retrieval, Faculty of Artificial Intelligence, Kafrelsheikh University, Egypt.*
[3]*Department of Electrical Engineering, Faculty of Engineering, Kafrelsheikh University, Kafrelsheikh, Egypt.*
osama_emam@fci.helwan.edu.eg, Mahmoud_Mohsen@fci.kfs.edu.eg, mona_gafar@fci.kfs.edu.eg, tmedhatm@eng.kfs.edu.eg

Abstract— **Software Defect Prediction (SDP) is a crucial and helpful method for upgrading software reliability and quality. It enables more effective project management by predicting potential release delays early on and facilitating cost-effective corrective actions to enhance software quality. This is achieved by forecasting which modules in a large software product are likely to have the highest number of defects in the next version. However, creating reliable defect forecasting models remains a challenging issue, leading to the presentation of numerous methods in literature. Typically, machine learning (ML) classifiers are employed, using manually designed attributes (like complexity measures) to identify problematic code. However, these attributes often fail to capture the full structural and semantic details of the software. Incorporating this information is crucial for the development of accurate defect prediction models. This study covers various defect prediction strategies and explores recent research on ML methodologies for SDP, aiming to bridge the gap between software semantics and defect forecasting attributes. By doing so, it seeks to produce more precise and accurate forecasting.**

Index Terms— **Predicting Software Defect, Machine Learning (ML), Software Testing, Software Metrics, Algorithms of Deep learning (DL)**

I. Introduction

TODAY, developers' focus has gradually evolved over the past few years towards software-based systems driven by the belief that software quality and reliability are the most crucial components of user performance. The increased computerization in recent years has resulted in the development of a wide range of software. However, steps must be taken to ensure that this software is error-free. Complex source code raises the likelihood of software exhibiting flaws, ultimately leading to software failure [1]. Finding software bugs will be the focus of future study in the area of software engineering since it makes it easier for software engineers and developers to find bugs quickly and accurately [2].

Early and accurate SDP is necessary for the scientifically based management of the software testing stage. Business typically uses SDP model development; these models aid in further defect prediction, testing, effort calculation, software dependability, software quality, assessment of hazards, etc., during the developing phase.

All software development organizations find that the Software Quality Assurance (SQA) is the most challenging and costly operation [3], as the quality assurance teams typically devote a considerable portion of their efforts and time to thoroughly reviewing the existing software instead of developing a new functionality. Tasks for SQA teams, such as software examination, help software developers locating possible issues and prioritize their efforts for testing. They have significant influences on the building of trustworthy, software with higher quality. In several research literatures focusing on SDP approaches, significant emphasis has been placed on testing and repairing software. The core concept of SDP revolves around constructing classifiers capable of predicting modules or sections of code at the highest risk of failure [4] [5]. The majority of these methods concentrate on creating characteristics (such as complexity measures) that are related to possibly defective code. SDP can be used to detect software modules, count the amount of faults in a particular module, and judges the effectiveness of the model designed to predict defects. The majority of SDP models were created with the goal of classifying defects [6]. The amount of time and resources needed to fix the defect in the software components can be better understood by forecasting the number of defects [7].

SDP is carried out by taking into account several software metrics. Software metrics utilized in SDP [8] include requirements and change metrics, code churn, software size, lines of code, software difficulty, McCabe's [9], Halstead's metrics [10], etc. Software metrics encompass diverse types, including object-oriented, procedural, hybrid, and miscellaneous metrics. The SDP model is developed based on these metrics.

The SDP work has existed since the early 1900s [11]. ML algorithms have been used in the SDP field over the years. ML can identify components that pose a significant danger. potential failure (those are probably faulty) and then assign the highest priority to more risky test scenarios in order to create an effective testing plan with the least amount of efforts, time, and costs [12]. Establishing a mechanism for predicting software defects so that testing and maintenance costs can be reduced is the most important task in the testing stage of the Software Development Life Cycle (SDLC). It identifies the components that need extensive testing and are error-prone. Regression-based methods have been utilized for several decades to predict defective code; in recent years, both supervised and unsupervised ML techniques have been employed to predict defective code utilizing an existing set of training data that includes historical data from software repositories [1].

Recent ML models are combined to create ensemble models [13] because conventional ML algorithms are not as effective . For cross-project [14] and within-project [15] prediction, SDP approaches can be utilized. When it comes to SDP, within-project methodology involves collecting data from a single project for both training and testing, compared to cross-project methodology involves gathering training and testing data from different projects. Classification and regression have been carried out using several types of ML techniques, such as Artificial Neural Networks (ANN), Support Vector Machines (SVM), and ensemble techniques.

Numerous variables have an impact on SDP models. These include noise, feature selection, software metrics, cost parameters, and over-fitting of the model. Also, data imbalance is one of the most critical problems with failure datasets [16], [17]. An unbalanced dataset is one that contains an unbalanced distribution of labels. A majority of one label of data exists, while a minority of another label of data does. Applying a technique to this dataset leads to skewed results and an unreliable assessment of the system. Numerous sampling strategies are employed to address this problem [12, 10]. To develop better performance SDP models, these sampling methods are integrated with ML algorithms.[16], [18]. To tackle class imbalance problems, over-sampling and under-sampling techniques are frequently employed [17], [19]. One of two methods may be utilized to upgrade the reliability of the SDP method. Graphical and numerical metrics can be employed as performance assessment measures for SDP. Some graphical measurements include the cost curve, receiver operating characteristic curve, and area under the curve. The effectiveness of the model is predicted utilizing numerical performance indicators like accuracy, precision, F1-score, G-measure, and others. The reliability of various models can be upgraded depending on the dataset, assessment measures, and usability.

Only the most recent SDP methods addressing performance issues, such as accuracy, are considered in this research. Researchers can analyze various algorithms and select the most suitable one based on factors such as software metrics and the dataset.

II. Background

This study briefly explains the SDP method and related researches. Researchers are working tirelessly these days to enhance the functionality of SDP models. The accuracy of the ML method depends on the quality of the historical data, which is divided into defective and non-defective components. The following subsection presents software defect prediction, SDP process, and summary of public datasets, respectively.

1- Software Defect Prediction (SDP)

SDP is one of the most crucial tasks of the SDLC testing stage. It determined which components need more testing because they are prone to bugs. It is possible to use the test resources effectively without going overboard. Even while SDP is a great tool for testing, it's not always simple to identify the problematic components. Various issues hamper the seamless operation and application of defect prediction models. Predicting software flaws at an early stage enhances software quality while lowering costs and facilitating efficient software management, hence this was an area of interest for research [20].

Before the testing stage of the Software Development Life Cycle (SDLC), SDP serves as a method for identifying defective code segments [21]. It enables the satisfactory delivery of software development and the effective utilization of resources [22]. To build SDP models, numerous ML methods are employed, each impacting the performance of the SDP process. These elements are crucial for creating a detailed and organized model. Some of these key elements include:

- Dataset problems.
- Performance Evaluation.
- ML algorithms.
- Metrics of software
- Feature selection.
- Datasets.

Designers encounter various challenges while creating a model to identify defective software components. These elements can significantly affect the model's effectiveness and accuracy. Developers must consider all these factors when creating a model. Here are a few of them:

- Data uncertainty.
- Imbalanced data.
- Model over-fitting.
- Noisy.
- Estimating cost parameters.

The following are the goals of SDP: (i) early -phases fault prediction; (ii) identifying critical components that need additional focus and resources; (iii) software quality enhancements; (iv) Cost savings; and (v) software efficiency [20].

2- SDP process

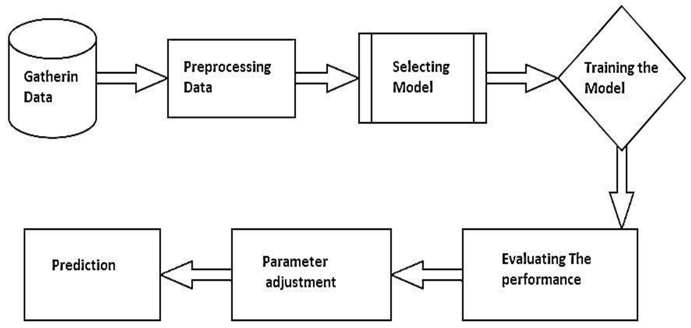

Fig. 1 General SDP Stages

Figure 1 depicts the standard process followed by SDP model [8], [21]:

Stage 1, Gathering Data: For defect prediction, a variety of datasets are accessible. In this stage, either a dataset is developed or one from the publicly accessible datasets is used. The publicly accessible datasets are briefly described in the following Subsection.

Stage 2, Preprocessing Data: The datasets used to predict defects may have issues with class imbalance, outliers, uncertainty, noise, and irrelevant data. This stage involves applying different sampling and noise reduction algorithms to the dataset. Applying ML methods to these datasets improves the model's performance.

Stage 3, selecting the Model: ML model is chosen depending on the dataset and the goal. One can choose among conventional approaches, edit an existing model of ML, or build a new one.

Stage 4, during this stage, a dataset is used to develop the model. The model self-learned utilizing the trained data and the designated methodology. Testing and training data are separated from the datasets.

Stage 5, evaluating the model performance: It is evaluated utilizing the data used for testing. Many of software metrics like accuracy are utilized for evaluating the model's reliability and effectiveness in predicting defective software.

Stage 6, Parameter adjustment: The model's parameters are changed in this stage to improve model’s accuracy and performance. Once the parameters have been adjusted, the model has been assessed again.

Stage 7, Prediction: the model of SDP is evaluated and applied on real world problems and the outcomes are documented.

3- Summary of Public datasets

The main problem with SDP in the past was the lack of publicly accessible datasets. It is difficult to evaluate the outcomes of various techniques in the absence of data. So, researchers established a few open repositories. Researchers discovered the PROMISE [23] store for the field of Software Engineering (SE) in 2005, motivated by the development of the ML repository established by California Irvine University [23]. In addition to this, numerous researchers have provided their personal publicly accessible datasets that are utilized in SDP [24]. The most well-known datasets for SDP that are publicly accessible are listed in the following table.

Table 1 Different datasets of SDP field.

Data	Explanation	projects	Features
ReLink [25]	Gathered by Zhang et.al [25] recovering link among defects and changes	3	26
NASA [26]	This dataset was made available by the NASA	13	20:40
PROMISE[23]	Open source SE	38	20
SOFTLAB[27]	Crated by a lab of software research in Turkey	5	29
ECLIPSE2 [28]	Collected by Gong et.al [28]	2	17
ECLIPSE1 [29]	Collected by Premraj [29] from eclipse projects.	3	31
AEEEM [27]	Gathered by Lanza [27]	5	61

III. LITERATURE REVIEW

In the area of SE, SDP has become a widely-studied topic. SDP not only finds defective software components but also promotes to boost software by fixing the defect in the preliminary phases of software designing. Recent research by academics in the field of SDP is described in this section. Studies pertaining to ML algorithms utilized for SDP are included in Section 1, whereas studies pertaining to DL are covered in Section 2.

For the concern of imbalanced data, Turabieh et al. [30] introduced a model of SDP dependent on optimized Moth Flame paired with an Adapted Synthetic Sampling technique. The suggested technique enhances the functionality of many classifiers when used with the dataset from PROMISE repository. Linear Discriminant algorithm has the greatest value of AUC, and K-Nearest Neighbor (K-NN) has the fastest completion time.

M.A.Elsabagh et al. [15] propose that the spotted hyena innovative meta-heuristic technique was used to forecast defects. By identifying the most suitable rules throughout individuals, an objective function helped the spotted hyena accomplish the task of a classifier based on support and confidence using NASA datasets.

A classification model centered on the spotted hyena algorithm was utilized by Marwa et al. [14] to forecast faults in cross-projects. Confidence and Support were combined to find the best predictor of forecasting defects. This classifier of forecasting is also applied to new software that try to forecast flaws or other studies with limited data.

Three distinct generated oversampling techniques, Wasserstein GAN plus Penalty of Gradient, Vanilla GAN, and Conditional GAN have been presented by Chouhan et al.

[31]. The Eclipse, JIRA, and PROMISE datasets were used in the test. The outcomes of the baseline methods were significantly enhanced when these sampling techniques are combined with these methods within evaluations on defective set of data.

Santosh et al. [32] used Bayesian regression techniques for SDP in within-projects and cross-projects. Along with Linear and Non-linear Bayesian Regression, SVM, Random Forest (RF), and Linear Regression (LR), Synthetic Minority Oversampling and data sampling techniques are used. On a sample of 46 separate projects, Non-linear Bayesian regression techniques outperformed LR techniques.

By autonomously learning attributes that describe a module and utilizing them for SDP, Pham et al. [33] proposed a methodology for forecasting software flaws. Their prediction approach is based on an abstract syntax tree source code modeling and a tree-structured long short-term memory network for effective DL.

Kanwal et al. [34] presented a comparison of several ensemble methods used for SDP. Classifiers include Naive Bayes (NB), Logistic Regression (LR), Decision Tree (DT), K-NN, and Multinomial NB. On accessing a data from the PROMISE, the study is run. The metric employed to assess effectiveness is called the F-measure. The findings show that model averaging beats stacking and voting ensemble techniques.

A model for forecasting software flaws utilizing ML techniques was reported by Alnabhan et al. [35]. They employed three supervised ML techniques to accurately anticipate software bugs based on data: NB, DT and ANN.

A method that relies on feature reduction was presented by Jayanthi et.al [36]. They used an ANN as a predictor to forecast the flaws and principal component analysis (PCA) augmented by the calculation of maximum likelihood to eliminate mistakes in PCA. P. D et.al [37] employed KEEL to analyze and validate the most popular ML technique, including DT, NB, ANN, Particle Swarm Optimization (PSO), and linear classifier. Chen et al. [38] utilized a methodology relied on the SVM as the main to handle the challenge of forecasting software errors. By using the grid search strategy, the model's parameters were optimized.

Chen et al. [39] observed that while traditional methodologies reduced unnecessary features using feature reduction techniques, some features remained significant and had an impact on performance when forecasting software problems. They applied the correlation method of maximal information to this problem, computing it from the chosen features and then clustering it subsequently.

Khan and M. Z. [40] compared well-known classifiers of composite ensembles with supervised learning using eight datasets. The findings demonstrated that bagging with an AdaBoost SVM produced excellent accuracy. Using NASA datasets, Kumar et al. [41] suggested two algorithms for predicting flaws utilizing partial least squares and asymmetric kernel PCA.

From the literature review of SDP, different issues and challenges may emerge:

1- SDP accuracy: SDP model still suffers from the accuracy, so researchers are working tirelessly these days to enhance the functionality of SDP models.

2- Feature selection: It might be difficult to determine which features or metrics of software are most likely to cause defects. There may not be agreement in the literature on the most important features, and it might be difficult to extract useful features from software data.

3- Differences techniques: A variety of approaches may be presented in the literature, making it challenging to determine which is the most reliable or efficient.

4- Uncertainty issues: uncertainties present in software features and predict defects with feasible accuracy.

5- Insufficient data: SDP still have a challenge especially in new projects or in projects with no historical data.

6- The core concept of SDP revolves around constructing classifiers capable of predicting modules or sections of code at the highest risk of failure

IV. DIFFERENT TYPES OF PREDICTION TECHNIQUES

The best-known approaches for predicting issues are ML techniques [42]–[44]. A number of algorithms, including transfer learning [45] [46], dictionary learning [4], multiple kernel ensemble learning [47] collaborative representation learning [48], and DL [49], [50], have been used to improve fault forecasts since new ML methods have been employed. We essentially classify the literature on associated fault prediction into the following three groups from the perspective of ML, as presented in Fig.2 [20]: unsupervised, semi-supervised, and supervised techniques. The utilization of all labeled data in a project to create fault forecasting model is known as supervised approaches. By using a project's enormous amount of data without labels and only a limited number of labeled training data, semi-supervised approaches build fault prediction models. Unsupervised techniques use the unlabeled data from a project to create fault predictive model instead of labeled data.

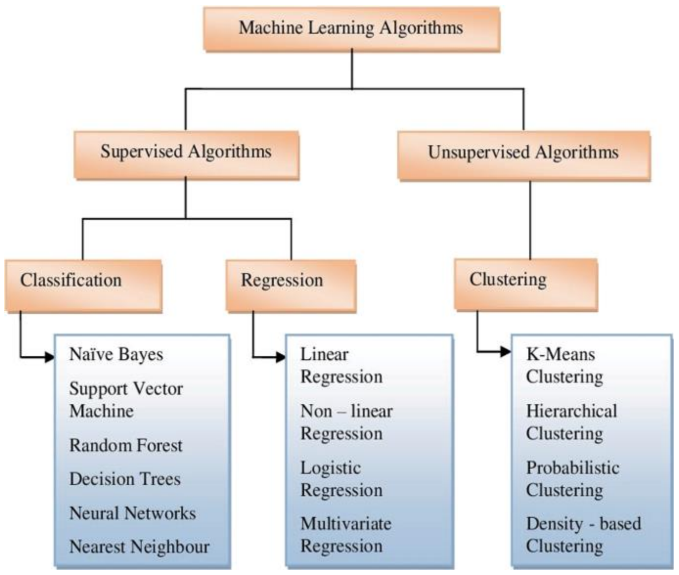

Fig.2 ML techniques in SDP.

1- *Supervised techniques*

Zhang et al. [4] created a cost-sensitive discriminative dictionary learning (CDDL) strategy to forecast fault using a recently discovered dictionary learning methodology. To solve the issue of imbalanced class , CDDL uses class knowledge from earlier data to enhance discriminant power and applies distinct misclassification costs by strengthening on type II misclassification (in other words, faulty class is forecasted as non-faulty).

A collaborative representation classification (CRC)-based Fault Anticipating (CSFA) technique was also proposed at the same time by Ying et al. [48]. The recently developed CRC approach, which is used by CSFA, is based on the principle that all other samples can be combined linearly to represent a single sample.

Xia et al. [51] developed two-layer ensemble learning (TLEL) solution for JIT fault anticipating using ensemble learning and decision tree methods. TLEL first constructed a random forest approach by merging bagging and decision tree. Then, using the random under-sampling approach, TLEL learned a variety of distinct random forest structures and again combines them using stacking ensemble.

2- *Semi-supervised techniques*

Liu et al. [52] introduced a non-negative sparse-based SemiBoost (NSSB) technique for SDP using ensemble learning and semi-supervised training . Using semi-supervised training, NSSB utilizes both an extensive amount of unlabeled cases and a limited size of labeled cases. On the other hand, NSSB uses ensemble learning to combine a variety of insufficient classifiers in order to decrease the bias brought on by the non-faulty.

A non-negative sparse graph-based label propagation (NSGLP) technique for SDP was put out by Jing et al. [53] using sparse representation learning techniques and graph-based semi-supervised training. To create a balanced data, NSGLP first made the cases that have been labeled as being non-defective. In order to better understand the link between the data, NSGLP then creates the weights utilizing the nonnegative sparse graph technique. Finally, using a label propagation strategy, NSGLP forecasts the unlabeled cases iteratively.

3- *Unsupervised techniques*

Enabling failure anticipating for new software or software lacking enough earlier data is a difficult topic. Kim and Nam [54] provided two unsupervised approaches to overcome this challenge. The fundamental concept behind these two techniques is the use of metric magnitude values to label an unlabeled data.

By contrasting and analyzing the above SDP model based on ML algorithms, as illustrated in Table 2, we may make the following conclusions:

i. For various forecasting contexts, fault forecasting approaches typically directly use or modify well-known ML techniques.

ii. To create classifiers, the majority of SDP strategies use supervised approaches. In general, supervised SDP approaches can increase the effectiveness of the anticipating, particularly the accuracy.

iii. Software code attributes provide the foundation for the majority of SDP techniques, which is made possible by the fact that they are simpler to gather than process metrics.

Table 2 Comparative analysis of SDP utilizing ML

Type	Name	Methods	Dataset
Unsupervised	Kim and Nam [54]	Feature selection, cluster	Relink
	Zheng et al.[55]	Spectral clustering	NASA
Semi-supervised	Jing et al. [53]	label propagation ,sparse graph & representation	NASA
	Zhang et al. [52]	Sparse representation, graph learning, SemiBoost.	NASA
Supervised	Xia et al. [51]	random forest, bagging, decision tree,	JIT
	Baik et al. [56]	boosting, cost-transfer, sensitive learning	PROMISE
	Penta et al. [57]	multi-objective optimization, Genetic Algorithm (GA)	PROMISE
	Shang et al. [46]	AdaBoost, transfer learning, GA	PROMISE
	Tan et al. [45]	Boosting, transfer learning	PROMISE
	Tan et al.[49]	DL	PROMISE
	Jing et al. [47]	Boosting, multiple kernel ensemble	NASA
	Ying et al. [48]	collaborative representation	NASA
	Zhang et al. [4]	cost-sensitive, dictionary learning,	NASA

V. Evaluation metrics

Numerous evaluation metrics have been widely utilized in [6], [24], [58]–[61] to assess SDP capability. The study of data in a confusion matrix (CM) [62] is typically the foundation for measuring SDP performance. This CM compares the different fault types' expected classifications to their real classifications. The CM and four SDP results are presented in Table 3. Here, true negative (TN), false positive (FP), false negative (FN), and true positive (TP) are the number of non-faulty samples that are expected as non-faulty, the number of non-faulty samples that are expected as faulty, the number of faulty samples that are expected as non-faulty, and the number of faulty samples that are expected as faulty, respectively.

Table 3 General CM.

	Expected faulty	**Expected non-faulty**
Real faulty	TP	FN
Real Non-faulty	FP	TN

The CM can define the following performance assessment metrics, commonly used in SDP research. Table 4 lists the most popular performance assessment metrics for SDP.

Table 3 Various performance assessment metrics

Metric	**Description**
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
FP rate	$\frac{FP}{FP+TN}$
G-measure	$\frac{2*Recall*(1-FP\ rate)}{recall+(1-FP\ rate)}$
F-measure	$\frac{2*Recall*Precision}{Recall+Precision}$

VI. Conclusion

As higher accuracy software systems are developed, (SQA) has grown to be one of the most crucial and costly phases. The complexity of Software keeps rising as software system take on more significance in our daily lives. Due to the growing complexity, quality assurance is extremely challenging to implement. fault -prone components can be identified by SDP models, allowing SQA teams to more efficiently allocate their limited code examination and testing resources by concentrating their efforts on these components. Recent years have seen a rapid emergence of new SDP approaches, issues, and applications. This paper makes an effort to comprehensively outline all common papers on SDP published lately. Depending on the findings of this study, this survey will assist academics and developers in more easily and effectively understanding earlier SDP studies from the perspectives of modeling methodologies, software metrics datasets, and evaluation metrics.

References

[1] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, "Literature Review: A Comparative Study of Software Defect Prediction Techniques," in *Proceedings of 3rd International Conference on Artificial Intelligence: Advances and Applications: ICAIAA 2022*, 2023, pp. 13–29.

[2] H. K. Dam *et al.*, "Lessons learned from using a deep tree-based model for software defect prediction in practice," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 46–57.

[3] R. Rana, M. Staron, J. Hansson, and M. Nilsson, "Defect prediction over software life cycle in automotive domain state of the art and road map for future," in *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, 2014, pp. 377–382.

[4] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 414–423.

[5] S. A. Sherer, "Software fault prediction," *J. Syst. Softw.*, vol. 29, no. 2, pp. 97–105, 1995.

[6] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, 2008.

[7] S. S. Rathore and S. Kumar, "An empirical study of some software fault prediction techniques for the number of faults prediction," *Soft Comput.*, vol. 21, pp. 7417–7434, 2017.

[8] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artif. Intell. Rev.*, vol. 51, pp. 255–327, 2019.

[9] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, no. 4, pp. 308–320, 1976.

[10] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[11] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4626–4636, 2011.

[12] C. Jin, "Software defect prediction model based on distance metric learning," *Soft Comput.*, vol. 25, pp. 447–461, 2021.

[13] L. Rokach, "Ensemble methods for classifiers," *Data Min. Knowl. Discov. Handb.*, pp. 957–980, 2005.

[14] M. A. Elsabagh, M. S. Farhan, and M. G. Gafar, "Cross-projects software defect prediction using spotted hyena optimizer algorithm," *SN Appl. Sci.*, vol. 2, no. 4, p. 538, 2020, doi: 10.1007/s42452-020-2320-4.

[15] M. A. Elsabagh, M. S. Farhan, and M. G. Gafar, "Meta-heuristic optimization algorithm for predicting software defects," *Expert Syst.*, vol. 38, no. 8, p. e12768, 2021.

[16] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. data Anal.*, vol. 6, no. 5, pp. 429–449, 2002.

[17] C. Tantithamthavorn, A. E. Hassan, and K.

Matsumoto, “The impact of class rebalancing techniques on the performance and interpretation of defect prediction models,” *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1200–1219, 2018.

[18] S. M. Abd Elrahman and A. Abraham, “A review of class imbalance problem,” *J. Netw. Innov. Comput.*, vol. 1, no. 2013, pp. 332–340, 2013.

[19] Q. Song, Y. Guo, and M. Shepperd, “A comprehensive investigation of the role of imbalanced learning for software defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1253–1269, 2018.

[20] M. Prashanthi and M. Chandra Mohan, “Survey on Innovative Techniques to Predict Software Defects,” in *Innovations in Computer Science and Engineering: Proceedings of the Ninth ICICSE, 2021*, Springer, 2022, pp. 697–707.

[21] S. Pandey and K. Kumar, “Software Fault Prediction for Imbalanced Data: A Survey on Recent Developments,” *Procedia Comput. Sci.*, vol. 218, pp. 1815–1824, 2023.

[22] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, “Machine learning based methods for software fault prediction: A survey,” *Expert Syst. Appl.*, vol. 172, p. 114595, 2021.

[23] “PROMISE Software Engineering Repository.” [Online]. Available: http://promise.site.uottawa.ca/SERepository/

[24] Z. Li, X.-Y. Jing, and X. Zhu, “Progress on approaches to software defect prediction,” *IET Softw.*, vol. 12, no. 3, pp. 161–175, 2018.

[25] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, “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*, 2011, pp. 15–25.

[26] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the nasa software defect datasets,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013.

[27] M. D’Ambros, M. Lanza, and R. Robbes, “Evaluating defect prediction approaches: a benchmark and an extensive comparison,” *Empir. Softw. Eng.*, vol. 17, pp. 531–577, 2012.

[28] S. Kim, H. Zhang, R. Wu, and L. Gong, “Dealing with noise in defect prediction,” in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 481–490.

[29] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in *Third International Workshop on Predictor Models in Software Engineering (PROMISE’07: ICSE Workshops 2007)*, 2007, p. 9.

[30] I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, “Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction,” *IEEE Access*, vol. 8, pp. 8041–8055, 2020.

[31] S. S. Rathore, S. S. Chouhan, D. K. Jain, and A. G. Vachhani, “Generative Oversampling Methods for Handling Imbalanced Data in Software Fault Prediction,” *IEEE Trans. Reliab.*, vol. 71, no. 2, pp. 747–762, 2022.

[32] R. Singh and S. S. Rathore, “Linear and non-linear bayesian regression methods for software fault prediction,” *Int. J. Syst. Assur. Eng. Manag.*, vol. 13, no. 4, pp. 1864–1884, 2022.

[33] H. K. Dam *et al.*, “A deep tree-based model for software defect prediction,” *arXiv Prepr. arXiv1802.00921*, 2018.

[34] E. Elahi, S. Kanwal, and A. N. Asif, “A new ensemble approach for software fault prediction,” in *2020 17th international Bhurban conference on applied sciences and technology (IBCAST)*, 2020, pp. 407–412.

[35] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, “Software bug prediction using machine learning approach,” *IJACSA) Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 2, 2018.

[36] R. Jayanthi and L. Florence, “Software defect prediction techniques using metrics based on neural network classifier,” *Cluster Comput.*, vol. 22, no. 1, pp. 77–88, 2019.

[37] P. D. Singh and A. Chug, “Software defect prediction analysis using machine learning algorithms,” in *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*, 2017, pp. 775–781.

[38] H. Wei, C. Hu, S. Chen, Y. Xue, and Q. Zhang, “Establishing a software defect prediction model via effective dimension reduction,” *Inf. Sci. (Ny).*, vol. 477, pp. 399–409, 2019.

[39] C. Shan, B. Chen, C. Hu, J. Xue, and N. Li, “Software defect prediction model based on LLE and SVM,” 2014.

[40] M. Z. Khan, “Hybrid Ensemble Learning Technique for Software Defect Prediction,” *Int. J. Mod. Educ. Comput. Sci.*, vol. 12, no. 1, p. 1, 2020.

[41] V. Kumar, J. K. Chhabra, and D. Kumar, “Parameter adaptive harmony search algorithm for unimodal and multimodal optimization problems,” *J. Comput. Sci.*, vol. 5, no. 2, pp. 144–155, 2014.

[42] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A systematic literature review on fault prediction performance in software engineering,” *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, 2011.

[43] C. Catal and B. Diri, “A systematic review of software fault prediction studies,” *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7346–7354, 2009.

[44] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Appl. Soft Comput.*, vol. 27, pp. 504–518, 2015.

[45] L. Chen, B. Fang, Z. Shang, and Y. Tang, “Negative samples reduction in cross-company software defects prediction,” *Inf. Softw. Technol.*, vol. 62, pp. 67–77, 2015.

[46] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, “Hydra: Massively compositional model for cross-project defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 42, no. 10, pp. 977–998, 2016.

[47] T. Wang, Z. Zhang, X. Jing, and L. Zhang, “Multiple

kernel ensemble learning for software defect prediction," *Autom. Softw. Eng.*, vol. 23, pp. 569–590, 2016.

[48] X.-Y. Jing, Z.-W. Zhang, S. Ying, F. Wang, and Y.-P. Zhu, "Software defect prediction based on collaborative representation classification," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 632–633.

[49] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 297–308.

[50] Y. Liu, M. Xie, J. Yang, and M. Zhao, "A new framework and application of software reliability estimation based on fault detection and correction processes," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 65–74.

[51] X. Yang, D. Lo, X. Xia, and J. Sun, "TLEL: A two-layer ensemble learning approach for just-in-time defect prediction," *Inf. Softw. Technol.*, vol. 87, pp. 206–220, 2017.

[52] T. Wang, Z. Zhang, X. Jing, and Y. Liu, "Non-negative sparse-based SemiBoost for software defect prediction," *Softw. Testing, Verif. Reliab.*, vol. 26, no. 7, pp. 498–515, 2016.

[53] Z.-W. Zhang, X.-Y. Jing, and T.-J. Wang, "Label propagation based semi-supervised learning for software defect prediction," *Autom. Softw. Eng.*, vol. 24, pp. 47–69, 2017.

[54] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 452–463.

[55] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 309–320.

[56] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Softw. Qual. J.*, vol. 25, pp. 235–272, 2017.

[57] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Defect prediction as a multiobjective optimization problem," *Softw. Testing, Verif. Reliab.*, vol. 25, no. 4, pp. 426–459, 2015.

[58] X. XUAN, L. O. David, X. XIA, and Y. TIAN, "Evaluating Defect Prediction using a Massive Set of Metrics.(2015)," in *SAC'15: Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13*, vol. 17, pp. 1644–1647.

[59] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *J. Syst. Softw.*, vol. 83, no. 1, pp. 2–17, 2010.

[60] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, vol. 1, pp. 789–800.

[61] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empir. Softw. Eng.*, vol. 13, pp. 561–595, 2008.

[62] F. Antaki, R. G. Coussa, G. Kahwati, K. Hammamji, M. Sebag, and R. Duval, "Accuracy of automated machine learning in classifying retinal pathologies from ultra-widefield pseudocolour fundus images," *Br. J. Ophthalmol.*, vol. 107, no. 1, pp. 90–95, 2023.