

A Systematic Review of AI-Driven Software Bug Prediction and Prioritization

¹*Dr. A. Antony Prakash, ²Ishwarya C, ³Praveen Raj S

¹Head and Assistant Professor, Department of Information Technology, St. Joseph's College, Tiruchirappalli, Tamil Nadu, India

^{2,3}PG Student, Department of Information Technology, St. Joseph's College, Tiruchirappalli, Tamil Nadu, India

*E-mail: aantonyprakash@gmail.com

Abstract: Software development projects often face challenges in managing and resolving a large number of bugs efficiently. Traditional bug tracking systems rely heavily on manual prioritization, which can be time-consuming and prone to human error. This paper surveys artificial intelligence and machine learning approaches for software bug prioritization and prediction. The review considers historical bug-report information such as severity, frequency, affected module, textual descriptions, developer activity, and past resolution time, and examines how these attributes can be used to predict priority levels and identify potentially defect-prone software components. Classification algorithms including Decision Trees, Random Forest, and Support Vector Machines can categorize newly reported bugs into priority classes such as high, medium, and low. Bug-report analysis can additionally use natural-language features to support categorization, severity prediction, and triaging, while software defect prediction models can learn patterns from code and project metrics. Recent literature highlights the usefulness of machine learning while also identifying recurring challenges involving class imbalance, noisy labels, dataset shift, explainability, data quality, and generalization across projects. The paper further presents a unified architecture for an AI-assisted bug management platform using Python, Flask, Scikit-learn, and a relational database. Such a system can combine bug submission, automated priority prediction, defect-risk prediction, history, and reporting in one interface. The survey concludes that AI-assisted bug prioritization can support software maintenance by reducing manual effort and helping teams focus attention on critical issues, while reliable deployment requires representative datasets, rigorous validation, transparent predictions, and human oversight.

Keywords: Artificial Intelligence, Machine Learning, Software Bug, Bug Prioritization, Bug Prediction, Software Defect Prediction, Random Forest, Support Vector Machine, Software Maintenance.

I. INTRODUCTION

Software projects generate large numbers of defect reports during development, testing, deployment, and maintenance. When many bugs are reported simultaneously, development teams must decide which issues should be investigated and fixed first. Traditional bug tracking systems commonly depend on developers or testers to assign severity and priority manually. This process can require substantial effort and may produce inconsistent decisions when reports are numerous, incomplete, or submitted by different users.

Machine learning provides a data-driven approach to software maintenance by learning relationships between historical bug reports, project characteristics, and later outcomes. Systematic reviews of software defect prediction show extensive use of supervised learning, ensemble methods, deep learning, and software metrics for identifying defect-prone modules [1], [2]. Recent research also emphasizes that defect prediction

should produce more informative and explainable outputs rather than only a binary defective/non-defective label [3].

For an AI-assisted bug management system, the research challenge is not simply selecting a classifier. Bug reports may contain structured attributes, free-text descriptions, categorical labels, timestamps, component information, and historical developer activity. The model must therefore handle missing values, noisy labels, class imbalance, text representation, and differences between software projects. A practical platform should keep prediction models separate from the application interface so that models can be retrained or replaced without redesigning the complete system.

The remainder of this paper is organized as follows. Section 2 reviews recent literature on software defect prediction, bug triaging, and bug-report analysis. Section 3 states the problem addressed by the survey. Section 4 summarizes machine-learning techniques relevant to bug prioritization and

prediction. Section 5 proposes a layered system architecture. Section 6 discusses the prediction models and evaluation measures. Section 7 lists candidate technologies for implementation. Sections 8 through 10 discuss applications, challenges, and advantages, and Section 11 concludes the paper with directions for future work.

II. LITERATURE REVIEW

Software defect prediction has been studied extensively as a way of identifying defect-prone software components before failures become costly. A systematic literature review of supervised machine-learning techniques examined datasets, preprocessing, algorithms, and evaluation approaches used for software defect classification [1]. Another systematic review of artificial-intelligence-based defect prediction emphasized the importance of dataset quality, validation methods, prediction approaches, and tools [2].

Recent work has also focused on making defect-prediction outputs more informative. A 2024 systematic literature review analyzed 132 selected studies and reported that many models still provide binary outputs, while explainability is not widely incorporated [3]. This is relevant to bug prioritization because development teams often need actionable information such as severity, priority, defect type, or likely affected component rather than a simple prediction of whether a defect exists.

Bug triaging research considers the automation of activities such as categorization, severity prediction, assignment, and prioritization. A review of artificial-intelligence frameworks for software bug triaging describes machine-learning approaches for handling large volumes of bug information and discusses challenges in automation [4]. A recent systematic literature review of bug-report learning examined a large body of studies and identified bug categorization, localization, and severity prediction as common tasks, with increasing use of deep-learning representations [5].

Industrial defect-prediction research also reports frequent use of Support Vector Machines and Random Forest, together with metrics such as Accuracy and F1-score [6]. Ensemble learning has been investigated because combining multiple classifiers can reduce dependence on the weaknesses of an individual model [7]. Taken together, the literature suggests that AI can support software maintenance, but model performance should be interpreted in the context of project-specific data, class imbalance, explainability, validation strategy, and changes in software development practices.

Table 1: Survey of Recent Research

Research area	Focus	Key research issue
Software defect prediction	ML classification of defect-prone modules	Dataset quality and generalization
Bug triaging	AI-assisted categorization and prioritization	Automation and noisy labels
Bug-report analysis	Text and structured report learning	Representation and explainability
Industrial prediction	ML models for software quality	Class imbalance and validation
Explainable prediction	Actionable and interpretable outputs	Transparency of model decisions

III. PROBLEM STATEMENT

Software projects may receive hundreds or thousands of bug reports with different levels of severity and business impact. Manual prioritization can become difficult when reports arrive continuously and when the available information is incomplete or inconsistent. Existing bug-tracking tools can record reports effectively, but the decision about which issue should be addressed first may still depend heavily on human judgment. There is therefore a need for an intelligent framework that can learn from historical bug data and provide consistent priority predictions.

The problem is to investigate how machine-learning methods can analyze historical bug information and predict the priority of newly reported software bugs while also identifying patterns associated with future defects. The proposed approach considers two related tasks: (i) prioritizing reported bugs into categories such as high, medium, and low, and (ii) predicting potential defects from historical software and project patterns. The system should assist developers rather than replace their judgment, because priority may also depend on business requirements, security implications, release schedules, and domain knowledge that are not completely represented in historical data.

IV. MACHINE LEARNING TECHNIQUES FOR BUG PRIORITIZATION AND PREDICTION

This section summarizes supervised and ensemble learning methods relevant to AI-assisted software bug management. The

techniques are presented from relatively interpretable classifiers to more complex models. The choice of algorithm should be validated on representative historical data rather than assumed to be optimal for every software project.

4.1 Logistic Regression

Logistic Regression can be used as a baseline classifier for priority or defect categories. It estimates class probabilities from input features and is relatively easy to train and interpret. For bug reports, numerical project metrics and encoded categorical or text-derived features can be used as predictors. Its coefficients can provide an initial indication of how features are associated with the predicted class.

4.2 Decision Tree

Decision Trees classify observations through a sequence of feature-based rules. For bug prioritization, a tree may learn rules involving severity indicators, affected modules, frequency, historical resolution time, or other report attributes. The resulting rules are comparatively easy to explain, but an uncontrolled tree may overfit historical reports. Depth control, pruning, and validation are therefore important.

4.3 Random Forest

Random Forest combines multiple decision trees and aggregates their predictions. It can represent nonlinear relationships and interactions among software metrics and bug-report attributes. Random Forest is particularly useful when the relationship between features and priority is not simple or linear. Feature importance measures can also provide supporting information, although they should not automatically be interpreted as causal explanations.

4.4 Support Vector Machine

Support Vector Machines construct decision boundaries between classes and can represent nonlinear relationships through kernels. They can be applied to structured software metrics and appropriately represented text features. Feature scaling and parameter selection can substantially affect performance, and model settings should be selected using validation procedures rather than fixed without evidence.

4.5 K-Nearest Neighbors

K-Nearest Neighbors predicts a new bug report by comparing it with nearby examples in the training data. The method is simple and can be useful as a baseline, but its

performance depends strongly on feature scaling, neighborhood size, and the representation of bug reports. Large or high-dimensional datasets may also make prediction more computationally expensive.

4.6 Naive Bayes

Naive Bayes applies Bayes' theorem with a conditional-independence assumption. It is lightweight and can be useful for text classification tasks such as categorizing bug reports. Although the independence assumption is simplified, the method can provide a fast baseline for comparing automated bug-report classification approaches.

4.7 Deep Learning

Deep-learning models can learn complex representations from large collections of software artifacts and bug reports. Neural language models, recurrent architectures, convolutional models, and transformer-based methods have been explored for software-engineering tasks. However, these methods may require larger datasets and more computational resources, and their predictions can be difficult to explain. For production use, performance, reproducibility, data leakage, and explainability should be evaluated carefully.

Table 1a: Comparative Summary of Techniques

Technique	Strength	Limitation
Logistic Regression	Simple and interpretable baseline	Limited nonlinear representation
Decision Tree	Readable rule structure	Can overfit without control
Random Forest	Handles nonlinear interactions	Less transparent than a single tree
SVM	Effective classification boundaries	Sensitive to scaling and parameters
K-Nearest Neighbors	Simple comparison baseline	Sensitive to scale and dimensionality
Naive Bayes	Fast and useful for text classification	Independence assumption
Deep Learning	Learns complex representations	Needs more data and computation

V. SYSTEM ARCHITECTURE

A practical AI-assisted bug management system can be organized into five layers. The user interface collects and displays bug reports. The application layer validates requests and manages authentication and routing. The machine-learning layer preprocesses historical and new bug data and invokes the appropriate prediction model. The database layer stores user records, bug reports, model outputs, and prediction history. A reporting layer presents priority predictions, defect-risk information, and historical summaries.

Table 2: Layered Architecture

Layer	Function
User Interface	Collects bug reports and displays predictions.
Application Layer	Flask validates input, manages sessions, and routes requests.
ML Layer	Preprocesses data and invokes priority/defect models.
Database Layer	Stores users, bug reports, predictions, and history.
Reporting Layer	Presents priority, risk, confidence, and historical summaries.

In this architecture, the bug-prioritization model and defect-prediction model can be trained and versioned independently. The application sends validated features to a common prediction interface and receives a predicted class together with supporting confidence information. This separation allows the models to be retrained as new bug reports become available without requiring changes to the user interface, authentication, or database components. Human review remains part of the workflow so that unusual or high-impact bugs can be escalated even when the model assigns a lower predicted priority.

VI. BUG PRIORITIZATION AND PREDICTION MODELS

6.1 Bug Priority Prediction

The priority model can use historical attributes such as severity, frequency, affected module, report age, previous resolution time, reporter information, and selected textual features. After preprocessing and feature encoding, a classification algorithm predicts a priority class such as high,

medium, or low. The training data should be divided carefully to avoid leakage from future information into historical training records.

6.2 Bug Severity and Category Prediction

Severity and category prediction can provide additional information for triaging. Textual descriptions may be transformed using techniques such as TF-IDF or learned embeddings, while structured attributes can be encoded separately. A model may then predict severity, category, or other labels that help route a report to an appropriate development team. These predictions should be evaluated separately because a model that performs well for one label may not perform equally well for another.

6.3 Defect-Prone Module Prediction

Software defect prediction can use historical code and project metrics to identify modules that may be more likely to contain defects. Possible features include code size, complexity, coupling, change frequency, previous defects, and development activity. The purpose is to support testing and maintenance planning by identifying areas that may deserve additional attention before or during a release.

6.4 Evaluation Measures

Model evaluation should use more than one metric because bug-priority datasets may be imbalanced. Accuracy measures the overall proportion of correct predictions, while Precision measures the proportion of predicted positive cases that are correct. Recall measures how many actual positive cases are identified, and F1-score balances Precision and Recall. For multiclass priority prediction, macro-averaged and class-specific measures can reveal whether a model performs poorly on minority priority classes. Cross-validation and, where possible, time-aware or project-aware validation can provide stronger evidence of generalization.

Table 3: Common Evaluation Measures

Metric	Purpose
Accuracy	Overall proportion of correctly classified reports.
Precision	Proportion of predicted positive cases that are correct.
Recall /	Ability to identify actual positive cases.

Sensitivity	
F1-score	Harmonic mean of Precision and Recall.
ROC-AUC	Measures discrimination across classification thresholds.
Macro F1	Balances performance across multiple priority classes.

VII. TECHNOLOGIES AND PLATFORMS

The proposed implementation can use Python for data processing and model development and Flask for web deployment. Scikit-learn provides common preprocessing and classical machine-learning algorithms, while Pandas and NumPy support data manipulation. HTML, CSS, JavaScript, and Bootstrap can provide the web interface, and MySQL can store users, bug reports, predictions, and model history. A version-control platform can also be used to maintain source code and model versions.

Table 4: Technology Stack

Component	Technology / Tool
Programming	Python
ML Framework	Scikit-learn
Data Processing	Pandas, NumPy
Web Framework	Flask
Front End	HTML, CSS, JavaScript, Bootstrap
Database	MySQL

VIII. APPLICATIONS

- Automated bug prioritization: Predicts the priority level of newly reported bugs from historical information.
- Defect prediction: Identifies software components or patterns associated with increased defect risk.
- Bug triaging support: Helps organize large numbers of incoming reports into useful categories.
- Faster maintenance: Helps teams identify reports that require earlier human attention.
- Prediction history: Stores previous predictions and outcomes for analysis and model improvement.
- Research and education: Demonstrates the integration of artificial intelligence, machine learning, and full-stack software development.

8.1 Challenges and Limitations

Data quality strongly affects bug-prioritization and defect-prediction models. Historical bug reports may contain missing fields, duplicate reports, inconsistent severity labels, noisy text, or changes in reporting practices over time. Class imbalance can cause a model to perform well on common low-priority cases while performing poorly on rare high-priority cases. Data leakage is another important risk when information that becomes available only after a bug is resolved is accidentally used as a training feature.

Generalization across projects is also difficult because different organizations use different workflows, labels, programming languages, reporting systems, and release processes. Explainability is important when predictions influence maintenance decisions, especially when developers need to understand why a bug was assigned a particular priority. Models should therefore be validated on realistic data and treated as decision-support tools. Security, privacy, and access control are also required when bug reports contain confidential source-code or customer information.

IX. CONCLUSION AND FUTURE WORK

Artificial intelligence and machine learning provide promising approaches for assisting software bug prioritization and prediction. The surveyed literature demonstrates the use of supervised learning, ensemble models, deep learning, and automated bug-report analysis for software maintenance tasks. Decision Trees, Random Forest, Support Vector Machines, and related classifiers can be applied to historical bug information to predict priority and defect-related outcomes. At the same time, recurring challenges include class imbalance, noisy labels, data leakage, limited explainability, and differences between software projects.

Future work should focus on larger and more representative datasets, time-aware validation, explainable predictions, improved text representation, and integration of structured and unstructured software data. The proposed platform can be extended with transformer-based language models, continuous learning, developer feedback, automatic duplicate detection, and integration with issue-tracking systems. Future studies should also compare model performance across projects and evaluate whether AI-assisted prioritization reduces resolution time without creating harmful prioritization errors. In practical deployment, model outputs should support—not replace—developer review and project-specific decision making.

REFERENCES

- [1] F. Matloob *et al.*, “Software Defect Prediction Using Supervised Machine Learning Techniques: A Systematic Literature Review,” *Intelligent Automation & Soft Computing*, vol. 29, no. 2, pp. 403–421, 2021.
- [2] “A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools,” *Engineering Applications of Artificial Intelligence*, vol. 111, 104773, 2022.
- [3] N. C. Grattan *et al.*, “The need for more informative defect prediction: A systematic literature review,” *Information and Software Technology*, vol. 171, 107456, 2024.
- [4] “An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review,” *International Journal of Information Management Data Insights*, vol. 3, no. 1, 100153, 2023.
- [5] G. Long, J. Gong, H. Fang, and T. Chen, “Learning Software Bug Reports: A Systematic Literature Review,” 2025.
- [6] A. Daza *et al.*, “Industrial applications of artificial intelligence in software defects prediction: Systematic review, challenges, and future works,” *Computers and Electrical Engineering*, vol. 124, Part B, 110411, 2025.
- [7] “Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review,” *IEEE*, 2021.
- [8] “Software Fault Prediction Using Data Mining, Machine Learning and Deep Learning Techniques: A Systematic Literature Review,” *Computers & Electrical Engineering*, 2022.
- [9] M. A. Olaleye, “Predictive Analytics and Software Defect Severity: A Systematic Review and Future Directions,” *Scientific Programming*, vol. 2023, article 6221388, 2023.
- [10] Matloob, F., Aftab, S., Ahmad, M., Khan, M. A., Fatima, A., *et al.* (2021). Software Defect Prediction Using Supervised Machine Learning Techniques: A Systematic Literature Review. *Intelligent Automation & Soft Computing*, 29(2), 483–497. DOI: 10.32604/iasc.2021.017562.

Citation of this Article:

Dr. A. Antony Prakash, Ishwarya C, & Praveen Raj S. (2026). A Systematic Review of AI-Driven Software Bug Prediction and Prioritization. *Journal of Artificial Intelligence and Emerging Technologies (JAJET)*. 3(10), 1-6. Article DOI: <https://doi.org/10.47001/JAIET/2026.310001>

*** End of the Article ***