

A Comparative Review of Software Defect Prediction using Machine Learning Technique

¹Praveen Prakash Ranjan, ²Dr. Ankit Temurnikar

M. Tech. Scholar, Department of Computer Science and Engineering, Madhyanchal
Professional University (MPU), Bhopal, India¹

Professor, Department of Computer Science and Engineering, Madhyanchal Professional
University (MPU), Bhopal, India²

Abstract- Software defect prediction is an important research area in software engineering that aims to identify defect-prone software modules before software deployment. The early detection of software defects helps improve software quality, reduce maintenance costs, and minimize testing efforts. Traditional software testing methods often require considerable time and resources, making machine learning techniques valuable for automated defect prediction. This study presents a comparative review of software defect prediction using various machine learning techniques, including Decision Tree, Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression, Naïve Bayes, and ensemble learning approaches. The review examines the role of software metrics, feature selection, data preprocessing, and classification algorithms in predicting defective and non-defective software modules. Furthermore, it discusses the advantages and limitations of different machine learning techniques in terms of prediction accuracy, precision, recall, F1-score, and computational efficiency. Particular attention is given to class imbalance, feature redundancy, dataset variability, and model generalization, which significantly influence prediction performance. The comparative analysis aims to identify methodological trends, research challenges, and opportunities for developing more reliable and efficient software defect prediction systems. The findings of this review can support researchers and software developers in selecting suitable machine learning approaches for software quality assurance and improving defect detection during the software development lifecycle.

Keywords: Software Defect Prediction, Machine Learning, Software Quality Assurance, Feature Selection, Classification, Ensemble Learning

I. INTRODUCTION

Software development has become an essential part of modern technological advancement, supporting applications in healthcare, education, banking, communication, business, and industrial automation. The increasing complexity of software systems has created significant challenges in maintaining software quality, reliability, and performance. Software defects are errors, faults, or flaws in software programs that may lead to incorrect outputs, system failures, security vulnerabilities, and unexpected behavior. These defects can increase development costs, delay project completion, and reduce user satisfaction. Therefore, the early identification and prevention of software defects are important objectives in software engineering. Software defect prediction (SDP) has emerged as an effective approach for identifying potentially defective software modules before the testing and deployment stages.

Traditional software testing techniques primarily depend on manual inspection, code review, and conventional testing procedures. Although these methods are useful for identifying software defects, they may require considerable time, effort, and computational resources, particularly for large and complex software projects. Furthermore, identifying all possible defects through traditional testing can be difficult because software systems contain numerous interconnected components [1, 2].

To overcome these challenges, researchers have introduced data-driven techniques that analyze historical software project information and identify patterns associated with defective modules. Software defect prediction uses historical defect records, software metrics, and development-related information to estimate the probability that a software component contains defects. This process helps developers prioritize testing activities and allocate resources to the modules that require greater attention [3].

Machine learning (ML) techniques have gained considerable attention in software defect prediction because of their ability to analyze large datasets and identify complex relationships among software metrics. Various machine learning algorithms, including Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression (LR), and Naïve Bayes (NB), have been widely investigated for software defect classification. These algorithms learn from historical software data and classify software modules as defective or non-defective. Decision Tree provides an interpretable classification structure, while Random Forest combines multiple decision trees to improve prediction stability. SVM identifies decision boundaries between different classes, whereas KNN classifies software modules based on the similarity of their feature values. Logistic Regression estimates the probability of defect occurrence, and Naïve Bayes applies probabilistic principles for classification. The performance of these algorithms depends on dataset characteristics, feature selection, data preprocessing, and parameter optimization [4].

Ensemble learning techniques have also become important in software defect prediction because they combine multiple machine learning models to improve classification performance. Methods such as bagging, boosting, and voting classifiers can enhance prediction reliability by integrating the outputs of different learning algorithms. Random Forest and Gradient Boosting are commonly investigated ensemble approaches for software defect prediction. In addition, hybrid approaches combining feature selection, data balancing, and classification algorithms can help address challenges associated with high-dimensional software metrics and imbalanced datasets. Since defective software modules are often less frequent than non-defective modules, class imbalance may cause prediction models to favor the majority class. Consequently, evaluation using accuracy alone may not provide a complete understanding of model performance. Metrics such as precision, recall, F1-score, sensitivity, specificity, and area under the ROC curve are important for evaluating defect prediction systems [5, 6].

Despite the development of several machine learning approaches, software defect prediction continues to face various research challenges. The performance of predictive models may vary across software projects because of differences in programming languages, software

development practices, dataset distributions, and software complexity. Irrelevant or redundant features can also reduce model efficiency and affect prediction accuracy. Furthermore, models trained on one software project may not perform consistently when applied to another project [7, 8]. Therefore, selecting suitable machine learning algorithms and evaluating their generalization capabilities are important aspects of software defect prediction research. Comparative evaluation of different algorithms using consistent datasets, preprocessing procedures, and performance metrics can provide a better understanding of their strengths and limitations [9].

The present study focuses on a comparative review of software defect prediction using machine learning techniques. It examines different classification algorithms, their underlying principles, and their applications in identifying defective software modules. The study also discusses the importance of software metrics, feature selection, data preprocessing, class imbalance handling, and performance evaluation. Particular attention is given to comparing the prediction capabilities of individual machine learning algorithms and ensemble-based approaches [10]. The objective is to identify the major research challenges and methodological developments in software defect prediction and provide a foundation for selecting suitable techniques for future research. This comparative review can assist researchers and software developers in understanding machine learning-based defect prediction methods and their potential contribution to software quality assurance, testing optimization, and reliable software development.

II. SOFTWARE DEFECT PREDICTION

The SDP administers assets of the heads which are in programming quality insistence and this registers the flaws and their district of the source code which depends upon the experience and the information gathered from the experience which is reliably inauspicious thusly costly. It further joins the information mix which is from two of the different cycles that utilizes different vaults: there is a bug following in much the same way as a source code submitting. The clarification behind these stories is that it bars any product advancement process and the designs won't be related irrefutably as a mind blowing obstruction that causes utilizing the programming advancement cycle in the lifecycle of programming progress which is amazing. There is besides an essential for an intentional cycle that is utilized for motorizing of the example of information bunching.

The SDP plays a vivacious piece in guaranteeing programming quality by supervising the allotment of assets to the heads capable for programming quality confirmation. In this cycle, the SDP distinguishes and records abandons in the source code and decides their particular area. This essential data is gotten from a blend of involvement and information gathered from dreary assessments, which can be tedious and exorbitant. Additionally, the SDP flawlessly coordinates information from two unmistakable cycles: bug following and source code accommodation, utilizing various storehouses to upgrade effectiveness and efficiency.

The charming reasoning behind these enthralling stories lies in the disallowance of the SDP, thwarting the consistent joining of designs. This causes an exceptional obstruction in

programming improvement, as the SDP stays tricky. Besides, it becomes clear that a conscious cycle is fundamental for the computerization of the information grouping design. Therefore, to explore and appreciate these stories completely, it becomes basic to dive into the complexities of the SDP's prohibition and the basic job of deliberate cycles in smoothing out information grouping processes.

The SDP dataset includes an extensive variety of significant data gathered into two particular classifications. It, right off the bat, catches fundamental programming measurements for example, size and intricacy appraisals, which assist with assessing programming quality.

Furthermore, it records programming with measure of bugs present in individual module, giving important bits of knowledge into its general security. This refined framework works with the greatest amount of productivity, as it proficiently normalizes and mechanizes the assortment of SDP information. This liberates the client from specialized particulars. Its consistent usefulness works on the client experience, empowering them to zero in exclusively on the project within reach.

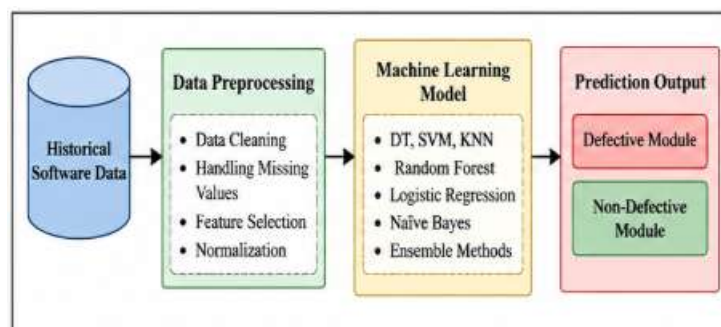


Figure 1: Software defect prediction

Guaranteeing the nature of a thing relies upon two variables: the originator's aptitude and the philosophy utilized to recognize blemishes. The SDP presents a double move toward in its continuous assessment, arranging itself as deformity situated or non-defect-arranged. By analyzing the inborn vulnerability related with size-based assessments and their powerful allotment of testing assets, one can reveal the viability of one or the other activity or testing. This complicated interchange between different components ensures an elevated requirement of item greatness.

Inconsistent frameworks have prompted a scope of working proficiency and shortcoming discovery models that address the innate disappointment related with them. These models are unpredictably founded on the utilization of a strong design. This includes information obtaining through item imperfection distinguishing proof, check, and thorough testing techniques. These basic advances guarantee the distinguishing proof of potential deficiencies and make ready for enhancements. These savvy discoveries can then be utilized to expect and correct any likely disfigurements, laying out a establishment for upgraded execution and consumer loyalty [6, 7].

The avoidance of both monetary and human calamities requires an fundamental comprehension of the critical job in keeping up with the trustworthiness of the complex

frameworks that manage ethereal alterations plays. Any imperfections inside these frameworks might possibly cause underlying disappointments yet in addition trigger prompt location. Perceiving that bends inside these parts can have extreme suggestions, it becomes basic to focus on their recognizable proof and redress them. By grasping the criticality of resolving these issues speedily, it tends to be protected against possibly unfavorable results from split the difference foundational layouts.

Quality confirmation for item is urgent, requiring careful oversight furthermore, cautious data scattering. In this specific circumstance, the product advancement process expects an outstanding job, filling in as an extensive assessment device for programming execution and module mix. The SDP works with immaculate programming execution by distinguishing and overseeing modules with inborn blemishes.

This deliberate methodology guarantees potential deformities are prudently settled, empowering consistent programming usefulness. Utilizing the SDP ideally permits associations to convey remarkable programming arrangements, further supporting their notoriety for greatness in the field. With its significant help, the SDP engages organizations to lift programming advancement cycles to exceptional levels. Through the usage of simulated intelligence (Computerized reasoning) calculations, the programming Improvement Interaction has earned respect for its capacity to actually examine and deal with a significant measure of neighborhood preparing information. This is especially advantageous for modules that are probably going to be taken out. This underlines the meaning of preparing information that displays comparative development to test information, as [7, 8] stressed. This exhibits SDP's capability in utilizing artificial intelligence advances to guarantee ideal execution and achievement.

III.LITERATURE REVIEW

Zhou, Zhu, Yu, and Li [1] proposed a software defect prediction method based on multi-feature fusion. The study focused on improving defect prediction by integrating different software features to obtain more comprehensive representations of software modules. Multi-feature fusion can help capture relationships among software metrics and improve the identification of defect-prone components. The approach highlights the importance of combining multiple feature sources rather than relying on individual metrics. The study is relevant to software quality assurance because early defect identification can support testing prioritization and reduce software maintenance costs. However, the effectiveness of the approach depends on feature quality, dataset characteristics, and model generalization.

Siddika et al. [2] presented a study titled “Enhancing Software Defect Prediction Using Ensemble Techniques and Diverse Machine Learning Paradigms.” The research focused on improving software defect prediction by investigating ensemble learning techniques and different machine learning approaches. Ensemble methods combine the outputs of multiple predictive models to improve the reliability of classification. The study emphasizes the importance of evaluating different machine learning paradigms for identifying defective and non-defective software modules. Such approaches can support software developers in

detecting potential defects during the early stages of software development. The research also highlights the relevance of predictive modeling in improving software quality and reducing testing efforts. However, the effectiveness of ensemble techniques depends on the quality of the training data, model selection, and evaluation methodology. The study provides a foundation for further investigation into hybrid and optimized machine learning models for software defect prediction.

Setia et al. [3] investigated software defect prediction using machine learning techniques. The study addressed the application of machine learning in identifying software components that may contain defects based on available software metrics. Machine learning algorithms can analyze historical software data and identify patterns associated with defective modules. This approach can help developers prioritize software testing and allocate resources to components that require greater attention. The research highlights the potential of data-driven techniques in improving software quality assurance compared with conventional manual defect identification methods. Furthermore, machine learning-based prediction can support early defect detection and assist software development teams in making informed testing decisions. However, prediction performance may be affected by imbalanced datasets, irrelevant features, and variations in software project characteristics. The study provides a basis for further research into efficient and reliable machine learning-based software defect prediction systems.

Albattah and Alzahrani [4] presented an empirical study on software defect prediction based on machine learning and deep learning techniques. The research explored the application of these approaches to identify defective software components using software-related data. Machine learning techniques provide effective classification capabilities, while deep learning models can learn complex patterns and feature relationships from data. The empirical approach highlights the importance of evaluating predictive models using experimental data to understand their effectiveness in software defect prediction. Such techniques can assist developers in identifying defect-prone modules and improving software testing strategies. The study also emphasizes the role of intelligent prediction methods in supporting software quality assurance. However, the performance of machine learning and deep learning models can vary depending on dataset size, feature representation, and model configuration. The research provides a foundation for further comparative analysis and the development of advanced defect prediction models.

Muhammad Azam et al. [5], one of the most active areas in the software development community is defect prediction. Software success depends on bridging the gap between data mining and software development. Identifying errors in source code before testing makes programming exceed expectations. Software defect prediction techniques such as clustering, statistical methods, mixture algorithms, neural network-based metrics, black box testing, white box testing, and machine learning are widely used to study the scope of software. The key task of this study is to improve the accuracy of computer intelligence classifiers in pre-linguistic ability deserts using component determination. In this study, the accuracy of deformation prediction in five NASA information indicators was further developed: PC1, CM1, JM1, KC2, KC1, and these NASA information ranges are publicly available. This study combines element determination methods with AI techniques. Random forests, logistic

regression, multilayer perceptrons, Bayesian nets, rule ZeroR, J48, lazy IBK, neural networks, and decision stamps were used to achieve higher defect prediction accuracy than without feature selection (WOFS). The Research Workbench applies the aforementioned classifiers and uses WEKA (Waikato Environment for Knowledge Analysis) machine learning tools to preprocess and refine the data. The Minitab Statistics tool is used when evaluating statistical analysis. This study shows that the accuracy of WOFS is not the accuracy of WOFS, but the accuracy of WFS (Absconds Expectation with Include Determination).

A. Perera et al. [6], coverage information is the most important guide for automatic test generators, such as: B. Search-Based Software Testing (SBST) techniques, and therefore very suitable for covering large amounts of code. But is code coverage alone enough to effectively detect errors? In this article, we propose another SBST strategy, Prescient Multiple Goal Arrangement Calculation (PreMOSA), which augments inclusion data with transformation prediction data to determine where to create test inclusions within the class under test (CUT). We show that PreMOSA outperforms the current state-of-the-art Dynamic Multi-Objective Classification Algorithm (DynaMOSA) in defect detection when compared to acceptable defect predictors. H. A defect predictor with a 75% hit rate and accuracy in an experimental evaluation with 420 labeled defects on the Defects4J benchmark. With a time limit of 2 minutes for test generation per CUT, PreMOSA typically detects up to 8.3% more flagged errors than DynaMOSA.

M. A. Kabir et al. [7], IRDP is a real-world scenario in which a prediction model is built using a dataset of previous versions to predict defects in the current version within the same software project. Real software projects have multiple versions, and data for each version arrives in chronological blocks. The evolving information in each delivery introduces new ideas, called idea float, into the model, which negatively impacts the performance of the IRDP model. This review aims to analyze and evaluate the impact of factor determination (FS) on the performance of IRDP models and the robustness of the models to idea drift. In the empirical experiments, we use 36 versions of 10 open source projects. The results of Friedman and Nemenyi's post-hoc tests show that the prediction results with and without FS techniques are statistically different. The IRDP model built on the data of the latest publications is not always the best model. A prediction model trained with carefully selected features can also reduce idea drift.

S. Madipati et al. [8], there were a number of serious concerns regarding the credibility and integrity of the conference organization and peer review process. After a thorough investigation, it was determined that the peer review process did not meet the high standards of Materials Today: Proceedings. There is still significant uncertainty about the accuracy of the conference, so the entire proceedings have been removed to address the academic record.

Alsghaier, H. et al. [9], software defect prediction is a modularization process that helps developers identify modules that require additional refactoring during maintenance and identify defective classes or modules early in the development lifecycle. Software reliability means the existence of the possibility of failures over time. When we say a system is unreliable, we mean that it has many errors. While these errors are fine for some systems, they can cause serious problems for critical systems such as space shuttles, aircraft, and medical systems. Therefore, finding faulty programming modules is a fundamental step as it helps to identify modules that require serious rework or real testing. In this paper, we develop a methodology to predict programming errors by combining a genetic algorithm (GA)

with assistant vector machine (SVM) classifiers and molecular swarm algorithms as an alternative to better methods for predicting programming errors. The new methodology is applied to 24 datasets, of which 12 are NASA MDPs and 12 Java open source projects. The NASA MDP is a large data set, while the Java open source project is a small data set. When applied to both large and small datasets, combining GA with SVM and particle swarm algorithms improves the performance of the software fault prediction process and overcomes the limitations of previous research.

Yu et al. [10], some learning-to-rank algorithms have been compared in previous empirical studies on only a few datasets, without robust statistical testing techniques, and using single or inadequate performance measures. We evaluate the practical impact of 23 learning-to-rank algorithms using a module-based effort-aware performance measure (FPA) and a source code lines (SLOC)-based effort-aware performance measure (Norm(P opt)). To address these concerns and investigate the impact of learning-to-rank algorithms on the performance of EADP models, we utilized 41 incomplete datasets available from the PROMISE repository. We also compare how these algorithms perform when trained on more relevant feature subsets, determined by an "information gain" feature selection method. We find statistically significant differences between these algorithms in terms of FPA and norm (P opt), with Bayesian Ridge Regression (BRR) and learning-to-rank (LTR) achieving the best results in terms of norm (P opt). If there is a large difference between FPA and Norm (P opt), LTR and BRR still perform best when trained on a more relevant feature subset selected by information gain. Therefore, we recommend building EADP models with BRR and LTR so that more errors can be found by examining a given number of modules or lines of code.

Problem Formulation: Software defect prediction is formulated as a binary classification problem in which software modules are categorized as defective or non-defective based on their software characteristics. The primary objective is to develop an effective machine learning model that can identify defect-prone software modules using historical software project data. Each software module is represented by a set of software metrics, including lines of code, cyclomatic complexity, coupling, cohesion, and Halstead metrics. These features provide information about the size, complexity, and structural characteristics of software components. The dataset consists of multiple software modules, where each module is associated with a feature vector and a corresponding class label. The class label represents whether the module contains defects or is free from defects. Machine learning algorithms such as Decision Tree, Random Forest, Support Vector Machine, K-Nearest Neighbors, Logistic Regression, Naïve Bayes, and ensemble learning techniques can be applied to learn patterns from historical data and predict the defect status of previously unseen modules.

IV. MACHINE LEARNING

Machine Learning (ML) is a branch of artificial intelligence that enables computer systems to learn patterns from historical data and make predictions without being explicitly programmed for every task. In software defect prediction, machine learning techniques are used to identify defective and non-defective software modules by analyzing software metrics such as lines of code, cyclomatic complexity, coupling, cohesion, and Halstead metrics. These metrics provide important information about the structural characteristics and complexity of software components. Machine learning algorithms learn relationships between software metrics and previously identified defects to predict the defect status of new software modules. This

approach helps software developers identify defect-prone components before software deployment and improve the overall software development process.

Various machine learning algorithms are applied to software defect prediction, including Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression (LR), and Naïve Bayes (NB). Decision Tree uses a tree-based structure to classify software modules, while Random Forest combines multiple decision trees to improve prediction stability. SVM identifies an optimal decision boundary between defective and non-defective modules, whereas KNN classifies modules based on their similarity to previously observed samples. Logistic Regression estimates the probability of defect occurrence, and Naïve Bayes performs classification using probabilistic principles. Ensemble learning techniques, such as boosting and voting classifiers, combine multiple models to improve prediction performance.

Supervised Learning

This is an AI task of deriving features from labeled training data. A set of training examples forms the training data. Each sample in supervised learning is a pair of an input object and a desired output value. As we know, there are two tasks in supervised learning: regression and classification. Classification involves building a predictive model for a function in a discrete range, while regression involves building a model for a continuous range. Supervised learning has attracted the attention of many researchers in the field of machine learning. Concept learning, classification, rule learning, instance-based learning, Bayesian learning, linear regression, neural network learning, and support vector machine learning are the most commonly used techniques of supervised machine learning.

Unsupervised learning is also known as learning by observation. In unsupervised learning, the system has to examine all the patterns based only on the common characteristics of the examples, without knowing how many patterns there are or if there are any at all. Clustering, association rule mining, and sequential pattern mining are the most common methods of unsupervised learning. Predicting Programming Errors with AI the field of AI is evolving rapidly, giving rise to a large number of learning algorithms for various applications.

ML used in SDP

The final value of these algorithms is usually determined by the condition that determines the progress in solving real problems. Therefore, the development of this field depends on the replication of algorithms and their application to new tasks. Nevertheless, various AI researchers are already working on developing models to predict programming errors. Currently, we classify successful programming variant models into three categories based on their characterization, grouping, and collection strategies.

machine learning is not a hard science. Software developers can use machines to automatically learn from training data and create small summaries of data and existing systems, speeding up system development and saving time and cost. According to Cannon et al., the development of software engineering based on machine learning is a barrier to integrating machine learning into software development. As with any other application, before applying machine learning, it is necessary to preprocess the complexity of software development data and patterns. Currently, developers pay a lot of attention to component-based error prevention methods, because splitting a project into separate components reduces development costs and time, and improves the quality and reliability of the project. However, this method is only suitable for identifying quality issues in individual components.

V. CONCLUSION

Software defect prediction using machine learning techniques plays an important role in improving software quality, reliability, and maintenance efficiency. This study presents a comparative review of various machine learning algorithms, including Decision Tree, Random Forest, Support Vector Machine, K-Nearest Neighbors, Logistic Regression, Naïve Bayes, and ensemble learning techniques for identifying defective and non-defective software modules. These algorithms analyze historical software data and software metrics to identify patterns associated with defect occurrence. The review highlights the importance of data preprocessing, feature selection, class imbalance handling, and appropriate performance evaluation in developing reliable prediction models. Performance metrics such as accuracy, precision, recall, F1-score, sensitivity, and specificity provide valuable information for evaluating classification performance. However, challenges such as high-dimensional features, imbalanced datasets, and variations in software project characteristics can affect prediction reliability and generalization. Future research should focus on developing hybrid machine learning models, applying advanced optimization techniques, and improving cross-project defect prediction capabilities. The integration of explainable artificial intelligence may also help developers understand the factors contributing to defect predictions. Overall, machine learning-based software defect prediction provides an effective data-driven approach for identifying potential software defects, prioritizing testing activities, reducing maintenance costs, and supporting the development of reliable software systems.

REFERENCES

- [1] Z. Zhou, Y. Zhu, Q. Yu and J. Li, "Software Defect Prediction Method Based on Multi-Feature Fusion," *2025 25th International Conference on Software Quality, Reliability and Security (QRS)*, Hangzhou, China, 2025, pp. 529-539.
- [2] Siddika, Ayesha, Momotaz Begum, Fahmid Al Farid, Jia Uddin, and Hezerul Abdul Karim. 2025. "Enhancing Software Defect Prediction Using Ensemble Techniques and Diverse Machine Learning Paradigms" *Eng* 6, no. 7: 161.
- [3] S. Setia, K. K. Ravulakollu, K. Verma, S. Garg, S. K. Mishra and B. Sharan, "Software Defect Prediction using Machine Learning," *2024 11th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2024.
- [4] Albattah, Waleed, and Musaad Alzahrani. 2024. "Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach" *AI* 5, no. 4: 1743-1758.
- [5] Muhammad Azam, Muhammad Nouman and Ahsan Rehman Gill, "Comparative Analysis of Machine Learning techniques to Improve Software Defect Prediction", *KIET Journal of Computing & Information Sciences [KJCIS]* Vol. 5, Issue 2, 2022.
- [6] Perera, A., Aleti, A., Turhan, B. & Boehme, M. An experimental assessment of using theoretical defect predictors to guide searchbased software testing. *IEEE Trans. Sofw. Eng.* Vol. 1, issue 1, 2022.

- [7] Kabir, M. A., Keung, J., Turhan, B. & Bennin, K. E. Inter-release defect prediction with feature selection using temporal chunkbased learning: An empirical study. *Appl. Sof Comput.* 113, 107870, 2021.
- [8] Maddipati, S.; Srinivas, M. Machine learning approach for classification from imbalanced software defect data using PCA and CSANFIS. *Mater. Today Proc.* 52, 471, 2021.
- [9] Alsghaier, H. & Akour, M. Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier. *Sofw. Pract. Exp.* 50(4), 407–427, 2020.
- [10] Yu, X., Bennin, K. E., Liu, J., Keung, J. W., Yin, X., & Xu, Z. An empirical study of learning to rank techniques for effort-aware defect prediction. In *SANER 2019 - Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering*, pp. 298–309, 2019.
- [11] Bach, T., Andrzejak, A., Pannemans, R. & Lo, D. The impact of coverage on bug density in a large industrial software project. *ACM/ IEEE Int. Symp. Empirical Sofw. Eng. Meas. (ESEM)*, 307–313, 2017.
- [12] Mohammed, R., Rawashdeh, J., & Abdullah, M. Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. In *11th International Conference on Information and Communication Systems (ICICS)*, pp. 243–248, 2020.
- [13] Aljamaan, H.; Alazba, A. Software defect prediction using tree-based ensembles. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Virtual USA*, 8–9 November, pp. 1–10, 2020.
- [14] Li, R.; Zhou, L.; Zhang, S.; Liu, H.; Huang, X.; Sun, Z. Software Defect Prediction Based on Ensemble Learning. In *Proceedings of the 2nd International Conference on Data Science and Information Technology*, Seoul, Republic of Korea, 19–21, pp. 1–6, 2019.
- [15] Alsaeedi, A.; Khan, M.Z. Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study *J. Softw. Eng. Appl.* 12, 85–100, 2019.
- [16] Tua, F.M.; Sunindyo, W.D. Software Defect Prediction Using Software Metrics with Naïve Bayes and Rule Mining Association Methods. In *Proceedings of the 5th International Conference on Science and Technology (ICST)*, Yogyakarta, Indonesia, pp. 1–5, 2019.
- [17] Manjula, C.; Florence, L. A Deep neural network based hybrid approach for software defect prediction using software metrics. *Clust. Comput.*, 22, 9847–9863, 2019.
- [18] Manjula, C.; Florence, L. Hybrid approach for software defect prediction using machine learning with optimization technique. *Int. J. Comput. Inf. Eng.*, 12, 28–32, 2018.