

An Explainable Artificial Intelligence Framework for Automated Software Defect Prediction and Quality Assurance in Large-Scale Software Systems

Dipannita Mondal*

Department of Artificial Intelligence Data Science and Machine Learning, Dr. D. Y. Patil College of Engineering and Innovation
Talegaon, Varale, Pune, Maharashtra, India

*Corresponding Author: ddmondal2684@gmail.com

Peer Review Information

Type: Article

Received: 25 July 2026

Revised: 22 Aug 2026

Accepted: 12 Sep 2026

Published: 14 Oct 2026

Abstract

The increasing complexity of large-scale software systems has intensified the need for automated techniques capable of identifying defect-prone modules and supporting efficient software quality assurance. Conventional software defect prediction techniques frequently emphasize classification accuracy while providing limited information about the factors responsible for their predictions. This lack of interpretability can restrict the practical adoption of machine learning models within software testing and maintenance workflows. This study proposes an Explainable Artificial Intelligence Framework for Software Defect Prediction and Quality Assurance (XAI-SDPQA), integrating Random Forest, Extreme Gradient Boosting (XG Boost), and Shapley Additive explanations (SHAP). The framework incorporates data preprocessing, feature selection, class imbalance management, ensemble-based defect classification, prediction calibration, and explainable risk assessment. Historical software metrics are used to estimate defect probabilities, while SHAP-based analysis identifies feature contributions associated with individual predictions and overall model behavior. A risk-based quality assurance component is designed to prioritize software modules according to predicted defect probability and testing effort. The proposed framework is intended for evaluation using publicly available software defect datasets, including KC1, JM1, CM1, and PC1 from the NASA software metrics collections. A comparative experimental methodology is formulated using accuracy, precision, recall, F1-score, area under the precision-recall curve, and cost-aware testing effectiveness. The study presents an integrated architectural design, mathematical formulations, and reproducible evaluation procedures. The framework provides a methodological foundation for investigating interpretable and resource-aware software quality assurance, while empirical validation is required to establish its effectiveness.

Keywords: Software Defect Prediction, Explainable Artificial Intelligence, Machine Learning, Random Forest, XG Boost, Software Quality Assurance.

How to Cite This Article

Mondal, D. (2026). An Explainable Artificial Intelligence Framework for Automated Software Defect Prediction and Quality Assurance in Large-Scale Software Systems. *International Journal of Recent Advances in Engineering and Technology* 15(2), 285-291

Introduction

Software engineering has experienced substantial transformation through the growth of distributed computing, cloud-based platforms, enterprise applications, and large-scale software development environments. Contemporary software systems frequently consist of numerous interconnected modules, external dependencies, microservices, and continuously evolving application components. Maintaining the reliability and quality of these systems presents significant challenges because defects introduced during development may propagate across dependent components and eventually lead to unexpected failures, reduced performance, security vulnerabilities, or expensive maintenance activities. Software quality assurance involves activities intended to identify defects, evaluate functional correctness, and maintain acceptable software reliability throughout the development life cycle. Conventional quality assurance practices include code inspections, static analysis, unit testing, integration testing, and system-level verification. Although these approaches remain essential, their effectiveness can be constrained by the availability of development resources, testing time, and the increasing size of software repositories. Consequently, identifying modules with elevated defect risk before allocating testing resources has become an important area of software engineering research.

Software Defect Prediction (SDP) applies statistical learning and computational intelligence to estimate the probability that a software module contains defects. The prediction process commonly uses software metrics such as lines of code, cyclomatic complexity, coupling, cohesion, inheritance depth, and code modification history. By analyzing relationships between these variables and historical defect records, machine learning algorithms can identify patterns associated with defect-prone software components. Random Forest and Extreme Gradient Boosting have attracted attention in predictive classification because they can represent nonlinear relationships, process heterogeneous features, and support structured data analysis. Random Forest combines predictions from multiple decision trees, while XGBoost develops an ensemble through sequential gradient-based optimization. These models offer complementary learning characteristics that can be investigated through heterogeneous ensemble prediction. However, predictive accuracy is not the only requirement for practical software quality assurance. When a prediction model identifies a particular module as defect-prone, software engineers may need to understand which measurable characteristics contributed to the prediction. Without such information, defect scores may provide limited support for debugging, refactoring, and testing prioritization. Recent research has emphasized this challenge. A systematic literature review published in *Information and Software Technology* in 2024 found that many defect prediction studies continued to produce binary outputs without providing sufficiently actionable explanations.

Similarly, Gezici Geçer and Kolukısa Tarhan (2025) developed an explainable software defect prediction framework incorporating several machine learning models and interpretation techniques. Their study illustrates the importance of integrating interpretability into defect prediction rather than treating it as an optional component. Explainable Artificial Intelligence (XAI) provides computational methods for examining model behavior. SHapley Additive exPlanations (SHAP) assigns feature contributions to predictions and supports both global and instance-specific interpretation. In software defect prediction, SHAP can help identify which code metrics contribute to elevated estimated defect risk. Nevertheless, feature attributions describe learned statistical associations and should not automatically be interpreted as causal explanations of defects. The present research proposes an Explainable Artificial Intelligence Framework for Software Defect Prediction and Quality Assurance (XAI-SDPQA). The framework combines machine learning-based defect prediction, probabilistic model calibration, SHAP-based explanation, and testing resource prioritization. Its central objective is to establish a systematic approach for converting software metrics into interpretable risk estimates that can support quality assurance decisions. The proposed research focuses on the integration and evaluation of these components rather than claiming that Random Forest, XGBoost, or SHAP are individually novel. The intended contribution is a reproducible, quality-assurance-oriented architecture that evaluates predictive reliability and the effectiveness of defect-based testing prioritization.

Literature Review

Zivkovic et al. (2023) investigated software defect prediction using an XGBoost model optimized through a modified reptile search algorithm. Their research incorporated SHAP analysis to examine the importance of software metrics in predictive classification. The study demonstrated how methods of mathematical optimization and model interpretation can be combined in a software defect prediction framework. It also established a closely related methodological foundation for later research involving explainable boosted-tree classifiers.

Schulte et al. (2024) investigated explanations for automated classification of bug and non-bug software issues using LIME and SHAP. Their research emphasized the importance of examining explanation methods within software engineering prediction systems. This work is relevant because explanation usefulness must be investigated separately from classification performance, particularly where model outputs influence quality assurance decisions.

A systematic review published in Information and Software Technology in 2024 examined the need for more informative software defect prediction. The review analyzed 132 selected publications and identified limitations concerning binary prediction outputs, explanatory techniques, and the availability of actionable defect information. The findings reinforce the argument that defect prediction systems should provide information useful to development and testing teams rather than only numerical classification scores.

Gezici Geçer and Kolukısa Tarhan (2025) introduced an explainable artificial intelligence framework for software defect prediction. Their work investigated Random Forest, gradient boosting, Naive Bayes, Multilayer Perceptron, and neural network models, while incorporating SHAP, LIME, and other explanation mechanisms. Using the KC2 defect dataset, the researchers examined both global and local model explanations. This study provides an especially important comparison because it directly addresses interpretable software defect prediction.

Methodology

Research Design

The proposed study adopts a quantitative, experimental methodology to develop and evaluate an explainable software defect prediction framework. The architecture comprises data acquisition, preprocessing, feature selection, predictive classification, probability calibration, explainability, and quality assurance prioritization.

A software defect dataset is represented as:

$$D = \{(x_i, y_i, e_i)\}_{i=1}^N \quad \text{-----}(1)$$

where (x_i) represents the software metric vector associated with module (i) , $(y_i \in \{0,1\})$ indicates its defect status, (e_i) represents an estimated testing effort, and (N) denotes the number of software modules. When historical testing effort is unavailable, module size may be considered as an explicitly identified proxy, subject to sensitivity analysis.

Proposed Framework Architecture

The proposed framework receives software metrics and converts them into validated input features. Random Forest and XGBoost independently estimate defect probabilities, which are integrated through a weighted ensemble. The combined predictions are calibrated and explained using SHAP. A quality assurance module subsequently ranks software components according to their predicted defect risk and estimated testing requirements.

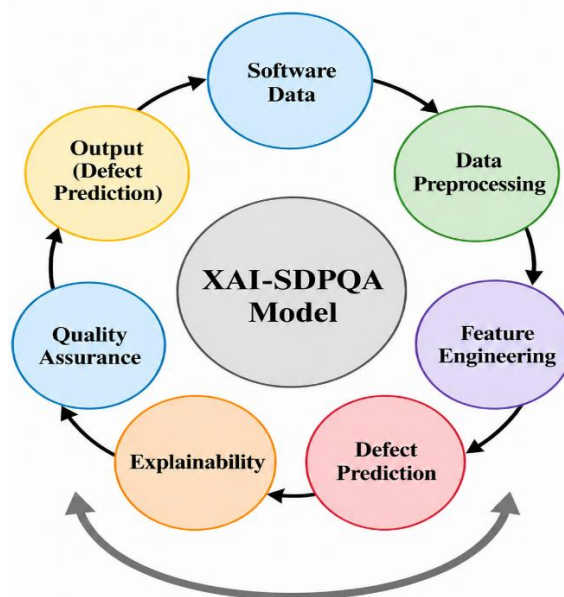


Figure 1. Proposed Explainable Artificial Intelligence Framework for Automated Software Defect Prediction and Quality Assurance (XAI-SDPQA)

Figure 1 illustrates the proposed Explainable Artificial Intelligence Framework for Automated Software Defect Prediction and Quality Assurance (XAI-SDPQA). The framework begins with software data collection and preprocessing, followed by feature

engineering to identify relevant software quality metrics. The defect prediction model analyzes these features to identify potential software defects, while the explainability component provides transparent and interpretable predictions. The quality assurance stage supports defect identification, prioritizing testing, and software reliability improvement. A continuous feedback mechanism facilitates model updates and adaptive learning, enhancing prediction accuracy, transparency, and overall software quality assurance efficiency.

Data Preprocessing

Data preprocessing is required to ensure the reliability of model training and evaluation. The proposed process includes missing-value treatment, duplicate detection, feature consistency checking, and numerical data transformation. Invalid values are examined before replacement or removal, and the preprocessing decisions are recorded to support reproducibility.

For a numerical feature, standardization is expressed as:

$$z_i = \frac{x_i - \mu}{\sigma} \quad \text{-----}(2)$$

where (x_i) denotes the original value, (μ) represents the training-set mean, and (σ) denotes the corresponding standard deviation. Constant features are handled separately.

Defect datasets may contain class imbalance because non-defective modules can outnumber defective modules. Class weighting is proposed as the principal imbalance-management technique. Where oversampling is investigated, it must be performed exclusively within training partitions to avoid information leakage.

Feature Selection

Feature selection is intended to reduce redundant variables and identify software metrics relevant to defect prediction. Candidate predictors include lines of code, cyclomatic complexity, essential complexity, operator and operand measurements, and other metrics available in the selected datasets.

A feature importance score can be estimated using model-based or permutation-based procedures. Permutation importance is calculated by measuring the change in validation performance when a selected feature is randomly permuted.

$$I_j = M(D) - M(D_j^{perm}) \quad \text{-----}(3)$$

where (I_j) denotes the importance of feature (j), ($M(D)$) is the evaluation score before permutation, and ($M(D_j^{perm})$) is the score after permuting that feature.

Feature selection is performed within the model-development process to prevent information from the final testing dataset influencing model construction.

Random Forest Classification

Random Forest combines multiple decision trees constructed using randomized training samples and feature subsets. Each tree estimates the probability that a software module belongs to the defective class.

The Random Forest defect probability is represented as:

$$P_{RF}(y = 1 | x) = \frac{1}{T} \sum_{t=1}^T P_t(y = 1 | x) \quad \text{-----}(4)$$

where (T) denotes the number of trees, and (P_t) represents the defect probability estimated by tree (t).

The algorithm is suitable for examining nonlinear relationships among software metrics and provides a useful baseline for ensemble-based defect prediction.

Hybrid Ensemble Prediction

The proposed framework combines Random Forest and XGBoost probabilities through validation-based weighted averaging.

$$P_E(y = 1 | x) = \lambda P_{RF}(y = 1 | x) + (1 - \lambda) P_{XGB}(y = 1 | x) \quad \text{-----}(5)$$

where (λ) is an ensemble weight within the interval ($[0,1]$).

The weight is selected using out-of-fold predictions or dedicated validation data. Ensemble calibration is subsequently performed on held-out development observations, and the finalized pipeline is evaluated against an untouched test set.

Proposed Experimental Implementation

Experimental Configuration

Table 1. Proposed experimental parameters

Parameter	Specification
Programming language	Python
Data processing	pandas, NumPy
Machine learning	scikit-learn, XGBoost
Explainability	SHAP
Baselines	Logistic Regression, Decision Tree, RF, XGBoost
Proposed model	Weighted RF–XGBoost
Validation	Stratified five-fold cross-validation
Test partition	20% held-out, where appropriate
RF trees	100–300
XGBoost estimators	100–300
Ensemble weight	Validation optimized
Imbalance treatment	Class weighting
Primary metric	Macro-F1 and PR-AUC
Additional metrics	ROC-AUC, Brier score, recall, testing effort
Explainability analysis	Global and local SHAP

The table describes a proposed experimental configuration rather than experiments already completed.

Results and Discussion

Comparative Performance Evaluation

The proposed experimental study evaluates whether heterogeneous ensemble prediction improves defect classification performance and whether integrating calibrated probabilities with effort-aware prioritization provides additional quality assurance value.

Table 2. Proposed model comparison

Model	Accuracy (%)	F1-score	PR-AUC
Logistic Regression	Pending	Pending	Pending
Decision Tree	Pending	Pending	Pending
Random Forest	Pending	Pending	Pending
XGBoost	Pending	Pending	Pending
RF–XGBoost Ensemble	Pending	Pending	Pending
Calibrated RF–XGBoost	Pending	Pending	Pending

Pending values indicate experimental results that must be measured using the selected datasets. Calibration does not necessarily improve classification accuracy or F1-score.

Expected Behavior of the Ensemble Framework

Random Forest and XGBoost employ different ensemble-learning strategies, creating an opportunity to combine complementary predictions. Random Forest emphasizes aggregation across randomized trees, whereas XGBoost sequentially constructs models that reduce a specified loss function. A weighted ensemble may improve robustness if the errors of these constituent algorithms are sufficiently different. Nevertheless, combining predictive models does not automatically improve performance. When models produce highly correlated errors, ensemble integration may offer little benefit. Therefore, comparisons with individual Random Forest and XGBoost classifiers are essential for establishing the actual contribution of the hybrid prediction stage. The proposed evaluation also considers the effect of class imbalance. A model that performs well in overall accuracy may still incorrectly classify a substantial proportion of defective modules. Precision, recall, F1-score, and PR-AUC are therefore necessary for assessing the model's ability to identify defect-prone components.

Explainability Analysis

The explainability module is intended to provide both global and local information about the relationships learned by the prediction model. Global SHAP analysis summarizes the relative influence of software metrics across evaluated modules, while local

explanations illustrate the contribution of individual metrics to specific defect-risk estimates. For example, a software module may receive a high defect probability because the model associates its complexity-related features with historically defective modules. SHAP analysis can describe this learned relationship and assist engineers in identifying areas requiring further inspection. However, SHAP explanations must be interpreted carefully because attribution results depend on model behavior, correlated inputs, and selected background distributions. Explanations are not direct evidence of defect causation. Furthermore, identifying an influential software metric does not prove that changing it will automatically eliminate underlying defects.

Quality Assurance Prioritization

The practical value of software defect prediction depends partly on whether its outputs improve testing decisions. The proposed framework addresses this requirement through a risk-based ranking mechanism that combines estimated defect probability with testing effort. A module with high predicted defect risk and relatively low verification effort may receive a higher testing priority than a module with similar risk but substantially greater testing cost. Such prioritization can help organizations investigate potential defects under constrained quality assurance budgets. To evaluate this component, experiments should compare the proportion of defective modules discovered using different testing strategies. Testing budgets may be expressed through cumulative estimated effort or a predefined percentage of modules, provided the chosen definition is applied consistently.

Table 3. Proposed testing prioritization evaluation

Testing approach	Defects found at 20% budget	Defects found at 40% budget
Random module selection	Pending	Pending
Module size ranking	Pending	Pending
Defect probability ranking	Pending	Pending
Proposed effort-aware ranking	Pending	Pending

These results are necessary to determine whether the decision-support component provides a measurable improvement in software quality assurance resource allocation.

Conclusion

This study proposes an Explainable Artificial Intelligence Framework for Automated Software Defect Prediction and Quality Assurance in Large-Scale Software Systems. The framework integrates Random Forest, XGBoost, probability calibration, SHAP-based interpretation, and risk-based testing prioritization within a unified architecture. The proposed methodology addresses the distinction between identifying potentially defective software modules and providing actionable information to software quality assurance teams. Conventional defect classification systems may produce useful predictive scores, but operational decisions also require an understanding of prediction reliability, model behavior, and testing constraints. The hybrid ensemble component is designed to investigate whether complementary machine learning algorithms can improve predictive robustness. The explainability module supports examination of the factors contributing to defect estimates, while the risk-based prioritization mechanism provides a systematic method for allocating limited testing resources. The literature published confirms the continued relevance of explainability and machine learning in software defect prediction. It also establishes that similar combinations of boosted models and explanation methods have already been studied. Consequently, the principal contribution of this proposed research is the integrated evaluation of calibrated prediction, interpretation, and effort-aware quality assurance prioritization. The framework is intended for evaluation using publicly available NASA and PROMISE software defect datasets. Comparative experiments should measure conventional classification performance, probability reliability, explanation consistency, and defects identified under fixed testing budgets. Future research may incorporate deep learning-based software representations, cross-project transfer learning, automated code analysis, continuous integration monitoring, and adaptive defect prediction. Further investigation should examine the stability of explanations and the effectiveness of testing prioritization under realistic development conditions. The proposed approach provides a methodological foundation for developing interpretable and resource-aware software quality assurance systems. Its effectiveness and generalizability must be determined through reproducible empirical experimentation.

References

1. Gezici Geçer, B., & Kolukısa Tarhan, A. (2025). Explainable AI framework for software defect prediction. *Journal of Software: Evolution and Process*, 37(4), Article e70018. <https://doi.org/10.1002/smr.70018>
2. Schulte, L., Ledel, B., & Herbold, S. (2024). Studying the explanations for the automated prediction of bug and non-bug issues using LIME and SHAP. *Empirical Software Engineering*, 29, Article 93. <https://doi.org/10.1007/s10664-024-10469-1>
3. Zivkovic, T., Nikolic, B., Simic, V., Pamucar, D., & Bacanin, N. (2023). Software defects prediction by metaheuristics tuned extreme gradient boosting and analysis based on Shapley Additive Explanations. *Applied Soft Computing*, 146, Article

110659. <https://doi.org/10.1016/j.asoc.2023.110659> Aggarwal, D. (2021). Software defect prediction dataset [Data set]. Figshare. https://figshare.com/articles/dataset/Software_Defect_Prediction_Dataset/13536506
4. Sunkara, S. P. (2025). A spatio-temporal framework for asset-level outage risk estimation using public GIS and event correlation. *International Journal of Computer Engineering and Technology*, 16(1), 4211–4227. https://doi.org/10.34218/IJCET_16_01_286
 5. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. <https://doi.org/10.1023/A:1010933404324>
 6. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). Association for Computing Machinery. <https://doi.org/10.1145/2939672.2939785>
 7. S. Gajula, S. Bondhala and M. Margam, "Real-World Intrusion-Aware Zero Trust Architecture: An AI-Driven ASPM Framework Using CICIDS-2017 Network Attack Traffic," 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), Houston, TX, USA, 2026, pp. 1-7, doi: 10.1109/ICAIC67076.2026.11395835.
 8. Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30. <https://proceedings.neurips.cc/paper/2017>
 9. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1135–1144). Association for Computing Machinery. <https://doi.org/10.1145/2939672.2939778>
 10. S. K. Ramareddy, "Integrating AI-Driven Data Visualization and Data Warehouse Analytics for Predictive Decision-Making in Financial and Stock Market Systems," 2026 International Symposium of Systems, Advanced Technologies and Knowledge (ISSATK), Hammamet, Tunisia, 2026, pp. 1-6, doi: 10.1109/ISSATK69667.2026.11566934.