Proceedings of the 5th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/99/20251769

A Brief Analysis of the Progress and Trends in Software Defect Prediction Methods

Ruxi Jia[1,a,*]

[1]School of Computer Science and Technology, Zhejiang University of Technology, No. 18, Chaowang Road, Gongshu District, Hangzhou City, Zhejiang Province, China
a. jiaruxi@zjut.edu.cn
**corresponding author*

Abstract: Software defect detection is particularly important for modern society, as it is a crucial step in ensuring the quality and reliability of software systems. With the emergence of artificial intelligence (AI), research in software defect detection has evolved from traditional methods to more complex approaches that utilize deep learning and large language models (LLMs). The advent of LLMs has fundamentally changed the paradigm of software development and defect detection, bringing new challenges and confusion to the field of software defect prediction research. To address these issues, we compare software defect detection methods based on traditional techniques, deep learning approaches, and LLMs through a literature review. We analyze the changes brought about by the introduction of LLMs to software development and propose new insights. Additionally, we examine the progress and trends in software defect prediction to provide inspiration for subsequent research.

Keywords: Software Defect Prediction, Machine Learning, Deep Learning, Large Language Model.

1. Introduction

Software now plays a crucial role in various fields such as national economy, defense, government affairs, and daily life. The performance and complexity of these systems largely depend on their stability. Defects in software may be potential causes of system failures, crashes, and even equipment damage and personnel injuries. However, with the development of software technology, no inspection or verification method can detect and eliminate all defects. Although software does not wear out, it may malfunction or fail at any time due to reasons that are difficult to detect. Ensuring software quality is crucial and costly in the development of high reliability software systems. As software systems play an increasingly important role in our daily lives, their complexity is also constantly increasing [1]. Therefore, software defect detection is particularly important for modern society. It is a crucial step in ensuring the quality and reliability of software systems. In the 1970s, the use of statistical learning techniques emerged to predict the number and types of defects in software systems based on historical data, along with software measurement data such as discovered defects. Defect prediction technology has played a vital role in improving and ensuring software quality and has also significantly promoted the development of software engineering technology since the 1970s.

Taking cross projects software defect detection as an example, we first searched for papers related to the review topic in important academic search engines [2]. Then, we filtered out papers unrelated to the review topic by systematically analyzing the relevant content and supplemented the missed papers by analyzing the relevant job descriptions of the selected papers. The final number of relevant papers published from 2002 to 2016 is shown in Figure I.

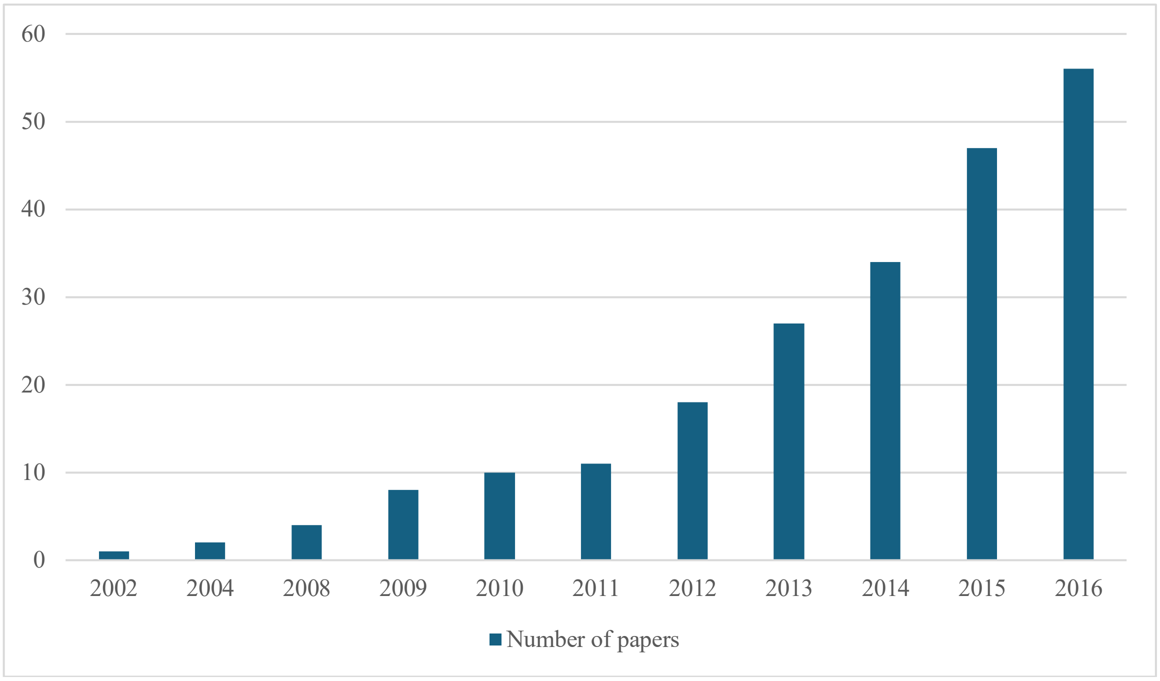

Figure 1: Statistics on the cumulative number of published papers each year.

Figure 1 shows that the number of published papers was relatively low before 2011, while since 2012, the number of related papers has shown a polynomial growth trend. The academic community's attention to software defect detection has increased since 2012.

Software defect prediction techniques have generally been divided into two types: static and dynamic defect prediction techniques. Static prediction technology refers to the technique of predicting the number or distribution of defects based on defect-related measurement data. Dynamic technology predicts the distribution of system defects over time based on the timing of defect or failure occurrences. With the emergence of AI, the field of software defect detection has evolved from traditional methods to more complex approaches that utilize deep learning and LLMs. For example, Li et al. introduced VulDeePecker, a deep learning system for vulnerability detection that processes code snippets and employs bidirectional LSTM to address gradient issues and dependencies. It achieved an F1 score of 90.5, outperforming traditional detection methods and reducing false positives[3]. Xu et al.'s "Contextual LSTM" integrates CNN for local feature extraction with RNN for sequential dependency analysis, achieving superior F1 scores over individual CNN and LSTM models on the Sard dataset [4].

The rise of large-scale language models has changed the mode of software development and defect detection, while bringing new challenges and uncertainties to the research of software defect prediction. However, there is currently a lack of systematic analysis in this area. To address this problem, we conduct a literature review, comparing and contrasting software defect detection methods based on traditional approaches, deep learning techniques, and LLMs. This review delves into the transformative impact of LLMs on software development processes and defect detection methods. By examining historical processes and current trends, we aim to clarify the advantages and limitations of each method.

In addition, we discuss the impact of large language model integration on software defect prediction, emphasizing the shift in research focus and the necessity of new strategies to address emerging complexities. The analysis presented in this study lays the foundation for future empirical verification and provides insights into the direction of software defect prediction research. It also helps guide practitioners and researchers in selecting and developing more effective defect detection tools.

2. Literature Review of Software Defect Prediction

2.1. Classical Software Defect Prediction Methods

Since the 1970s, statistical learning methods have emerged, utilizing historical records and software metrics (such as identified defects) to predict the total number and classification of defects in software systems. Defect prediction technology aims to assess whether a software system is ready for delivery. This technology tallies the number of confirmed defects in the system and estimates the number of potential defects that may not have been discovered yet. Defect prediction technology plays a crucial role in improving software quality and ensuring its reliability, while also driving significant development in the field of software engineering. In the early stages of software development, defect detection mainly relied on manual labor, including code reviews, design, and execution of test cases. Although this method was time-consuming and labor-intensive, it was the only option available at the time.

Since the 1970s, software defect prediction techniques have generally been divided into two types: static defect detection techniques and dynamic time-based software defect prediction techniques. The development of static defect prediction technology has a long history, initially focusing on defect prediction based on metrics such as software size. In order to predict the possible number of defects in software, researchers are committed to studying the relationship between defects and basic characteristics such as software size and complexity. In the early 1990s, people began to realize that defects in software were not uniformly or completely randomly distributed. It promoted the development of prediction techniques for defect distribution characteristics. Moreover, the introduction and removal of defects at different stages of the software development lifecycle significantly impact residual defects. Corresponding software defect prediction models have achieved remarkable results in practice[5].

Another crucial defect prediction technique is based on temporal relationships. Many dynamic reliability models adopt this type of prediction method. This area of research focuses on uncovering the distribution patterns of software defects over time during the software lifecycle or in specific phases through empirical studies and statistical methods. Dynamic testing identifies defects by executing the software, monitoring its behavior, simulating real-world usage environments, and detecting issues that are difficult to uncover through static analysis.

2.2. Software Defect Prediction Method Based on Deep Learning

In research on defect detection for large and complex software systems, it is commonly believed that most defect-related information can be gleaned through in-depth analysis of the code. Therefore, the ability to analyze code data is of great significance for how to efficiently detect software defects. Machine learning methods, especially those based on big data analysis, can identify and learn patterns from large amounts of data. Although early defect detection research primarily relied on machine learning techniques, these methods exhibit significant shortcomings in efficiency and accuracy. Most machine learning approaches depend on features manually crafted by experts, and their performance is limited by the quality of feature engineering.

With the remarkable success of deep neural networks have achieved great success in technologies such as image recognition and natural language processing, some researchers have begun to explore their application in source code defect detection. For example, S. Sivapurnima et al. developed an efficient adaptive deep learning model (ADLM) for the automatic detection and classification of duplicate error reports. In the preprocessing stage, data is collected from online systems. Useless information is removed through text cleaning, data type conversion, and null value replacement, encompassing stop word removal and stem extraction. Four types of feature extraction methods, namely context, classification, time, and text features, are adopted. After independently generating the LSTM and CRF models during the model construction phase, the LSTM is integrated into the CRF structure. Although CRF can independently utilize features for decision-making, classification may need to be improved due to solid dependencies on the output. LSTM, with its advantage of processing time series information, updates the hidden layer of CRF through its gating mechanism (including the forget gate, input gate, output gate, and memory unit), thus enhancing the model performance. Meanwhile, the characteristics of the Dingo Optimizer mathematical model are utilized to select the optimal weight parameters in CRF-LSTM. Compared with existing methods for this purpose, it demonstrates high accuracy[6]. Preliminary results indicate that deep learning exhibits advantages in defect detection that traditional methods and early machine learning approaches do not possess[7]. Moreover, there is considerable potential for further research in this field.

2.3. Software Defect Prediction Method Based on Large Language Model

The Large Language Model (LLM), taking ChatGPT as an example, has attracted great interest from the computational and data science communities due to its wide range of applications and powerful performance. Its significant effectiveness in understanding natural language and generating meaningful content has sparked interest in various disciplines, including software engineering. Therefore, after the proposal of the LLM, some researchers began to carry out software defect detection based on it. For example, X. Wang et al. proposed a novel framework based on a Large Language Model for software defect detection[8]. I t leveraged the pre-trained CodeT5+ and (IA)3 for parameter-efficient software defect detection. S. Hossain et al. demonstrated their success in APR technology based on the framework Toggle proposed by the LLM, establishing a new benchmark test for CodeXGLUE code optimizations[9]. It demonstrated significant performance on multiple datasets, including Defects4J.

The LLM has shown significant potential in the field of software development. It enables tasks such as code generation, program defect repair, code documentation, and the creation of test cases[10]. In the process of automating program debugging and repair, fault localization technology plays a crucial role and has become a highlight in the release of ChatGPT-4[11].

3. Comparative Analysis

3.1. Comparison Analysis of the Traditional Software Defect Prediction Methods

Table 1: Comparison Analysis of the Traditional Software Defect Prediction Methods.

Software Defect Prediction		Authors	Year	Techniques	Usage Scenario	Weakness
Static Software Defect Prediction	Defect Prediction Model	S. Chulani et al[12]	1999	COQUALMO model	Used in the planning phase	Static Software Defect Prediction
		M. Fagan[13]	1999	DRE model	Suitable for stable life cycle and process models	Reliable defect classification and matrix methods are needed to map to the source and discovery of defects
		N. Fenton et al[14]	2007	Bayesian model	Used in scenarios lacking defect data and requiring risk analysis	Needs expert experience and judgment.
		L. Briand et al[15]	2000	Capture-recapture model	Determine whether re-inspection of the software product is necessary	Requires stringent assumptions.
	Defect Distribution Prediction	T, Khoshgoftaar et al [16]	2000	Classification technology, Regression technology	Applying to make planning and testing	Cannot well balancing Type I and Type II misclassifications
	Metric-Based technology	T, Khoshgoftaar et al[17]	2006	Size-Based, Complexity-Based	Used in planning. The required data size, complexity, coupling degree, etc.	The density of faults during operation may not be accurately predicted by the defect density of the module; In order to maintain consistency between different systems, it is necessary to calibrate the model and assumptions.
		P. Li et al[18]	2005	Process metric-based		
		V. Basili et al[19]	1996	OO metric-based, Web metric-based		
Dynamic Defect Prediction Technology				Rayleigh model	Need historical defect density data and continuously track defect counts	Unable to adjust based on changes in products, personnel, platforms, or projects that may affect defect prediction.
		A. Goel et al[20]	1979	Exponential model	Applying to the formal testing phase	Testing effort is homogeneous.
		K. Trivedi[21]	1984	S-curves arrival distribution models	Excellent adaptation to large software projects with numerous defects	More complicated than the above two models.

The comparative analysis of traditional software defect prediction methods is presented in Table 1. It indicates that the selected papers were predominantly published in the late 20th and early 21st centuries. Moreover, the time span between these publications is relatively extensive. Traditional software defect detection methods have undergone extensive development and have been thoroughly researched and discussed in the academic community. For example, M. Fagan introduced the DRE model, which is suitable for cyclic and process models with stable lifetimes[13]. Furthermore, the DRE model examines the characteristics of diverse project stages and processes and maps defects onto corresponding stages and process segments. For instance, syntax and logic errors that might emerge during the encoding stage can be mapped within the DRE model, thereby facilitating a more precise location of the defect source. It formulates a relatively standardized defect handling process framework, incorporating defect classification and mapping methods. Subsequent research can extend and optimize this framework, offering a reference and a basis for developing more appropriate defect prediction models for various project types. L. Briand et al. proposed a capture-recapture model, which is used to determine whether software products need to be rechecked [15]. It is based on the capture-recapture principle in biology. Considering the distinct characteristics between software inspection and animal capture in biology, such as the varying abilities of different inspectors (time response factor) and the diverse probabilities of defect detection (heterogeneity factor) in software inspection, a model incorporating multiple sources of variation is established. For instance, four models are formulated: MO (with no variation), Mh (considering only heterogeneity variation), Mt (considering only time response variation), and Mth (considering both time response and heterogeneity variations). These models possess different assumptions regarding the detection probabilities of inspectors and defects, thereby being more congruent with the actual circumstances of software inspection. Simultaneously, the "virtual inspection" method systematically investigates the impacts of the number of inspectors and the total number of defects on model performance. The systematic variations of these two factors within appropriate ranges are analyzed. Ultimately, based on a comprehensive assessment of the accuracy, bias, and variability of different models and estimators under diverse conditions, recommendations for model selection are put forward for different numbers of inspectors and defects. This model is typically applied during software development's planning and formal testing stages, particularly in environments lacking defect data, where risk analysis is required and large projects are handled. This method is also crucial for retesting decisions and relies on historical defect data for continuous improvement.

Although traditional defect detection methods have a long history, their inherent patterns are inevitably flawed. Traditional static analysis methods typically rely on manually defining defect patterns. As software and defect complexity increase, the cost and difficulty of manual definition become prohibitively high. Additionally, the rates of false positives and false negatives may undergo significant changes due to subjective differences in human understanding. In dynamic analysis methods, the automatic generation and variation of test cases introduce significant uncertainty. These factors may lead to testing redundancy, unclear testing attack surfaces, difficulties in discovering access control vulnerabilities, and design logic errors. Therefore, although these early defect detection methods have achieved certain results in detecting defects in small software, they often cannot meet the requirements when faced with large, complex software systems and diverse new types of defects.

3.2. Comparison Analysis of Different Software Defect Prediction Methods

Table 2: Comparison Analysis of Different Software Defect Prediction Methods.

Method	Authors	Year	Techniques	Advantages
Classical Machine Learning Technology	N. Khleel et al[22]	2021	A software bug prediction model based on supervised machine learning algorithms	In four publicly available databases from NASA, the model demonstrated higher accuracy and efficiency in identifying potential software defects.
	H. Perl et al[23]	2015	VCCFinder, an approach to improve code audits	Compared to the vulnerability finder Flawfinder, VCCFinder reduces the number of false alarms by over 99% at the same level of recall.
	K. Elish[24]	2008	SVM is a supervised learning algorithm. It can be used for classification and regression problems, and provide relatively accurate prediction results.	To provide better prediction results, different kernel functions are used. Simultaneously reducing computing power requirements.
Deep Learning Technology	A. Mishra et al[25]	2024	A novel deep learning (DL)-based CI/CD software defect prediction technique	Compared with existing methods, the proposed model generates less label noise and waiting time.
	L. Qiao et al[26]	2020	A deep learning-based model for software defect prediction	This method significantly reduces the mean squared error by over 14% and increases the squared correlation coefficient by over 8%.
	S. Wang et al[27]	2016	DBN, an unsupervised probabilistic deep learning algorithm	few restrictions on the labeled dataset.
	L. Qiao[28]	2017	CNN is a deep learning architecture that uses convolution operations in at least one level to replace traditional matrix multiplication.	It can detect important features by default without manual supervision.
Large Language Model Technology	H. Shen et al[29]	2023	A defect detection system based on graph convolutional neural network. It automatically extracts unique features from the AST of the program to explore the semantic and structural information of the code.	The popular deep learning methods on Java projects are not as good as it in terms of AUC and F1 metrics.
	A. Briciu et al[30]	2023	BERT-based language models for the detection of defective source codes	The CodeBERT MLM model trained on source code is more effective in detecting software defects than models that focus on natural language, such as RoBERTa.
	T. Le et al[31]	2024	CodeBERT and ChatGPT technologies for low-resource SV prediction	The effectiveness research of ChatGPT in low-resource SV prediction has great prospects.

The comparison analysis of different software defect prediction methods is shown in Table 2. The earliest paper was published in the early twenty-first century. This indicates that machine learning is a technology with a long history of development. It has received extensive and in-depth research within academia. As an important branch of machine learning, deep learning has rapidly developed and been widely applied over the past decade. Software defect detection technology, with the help of traditional machine learning and its derived deep learning algorithms, has achieved significant improvements in accuracy, efficiency, and utilization of computing resources. These methods provide effective tools for software quality assurance by reducing label noise and waiting time while improving prediction accuracy and enabling automatic feature detection.

Traditional software defect detection techniques, such as static code analysis, dynamic execution analysis, and manual inspection, often require a significant investment of human resources. However, they may not be as efficient when dealing with complex, large-scale software systems. In contrast, deep learning-based detection methods automatically extract features and recognize patterns from vast amounts of data. These techniques are capable of handling more complex defect scenarios and demonstrate superior generalization capabilities. By analyzing historical data from software projects, deep learning models can predict potential future defects. The popularity of research using deep learning for defect detection is continuously increasing [7].

Most selected deep learning and LLMs papers have been published within the past decade. The rise of large-scale language models represented by ChatGPT has brought new research directions to the academic community. Training traditional deep learning models relies on a large volume of labeled data. In contrast, LLMs use unsupervised learning to self-train many unlabeled text data. For example, H. Shen et al. proposed a defect detection framework based on graph convolutional neural networks (GCN)[29]. GCN combines the characteristics of convolutional neural networks and graph neural networks, extends the convolution operations to non-Euclidean space, conducts convolution operations on graphs via Fourier transform principles, aggregates node information with edge information to generate new node representations, and employs the GraphSAGE framework for learning node representations. It integrates the code semantics and descriptive semantic features and, after being processed by GraphSMOTE, constructs a model for prediction to capture the code information comprehensively. This framework outperforms popular deep learning methods in evaluation metrics such as AUC (Area Under the Curve) and F1 score. A. Briciu et al. proposed BERT-based language models to detect defective source code [30]. It adopts feature extraction rather than fine-tuning to obtain code embedding representations when utilizing pre-trained language models. It selects RoBERTa and CodeBERT-MLM, two BERT-based models, to automatically learn source code embeddings. It conducts a comparative study on the importance of features extracted from natural language datasets and code-specific semantic and syntactic patterns pre-trained from source code datasets for software defect prediction. This model demonstrates higher software defect detection efficiency than models focusing on natural language processing. These models are not confined to specific tasks like image recognition or speech processing. They generate coherent text and are suitable for various applications, including chatbots and content creation. There are significant differences between LLMs and traditional deep learning models regarding model structure, training methods, and application areas. The emergence of LLMs has provided us with a new and powerful tool for handling various complex tasks. Therefore, LLMs' research and application prospects in software defect detection are extensive.

4. Conclusion

In summary, the critical role of software in systems has grown. It has become a key factor in cost and risk, especially with the rise of complex systems. Defect prediction in software engineering is essential and continually evolving. Large language models (LLMs) offer a new, more adaptable approach to defect detection. LLMs enhance the accuracy of defect prediction.

We compare traditional software defect detection methods, deep learning-based detection techniques, and detection methods utilizing LLMs. We highlight LLMs' significant impact on software development. The open-source nature of LLMs like ChatGPT facilitates their integration into existing processes, improving development efficiency. Despite their theoretical and practical promise, further research is needed to address emerging challenges in this field.

References

[1] 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, doi: 10.1049/iet-sen.2017.0148.

[2] Chen, Xiang & Wang, L.-P & Gu, Q. & Wang, Z. & Ni, C. & Liu, W.-S & Wang, Q.-P. (2018). A Survey on Cross-Project Software Defect Prediction Methods. Jisuanji Xuebao/Chinese Journal of Computers. 41. 254-274. 10.11897/SP.J.1016.2018.00254.

[3] Z. Li et al., "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection," 25th Annu. Netw. Distrib. Syst. Secur. Symp. NDSS 2018, no. February, 2018, doi: 10.14722/ndss.2018.23158.

[4] I. Conference, "2018 5th International Conference on Systems and Informatics, ICSAI 2018," 2018 5th Int. Conf. Syst. Informatics, ICSAI 2018, no. Icsai, pp. 1225–1230, 2019.

[5] X. Chen, Q. Gu, W. S. Liu, S. L. Liu, and C. Ni, "Survey of static software defect prediction," Ruan Jian Xue Bao/Journal Softw., vol. 27, no. 1, pp. 1–25, 2016, doi: 10.13328/j.cnki.jos.004923.

[6] S. Sivapurnima and D. Manjula, "Adaptive Deep Learning Model for Software Bug Detection and Classification," Comput. Syst. Sci. Eng., vol. 45, no. 2, pp. 1233–1248, 2023, doi: 10.32604/csse.2023.025991.

[7] X. Deng, W. Ye, R. Xie, and S. K. Zhang, "Survey of Source Code Bug Detection Based on Deep Learning," Ruan Jian Xue Bao/Journal Softw., vol. 34, no. 2, pp. 625–654, 2023, doi: 10.13328/j.cnki.jos.006696.

[8] X. Wang, L. Lu, Z. Yang, Q. Tian, and H. Lin, "Parameter-Efficient Multi-classification Software Defect Detection Method Based on Pre-trained LLMs," Int. J. Comput. Intell. Syst., vol. 17, no. 1, 2024, doi: 10.1007/s44196-024-00551-3.

[9] S. B. Hossain et al., "A Deep Dive into Large Language Models for Automated Bug Localization and Repair," Proc. ACM Softw. Eng., vol. 1, no. FSE, pp. 1471–1493, 2024, doi: 10.1145/3660773.

[10] Q. Zhang et al., "A Critical Review of Large Language Model on Software Engineering: An Example from ChatGPT and Automated Program Repair," 2023, [Online]. Available: http://arxiv.org/abs/2310.08879

[11] Y. Wu, Z. Li, J. M. Zhang, M. Papadakis, M. Harman, and Y. Liu, "Large Language Models in Fault Localisation," vol. 1, no. 1, 2023, [Online]. Available: http://arxiv.org/abs/2308.15276

[12] S. Chulani and B. Boehm, "Modeling Software Defect Introduction and Removal: COQUALMO (COnstructive QUALity MOdel)," USC-CSE Tech. Rep., no. August, pp. 99–510, 1999.

[13] M. E. Fagan, "Design and code inspections to reduce errors in program development," IBM Syst. J., vol. 38, no. 2, pp. 258–287, 1999, doi: 10.1147/sj.382.0258.

[14] N. Fenton et al., "Predicting software defects in varying development lifecycles using Bayesian nets," Inf. Softw. Technol., vol. 49, no. 1, pp. 32–43, 2007, doi: 10.1016/j.infsof.2006.09.001.

[15] L. C. Briand, K. El Emam, B. G. Freimut, and O. Laitenberger, "A comprehensive evaluation of capture-recapture models for estimating software defect content," IEEE Trans. Softw. Eng., vol. 26, no. 6, pp. 518–540, 2000, doi: 10.1109/32.852741.

[16] T. M. Khoshgoftaar, X. Yuan, and E. B. Allen, "Balancing misclassification rates in classification-tree models of software quality," Empir. Softw. Eng., vol. 5, no. 4, pp. 313–330, 2000, doi: 10.1023/A:1009896203228.

[17] T. M. Khoshgoftaar, A. Herzberg, and N. Seliya, "Resource oriented selection of rule-based classification models: An empirical case study," Softw. Qual. J., vol. 14, no. 4, pp. 309–338, 2006, doi: 10.1007/s11219-006-0038-1.

[18] P. L. Li, J. Herbsleb, and M. Shaw, "Forecasting field defect rates using a combined time-based and metrics-based approach: A case study of OpenBSD," Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE, vol. 2005, pp. 193–202, 2005, doi: 10.1109/ISSRE.2005.19.

[19] V. R. Basili, L. C. Briand, W. L. Melo, and I. C. Society, "Tse1996-Basili-Validation of Oo Metrics," IEEE Trans. Softw. Eng., vol. 22, no. 10, 1996.

[20] A. L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," IEEE Trans. Reliab., vol. R-28, no. 3, pp. 206–211, 1979, doi: 10.1109/TR.1979.5220566.

[21] K. S. Trivedi, R. M. Geist, and I. Trans, "S-shaped reliability growth modeling for software error de- Decomposition in reliability analysis of fault-tolerant systems . Generalized preventive maintenance policies for a system Addendum to : computing failure frequency via mixed products A multiple ," vol. 48, p. 1984, 1984.

[22] N. A. A. Khleel and K. Nehez, "Comprehensive Study on Machine Learning Techniques for Software Bug Prediction," Int. J. Adv. Comput. Sci. Appl., vol. 12, no. 8, pp. 726–735, 2021, doi: 10.14569/IJACSA.2021.0120884.

[23] H. Perl et al., "VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits," Proc. ACM Conf. Comput. Commun. Secur., vol. 2015-Octob, pp. 426–437, 2015, doi: 10.1145/2810103.2813604.

[24] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," J. Syst. Softw., vol. 81, no. 5, pp. 649–660, 2008, doi: 10.1016/j.jss.2007.07.040.

[25] A. Mishra and A. Sharma, "Deep learning based continuous integration and continuous delivery software defect prediction with effective optimization strategy," Knowledge-Based Syst., vol. 296, no. January, p. 111835, 2024, doi: 10.1016/j.knosys.2024.111835.

[26] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," Neurocomputing, vol. 385, pp. 100–110, 2020, doi: 10.1016/j.neucom.2019.11.067.

[27] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," Proc. - Int. Conf. Softw. Eng., vol. 14-22-May-, pp. 297–308, 2016, doi: 10.1145/2884781.2884804.

[28] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," Proc. - 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2017, pp. 318–328, 2017, doi: 10.1109/QRS.2017.42.

[29] H. Shen, X. Ju, X. Chen, and G. Yang, "EDP-BGCNN: Effective Defect Prediction via BERT-based Graph Convolutional Neural Network," Proc. - Int. Comput. Softw. Appl. Conf., vol. 2023-June, pp. 850–859, 2023, doi: 10.1109/COMPSAC57700.2023.00114.

[30] A. Briciu, G. Czibula, and M. Lupea, "A study on the relevance of semantic features extracted using BERT-based language models for enhancing the performance of software defect classifiers," Procedia Comput. Sci., vol. 225, pp. 1601–1610, 2023, doi: 10.1016/j.procs.2023.10.149.

[31] T. H. M. Le, M. A. Babar, and T. H. Thai, "Software Vulnerability Prediction in Low-Resource Languages: An Empirical Study of CodeBERT and ChatGPT," ACM Int. Conf. Proceeding Ser., pp. 679–685, 2024, doi: 10.1145/3661167.3661281.