Journal of Technology Informatics and Engineering (JTIE)
Vol. 5 No. 2 August 2026
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 167-187
DOI: https://doi.org/10.51903/jtie.v5i2.545

A Systematic Literature Review on Software Testing Prediction Models

Job Onyinkwa Osoro*[1], John Ndia[2]
Email: jobonyi@gmail.com, jndia@mut.ac.ke
Orcid: https://orcid.org/0009-0001-6547-2573, https://orcid.org/0000-0002-9223-8096
[1]Computer Science. School of Computing and Information Technology. Murang'a University of Technology. Murang'a, Kenya. 75-10200
[2]Information Technology. School of Computing and Information Technology. Murang'a University of Technology. Murang'a, Kenya. 75-10200
*Corresponding Author

Abstract
Software testing prediction models play a critical role in improving software quality, reducing faults, and optimizing testing. Although past studies show that these models have improved over time, little focus has been given to them. Many studies rely on limited datasets that do not fully capture the complexity of real software systems, which limits how well the models can generalize. There is also insufficient evidence on how these models use existing datasets in practical, real-life settings, since most evaluations are conducted under controlled or experimental conditions. In addition, key aspects such as interpretability, generalizability, and practical usability are still not adequately addressed, which reduces trust and slows down adoption in practice. This study presents a systematic literature review of recent empirical research on predictive approaches in software testing, focusing on machine learning, deep learning, and hybrid techniques. A structured methodology was used, including clearly defined inclusion and exclusion criteria, systematic searches across major academic databases, quality assessment, and data extraction from 22 selected studies.The analysis considered model types, datasets, feature methods, evaluation metrics, and methodological approaches. The findings show a shift from traditional statistical models to advanced artificial intelligence techniques, including graph neural networks, contrastive learning, and deep fuzzy clustering. In addition, optimization and data balancing techniques improve predictive performance, while explainable artificial intelligence enhances model interpretability. However, challenges such as limited cross-project generalization and insufficient industrial validation still exist.

Keywords: *Software testing prediction, Software defect prediction, Machine learning, deep learning, Explainable AI.*

I. INTRODUCTION

Software plays an important role in modern society by supporting sectors such as healthcare, finance, education, communication, and transportation. As reliance on software systems continues to grow, maintaining software quality and reliability has become increasingly important. However, modern software systems are becoming more complex and are often developed under tight deadlines, increasing the likelihood of defects. When faults are not identified early, they may lead to system failures, financial losses, and reduced user confidence. For this reason, software testing remains a key phase of the software development lifecycle, aimed at detecting and correcting defects before deployment (Choras et al., 2020; Boloori et al., 2024; EskandariNasab et al., 2026).

Traditional software testing mainly depends on manual processes and predefined testing strategies. Although these methods can be effective, they are often time-consuming, costly, and

Received on April 2026; Revised on May 2026; Accepted on June 2026; Published on August 2026

difficult to scale in agile and continuous integration environments. The growing need for faster and more efficient testing has encouraged the adoption of intelligent and automated approaches. One major development is the use of software testing prediction models that help identify defect-prone components before extensive testing is performed (Gurcan et al., 2022; Behera et al., 2025; Aziz et al., 2019).

Software defect prediction (SDP) models use historical software data such as code metrics, change history, and previous defect records to estimate the likelihood of faults in software modules. This helps testers concentrate on high-risk areas and improve testing efficiency. Early studies mainly relied on statistical techniques and traditional machine learning algorithms based on metrics such as software complexity, size, and inheritance relationships (Aziz et al., 2019; Arshad et al., 2018). While these approaches provided useful insights, they had limited ability to capture deeper structural relationships within software systems.

Recent advances in artificial intelligence and machine learning have significantly transformed software-testing prediction. Current studies increasingly apply deep learning, graph neural networks, and transformer-based models that can learn complex structural and semantic patterns from large datasets (Dai et al., 2026; Malhotra & Patidar, 2025; Phung et al., 2025). Researchers have also explored hybrid models, optimization techniques, and feature selection methods to improve prediction accuracy and efficiency (Boloori et al., 2024; Tao et al., 2019; Urbikain-Pelayo et al., 2025).

Despite these developments, several challenges still exist. Data imbalance often causes models to favor non-defective modules, reducing the accuracy of defect detection (Boloori et al., 2024; Khatibsyarbini et al., 2019). Many models also struggle to generalize across different projects and domains (Gong et al., 2020; Tumar et al., 2020). In addition, differences in datasets, evaluation methods, and experimental setups make it difficult to compare findings across studies (Chan & Keung, 2024; De Silva & Hewawasam, 2024). Advanced machine learning models also tend to lack interpretability, which limits trust and practical adoption (Abdulwareth & Al-Shargabi, 2021; Fatima et al., 2024).

To address these limitations, recent studies have focused on explainable AI, domain adaptation, and automated testing frameworks to improve transparency, adaptability, and efficiency (Mehmood et al., 2024; Li & Fang, 2025; Aggarwal et al., 2024). Researchers are also integrating prediction models with reliability analysis and test optimization techniques to strengthen software quality assurance practices (Komee & Pachauri, 2025; Faraji & Pombo, 2025; Wang et al., 2025).

Given the rapid growth of this field and the diversity of existing approaches, there is a need for a systematic literature review (SLR) to synthesize current knowledge, identify research trends, and

highlight existing gaps. Therefore, this study reviews software testing prediction models by examining prediction techniques, datasets and features, evaluation methods, challenges, and future research directions. The review aims to support the development of more accurate, scalable, and interpretable prediction systems for modern software engineering.

Research Questions for this research are:

RQ1. What software testing prediction techniques have been proposed in the literature?

RQ2.What types of data and features are commonly used in software testing prediction models?

RQ3.What evaluation metrics and validation methods are frequently applied?

RQ4.What challenges and limitations affect software testing prediction models?

RQ5.What trends, gaps, and future research directions exist in this field?

The remainder of this paper is organized as follows: Section 2 presents related works, Section 3 describes the methodology, Section 4 presents the results, Section 5 discusses the findings, and Section 6 concludes the study and provides recommendations for future work.

II. RELATED WORK

Several systematic literature reviews have been conducted on software defect prediction, reflecting the growing interest in data-driven approaches for improving software quality. These reviews show how the field has evolved from traditional machine learning techniques to more advanced intelligent methods. Early reviews such as (Wahono, 2015) and (Tosun et al., 2017) focused on foundational machine learning approaches for defect prediction. Their findings showed that these models could effectively identify fault-prone software components and support testing activities. However, they also noted that most studies were carried out in controlled experimental settings, with limited evidence of performance in real industrial environments. This raised concerns about the practical applicability of the findings.

As the field advanced, reviews by (Ali & Gravino, 2019), (Li et al., 2020), and (Hemandez-Molinos et al., 2021) expanded the scope by examining a broader range of machine learning-based software testing techniques. These studies generally reported improvements in prediction accuracy and model performance. Nevertheless, they also identified recurring challenges such as poor-quality datasets, heavy dependence on benchmark datasets, and limited model generalization across software projects. Similarly, (Chengxiao et al., 2023) and (Navarro Cedeño et al., 2023) emphasized that many prediction models still lack adequate validation using real-world industrial data, limiting their practical usefulness.

More recent reviews, including (Olivares-Galindo et al., 2023), (Durrani et al., 2024), (Boukhlif et al., 2024), and (Arani et al., 2024), show a clear shift toward advanced approaches such as deep learning, hybrid models, and explainable artificial intelligence. These studies reported improvements in predictive performance and the use of more sophisticated modeling techniques. However, they consistently highlighted unresolved issues related to interpretability, cross-project generalization, and practical usability. Emerging studies such as (Weng et al., 2026) continue to emphasize the need for models that are more robust, transparent, and suitable for industrial environments.

Overall, the reviewed studies indicate that although significant progress has been made in software testing prediction models, there is still limited understanding of how these models utilize datasets in real operational settings. Most evaluations continue to focus on experimental or benchmark environments rather than real-world deployment. This gap demonstrates the need for a more comprehensive systematic literature review that not only captures recent advancements but also critically examines real-world applicability, dataset usage, and challenges affecting the practical adoption of software testing prediction models.

A. Approaches to Software Testing Prediction Models

Software testing prediction models have evolved from classical statistical techniques to advanced AI-based frameworks. These approaches are broadly grouped into statistical models, machine learning methods, deep learning models, metaheuristic optimization techniques, hybrid approaches, domain-specific frameworks, and data-driven enhancements.

1. Statistical and Reliability Growth Models

Traditional statistical models continue to play an important role in software reliability prediction. Among these, Non-Homogeneous Poisson Process (NHPP) models are widely used to estimate software failure rates and measure reliability growth during the testing process. These models also consider important aspects such as testing effort and software change-points, making them useful for evaluating software quality and predicting system performance over time (Aggarwal et al., 2024; Kim et al., 2023). However, they are less effective with complex, high-dimensional datasets.

2. Machine Learning-Based Approaches

Machine learning methods are widely used in defect prediction. Supervised algorithms such as decision trees, random forests, support vector machines, and ensemble models detect fault-prone modules using code metrics, defect history, and process data (Aziz et al., 2019; Mehmood et al.,

2024). More recent methods, including supervised contrastive learning with CodeBERT, improve semantic understanding of source code and boost prediction accuracy.

3. *Deep Learning and Neural Network Models*

Deep learning approaches such as graph neural networks (GNNs), convolutional neural networks (CNNs), and recurrent neural networks enable deeper analysis of software artifacts. GNN-based models can jointly predict defects and evaluate code quality by capturing structural and semantic relationships in codebases (Dai et al., 2026). Semi-supervised deep fuzzy clustering is also used to detect hidden fault patterns in limited-label datasets (Arshad et al., 2018).

4. *Metaheuristic and Optimization-Based Methods*

Metaheuristic algorithms inspired by natural processes are applied to improve prediction and testing efficiency. Methods such as firefly optimization, binary moth-flame optimization, and simulated annealing are used for feature selection, test case prioritization, and reliability improvement (Komee & Pachauri, 2025). These techniques help focus testing on critical components and reduce computational cost.

5. *Hybrid and AI-Driven Testing Frameworks*

Hybrid approaches combine statistical methods, machine learning, and AI techniques. AI-based testing frameworks improve test coverage, detect flaky tests, and support decision-making (Gong et al., 2020; Abdulwareth & Al-Shargabi, 2021). Large language models are also used for predicting flaky test fixes and automating test code repair, linking defect prediction with test automation.

6. *Domain-Specific and Emerging Approaches*

Some models are designed for specific domains such as serverless systems, blockchain applications, and quantum software (Gordieiev et al., 2024; Li & Fang, 2025). These frameworks integrate domain-specific metrics and environmental factors to improve prediction accuracy. Emerging methods such as explainable AI and counterfactual analysis are also used to improve interpretability and developer trust (Zou et al., 2025).

7. *Data-Driven Enhancements and Feature Engineering*

Across all approaches, data quality and feature selection are critical. Techniques such as dataset balancing, feature optimization and error-type integration improve model performance. Feature selection methods, including AI-based and metaheuristic techniques, reduce dimensionality while maintaining predictive accuracy (Boloori et al., 2024).

III. METHOD

This study uses a Systematic Literature Review (SLR) to examine software testing prediction models in a structured and reliable way. An SLR is widely used because it follows a clear process that reduces bias and improves the credibility of findings (Behera et al., 2025; Faraji & Pombo, 2025). The study follows the PRISMA framework, which provides standard guidelines for conducting and reporting systematic reviews. PRISMA structures the review into four stages: identification, screening, eligibility, and inclusion. In the identification stage, studies are gathered from academic databases using keywords related to software testing and defect prediction. In the screening stage, duplicates and irrelevant studies are removed based on titles and abstracts. The remaining studies are then assessed in detail against predefined criteria to ensure only relevant and high-quality papers are included (Mehmood et al., 2024; Boloori et al., 2024).

Using PRISMA improves transparency because each step is clearly documented and justified, making the process easier to replicate. It also strengthens reliability by ensuring study selection is systematic rather than subjective (Chan & Keung, 2024). Beyond selection, the study also includes structured data extraction and quality assessment. Key information such as prediction techniques, datasets, feature types, and evaluation methods is collected from each study. The quality of studies is assessed based on clarity of methods, validity of results, and suitability of evaluation approaches (Aziz et al., 2019; Aggarwal et al., 2024). Overall, combining SLR with PRISMA ensures a comprehensive, transparent, and methodologically strong review, providing a solid basis for identifying trends, comparing techniques, and highlighting research gaps in software testing prediction models.

A. Research Design

This study adopts a Systematic Literature Review (SLR) approach to explore and analyze software testing prediction models. The SLR method was selected because it offers a clear, structured, and transparent way of identifying, evaluating, and synthesizing existing research (Behera et al., 2025; Faraji & Pombo, 2025). To further strengthen the rigor of the review, the study follows the PRISMA guidelines, which help ensure a systematic and well-documented selection process. The main goal is to gather reliable insights into the techniques, data sources, and evaluation methods used in predicting software testing outcomes. By combining a systematic, approach with PRISMA guidelines, the study reduces the risk of bias and ensures that the findings are both comprehensive and reproducible (Mehmood et al., 2024; Chan & Keung, 2024).

B. Data Sources and Search Strategy

To ensure a comprehensive and representative coverage of relevant studies, several well-established academic databases were searched, including IEEE Xplore, ScienceDirect,

SpringerLink, the ACM Digital Library, and Google Scholar, all widely recognized for high-quality research in software engineering. Using multiple sources helped reduce the risk of missing key studies and improved the completeness of the review (Behera et al., 2025; Faraji & Pombo, 2025). A structured search strategy was applied using keywords such as "software testing prediction models," "defect prediction," "machine learning in software testing," "software quality prediction," and "test case prediction." These terms were combined using Boolean operators (AND, OR) to refine and improve relevance. The search was also restricted to English-language publications within a defined time range to capture recent developments and current trends in the field (Mehmood et al., 2024; Boloori et al., 2024).

C. *Inclusion and Exclusion Criteria*

1. Inclusion Criteria

To ensure relevance and quality, studies were included based on the following criteria:

First, only studies that explicitly address software testing prediction or software defect prediction were considered. This includes research that proposes, evaluates, or improves predictive models. Examples include deep learning-based approaches (Sahar et al., 2024), graph neural network models (Dai et al., 2026), and clustering-based prediction techniques (Arshad et al., 2018; Kim et al., 2023). Second, empirical studies with clear experimental validation were included. Selected papers were required to present datasets, model implementation details, and evaluation results. Studies such as (Aziz et al., 2019) and (Boloori et al., 2024) were included due to their strong empirical validation.

Third, studies employing machine learning, deep learning, or hybrid approaches were prioritized. This includes supervised, unsupervised, and semi-supervised methods such as domain adaptation (Gong et al., 2020), contrastive learning (Sahar et al., 2024), and advanced language models (Malhotra & Patidar, 2025). Fourth, studies that incorporate feature engineering, optimisation, or data balancing techniques were included. These contributions are essential for improving prediction performance, as demonstrated in (Tumar et al., 2020) and (Malhotra & Patidar, 2025).

Fifth, studies addressing explainability and interpretability in prediction models were included. Research such as (Zou et al., 2025), which focuses on counterfactual explanations, was considered important for improving trust and adoption of predictive models. Sixth, only recent studies within the selected period (2015–2026) were included to ensure relevance to current trends, including AI-driven testing and automation (Faraji & Pombo, 2025; Fatima et al., 2024) and intelligent reliability prediction (Behera et al., 2025). Finally, only peer-reviewed journal and conference publications were included to ensure academic quality and credibility.

2. Exclusion Criteria

To maintain focus and methodological rigor, several types of studies were excluded. First, studies not directly related to software testing prediction or defect prediction were removed. This includes work on general testing frameworks, process improvement, or tool selection without predictive modelling (Abdulwareth & Al-Shargabi, 2021; Choras et al., 2020). Second, studies outside the software domain were excluded, such as prediction models applied to unrelated fields like solar flare prediction (EskandariNasab et al., 2026) or manufacturing systems (Urbikain-Pelayo et al., 2025).

Third, non-empirical studies were excluded, including papers without experiments, datasets, or measurable results. While surveys such as (Behera et al., 2025) and (Faraji & Pombo, 2025) were useful for background, only those contributing empirical insights were considered. Fourth, duplicate or overlapping studies were removed, with only the most complete versions retained when multiple versions existed (Arshad et al., 2018).

Fifth, studies lacking sufficient methodological detail were excluded, particularly those that did not clearly describe datasets, models, or evaluation metrics, making comparison or replication difficult. For example, general discussions without predictive evaluation (De Silva & Hewawasam, 2024) were not included. Sixth, studies focusing only on optimisation or testing techniques without a prediction component, such as test case prioritization (Khatibsyarbini et al., 2019) or test suite optimisation (Mehmood et al., 2024), were excluded unless directly linked to prediction modelling. Finally, studies outside the defined time range or not aligned with modern machine learning approaches were excluded to ensure the review reflects current developments in the field.

D. Study Selection Process

The study selection process was carried out in a systematic and transparent way to ensure that only relevant and high-quality studies were included in this review. The study selection process for this systematic literature review followed a structured PRISMA approach to ensure transparency, traceability, and rigor in identifying relevant literature on software testing prediction models. A total of 520 records were initially identified from major academic databases, including IEEE Xplore, SpringerLink, ACM Digital Library, ScienceDirect, and Google Scholar. This wide search ensured that all potentially relevant studies were captured across different publication platforms.

However, because many studies were indexed in more than one database, a significant number of 210 duplicate records were identified and removed during the data cleaning stage. This resulted

in 310 unique records that proceeded to the screening phase. During the screening phase, titles and abstracts were carefully reviewed to assess relevance to the research topic. At this stage, 276 records were excluded because they did not focus on software testing prediction models, lacked empirical evaluation, or were outside the scope of software defect prediction, reliability modeling, or machine learning-based testing approaches.

As a result, 34 reports were considered potentially relevant and were sought for full-text retrieval. Out of these, 6 reports could not be retrieved due to restricted access, unavailable full-text versions, or limitations in institutional subscriptions. Consequently, 28 full-text reports were successfully retrieved and assessed for eligibility. Following a detailed full-text review, 22 studies were included in the final synthesis. The remaining 6 studies were excluded because they did not fully meet the inclusion criteria, such as insufficient methodological detail, limited relevance to prediction models, or overlap with already included studies.

E. *Data Extraction and Analysis*

After applying the inclusion and exclusion criteria, relevant data were systematically extracted from each selected study to ensure consistency and comparability across the literature. A structured approach was used to capture key information, including the type of prediction model, techniques employed (such as machine learning, deep learning, or hybrid approaches), dataset characteristics, feature types, preprocessing methods, and evaluation metrics. For instance, studies utilizing advanced techniques such as graph neural networks (Dai et al., 2026) and contrastive learning approaches (Sahar et al., 2024) were carefully examined alongside those incorporating optimization strategies like metaheuristic-based feature selection (Tumar et al., 2020).

In addition, important findings and identified limitations were recorded to support critical analysis. The extracted data were then analyzed using a comparative and thematic approach, where studies were categorized based on their modelling techniques and evaluated using common performance metrics such as Precision, Recall, F1-score, and AUC. Particular attention was given to the impact of feature engineering, dataset quality, and model generalization, including approaches such as domain adaptation (Gong et al., 2020) and data balancing techniques (Malhotra & Patidar, 2025). Furthermore, the analysis explored emerging trends in explainable artificial intelligence, with studies such as (Zou et al., 2025) highlighting the importance of model interpretability in practical adoption. Through this process, key patterns, strengths, and research gaps were identified, including issues related to class imbalance, lack of standardized evaluation, and limited cross-project validation. This systematic data extraction and analysis process ensured

a comprehensive understanding of existing software testing prediction models and provided a strong foundation for the development of an improved predictive framework.

F. *Synthesis of Findings*

The synthesis of findings from the selected studies reveals a clear evolution in software testing prediction models, moving from traditional statistical approaches to more advanced machine learning and deep learning techniques. The majority of studies demonstrate that machine-learning models, particularly ensemble methods, consistently outperform classical statistical models in predicting software defects. More recent approaches, such as deep learning models and graph-based techniques (Dai et al., 2026), as well as contrastive learning methods (Sahar et al., 2024), show promising improvements in capturing complex patterns within software data. Additionally, optimization techniques and data balancing strategies (Malhotra & Patidar, 2025; Tumar et al., 2020) have been found to significantly enhance model performance, especially in addressing class imbalance issues commonly present in defect datasets.

Despite these advancements, several challenges persist across the literature. Many studies rely heavily on publicly available datasets, limiting the generalizability of results to real-world environments. Furthermore, cross-project prediction remains a major limitation, with models often performing poorly when applied to different datasets or domains. Another key observation is the growing emphasis on explainable artificial intelligence, where approaches such as counterfactual explanations (Zou et al., 2025) aim to improve model transparency and user trust. However, the integration of explain ability into practical software testing environments is still at an early stage. Overall, the synthesis highlights that while significant progress has been made in improving predictive accuracy, there is still a need for more generalizable, interpretable, and context-aware models, particularly in underrepresented environments such as emerging software ecosystems

G. *Quality Assessment*

To ensure the reliability and credibility of the findings, a quality assessment process was conducted on all selected studies included in this systematic literature review. Each study was evaluated based on predefined criteria, including clarity of research objectives, adequacy of the experimental design, quality of the dataset used, and appropriateness of the evaluation metrics. Studies that clearly described their methodology, including data preprocessing, model development, and validation procedures, were rated more highly. Particular attention was given to whether the studies employed robust evaluation techniques such as cross-validation and whether they used standard performance metrics such as Precision, Recall, F1-score, and AUC.

Journal of Technology Informatics and Engineering (JTIE)
Vol. 5 No. 2 August 2026
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 167-187
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

Additionally, the reproducibility of results and transparency of the research process were considered important indicators of quality. For instance, studies incorporating advanced models such as graph neural networks (Dai et al., 2026) and explainable approaches (Zou et al., 2025) were assessed not only on performance but also on how well they justified their design choices and reported their findings. Conversely, studies lacking sufficient methodological detail, clear experimental validation or proper evaluation were considered lower in quality and were either excluded or given less weight in the synthesis. This quality assessment process ensured that the review was based on robust and credible evidence, thereby strengthening the validity of the conclusions drawn.

Table 1: Quality assessment process

Assessment Criterion	Description	Quality Indicator
Clarity of Research Objectives	Whether the study clearly explained what it wanted to achieve	Studies with clear and well-focused objectives were considered stronger
Experimental Design Adequacy	How well the study was planned and structured.	Well-organized and properly designed studies were rated highly
Dataset Quality	The relevance, reliability, and clarity of the data used.	Studies using high-quality and well-explained datasets were preferred
Evaluation Metrics Used	Use of standard metrics such as Precision, Recall, F1-score, and AUC.	Studies using standard and appropriate metrics were preferred.
Methodological Transparency	Clear description of preprocessing, model development, and validation steps.	Detailed and transparent methodologies were rated higher.
Validation Techniques	Use of robust evaluation methods such as cross-validation.	Studies using strong validation techniques were considered more reliable
Reproducibility of Results	Whether results could be replicated based on provided details	High reproducibility increased study credibility.
Advanced Model Justification	Evaluation of studies using advanced models (e.g., GNNs, explainable AI) and how well they justified design choices.	Well-justified advanced methods improved quality rating.
Reporting Quality	Clarity and completeness of how findings were presented.	Clear and well-reported studies were rated higher.
Exclusion/Weighting Decision	Studies with weak methodology or insufficient detail.	Lower-quality studies were excluded or given less weight

Source: Developed by the author based on an adapted synthesis of SLR methodology standards (Kitchenham & Charters, 2007; Petersen et al., 2021; Wohlin et al., 2022).

H. Ethical Considerations

Ethical considerations were carefully observed throughout this systematic literature review to ensure integrity, transparency, and respect for intellectual property. All sources included in the study were properly acknowledged through accurate citation and referencing, thereby avoiding

plagiarism and giving due credit to original authors. The review relied solely on publicly available and peer-reviewed literature, ensuring that no confidential or sensitive data were accessed or misused. In addition, care was taken to present findings objectively, without misinterpretation or bias, and to accurately reflect the contributions and limitations of each study. The inclusion and exclusion criteria were applied consistently to avoid selective reporting and ensure fairness in the selection process. Furthermore, since this study does not involve human participants or primary data collection, issues related to consent and privacy were not applicable. However, the ethical use of data and responsible reporting of results remain central to maintaining academic integrity and credibility in this research.

IV. RESULT

A. RQ1: What software testing prediction techniques have been proposed in the literature?

The reviewed studies show that software testing prediction techniques have evolved from statistical models to advanced AI-based methods. These approaches are grouped into statistical and reliability models, machine learning, deep learning, metaheuristic optimization, hybrid frameworks, and domain-specific models.

1. Machine Learning-Based Techniques

Machine learning is widely used for defect prediction. Methods such as decision trees, support vector machines (SVM), random forests, and ensemble models classify software modules as fault-prone or not using historical data and software metrics (Aziz et al., 2019; Mehmood et al., 2024). Recent improvements include feature engineering and data balancing to handle imbalanced datasets. Some studies also integrate domain-specific metrics like error-type indicators to improve risk prediction. Unsupervised and semi-supervised methods are also used when labeled data is limited. For example, clustering methods such as deep fuzzy c-means detect hidden fault patterns without labeled data (Arshad et al., 2018), while metamorphic testing supports unsupervised validation (Chan & Keung, 2024).

2. Deep Learning and Representation Learning

Deep learning models capture complex relationships in software data. Graph neural networks (GNNs) model code structure and support both defect prediction and code quality analysis (Fatima et al., 2024). Transformer-based models and embeddings like CodeBERT improve semantic understanding of code, leading to better defect detection. Contrastive learning is also used to better separate faulty and non-faulty code.

3. Metaheuristic and Optimization-Based Techniques

Metaheuristic methods such as firefly algorithms, binary moth-flame optimization, and simulated annealing are used for feature selection, parameter tuning, and test prioritization (Komee & Pachauri, 2025). These methods improve performance by selecting relevant features and reducing computational cost.

4. *Hybrid and AI-Driven Techniques*

Hybrid models combine statistical, machine learning, and optimization methods to improve accuracy and robustness. AI-driven testing frameworks support defect prediction, test optimization, and coverage improvement (Gong et al., 2020; Phung et al., 2025). Large language models are also used for flaky test prediction and test code repair, improving automation.

5. *Domain Adaptation and Transfer Learning*

Domain adaptation methods help models generalize across projects. Techniques such as conditional domain adversarial adaptation transfer knowledge between projects, improving cross-project prediction (Sahar et al., 2024).

6. *Explainable and Interpretable Techniques*

Explainable AI methods such as counterfactual explanations improve transparency and trust by making predictions easier to understand (Zou et al., 2025).

7. *Domain-Specific and Emerging Techniques*

Recent models are tailored for domains such as serverless systems, blockchain, and quantum software (Gordieiev et al., 2024; Li & Fang, 2025). These use specialized metrics to improve accuracy in specific environments. However, AI testing systems also introduce challenges such as non-deterministic behavior. Overall, techniques have progressed from statistical models (NHPP, Weibull) to machine learning, deep learning, and AI-driven systems. Metaheuristics improve optimization, while hybrid and explainable models enhance automation and transparency. However, issues such as data quality, scalability, and practical deployment remain.

B. RQ2: What types of data and features are used in software testing prediction models?

The reviewed studies show that software testing prediction techniques have evolved from statistical models to advanced AI-based methods. These approaches are grouped into statistical and reliability models, machine learning, deep learning, metaheuristic optimization, hybrid frameworks, and domain-specific models. Prediction models use diverse data sources grouped into code-related, process-related, historical defect, testing, and domain-specific data (Boloori et al., 2024). Code-related data includes metrics such as complexity, size, coupling, cohesion, and inheritance, which help identify fault-prone modules (Aziz et al., 2019).

Process-related data includes code churn, commit history, and developer activity, reflecting development behavior and helping improve prediction accuracy. Historical defect data such as bug reports and defect logs is essential for supervised learning and reliability models (Mehmood et al., 2024; Aggarwal et al., 2024). Testing data includes test execution results, coverage, and test logs, used for test prioritization and defect detection (Khatibsyarbini et al., 2019). Semantic features from deep learning models (e.g., embeddings and graph representations) capture code meaning and structure, improving prediction performance (Fatima et al., 2024).Feature engineering techniques such as data balancing, normalization, and feature selection improve performance by reducing noise and imbalance (Zou et al., 2025). Domain-specific data is used in specialized systems like blockchain and serverless applications, improving accuracy but reducing generalizability (Gordieiev et al., 2024; Li & Fang, 2025).Overall, models combine structural, historical, process, testing, and semantic data. Despite improvements, challenges such as data quality and imbalance remain.

C. RQ3: What evaluation metrics and validation methods are commonly applied?

1. Evaluation Metrics

Common metrics include accuracy, precision, recall, and F1-score. However, accuracy is less reliable for imbalanced datasets. Precision and recall measure correctness and detection ability, while F1-score balances both (Mehmood et al., 2024). AUC-ROC is widely used because it handles imbalance better (Aziz et al., 2019).For reliability models, MSE and RMSE measure prediction error (Aggarwal et al., 2024). In testing optimization, metrics such as coverage, fault detection rate, and cost reduction are used (Komee & Pachauri, 2025). Some studies also use interpretability metrics for explainable AI models.

2. Validation Methods

K-fold cross-validation is widely used to improve reliability by repeatedly training and testing models on different data splits (Mehmood et al., 2024), while holdout validation is also common but depends on a single train–test split. Cross-project validation is used to assess how well models generalize across different software systems (Sahar et al., 2024), and time-based validation is applied when working with software evolution data. For unsupervised models, metamorphic testing is used when labeled data is limited (Chan & Keung, 2024). Overall, evaluation relies on a mix of classification, regression, and testing-specific metrics, but comparison across studies remains difficult due to a lack of standardization.

D. RQ4: What are the key challenges and limitations?

A key challenge is poor data quality, including noise, missing values, and inconsistency, which lowers prediction accuracy (Abdulwareth & Al-Shargabi, 2021; Aggarwal et al., 2024). Class imbalance is another issue, where defective cases are fewer than non-defective ones, leading to biased results (Aziz et al., 2019). Generalization is also limited, as many models perform well on one project but fail on others (Arshad et al., 2018; Boloori et al., 2024). Feature selection remains difficult due to irrelevant or missing important features, which affects performance (Choras et al., 2020). Deep learning models also face interpretability issues, making predictions harder to trust (Dai et al., 2026). In addition, inconsistent evaluation practices across studies, including different datasets and metrics, make comparison difficult (Boloori et al., 2024). High computational cost also limits the scalability of complex models (Chan & Keung, 2024). Many studies rely mainly on experimental datasets, with limited real-world validation (Choras et al., 2020). Finally, the lack of a standard benchmarking framework leads to inconsistent evaluation across studies.

E. RQ5: What trends, gaps, and future directions exist?

The field has progressed from statistical models to machine learning, then to deep learning and AI-driven systems. Early approaches relied on basic metrics such as code size and complexity (Aziz et al., 2019), but they struggled with scalability and generalization. From 2020, hybrid and ensemble models became more common, improving accuracy but reducing interpretability (Choras et al., 2020). Later research focused more on real-world use and scalability (EskandariNasab et al., 2026).

Recent advances in software testing prediction models show a clear shift toward deep learning, graph neural networks, and transformer-based approaches, which are better at capturing both semantic and structural information in source code (Fatima et al., 2024). At the same time, there is increasing interest in domain-specific solutions tailored for emerging areas such as blockchain systems, serverless environments, and even quantum software, alongside growing efforts to make models more explainable and trustworthy (Li & Fang, 2025). Despite these improvements, important challenges still persist, including poor generalization across projects, lack of standardized datasets, data imbalance issues, and high computational costs that limit real-world deployment (Mehmood et al., 2024; Tumar et al., 2020). Overall, while the field has made significant progress, future work should focus on developing lightweight and scalable models, improving cross-project learning, strengthening explainability, and better integrating prediction models into real-world development pipelines such as CI/CD and automated testing.

V. DISCUSSION

The review of software testing prediction models shows a clear evolution from statistical and basic machine learning methods to intelligent, automated, and context-aware systems. Compared to earlier SLRs, which mainly focused on listing techniques or reporting performance improvements, this study provides a more structured synthesis across techniques, data, evaluation, challenges, and emerging trends, while also highlighting gaps in real-world adoption. Although predictive performance has improved over time, practical deployment in industry remains limited, indicating a persistent gap between research and application.

In terms of techniques, earlier studies relied on decision trees, support vector machines, and Bayesian networks, while later work improved performance using ensemble learning, feature selection, and deep learning. Hybrid models became common due to better accuracy and robustness. More recent approaches such as graph neural networks, contrastive learning, and large language models capture deeper structural and semantic relationships in software. Unlike many previous reviews that treat these methods separately, this study shows their progression and integration into CI/CD and modern development pipelines.

The data and features used have also become more diverse. Early work focused on structured metrics like code size, complexity, inheritance, and change history. Newer studies include process metrics, developer activity, and unstructured data such as bug reports and code comments, supported by natural language processing. However, data quality issues such as noise, missing values, and inconsistency remain a key limitation, reinforcing the importance of preprocessing and feature engineering.

For evaluation, studies commonly use accuracy, precision, recall, and F1-score, with AUC and error-based metrics applied in imbalanced cases. Cross-validation and holdout methods are widely used, but differences in datasets, metrics, and validation strategies make comparisons across studies difficult. This review highlights this inconsistency more explicitly than earlier SLRs, which often reported metrics without critically comparing evaluation practices. Several challenges remain consistent across the literature. These include poor data quality, class imbalance, and limited generalization across projects, and low interpretability of complex models. High computational cost also limits practical use in real environments. While earlier reviews often listed these issues separately, this study emphasizes their combined effect on limiting real-world adoption.

Current trends show a shift toward adaptive and intelligent systems, including AI-driven testing, self-learning models, and continuously updating systems. Explainable AI and counterfactual methods are gaining attention to improve transparency and trust. Unlike previous SLRs, this

review also emphasizes the need for standard datasets, benchmarking frameworks, and consistent evaluation guidelines to support reproducibility. It further highlights integration into real development workflows as a key requirement for practical impact.

VI. CONCLUSION AND FUTURE WORK

This review shows that software testing prediction models have evolved from simple statistical and traditional machine learning techniques to more advanced approaches such as ensemble learning, deep learning, and adaptive systems. Early models mainly relied on basic software metrics like code complexity, size, and change history, while recent methods focus on handling more complex data and improving prediction accuracy. Although these developments have improved performance, many models still face challenges when applied in real-world and cross-project environments.

Future research should focus on making these models more practical and reliable by improving generalization across projects and addressing issues such as data imbalance and noise. There is also a need for standardized datasets and evaluation methods to ensure fair comparison between approaches. In addition, future work should aim to develop lightweight and efficient models that can be integrated into real-time systems like CI/CD pipelines, while also improving explainability so that developers can better understand and trust model predictions.

REFERENCES

Abdulwareth, A. J., & Al-Shargabi, A. A. (2021). Toward a Multi-Criteria Framework for Selecting Software Testing Tools. *IEEE Access*, *9*, 158872–158891. https://doi.org/10.1109/ACCESS.2021.3128071

Aggarwal, A., Kumar, S., & Gupta, R. (2024). Testing coverage based NHPP software reliability growth modeling with testing effort and change-point. *International Journal of System Assurance Engineering and Management*, *15*(11), 5157–5166. https://doi.org/10.1007/s13198-024-02504-7

Ali, A., & Gravino, C. (2019). A systematic literature review of software effort prediction using machine-learning methods. *Journal of Software: Evolution and Process, 31*(10), e2211. https://doi.org/10.1002/smr.2211

Arani, A. K., Le, T. H. M., Zahedi, M., & Babar, M. A. (2024). Systematic Literature Review on Application of Learning-Based Approaches in Continuous Integration. *IEEE Access*, *12*, 135419–135450. https://doi.org/10.1109/ACCESS.2024.3424276

Arshad, A., Riaz, S., Jiao, L., & Murthy, A. (2018). The Empirical Study of Semi-Supervised Deep Fuzzy C-Mean Clustering for Software Fault Prediction. *IEEE Access*, *6*, 47047–47061. https://doi.org/10.1109/ACCESS.2018.2866082

Aziz, S. R., Khan, T. A., & Nadeem, A. (2019). Experimental Validation of Inheritance Metrics' Impact on Software Fault Prediction. *IEEE Access*, *7*, 85262–85275. https://doi.org/10.1109/ACCESS.2019.2924040

Behera, A. K., Chaudhury, P., & Dash, Ch. S. K. (2025). A comprehensive survey on intelligent software reliability prediction. *Discover Computing*, *28*(1), 90. https://doi.org/10.1007/s10791-025-09597-z

Boloori, A., Zamanifar, A., & Farhadi, A. (2024). Enhancing software defect prediction models using metaheuristics with a learning to rank approach. *Discover Data*, *2*(1), 11. https://doi.org/10.1007/s44248-024-00016-0

Boukhlif, M., Hanine, M., Kharmoum, N., Ruigómez Noriega, A., García Obeso, D., & Ashraf, I. (2024). Natural Language Processing-Based Software Testing: A Systematic Literature Review. *IEEE Access*, *12*, 79383–79400. https://doi.org/10.1109/ACCESS.2024.3407753

Chan, P. Y. P., & Keung, J. (2024). Validating Unsupervised Machine Learning Techniques for Software Defect Prediction with Generic Metamorphic Testing. *IEEE Access*, *12*, 165155–165172. https://doi.org/10.1109/ACCESS.2024.3494044

Chengxiao, Y., Owais, J., Ahmed, A., Khan, M. O., Xiaoyang, Z., & Tunio, M. H. (2023). Software Defect Prediction Using Machine Learning—A Systematic Literature Review. *2023 20th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 1–9. https://doi.org/10.1109/ICCWAMTIP60502.2023.10387043

Choras, M., Springer, T., Kozik, R., Lopez, L., Martinez-Fernandez, S., Ram, P., Rodriguez, P., & Franch, X. (2020). Measuring and Improving Agile Processes in a Small-Size Software Development Company. *IEEE Access*, *8*, 78452–78466. https://doi.org/10.1109/ACCESS.2020.2990117

Dai, P., Zhu, H., Wu, J., & He, H. (2025). An integrated graph neural network model for joint software defect prediction and code quality assessment. *Scientific Reports*, *16*(1), 1677. https://doi.org/10.1038/s41598-025-31209-5

De Silva, D., & Hewawasam, L. (2024). The Impact of Software Testing on Serverless Applications. *IEEE Access*, *12*, 51086–51099. https://doi.org/10.1109/ACCESS.2024.3384459

Durrani, U. K., Akpinar, M., Fatih Adak, M., Talha Kabakus, A., Maruf Öztürk, M., & Saleh, M. (2024). A Decade of Progress: A Systematic Literature Review on the Integration of AI in Software Engineering Phases and Activities (2013-2023). *IEEE Access*, *12*, 171185–171204. https://doi.org/10.1109/ACCESS.2024.3488904

EskandariNasab, M., Hamdi, S. M., & Filali Boubrahimi, S. (2026). FlaPLeT: A full-stack web platform for end-to-end time series data processing and machine learning in solar flare prediction. *SoftwareX*, *33*, 102540. https://doi.org/10.1016/j.softx.2026.102540

Faraji, A., & Pombo, N. (2025). AI-Driven Software Test Automation: An AI4SE-Oriented Survey of Techniques, Tools, and Challenges. *IEEE Access*, *13*, 183296–183313. https://doi.org/10.1109/ACCESS.2025.3623944

Fatima, S., Hemmati, H., & C. Briand, L. (2024). FlakyFix: Using Large Language Models for Predicting Flaky Test Fix Categories and Test Code Repair. *IEEE Transactions on Software Engineering*, *50*(12), 3146–3171. https://doi.org/10.1109/TSE.2024.3472476

Gong, L., Jiang, S., & Jiang, L. (2020). Conditional Domain Adversarial Adaptation for Heterogeneous Defect Prediction. *IEEE Access*, *8*, 150738–150749. https://doi.org/10.1109/ACCESS.2020.3017101

Gordieiev, O., Rainer, A., Kharchenko, V., Pishchukhina, O., & Gordieieva, D. (2024). A Unified Approach to the Development of Technology-Based Software Quality Models on the Example of Blockchain Systems. *IEEE Access*, *12*, 118875–118889. https://doi.org/10.1109/ACCESS.2024.344827

Gurcan, F., Dalveren, G. G. M., Cagiltay, N. E., Roman, D., & Soylu, A. (2022). Evolution of Software Testing Strategies and Trends: Semantic Content Analysis of Software Research Corpus of the Last 40 Years. *IEEE Access*, *10*, 106093–106109. https://doi.org/10.1109/ACCESS.2022.3211949

Haddaway, N. R., Page, M. J., Pritchard, C. C., & McGuinness, L. A. (2022). PRISMA2020: An R package and Shiny app for producing PRISMA 2020-compliant flow diagrams, with interactivity for optimised digital transparency and Open Synthesis Campbell Systematic Reviews, 18, e1230. https://doi.org/10.1002/cl2.1230

Hemandez-Molinos, Ma. J., Sanchez-Garcia, A. J., & Barrientos-Martinez, R. E. (2021). Classification Algorithms for Software Defect Prediction: A Systematic Literature Review. *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 189–196. https://doi.org/10.1109/CONISOFT52520.2021.00034

Khatibsyarbini, M., Isa, M. A., Jawawi, D. N. A., Hamed, H. N. A., & Mohamed Suffian, M. D. (2019). Test Case Prioritization Using Firefly Algorithm for Software Testing. *IEEE Access*, *7*, 132360–132373. https://doi.org/10.1109/ACCESS.2019.2940620

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (EBSE Technical Report EBSE-2007-01). School of Computer Science and Mathematics, Keele University and Department of Computer Science, University of Durham

Kim, Y. S., Song, K. Y., & Chang, I. H. (2023). Prediction and Comparative Analysis of Software Reliability Model Based on NHPP and Deep Learning. *Applied Sciences*, *13*(11), 6730. https://doi.org/10.3390/app13116730

Komee, & Pachauri, B. (2025). Cost analysis of enhanced software reliability model with Weibull testing effort function using simulated annealing. *Scientific Reports*, *15*(1), 41927. https://doi.org/10.1038/s41598-025-25841-4

Li, N., Shepperd, M., & Guo, Y. (2020). *A Systematic Review of Unsupervised Learning Techniques for Software Defect Prediction* (arXiv:1907.12027). arXiv. https://doi.org/10.48550/arXiv.1907.12027

Li, W., & Fang, C.-C. (2025). Enhanced Software Testing Model Under Software Warranty Policy Considering Debugger Learning and Time Postponement Factors. *IEEE Access*, *13*, 85116–85130. https://doi.org/10.1109/ACCESS.2025.3569201

Malhotra, R., & Patidar, J. (2025). Enhanced software change proneness prediction in android applications using balanced data techniques and advanced language models. *Discover Computing*, *28*(1), 51. https://doi.org/10.1007/s10791-025-09552-y

Mehmood, A., Ilyas, Q. M., Ahmad, M., & Shi, Z. (2024). Test Suite Optimization Using Machine Learning Techniques: A Comprehensive Study. *IEEE Access*, *12*, 168645–168671. https://doi.org/10.1109/ACCESS.2024.3490453

Navarro Cedeño, G. O., Moya, K. C., Dormond, A. S., González-Torres, A., & Rojas-Hernández, Y. (2023). Systematic Literature Review: Machine Learning for Software Fault Prediction. *2023 IEEE 41st Central America and Panama Convention (CONCAPAN XLI)*, 1–6. https://doi.org/10.1109/CONCAPANXLI59599.2023.10517566

Olivares-Galindo, J. A., Sánchez-García, Á. J., Barrientos-Martínez, R. E., & Ocharán-Hernández, J. O. (2023). Ensemble Classifiers in Software Defect Prediction: A Systematic Literature Review. *2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 1–8. https://doi.org/10.1109/CONISOFT58849.2023.00011

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2021). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology, 133*, 106455. https://doi.org/10.1016/j.infsof.2020.106455

Phung, K., Ogunshile, E., & Aydin, M. E. (2025). Domain-specific implications of error-type metrics in risk-based software fault prediction. *Software Quality Journal*, *33*(1), 7. https://doi.org/10.1007/s11219-024-09704-1

Sahar, S., Younas, M., Khan, M. M., & Sarwar, M. U. (2024). DP-CCL: A Supervised Contrastive Learning Approach Using CodeBERT Model in Software Defect Prediction. *IEEE Access*, *12*, 22582–22594. https://doi.org/10.1109/ACCESS.2024.3362896

Tao, C., Gao, J., & Wang, T. (2019). Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices. *IEEE Access*, *7*, 120164–120175. https://doi.org/10.1109/ACCESS.2019.2937107

Tosun, A., Bener, A. B., & Akbarinasaji, S. (2017). A systematic literature review on the applications of Bayesian networks to predict software quality. *Software Quality Journal*, *25*(1), 273–305. https://doi.org/10.1007/s11219-015-9297-z

Tumar, I., Hassouneh, Y., Turabieh, H., & Thaher, T. (2020). Enhanced Binary Moth Flame Optimization as a Feature Selection Algorithm to Predict Software Fault Prediction. *IEEE Access*, *8*, 8041–8055. https://doi.org/10.1109/ACCESS.2020.2964321

Urbikain-Pelayo, G., Olvera-Trejo, D., López De Lacalle, L. N., & Elías-Zúñiga, A. (2025). Turn+: A MATLAB-based software for dynamic turning, chatter analysis, and surface roughness prediction. *SoftwareX*, *32*, 102401. https://doi.org/10.1016/j.softx.2025.102401

Wahono, R. S. (2015). A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks. *Journal of Software Engineering, 1*(1).

Wang, X., Ali, S., & Taibi, D. (2025). The Landscape of Quantum Software Testing Tools. *IEEE Software*, *42*(5), 136–140. https://doi.org/10.1109/MS.2025.3578154

Weng, S., Feng, Y., Yin, Y., Zhang, Z., & Xu, B. (2026). Data preparation and quality for code-centric generative software engineering tasks: A systematic literature review. *Frontiers of Computer Science, 20*(9), 2009203. https://doi.org/10.1007/s11704-025-41376-3

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2022). *Experimentation in software engineering* (2nd ed.). Springer

Zou, Q.-Y., Yang, Z.-Y., Qiu, X.-R., Yu, J.-H., Shi, Y.-Y., & Chen, N. (2025). Counterfactual Contrastive Explanations for Software Defect Prediction: Toward Better Model Understanding and Accuracy. *IEEE Transactions on Reliability*, *74*(4), 5000–5014. https://doi.org/10.1109/TR.2025.3611079