STM JOURNALS

International Journal of
Information Security Engineering

Volume 2, Issue 2, 2024
July–December
DOI (Journal): 10.37591/IJISE

https://journals.stmjournals.com/ijise

Review IJISE

Strategy for Improving Software Maintenance Using Machine Learning for Security Requirements: A Review

Maitri Manya[1,*], Raj Kumar Sharma[2]

Abstract

Within the area of software technical education, the significance of software defect discovery has increased as a research focus to enhance program reliability. By maximizing testing resources and assisting developers in identifying potential problems using program defect predictions, program dependability is increased. Applying software engineering (SE) techniques to critical and intricate systems, like networking and security systems, is imperative. Traditional methods of predicting software maintainability have limitations, particularly in balancing security concerns, maintainability, and system integrity. This work explores the application of machine learning (ML) techniques to predict and improve software maintainability by identifying key software metrics. The study explores several ML models, including deep learning, to increase the precision of the predictions made by software maintainability metrics. It also reviews existing research on software maintainability and defect prediction, identifying common research gaps such as model scalability, interpretability, and class imbalance issues. By employing ML classification techniques and addressing these gaps, this study aims to bridge the gap between security considerations and maintainability, providing more robust and efficient methods for software maintenance.

Keywords: Software engineering, software maintainability prediction, machine learning, metric, security

INTRODUCTION

In software development, maintenance takes up approximately 70% of the time and is expensive for developers. The complexity and scale of new software has expanded significantly in recent years, making it difficult to manage [1]. Software maintainability and a product or organization's financial performance are closely correlated [2]. We can forecast potential changes or software issues after launch using maintainability. Consequently, one of the qualities of software that influences its performance is maintainability. During the maintenance phase, developers receive assistance in estimating the likely quantity of alterations that may occur in software components. Because maintainability forecasts are accurate, software design can be modified to identify what must be changed to produce software modules [3].

A major issue facing the computer industry is software maintainability [4]. Thus, making the system automated is rational. Numerous approaches to artificial intelligence (AI) and machine learning (ML) have been employed. In most methods, testing has only been conducted using a limited set of software [5]. Previous studies have shown that no research has been conducted on the utilization of deep learning to

***Author for Correspondence**
Maitri Manya
E-mail: itsmaitrimanyahere2026@gmail.com

[1]Student, Department of Computer Science and Engineering, Lakshmi Narain College of Technology, Bhopal, Madhya Pradesh, India
[2]Assistant Professor, Department of Computer Science and Engineering, Lakshmi Narain College of Technology, Bhopal, Madhya Pradesh, India

Received Date: September 25, 2024
Accepted Date: October 03, 2024
Published Date: November 07, 2024

Citation: Maitri Manya, Raj Kumar Sharma. Strategy for Improving Software Maintenance Using Machine Learning for Security Requirements: A Review. International Journal of Information Security Engineering. 2024; 2(2): 36–48p.

forecast metrics for software maintenance. Furthermore, software upkeep predictability shows only minor improvements as a result of such operations. Datasets are insufficient for producing definitive findings that shed light on the input data of the system. The connected software-related activity Maintainability The primary method of predictive analytics is the application of various machine learning methods.

Among the several ML methods available for use in this investigation is deep learning. Instead of task-specific learning techniques, deep learning uses data representation [6]. This approach uses a hierarchical concept in which every layer picks up many levels of representation that correlate to various levels of complexity. The learning process can be completely supervised (e.g., classification), semi-supervised, or unsupervised (e.g., pattern analysis). The network decides which data to remember and which to forget after learning and remembering the order of incoming data in each layer. We can determine the metrics in which software sustainability will receive additions after the procedure, to generate precise predictions.

Problem Statement

A major difficulty in software development is finding an equilibrium between resolving security concerns, promoting maintainability, and preserving the system integrity. Conventional methods of determining and addressing security requirements are frequently imprecise and difficult to blend with the software development process. This study aims to investigate how ML classification techniques can be utilized to rank security needs according to how they affect software maintainability. This study aims to provide a more accurate and effective way to incorporate security consciousness into software maintenance methods by utilizing ML models and identifying critical metrics to bridge the gap between security considerations and maintainability.

Organized of this Paper

The rest of this paper is organized as follows: Section II discusses software maintenance in detail and explains its different types. This section discusses these activities. Section III focuses on the management of software maintenance and outlines key management functions. Section IV introduces ML concepts and their applications in software maintenance. Section V presents a comprehensive literature review, summarizing the existing research on the use of ML for software sustainability, reliability, and defect prediction. Finally, in Section VI, we conclude with limitations and future work.

SOFTWARE MAINTENANCE

A subfield of engineering science called “software engineering” is dedicated to creating software by applying scientific techniques and protocols. Software development life cycle (SDLC) is a procedure for creating programs that utilize software engineering methods and procedures. A series of tasks are carried out in the following stages of the SDLC generic process: requirements elicitation analysis, modeling and design, implementation, testing, and maintenance, as shown in Figure 1 [1]. According to IEEE, software maintenance is the process of making changes to a software product after it has been delivered to fix bugs, enhance performance or other features, or modify the product to fit into a different context [7]. Three primary factors—removing faults, including fresh attributes to adjust to the company context, or enhancing execution—are the causes of upkeep needs [2].

Professionals must perform considerable work during the maintenance procedure. Therefore, the proportion of upkeep expenditures is substantial; improvement requires one to two years, whereas maintenance takes five to ten years. According to Mishra et al. [3], the cost of upkeep is estimated to be four times that of development. Numerous difficulties are listed in a study [4, 5], and the most significant obstacle to the maintenance process is expense. As per the research [6], the overall cost of the program is more than 70% of the maintenance costs. Utilizing cloud computing is one way to potentially get around the maintenance problems. The maintenance process can be simplified using several features of cloud computing, including infrastructure, scalability, privacy, accessibility, availability, and constant technical assistance.

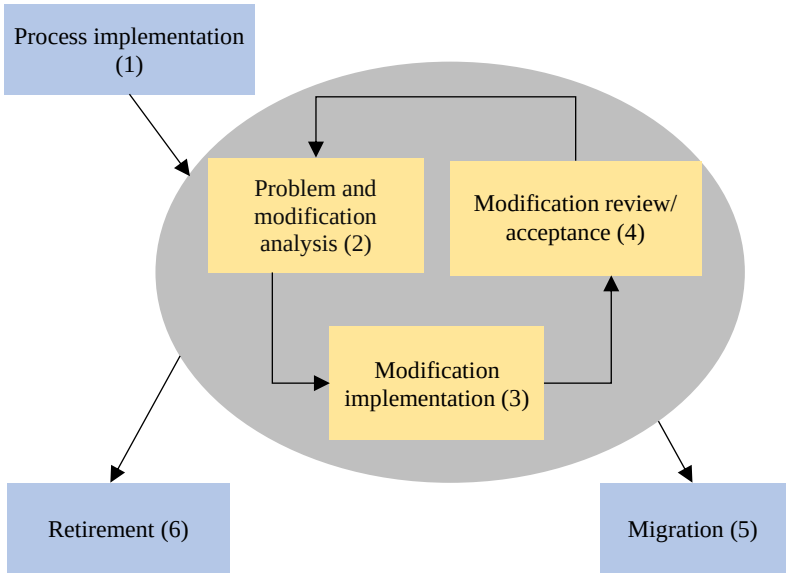

Figure 1. Overview of the maintenance process [8].

Types of Software Maintenance

Software maintenance comes in a variety of forms, and the goal of the designated upkeep is to determine why it is required [9]. Comprehending a particular weakness correctly can aid in resolving it without difficulty in any situation. Similarly, software maintainers can carry out maintenance with ease if they categorize issues according to their types [7].

- *Corrective maintenance:* After the software is delivered to the final user, errors must be fixed. These mistakes can be of many different types, including logical, design, and code flaws.
- *Adaptive maintenance:* Updates and changes are part of this type of maintenance, which keeps the system up-to-date with evolving corporate settings and technological advancements.
- *Perfective maintenance:* Updates and alterations are part of this form of maintenance, which aims to keep the system operating for as long as possible while integrating all requirements, functional and non-functional.
- *Preventive maintenance:* This kind of upkeep involves making adjustments and upgrades to stop issues that could arise later and lead to a system failure.
- *Emergency maintenance:* This type of maintenance is described by the IEEE standard as occurring when emergency mistakes arise and need to be fixed to maintain the system's continuity and consistency of operation [10].

Maintenance Reduction

There are several strategies for lowering software maintenance. Maintenance tasks can be greatly reduced if the preceding stages are closely examined and completed with extreme precision. Jobs like developing the right architecture, programming steps, adequate and unbiased testing, and good product documentation. The strategies listed below outline potential approaches for lower maintenance.

Re-documentation

If the product documentation is correctly updated in line with the latest trends and specifications, maintenance tasks can be cut in half or less. Re-documentation is found to lower costs by approximately 12%, which is significant, given that businesses spend a significant amount of money on product maintenance. Re-documentation may take the form of model-oriented re-documentation or incremental re-documentation.

Decreasing Turnovers

Lowering internal and external turnovers is a strategy to lower maintenance costs. While external turnovers are caused by external events and typically have an impact on employees through the

provision of more benefits or services, internal turnovers can be caused by a variety of internal factors occurring within the organization and may result in a massive turnover. Strict rules and work schedules that minimize turnovers in a project are implemented to mitigate this.

Dead Code Elimination

Eliminating dead codes improves the system performance and lowers the failure rates. Dead code is essentially a section of the code that is not utilized by the system and does not contribute to any component [11]. Generally speaking, this code remains in the system because it was utilized during the early stages of product building and was not removed by the programmer following upgrades and adjustments. Reducing the code size improves system performance and speed because it requires fewer resources and operates more quickly [12].

Understandability

To reduce the possibility of errors, maintenance staff must have a thorough understanding of the system before its installation. They must go through every stage of a product's development and be informed about the demands of the market today and the necessary implementation of changes.

Software Re-engineering

Software re-engineering is the process of updating and changing a product without impairing its operations. Every company undergoes a significant procedure to eliminate the faults and mistakes that users have reported. The code was redesigned to function flawlessly across all the platforms. It may seem more laborious to update legacy software than to create new software because it cannot be adjusted regularly owing to potential market developments. A product's fundamental functionality is one thing that never changes, even though it could age and become outdated, as shown in Figure 2.

Forward Engineering

This is the process of creating the desired product using specifications and data that were previously available following the reverse engineering step. However, certain steps in the software engineering process have already been completed. Except for one minor distinction, forward engineering occurs after reverse engineering; forward engineering functions similar to the software engineering process [13].

MANAGEMENT OF SOFTWARE MAINTENANCE

"The process of creating and preserving an atmosphere where people can effectively choose goals while collaborating in groups" is what management is defined as [14]. The basic goal of maintenance

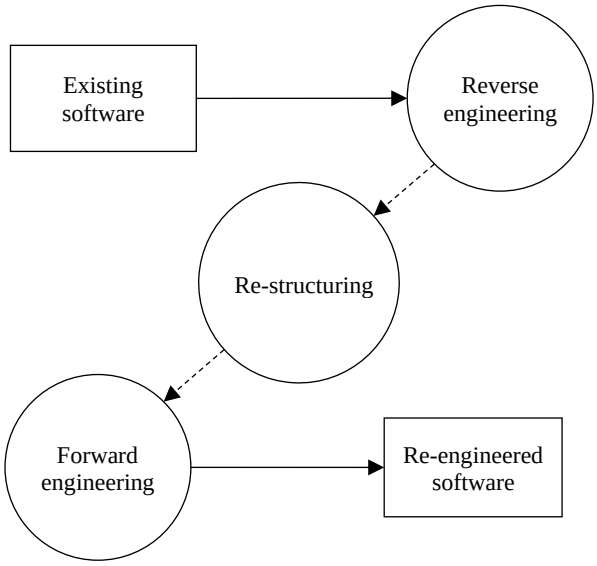

Figure 2. Software Re-engineering model.

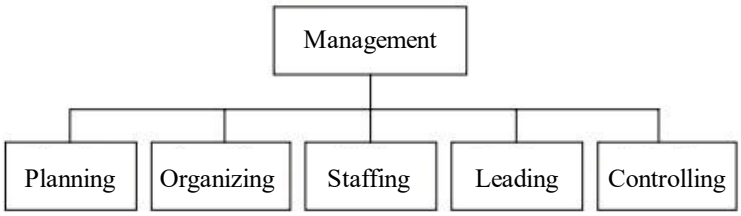

Figure 3. The functions of project management.

is to provide a software system with affordable support for its lifetime. Effectiveness and efficiency are implied by productivity and quality, respectively, which are concerns of management. Many authors [15] agree that management comprises five separate functions, as shown in Figure 3. Its functions are planning, organizing, staffing, leading (sometimes also called directing), and controlling.

Planning

Planning is the process of deciding on goals and missions and formulating a plan of action in advance to achieve them. The most important tasks in this role are action scheduling and commitment to material and human resources.

Organizing

The managerial task of organizing involves creating a purposeful framework of responsibilities for individuals to occupy within an organization. This means setting up the relationships between the roles and allocating the necessary authority and responsibilities.

Staffing

Staffing is the process of choosing and preparing candidates to fulfill open roles within a company. The two main tasks of this function are to assess and evaluate project staff members and to provide general development, that is, to enhance knowledge, attitudes, and abilities.

Leading

Creating a work environment and climate that will support and inspire individuals to help organizations and groups achieve their goals is the essence of leadership.

Controlling

Controlling compares actual performance to predetermined targets, and in the event of a discrepancy, develops remedial procedures. This means rewarding and reprimanding the project staff [16].

Cost of Maintenance

Reports indicate that maintenance is expensive, as shown in Figure 4. According to a study on software maintenance estimates, maintenance expenses account for up to 67% of the total cost of the software process cycle [17]. Software maintenance costs account for almost half of all SDLC stages on average. Numerous factors contribute to the increased maintenance expenses. Factors in the real world that impact maintenance costs:

- Structure of Software Program
- Programming Language
- Dependence on the external environment
- Staff reliability and availability

Maintenance Activities

An activity sequence for the maintenance procedure is provided by the IEEE, as shown in Figure 5. It can be expanded to encompass bespoke products and procedures and used iteratively [18]. These activities go hand-in-hand with each of the following phases.

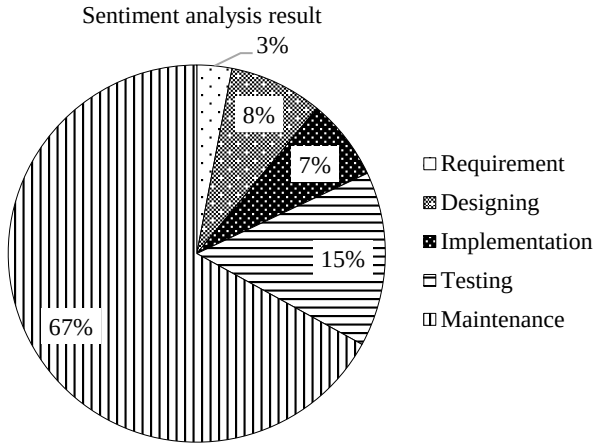

Figure 4. Cost of maintenance.

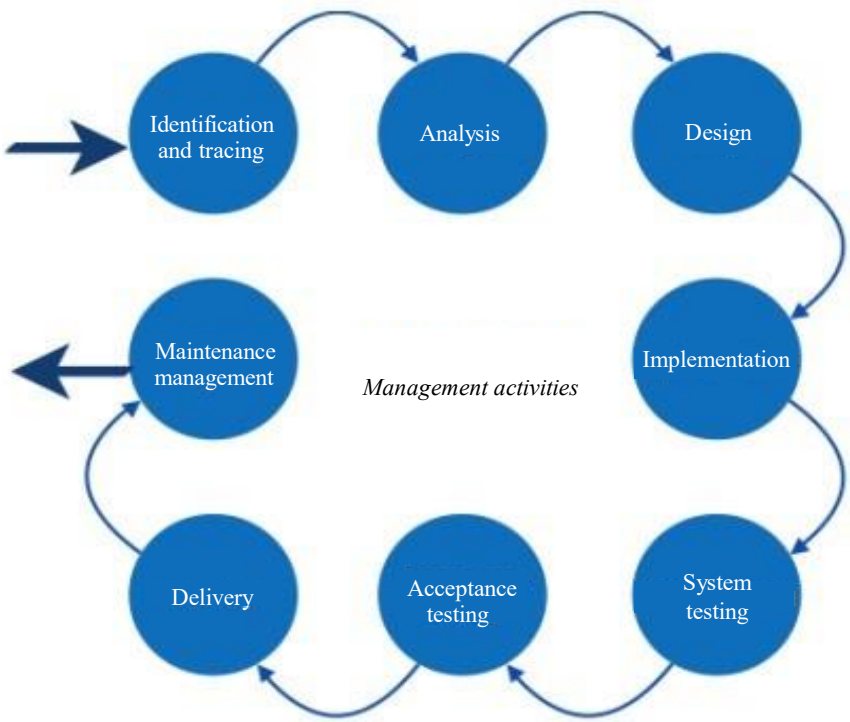

Figure 5. Activities of software maintenance [19].

Identification and Tracing

This entails tasks related to determining what needs to be maintained or modified. Either the user generates it, or the system reports it via error messages or logs.

Analysis

The impact of the alteration on the system was analyzed, considering any potential security and safety ramifications. If a different course of action is more likely to have a major impact, then a list of necessary changes materializes in requirement specifications. After analysis and estimation, the costs of modification and maintenance were determined.

Design

The requirements specified in the preceding steps were utilized to create new modules that required modifications or replacements. Test cases were developed to ensure validation and verification.

Implementation

The new modules are coded using a systematic design created during the design process. Unit testing was concurrently expected for every coder.

System Testing

Testing was performed for integration between the recently developed modules. Testing is also performed on the integration between the new modules and the system. Finally, regressive testing methods were used to test the system as a whole.

Acceptance Testing

The system was checked for acceptability with user assistance after internal testing. If users still have complaints at this point, they will either be handled or noted in a future edition.

Delivery

The system was either installed from scratch or distributed throughout the company via a simple update package following the acceptance test. As soon as the software was delivered, the client conducted the final test.

Maintenance Management

Configuration management is an important aspect of system maintenance. To regulate versions, semi-versions, or patch management, version control tools are helpful.

MACHINE LEARNING

Samuel popularized and initially presented the idea of ML in 1959 [20]. It was demonstrated that a computer can be trained to outperform its original programmer in a game of checkers. AI's ML field gives systems the capacity to learn and grow on their own without the need to be specifically programmed to do so [21]. Without training to carry out the task, it creates a model based on training data to make judgments or predictions automatically, as shown in Figure 6. The three primary classifications of ML techniques are reinforcement learning, unsupervised learning, and supervised learning. These divisions are based on the input received by the learning systems.

Supervised Learning

Machine learning is the most commonly used method for supervision [22]. Labeled datasets are used in supervised ML to train algorithms for precise data classification or outcome prediction. Labeled data is a single data label that could, for instance, specify the type of action depicted in an image or whether an animal is included in a video. The output data in supervised learning are labeled, indicating that specific outcomes are anticipated, and prior knowledge is required to label them, as shown in Figure 7.

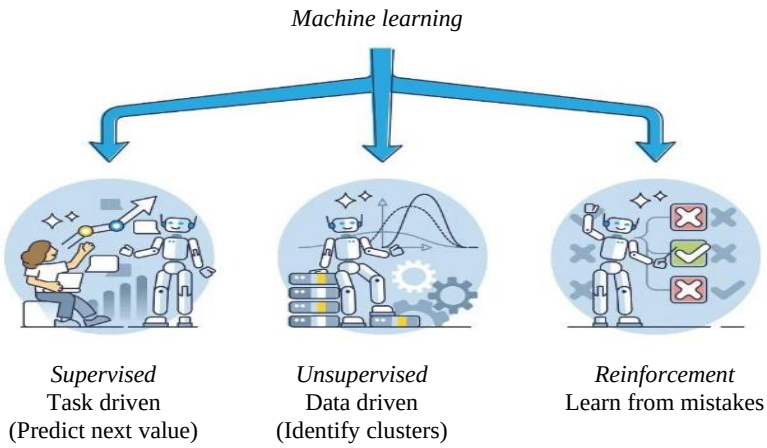

Figure 6. Machine learning.

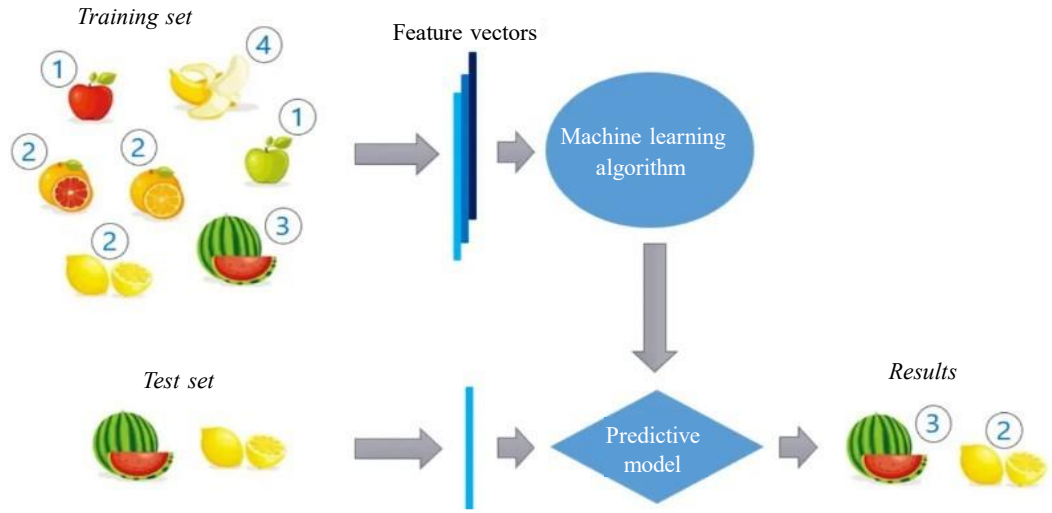

Figure 7. Supervised learning.

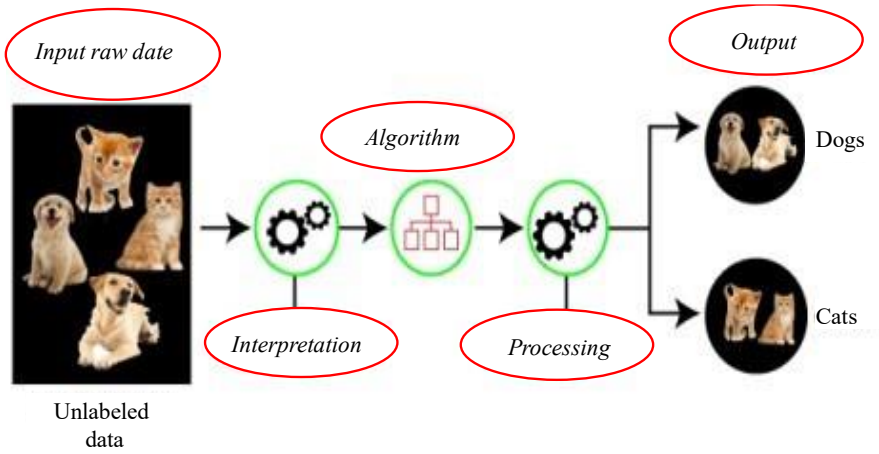

Figure 8. Unsupervised learning.

Unsupervised Learning

The training set for unsupervised learning consisted of unlabeled data. Except for validating the output variables, the machine learner in this instance attempts to solve the problems on its own. It searches the provided unlabeled data for hidden structures, patterns, etc. using these algorithms. Unsupervised learning models are applied to association, dimensionality reduction, and clustering. The clustering method was used to analyze the unlabeled data can be categorized based on their commonalities or variances, as shown in Figure 8.

Reinforcement Learning

The final of the three fundamental paradigms utilized in ML is reinforcement learning, as shown in Figure 9. The methodology of reinforcement learning lies between learning that is supervised and learning that is not. It deals with learning how to make decisions sequentially when there is little or no information [23]. A smart agent that investigates space or environment is used in reinforcement learning. This agent is typically represented by Markov decision processes. A formal framework for simulating decision-making in scenarios where the decision maker has some influence over some outcomes, but not all of them are provided by Markov decision processes.

LITERATURE REVIEW

This section provides a previous study on software maintenance in software engineering utilizing deep learning and machines, techniques, and methods provided below.

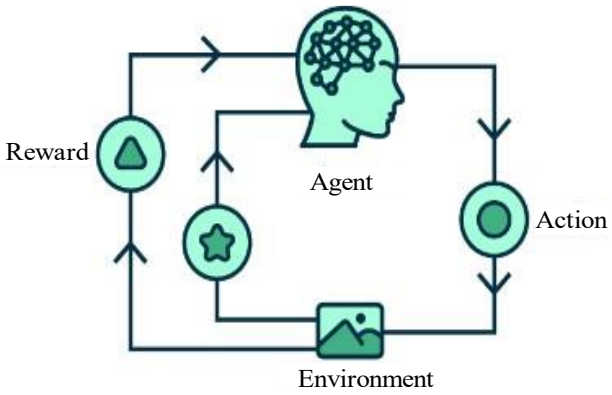

Figure 9. Reinforcement learning.

Silva, Bezerra, and Machado (2023) suggested white-box ML models to classify FM maintainability according to 15 measures. The process of developing the models involved the following steps: (i) two methods for evaluating FM maintainability using a human-based FM maintainability classification oracle were compared; (ii) pre-classification of the ML training dataset using the best method; (iii) creation of three machine learning models and assessment of the models using F1, AUC-ROC, recall, accuracy, and precision in classification; and (iv) use of the best model to develop a mechanism that can provide domain engineers with indications for improvements. The top model employed a decision tree technique and achieved a sum of 0.81 for accuracy, precision, and recall; 0.79 for F1-Score; and 0.91 for AUC-ROC. Using this approach, they may lower the required number of variables to assess FM maintainability from 15 to 9. To further enhance the maintainability of this artifact, they developed a method to recommend FM refactoring. The DyMMer tool streamlines the process of evaluating FM maintainability and the refactoring recommendation system [24].

In this study, Bombiri et al. (2023) recommended practicing a brief literature review of studies that have used machine learning (ML) techniques to enhance software quality. Software quality management using ML approaches is a vibrant area of study, as demonstrated by the lessons learned. Machine learning techniques have solved issues related to software defect prediction and maintainability prediction. This type of prediction model has been the primary focus of research on datasets, software metrics, and ML algorithms' abilities [25].

The study conducted by Wang and Zhang (2024) provides a model that uses software metrics derived from development data to quantify software dependability as a function of the residual fault count. The goal is to determine a statistical correlation between quality measurements and either product metrics (which deal with the product itself) or process metrics (which deal with the development process). Combining a pattern recognition algorithm method for fault proneness classification with fuzzy logic to software measurements for defect prediction may improve software reliability prediction during early development phases. Using historical failure data, this article proposes non-parametric methods, such as deep learning artificial neural networks (ANNs), to evaluate software reliability., and release readiness before launch [26].

In this paper, Al-Smadi et al. (2023) presented a novel system that uses 11 classification models for machine learning across 12 sets of data to forecast software problems. We use four distinct nature-inspired search algorithms for feature selection: particle swarm optimization, genetic algorithm, harmony algorithm, and ant colony optimization. Furthermore, to tackle the issue of data imbalance, we employ the synthetic minority oversampling technique (SMOTE). In addition, we emphasized the greatest determinative features using the Shapley additive explanation model. Based on the gathered data, decision trees, gradient boosting, stochastic gradient boosting, and categorical boosting performed better than the other tested models with accuracy and ROC-AUC of over 90%. Furthermore, we

discovered that the ant colony optimization approach works better than the other feature extraction methods used in the test [27].

In this study, Yimer, Molla, and Alemneh (2022) proposed a strategy for anticipating the type of maintenance, the effects of changes, and the duration of the maintenance to retrieve pertinent maintenance data from software repositories. Maintenance jobs and their impact on changes are predicted using Long Short-Term Memory (LSTM), Bi-LSTM, Random Forest, Logistic Regression, Multinomial Naïve Bayes, and Linear Support Vector Classifier. The time needed to resolve the modifications was estimated using an ANN. The experimental findings demonstrate that Random Forest and LSTM outperform other models, with accuracy rates of 94% and 95%, respectively. A Mean Square Error of 0.0028 was obtained by the ANN. The results of the Pearson and Spearman correlation analyses indicate that there is a positive association between maintenance type and maintenance time; however, there is a negative relationship between the type of maintenance, the amount of time it takes, and its effect [7].

According to Malhotra and Lata (2020), software engineers working on a maintainability prediction model could encounter cases where modules or classes with high levels of maintenance work are much less common than those with low-level maintenance efforts. The issue of class imbalance (CIP) arises because of this situation. Predictions made by minority classes, those with high maintainability effort requirements—present difficulties in this context. To that end, this research looks at 10 open-source programs using three methods for dealing with CIP to forecast software maintainability. To tackle CIP and create practical models for Sampling Model of Prediction (SMP), this empirical analysis supports the use of the resampling with replacement (RR) approach [28].

Reddivari and Raman (2019) provided a reliability- and maintainability-focused review of eight ML approaches. The number of flaws in a system is used to study its reliability, while the number of system modifications is used to analyze its maintainability. Software metrics served as important training parameters for both the defect and maintainability prediction models in our study. These are direct reflections of the various software characteristics. With an AUC of more than 0.8 in expect of both defects and maintenance, Random Forest yielded the best results out of the eight methods we tested [29].

Research Gaps

Table 1 provides a comprehensive summary of the above background studies on software maintenance using ML techniques. The common research gaps in these studies indicate several promising avenues that need to be explored further. Model generalizability and scalability across multiple software systems and domains are limits of ML approaches, notwithstanding their promising outcomes in software maintainability, defects, and reliability prediction. In theory, several models, including those that use neural networks, decision trees, and random forests, have been quite accurate; however, in practice, they have not been widely used. Additional work is required to address issues such as class imbalance, optimize feature selection, and address unstructured data. Further challenges include making ML models interpretable to non-experts and integrating them with real-time software development tools and procedures. Software quality management may be greatly improved if future studies validate these models using larger and more varied datasets, making them more applicable to real-world software systems.

Table 1. Comparative study on improving software maintenance using machine learning.

Study	Objective	Methods	Key Findings	Limitations/Research Gaps
Silva, Bezerra and Machado (2023) [24]	Classify FM maintainability using ML models based on 15 measures	Decision Trees, F1, AUC-ROC, Recall, Accuracy, Precision, Oracle-based pre-classification.	Achieved 0.81 for accuracy, precision, and recall, 0.79 for F1-Score, and 0.91 for AUC-ROC. Reduced variables from 15 to 9 for FM maintainability.	Further research is needed to generalize the model across different FM domains and enhance interpretability for non-experts.

Bombiri, Poda and Ouedraogo (2023) [25]	Enhance software quality through ML techniques	Literature review, ML for software defect prediction and maintainability.	ML successfully addresses defect and maintainability prediction, emphasizing the role of software metrics and algorithms.	Lack of practical application of findings; need for validation with real-world datasets and environments.
Jagtap, Katragadda and Satelkar (2022) [30]	Quantify software dependability using software metrics	Pattern recognition algorithm for fault proneness, fuzzy logic for defect prediction, ANNs for software reliability prediction.	ANNs and non-parametric methods provide effective early reliability prediction, improving fault classification and release readiness.	Limited scalability of the model across diverse software systems; need for comparison with alternative deep learning models.
Al-Smadi et al. (2023) [27]	Predict software defects using ML classifiers	11 ML classifiers, feature selection via PSO, GA, Harmony Search, Ant Colony Optimization, SMOTE, SHAP	Gradient Boosting, Stochastic Gradient Boosting, Decision Trees, and Categorical Boosting achieved over 90% accuracy and AUC-ROC.	Feature selection techniques may need improvement; real-time implementation in live systems is unexplored.
Yimer, Molla and Alemneh (2022) [7]	Predict maintenance tasks, impact of changes, and maintenance time	Linear Support Vector Classification (SVC), Random Forest, Logistic Regression, Naïve Bayes, LSTM, Bi-LSTM for tasks; ANN for maintenance time estimation.	Random Forest (94%) and LSTM (95%) outperformed others; ANN achieved an Mean Squared Error (MSE) of 0.0028 for time prediction.	Difficulty in handling highly unstructured data; integration with software development tools not fully addressed.
Malhotra and Lata (2020) [28]	Tackle class imbalance in maintainability prediction models (SMP)	Resampling with replacement approach to handle class imbalance (CIP).	Resampling approach effective in addressing CIP in maintainability prediction models across 10 open-source programs.	Limited focus on generalization to larger datasets and environments with more complex class imbalances.
Reddivari and Raman (2019) [29]	Evaluate ML techniques for reliability and maintainability prediction	Random Forest, other ML techniques, software metrics.	Random Forest yielded the best results with AUC > 0.8 for defect and maintenance prediction.	Limited exploration of other potential ML models; no clear exploration of the trade-offs between interpretability and performance.

CONCLUSION AND FUTURE WORK

As software development advances, maintainability continues to be a major challenge. Future approaches to maintainability may be affected by developments in automated code creation, ML, and AI. Nonetheless, the basic objective of developing software that is easily extensible and modifiable never changes. The idea of software maintainability has changed as a result of the necessity for efficient and economical maintenance, as well as the increasing complexity of software systems. Software system maintenance and improvement remains a crucial component of the software development process, as software becomes an essential part of contemporary life. This study examines the use of ML techniques to improve software maintainability prediction, addressing the challenges of escalating complexity and costs in software maintenance. Although several effective approaches have been identified, the research highlighted limitations such as the lack of generalizability and scalability of ML

models across diverse systems, restricted datasets, and the persistent issue of class imbalance, where high-maintenance modules are often underrepresented. In addition, the complexity of ML models can limit their practical use, particularly for non-experts. Future work should focus on enhancing model scalability, addressing class imbalance through advanced techniques, such as synthetic data generation, and integrating interpretable ML models into real-time development environments. Validating these models on larger, more varied datasets and embedding them into software engineering tools can significantly improve maintainability predictions and reduce maintenance costs.

REFERENCES

1. Huang Q, Shihab E, Xia X, Lo D, Li S. Identifying self-admitted technical debt in open source projects using text mining. Empir Softw Eng. 2018;23:418–51. DOI: 10.1007/s10664-017-9522-4.
2. Guo J, Yang D, Siegmund N, Apel S, Sarkar A, Valov P, et al. Data-efficient performance learning for configurable systems. Empir Softw Eng. 2018;23:1826–67. DOI: 10.1007/s10664-017-9573-6.
3. Mishra S, Sharma A. Maintainability prediction of object-oriented software by using adaptive network-based fuzzy system technique. Int J Comput Appl. 2015;119:24–7. DOI: 10.5120/21096-3799.
4. Mhawish MY, Gupta M. Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. J Comput Sci Technol. 2020;35:1428–45. DOI: 10.1007/s11390-020-0323-7.
5. Amarjeet JK, Chhabra JK. Improving package structure of object-oriented software using multi-objective optimization and weighted class connections. J King Saud Univ Comput Inf Sci. 2017;29:349–64. DOI: 10.1016/j.jksuci.2015.09.004.
6. Chug A, Malhotra R. Benchmarking framework for maintainability prediction of open source software using object-oriented metrics. Int J Innov Comput Inf Control. 2016;12:615–34.
7. Yimer ST, Molla YS, Alemneh E. Predicting software maintenance type, change impact, and maintenance time using machine learning algorithms. Int Conf Inf Commun Technol Dev Afr (ICT4DA). 2022;37–41. DOI: 10.1109/ICT4DA56482.2022.9971350.
8. Rantanen O. Artificial intelligence in software maintenance [Master's thesis]. Lappeenranta: Lappeenranta-Lahti University of Technology; 2021.
9. Aburakhia S, Shami A. SB-PdM: A tool for predictive maintenance of rolling bearings based on limited labeled data. Softw Impacts. 2023;16:100503.
10. IEEE 12207-2-2020. ISO/IEC/IEEE International Standard - Systems and software engineering--Software life cycle processes--Part 2: Relation and mapping between ISO/IEC/IEEE 12207:2017 and ISO/IEC 12207:2008. IEEE Computer Society. 2020.
11. Kemerer CF. Software complexity and software maintenance: A survey of empirical research. Ann Softw Eng. 1995;1:1–22. DOI: 10.1007/BF02249043.
12. Kaur U, Singh G. A review on software maintenance issues and how to reduce maintenance efforts. Int J Comput Appl. 2015;118:6–11. DOI: 10.5120/20707-3021.
13. Hall T, Rainer A, Baddoo N, Beecham S. An empirical study of maintenance issues within process improvement programmes in the software industry. Proc IEEE Int Conf Softw Maint. 2001;422–30. DOI: 10.1109/ICSM.2001.972755.
14. Joullié JE, Gould AM. Theory, explanation, and understanding in management research. BRQ Bus Res Q. 2023;26:347–60. DOI: 10.1177/23409444211012414.
15. Caulfield C, Veal D, Maj SP. Teaching Software Engineering Project Management – A Novel Approach for Software Engineering Programs. Mod Appl Sci. 2011;5. DOI: 10.5539/mas.v5n5p87.
16. Canfora G, Cimitile A. Handbook of software engineering and knowledge engineering. Fundam. Vol. 1. 2001;2010. DOI: 10.1142/9789812389718.
17. Karningsih PD, Puspitasari W, Singgih ML. Cost-integrated lean maintenance to reduce maintenance cost. J Optimasi Sistem Industri. 2023;22:69–80. DOI: 10.25077/josi.v22.n1.p69-80.2023.
18. Rana A, Koroitamana EVM. Measuring maintenance activity effectiveness. J Qual Maint Eng. 2018;24:437–48. DOI: 10.1108/JQME-11-2016-0061.

19. Tsang AH, Jardine AK, Kolodny H. Measuring maintenance performance: a holistic approach. International Journal of Operations & Production Management. 1999 Jul 1;19(7):691-715.
20. Awad M, Khanna R. Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers. Berkeley, CA: Apress; 2015. DOI: 10.1007/978-1-4302-5990-9.
21. Voskoglou MG, Salem ABM. Benefits and limitations of the artificial with respect to the traditional learning of mathematics. Math. 2020;8. DOI: 10.3390/math8040611.
22. Burkart N, Huber MF. A survey on the explainability of supervised machine learning. J Artif Intell Res. 2021;70:245–317. DOI: 10.1613/JAIR.1.12228.
23. Van Otterlo M, Wiering M. Reinforcement Learning and Markov Decision Processes. In: Wiering M, van Otterlo M, editors. Reinforcement Learning. Adaptation, Learning, and Optimization, vol 12. Springer, Berlin, Heidelberg; 2012. DOI: 10.1007/978-3-642-27645-3_1.
24. Silva P, Bezerra C, Machado I. Automating feature model maintainability evaluation using machine learning techniques. J Syst Softw. 2023;195. DOI: 10.1016/j.jss.2022.111539.
25. Bombiri O, Poda P, Ouedraogo TF. Application of machine learning in software quality: A mini-review. 2023 IEEE Multi-conference on Natural and Engineering Sciences for Sahel's Sustainable Development (MNE3SD), Bobo-Dioulasso, Burkina Faso. 2023. pp. 1–7. DOI: 10.1109/MNE3SD57078.2023.10079800.
26. Wang J, Zhang C. An open-source software reliability model considering learning factors and stochastically introduced faults. Appl Sci. 2024;14:708. DOI: 10.3390/app14020708.
27. Al-Smadi Y, Eshtay M, Al-Qerem A, Nashwan S, Ouda O, Abd El-Aziz AA. Reliable prediction of software defects using Shapley interpretable machine learning models. Egyptian Informatics J. 2023. DOI: 10.1016/j.eij.2023.05.011.
28. Malhotra R, Lata K. An empirical study on predictability of software maintainability using imbalanced data. Softw Qual J. 2020;28:1581–1614. DOI: 10.1007/s11219-020-09525-y.
29. Reddivari S, Raman J. Software quality prediction: An investigation based on machine learning. 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI), Los Angeles, CA, USA. 2019. pp. 115-122. DOI: 10.1109/IRI.2019.00030.
30. Jagtap M, Katragadda P, Satelkar P. Software Reliability: Development of Software Defect Prediction Models Using Advanced Techniques. Proceedings of the 2022 Annual Reliability and Maintainability Symposium (RAMS). Tucson, AZ, USA. IEEE Press; 2022. pp. 1–7. DOI: 10.1109/RAMS51457.2022.9893986.

