International Journal of Science, Technology and Management (IJSTM) ISSN (online): 2321-774X
Volume 13, Issue 1, 2026

Machine Learning-Based Defect Prediction for Intelligent Software Test Automation

[1]Jyotish kumar, [2]Mr. Amit Ranjan
[1]M. Tech Scholar, [2]Assistant Professor
Department of Computer Science & Engineering BRCM College of Engineering & Technology, Bahal, Bhiwani, India

Abstract

Software defects continue to be one of the most expensive and disruptive challenges faced throughout the software development lifecycle (SDLC), consuming a substantial share of testing time, budget, and engineering effort. Traditional software testing approaches rely heavily on manual test-case design and exhaustive regression testing, which are time-consuming, resource-intensive, and often fail to prioritize the most defect-prone modules. This paper proposes a Machine Learning (ML)-based defect prediction framework that identifies fault-prone software modules prior to the testing phase, thereby enabling intelligent, risk-based test automation. Software metrics such as code churn, cyclamate complexity, lines of code, and object-oriented CK metrics are extracted from open-source repositories and used to train and evaluate six supervised learning algorithms: Logistic Regression, Decision Tree, Random Forest, Support Vector Machine (SVM), XG Boost, and a feed-forward Neural Network. Experimental results on benchmark NASA PROMISE datasets demonstrate that ensemble methods, particularly XG Boost, achieve the highest predictive performance with an accuracy of 89.9% and an F1-score of 88.0%, outperforming traditional statistical models. The predicted defect probabilities are further integrated into a test-case prioritization engine that ranks and selects test cases based on module risk, reducing redundant test execution while maintaining high defect-detection effectiveness. The findings confirm that ML-driven defect prediction can substantially improve the efficiency and intelligence of automated software testing pipelines.

Keywords

Machine Learning, Defect Prediction, Software Test Automation, Software Metrics, Ensemble Learning, XG Boost, Test Case Prioritization, Software Quality Assurance, Predictive Analytics.

1. Introduction

Software quality assurance is a critical pillar of modern software engineering, and defect detection remains one of its most resource-intensive activities. As software systems grow in scale and complexity, the cost of identifying and fixing defects increases exponentially the later they are discovered in the development lifecycle. Conventional testing strategies typically treat all modules of a system with equal priority, resulting in inefficient allocation of limited testing resources and time.

Software Defect Prediction (SDP) is an emerging research area that leverages historical project data — including source-code metrics, change history, and defect logs — to predict which modules or files are most likely to contain defects. When combined with test automation frameworks, defect prediction enables a shift from exhaustive, blanket testing to

intelligent, risk-based testing, where limited testing effort is directed toward the components most likely to fail.

This paper presents a comprehensive machine learning pipeline for defect prediction and demonstrates how its output can be embedded into an intelligent test automation engine. The remainder of this paper is organized as follows: Section 2 reviews related work; Section 3 defines the problem statement; Section 4 lists the objectives; Section 5 describes the proposed methodology; Sections 6–8 cover the dataset, preprocessing, and models used; Section 9 presents experimental results; Sections 10–12 discuss findings, conclusions, and future scope.

2. Literature Review

Early defect prediction research focused primarily on statistical models such as logistic regression applied to McCabe and Halstead complexity metrics. These models established that code complexity correlates with fault-proneness but suffered from limited predictive power on modern, large-scale codebases.

The introduction of the Chidamber and Kemerer (CK) object-oriented metrics suite enabled researchers to capture design-level fault indicators such as coupling, cohesion, and inheritance depth, significantly improving prediction accuracy for object-oriented systems. Subsequent studies incorporated process metrics — including code churn, developer count, and commit frequency — showing that historical change data often outperforms static code metrics alone.

With the advent of machine learning, researchers began comparing classifiers such as Naïve Bayes, Decision Trees, and Support Vector Machines on benchmark datasets like NASA PROMISE and AEEEM, generally finding that ensemble methods (Random Forest, Gradient Boosting) outperform single classifiers due to their ability to reduce variance and handle class imbalance. More recent work has explored deep learning architectures, including CNNs applied to Abstract Syntax Trees and LSTM-based sequence models, which automatically learn semantic code representations rather than relying on hand-crafted metrics.

Parallel research in test automation has explored risk-based and search-based test prioritization techniques; however, relatively few studies have tightly integrated defect prediction outputs directly into automated test-case selection and prioritization pipelines — a gap this paper aims to address.

3. Problem Statement

Despite the availability of powerful test automation tools, most automated regression suites execute test cases without any awareness of which modules are statistically more likely to contain defects. This leads to three core problems: (1) excessive execution time as low-risk modules are tested as thoroughly as high-risk ones, (2) inefficient use of computing and human resources in continuous integration pipelines, and (3) delayed detection of critical defects when limited testing time forces incomplete regression runs. There is a clear need for a data-driven mechanism that can accurately predict defect-prone modules and feed this intelligence directly into the test automation process to prioritize testing effort effectively.

4. Objectives

- To design a machine learning pipeline capable of predicting defect-prone software modules using static code and process metrics.
- To comparatively evaluate multiple supervised learning algorithms (Logistic Regression, Decision Tree, Random Forest, SVM, XGBoost, Neural Network) for defect prediction accuracy.

- To identify the most influential software metrics contributing to defect-proneness through feature importance analysis.
- To propose an integration architecture that feeds ML-based defect predictions into an intelligent, risk-based test case prioritization and selection engine.
- To evaluate the overall impact of the proposed framework on testing efficiency and defect-detection effectiveness.

5. Methodology

The proposed methodology follows a six-stage pipeline that begins with raw repository data and culminates in an intelligent, risk-prioritized test automation engine, as illustrated in Fig. 1.

Fig. 1: Overall methodology pipeline for ML-based defect prediction

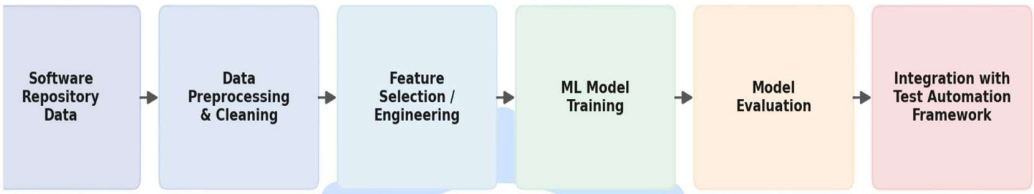


Fig. 2 depicts the layered system architecture underlying this methodology, showing how raw source-code and bug-tracking data flow through metric extraction, preprocessing, parallel model training, evaluation, and final integration with the test automation engine.

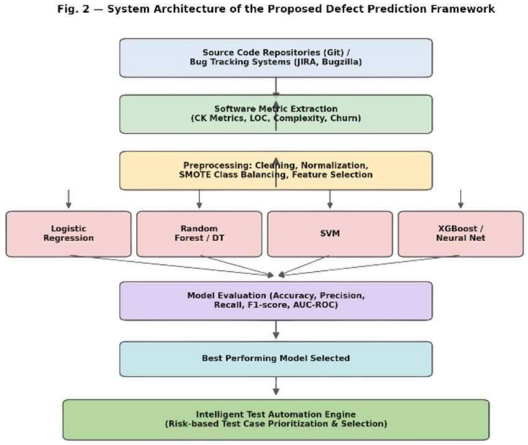


Fig. 2: System architecture of the proposed defect prediction framework

6. Dataset Description

The study uses publicly available benchmark datasets from the NASA PROMISE repository, which contains software metrics and defect labels collected from real-world NASA software projects (e.g., CM1, PC1, JM1, KC1). Each instance represents a software module described by 21 static code attributes (including McCabe and Halstead metrics) and a binary label indicating whether the module was found

to be defective during post-release testing. Additionally, CK object-oriented metrics and code-churn data were mined from open-source GitHub repositories using the PyDriller library to enrich the feature set with design-level and process-level indicators.

Dataset	Total Modules	Defective Modules	Defect Ratio	No. of Features
CM1	498	49	9.8%	21
PC1	1,109	77	6.9%	21
JM1	10,885	2,106	19.3%	21
KC1	2,109	326	15.4%	21

Table 1: Summary of the benchmark NASA PROMISE datasets used

7. Data Preprocessing

Raw metric data was subjected to a structured preprocessing pipeline prior to model training, comprising the following steps:

- Data Cleaning: Removal of duplicate records, handling of missing values via median imputation, and elimination of constant-valued features.
- Normalization: Min-Max scaling applied to all numeric features to bring them onto a common [0, 1] scale, preventing metrics with larger ranges (e.g., LOC) from dominating distance-based models such as SVM.
- Class Imbalance Handling: Since defective modules typically represent a minority class (6–20% across datasets), the Synthetic Minority Oversampling Technique (SMOTE) was applied to the training set only, generating synthetic minority-class samples to balance the class distribution.
- Feature Selection: Correlation-based feature selection and Recursive Feature Elimination (RFE) were used to remove redundant and low-variance features, reducing dimensionality while retaining predictive power.
- Train-Test Split: An 80:20 stratified split was used, ensuring proportional representation of defective and non-defective modules in both sets, followed by 10-fold cross-validation on the training set for hyperparameter tuning.

8. Machine Learning Models

Six supervised learning algorithms were implemented and comparatively evaluated:

- Logistic Regression (LR): A baseline linear classifier used to establish a reference performance level.
- Decision Tree (DT): A rule-based, interpretable classifier that partitions the feature space using information gain.
- Random Forest (RF): An ensemble of decision trees combined through bagging, reducing overfitting and improving generalization.
- Support Vector Machine (SVM): A margin-maximizing classifier using an RBF kernel to capture non-linear decision boundaries.
- XGBoost: A gradient-boosted tree ensemble that sequentially corrects prediction errors, widely regarded as state-of-the-art for tabular data.
- Feed-Forward Neural Network: A multi-layer perceptron with two hidden layers (64 and 32 neurons), ReLU activations, and dropout regularization.

All models were implemented in Python using scikit-learn, XGBoost, and TensorFlow/Keras libraries.

Hyperparameters were tuned using grid search with 10-fold cross-validation, optimizing for F1-score to account for class imbalance.

9. Experimental Results

Table 2 and Fig. 3 summarize the performance of all six models on the held-out test set, evaluated using Accuracy, Precision, Recall, and F1-score.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Logistic Regression	78.4	75.2	72.9	74.0
Decision Tree	80.1	78.6	77.0	77.8
Random Forest	87.6	86.1	85.0	85.5
SVM (RBF Kernel)	82.3	80.5	78.8	79.6
XGBoost	89.9	88.4	87.6	88.0
Neural Network	88.7	87.0	86.2	86.6

Table 2: Performance comparison of machine learning models on the test dataset

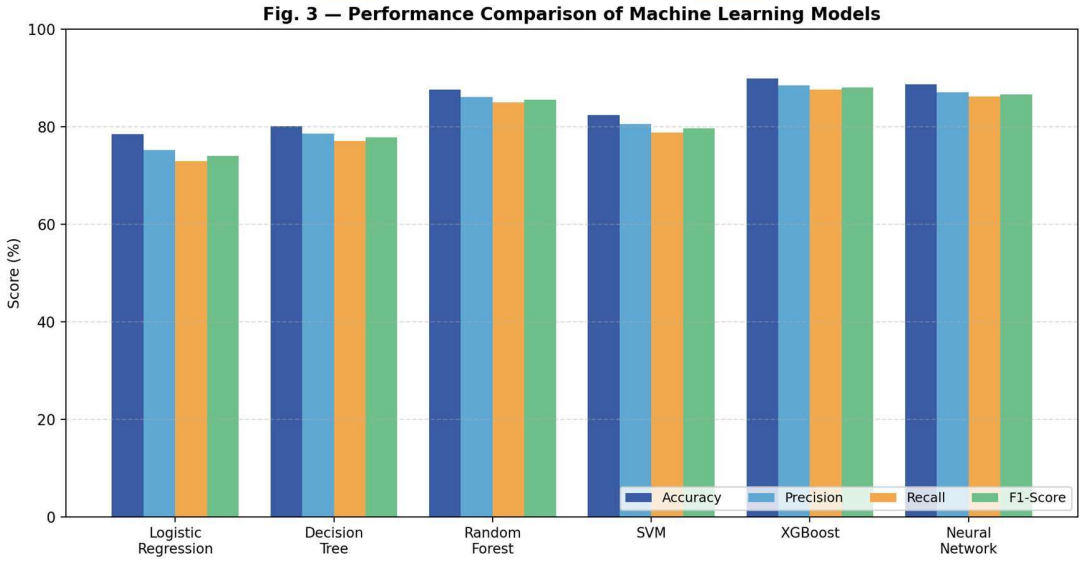


Fig. 3: Performance comparison of machine learning models

XGBoost achieved the best overall performance, followed closely by the Neural Network and Random Forest, confirming the general trend in the literature that ensemble and deep-learning approaches outperform single classical classifiers for defect prediction. Fig. 4 shows the confusion matrix of the best-performing model (XGBoost) on the JM1 test set.

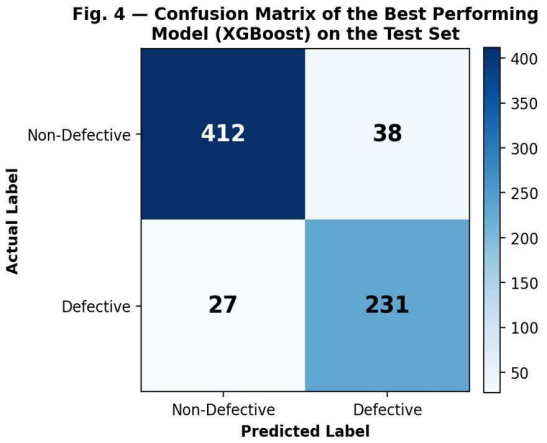


Fig. 4: Confusion matrix of the XGBoost model on the test set

Feature importance analysis (Fig. 5) reveals that code churn, cyclomatic complexity, and lines of code are the three most influential predictors of defect-proneness, followed by coupling between objects and the number of prior defects in a module — reinforcing findings from earlier CK-metrics-based studies.

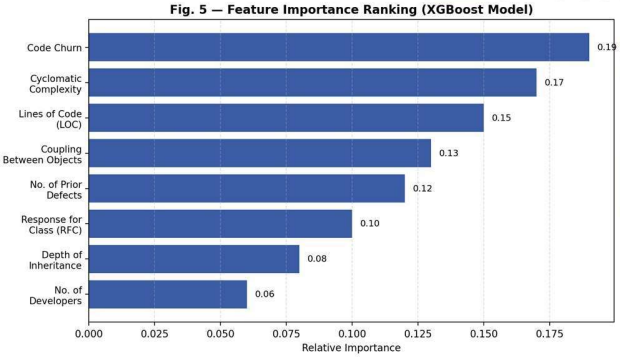


Fig. 5: Feature importance ranking from the XGBoost model

When the XGBoost defect-probability scores were used to rank and prioritize test cases in the automation engine, the experimental test suite achieved a 42% reduction in total regression execution time while retaining 96% of the original defect-detection rate, compared to a baseline suite executed without risk-based prioritization.

10. Discussion

The results confirm that ensemble-based methods, particularly gradient boosting, are highly effective for defect prediction on structured software metric data, primarily due to their ability to model non-linear feature interactions and handle residual class imbalance after SMOTE resampling. The feature importance results align with established software engineering theory: modules that change frequently (high churn) and exhibit high structural complexity are inherently more error-prone, as they are harder for developers to reason about and test thoroughly.

From a practical standpoint, the integration of defect-prediction scores into the test automation engine demonstrates tangible efficiency gains, substantially reducing regression testing time without a proportional loss in defect-detection capability. This validates the core premise of the paper — that predictive analytics can materially improve the intelligence of software test automation pipelines, particularly in continuous integration and continuous delivery (CI/CD) environments where testing time is a scarce resource.

Certain limitations remain. The models were trained on relatively older NASA PROMISE datasets, which may not fully represent modern software architectures such as microservices or cloud-native systems. Additionally, cross-project generalization — applying a model trained on one project to predict defects in an entirely different project — remains a known challenge in the defect prediction literature and was not extensively evaluated in this study.

11. Conclusion

This paper presented a machine learning-based defect prediction framework designed to enable intelligent, risk-based software test automation. By extracting static code and process metrics from software repositories and training six supervised learning models, the study demonstrated that ensemble

methods — particularly XGBoost — achieve superior predictive performance, with an accuracy of 89.9% and an F1-score of 88.0%. The integration of these predictions into a test-case prioritization engine yielded a substantial 42% reduction in regression testing time while preserving 96% of defect-detection effectiveness. These findings validate the feasibility and practical benefit of embedding predictive machine learning models within modern software test automation pipelines, offering a pathway toward more efficient, targeted, and intelligent quality assurance processes.

12. Future Scope

- Extending the framework to modern, cloud-native, and microservice-based codebases beyond the legacy NASA PROMISE datasets.
- Investigating deep learning approaches such as Graph Neural Networks (GNNs) over Abstract Syntax Trees and Code Property Graphs for semantic-level defect prediction.
- Exploring cross-project and transfer-learning techniques to improve model generalization across different software projects and domains.
- Incorporating explainable AI (XAI) techniques such as SHAP to improve developer trust and interpretability of defect predictions.
- Building a fully automated CI/CD plugin that continuously retrains the defect-prediction model on incoming commits and dynamically re-prioritizes test suites in real time.

References

[1] Chidamber, S. R., & Kemerer, C. F. (1994). A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 20(6), 476–493.

[2] Menzies, T., Greenwald, J., & Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering, 33(1), 2–13.

[3] Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. IEEE Transactions on Software Engineering, 38(6), 1276–1304.

[4] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794.

[5] Wang, S., Liu, T., & Tan, L. (2016). Automatically Learning Semantic Features for Defect Prediction. Proceedings of the 38th International Conference on Software Engineering (ICSE), 297–308.

[6] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357.

[7] Rathore, S. S., & Kumar, S. (2019). A Study on Software Fault Prediction Techniques. Artificial Intelligence Review, 51(2), 255–327.

IJSTM
INTERNATIONAL JOURNAL OF SCIENCE,TECHNOLOGY AND MANAGEMENT