

A Survey on Static and Dynamic Analysis ways for Software Security

Rounak Dey

Department of AI

FET.Jain Deemed-to-be University

Bengaluru-562112

rounakriju2003@gmail.com

Dr. K S Arvind

Department of CSE

FET.Jain Deemed-to-be University

Bengaluru-562112

Ks.arvind@jainuniversity.ac.in

Abstract—Software defect prediction is a crucial element of Software Quality Assurance (SQA), aimed at relating defect-prone modules beforehand in the development lifecycle. With the rapid growth in software complexity, traditional defect discovery approaches such as manual testing and rule-based systems have become inadequate. Machine learning (ML) techniques offer a data-driven result by analyzing historical software data and predicting implicit defects with high accuracy. This survey provides a comprehensive review of machine learning techniques for software defect prediction, including traditional models, ensemble learning, deep learning, and transformer-based approaches. It also examines datasets, feature engineering strategies, evaluation criteria, and cross-project prediction styles. Likewise, the paper discusses challenges such as class imbalance, data quality issues, and model interpretability, along with emerging trends like explainable AI and large language models. The study concludes that ML-based defect prediction significantly enhances software quality, although further exploration is needed to improve generalization and transparency.

Keywords—Software defect prediction, Machine learning, Deep Learning, Software Quality Assurance, Data Mining, AI

I. INTRODUCTION

Software systems are getting increasingly complex due to rapid technological advancements and growing user demands. This complexity increases the liability of defects, which can lead to system failures, security vulnerabilities, and financial losses. Detecting defects beforehand in the software development lifecycle is essential to reduce maintenance costs and improve overall software quality.

Traditional defect discovery styles, such as manual testing and static analysis, are frequently time-consuming and inefficient for large-scale systems. These styles also depend heavily on human expertise and may fail to detect hidden or complex defects.

Machine learning (ML) techniques have surfaced as a powerful solution for software defect prediction. By analyzing historical data and software metrics, ML models can identify patterns associated with defective modules and predict implicit defects before deployment. This enables developers to concentrate testing efforts on high-risk areas, improving efficiency and reducing costs.

This survey aims to provide a detailed analysis of ML techniques used in software defect prediction, covering datasets, feature engineering, models, challenges, and future directions.

II. BACKGROUND AND OVERVIEW OF SOFTWARE TESTING

Software defect prediction is a classification problem where modules are labeled as defective or non-defective based on historical data.

Traditional Approaches: Manual code reviews, Rule-based defect discovery, Static analysis tools.

Machine Learning Approaches: Supervised learning (classification), Unsupervised learning (clustering), Deep learning.

ML-based approaches significantly outperform traditional methods in terms of accuracy and scalability.

III. DATASETS USED IN DEFECT PREDICTION

PROMISE Dataset: The PROMISE dataset is one of the most extensively used datasets in defect prediction exploration. It contains multiple open-source Java systems with object-oriented metrics such as coupling, cohesion, and inheritance.

NASA Dataset: The NASA dataset consists of mission-critical software developed in C/C++. It includes metrics such as cyclomatic complexity, Halstead metrics, and code size, making it useful for assessing ML models.

Defects4J Dataset: Defects4J contains real-world bugs from Java systems along with test cases. It is extensively used for assessing defect prediction models in realistic scripts.

IV. FEATURE ENGINEERING

Feature engineering plays a critical role in improving model performance.

Code Metrics: Lines of Code (LOC) measures code size; Cyclomatic Complexity indicates code complexity.

Process Metrics: Code churn measures changes in code over time; Developer activity tracks contributions and variations.

Feature Selection Techniques: Correlation analysis, Recursive Feature Elimination (RFE), Principal Component Analysis (PCA).

VI. MACHINE LEARNING MODELS

A. Traditional Models

1) **Naive Bayes:** A probabilistic model that assumes feature independence. It is simple, fast, and effective for small datasets.

2) **Logistic regression:** A linear model used for binary classification. It provides interpretable results and is often used as a baseline.

3) **Decision Trees:** Tree-based models that split data based on feature values. They are easy to interpret but prone to overfitting.

4) **Support Vector Machines (SVM):** SVM finds an optimal boundary between classes. It provides high accuracy but requires high computational resources.

B. Ensemble Styles

1) **Random Forest:** Combines multiple decision trees to improve accuracy and reduce overfitting. It is robust and extensively used.

2) **XGBoost:** An advanced gradient boosting algorithm known for high performance and scalability.

C. Deep Learning Models

1) **Neural Networks:** Capable of learning complex patterns and connections in data.

2) **LSTM (Long Short-Term Memory):** Captures sequential dependencies in code changes.

3) **CNN (Convolutional Neural Networks):** Extracts features from code representations.

D. Transformer-Based Models CodeBERT, GraphCodeBERT, and GPT-based models provide deeper semantic understanding of code.

V. CROSS-PROJECT DEFECT PREDICTION

Cross-project defect prediction (CPDP) uses data from one project to predict defects in another. Techniques include transfer learning and domain adaptation. This approach is useful when project-specific data is limited.

VII. EVALUATION METRICS

Accuracy: Overall correctness. Precision: Ratio of correctly predicted defective modules to all predicted defective modules. Recall: Ratio of correctly predicted defective modules to all actual defective modules. F1-Score: Harmonic mean of precision and recall. ROC-AUC: Evaluates the model's capability to distinguish between classes across different thresholds.

VIII. RELATIVE ANALYSIS

Model	Accuracy	Complexity	Interpretability
Naive Bayes	Moderate	Low	High
SVM	High	High	Medium
Random Forest	High	Medium	Medium
Deep Learning	Very High	High	Low

IX. CHALLENGES

1) Class Imbalance in Datasets: Occurs when defective modules are much fewer than non-defective ones. Models may become biased toward the majority class.

2) Poor Data Quality: Noise, missing values, and inconsistent labeling can lead to unreliable models.

3) Lack of Model Interpretability: Advanced models like deep learning act as black boxes, making it difficult for developers to understand predictions.

4) Generalization Issues: Models often fail to generalize across different programming languages or development practices.

X. FUTURE DIRECTIONS

Explainable AI (XAI): Improving transparency. Federated Learning: Training models across organizations without sharing sensitive data. LLM-Based Prediction: Using large language models to analyze code and predict defects.

XI. DISCUSSION

Machine learning has significantly improved software defect prediction. Traditional models provide simplicity, while ensemble and deep learning models offer advanced accuracy. Combining different models and using hybrid approaches can improve performance.

XII. CONCLUSION

This survey highlights the significance of machine learning techniques in software defect prediction. ML models enable early identification of defect-prone modules, reducing development costs and improving software quality. Ensemble and deep learning models demonstrate superior performance, while emerging approaches such as transformer-based models and explainable AI offer promising future directions.

REFERENCES

- 1) T. Menzies et al., 'Data Mining Static Code Attributes,' IEEE, 2007.
- 2) N. Nagappan et al., 'Static Analysis Tools Comparison,' IEEE, 2006.
- 3) T. Zimmermann et al., 'Cross-Project Defect Prediction,' IEEE, 2009.
- 4) J. Lessmann et al., 'Benchmarking Classification Models,' IEEE, 2008.
- 5) PROMISE Repository, Software Defect Datasets.
- 6) NASA Metrics Data Program.
- 7) Defects4J Dataset Documentation.

- 8) Breiman, 'Random Forests,' Machine Learning, 2001.
- 9) Chen et al., 'XGBoost: A Scalable Tree Boosting System,' 2016.
- 10) Hochreiter & Schmidhuber, 'LSTM,' 1997.
- 11) Krizhevsky et al., 'CNN,' 2012.
- 12) Vaswani et al., 'Attention is All You Need,' 2017.
- 13) CodeBERT Research Paper, 2020.
- 14) GraphCodeBERT Paper, 2021.
- 15) GPT Models for Code Analysis, 2023.
- 16) Khoshgoftaar et al., 'Defect Prediction Models,' IEEE.
- 17) Hall et al., 'Systematic Literature Review on Defect Prediction,' 2012.
- 18) Sun et al., 'Deep Learning for Software Engineering,' 2019.
- 19) Zhang et al., 'Transfer Learning in Defect Prediction,' 2018.
- 20) IEEE Software Testing Standards, 2018.
- 21) M. D'Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in *Proc. MSR*, 2010.
- 22) Y. Shin and L. Williams, "Can Traditional Fault Prediction Models Be Used for Vulnerability Prediction?" in *Empirical Software Engineering*, 2013.
- 23) F. Rahman and P. Devanbu, "How, and Why, Process Metrics Are Better," in *Proc. ICSE*, 2013.
- 24) E. Giger, M. D'Ambros, M. Pinzger, and H. Gall, "Method-Level Bug Prediction," in *Proc. ESEC/FSE*, 2012.
- 25) S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting Faults from Cached History," in *Proc. ICSE*, 2007.
- 26) A. E. Hassan, "Predicting Faults Using the Complexity of Code Changes," in *Proc. ICSE*, 2009.
- 27) T. Mende and R. Koschke, "Effort-Aware Defect Prediction Models," in *Proc. ESEM*, 2010.
- 28) C. Catal and B. Diri, "A Systematic Review of Software Fault Prediction Studies," *Expert Systems with Applications*, 2009.
- 29) D. Radjenović, M. Heričko, R. Torkar, and A. Živković, "Software Fault Prediction Metrics: A Systematic Literature Review," *Information and Software Technology*, 2013.
- 30) Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A General Software Defect-Proneness Prediction Framework," *IEEE Transactions on Software Engineering*, 2011.