Classifying Code Quality in Java-Based Open Source Software Projects Using Machine Learning Techniques and Contribution Analysis

H. T. Welagedara and W. V. S. K. Wasalthilaka

Faculty of Computing, Sabaragamuwa University of Sri Lanka, Sri Lanka.
hashiniwel@gmail.com

Abstract. Geographically dispersed volunteer teams can achieve collaborative and transparent processes with Open Source Software Development (OSSD). While it outperforms traditional methodologies, challenges remain in preserving code quality, managing third-party dependencies, leading to compatibility issues, and inconsistencies in developer contributions that can lead to code redundancies. Java as the foundation of software development, has fostered numerous open-source projects, enhancing research dependability. This research proposes a machine learning model that classifies code quality in Java-based open-source software projects by analyzing contribution metrics. Popular machine learning techniques used for software quality prediction, such as Regression, Decision Trees, Random Forest, Support Vector Machine and Bayesian Learning, XGBoost and Multi-Layer Perceptron are used, as well as established software quality metrics to measure the developer's contribution using source code such as Lines of Code (LOC), Coupling Between Objects (CBO), Response for a Class (RFC), etc. The proposed model was evaluated using a dataset, containing over 200,000 observations and software metrics extracted from open-source projects. Performance was measured using Precision, Accuracy, Recall and F1-Score. XGBoost shows the highest model accuracy, with 82%. The system was built using XGBoost, which allows developers and others to upload Java files. Based on the derived quality metrics, the system classifies the code quality into three categories: high, medium, and low. This analysis enhances Java OSSD projects by accurately evaluating code contributions, ensuring reliability and sustainability. Refining code review, prioritizing refactoring, and leveraging the best ML approach to classify code quality can strengthen development processes and advance OSSD efficiency.

Keywords: Code Contributions, Code Quality, Machine Learning, Open Source Software Development, Software Quality Metrics.

1 Introduction

1.1 Background

Modern software development follows Open-Source Software Development (OSSD) as the key framework for creating software through remote collaborative teams that defeat conventional methods with greater innovation and availability [1]. The model encourages global participation through community management and utilizes Java's cross-platform capabilities and large ecosystem features to impact Java-based projects, including Apache Hadoop, Spring Framework, and Eclipse, which offer strong research foundations due to widespread usage and large datasets on GitHub platforms [2]. The projects showcase OSSD's capabilities through Java's system features, while also presenting assessment requirements that require advanced quality evaluation approaches.

OSSD challenges code quality in Java-based development due to varying skill levels and contribution practices. Code redundancies arise from inconsistent contributions, increasing maintenance costs and reducing efficiency due to repeated functions [3].

An increase in machine learning techniques emerges as a solution to forecast and classify software quality because it delivers data-driven analysis of code metrics. The established software quality metrics Lines of Code (LOC), Coupling Between Objects (CBO), Response for a Class (RFC), Weighted Methods per Class (WMC), Lack of Cohesion in Methods (LCOM), Depth of Inheritance Tree (DIT) and Number of Children (NOC) provide numerical indicators to assess code complexity and maintainability and cohesion which helps identify quality issues [4]. While previous research, such as Meher and Mall [5] focused on bug classification in open-source repositories using machine learning, this research distinguishes itself by specifically addressing code quality classification in Java-based OSS projects using contribution metrics and quality thresholds with machine learning techniques unlike approaches based on Natural Language Processing (NLP) which requires another structured form of data, which can be obtained by converting the source code into a token stream using a programming language processor [6]. The code quality classification system uses precision and accuracy and recall and F1-score performance metrics to help developers conduct detailed code assessments and determine the best ML approach for process improvement in OSSD development [7]. The combination of machine learning metrics LOC and CBO, and DIT provides data-based code quality classification systems that use XGBoost to reach 82% accuracy levels according to this research. This research evaluates more than 200,000 observations to improve Java OSS quality assessment methods.

1.2 Purpose of the Research

Due to the distributed nature of Open-Source Software Development (OSS) projects, it is difficult to ensure consistent code quality because of heterogeneous contributors with different skills and coding styles. This leads to plural redundant code, dependency compatibility problems and technical debt, diminishing reliability and sustainability standards. Automated systematic quality assurance is lacking, which prevents meeting global development needs, and manual code reviews are infeasible for large-scale OSS projects.

Research Questions

The research discovers suitable software quality metrics for Java OSS projects' code quality assessment because developers with different skills and techniques contribute code. The development of an accurate classification system requires identifying metrics that effectively show quality issues in collaborative OSS environments.

Also, this research uses ML algorithms for code quality classification into high, medium and low categories based on metrics discovered. The diverse nature of Java OSS codebases makes traditional manual assessment methods insufficient for the task. The research examines how Decision Trees and Random Forest, and Support Vector Machines (SVM) algorithms use quality metrics to conduct automated classification through a data-driven method that scales for quality assurance needs.

The evaluation metrics for assessing the performance of the ML model in code quality classification need to be determined. The evaluation of multi-class classification models requires Accuracy alongside Precision and Recall along with F1-Score to determine model effectiveness. The evaluation metrics question ensures thorough assessment of the model performance which provides both reliability evidence and practical utility information for Java OSS developers to validate system capability in meeting study goals.

Research Objectives

The main goal involved developing an innovative ML-based model for automatic code quality classification of Java OSS projects to tackle irregular developer contributions and their effects on software reliability and sustainability. The model implemented specific software quality metrics including LOC and CBO and RFC and WMC and LCOM and DIT, and NOC, to create a systematic data-based quality assessment solution. The system development focused on building a practical solution which enables developers to submit Java files for automated quality assessment leading to better code review processes and improved refactoring prioritization, and enhanced Java OSS project quality for lasting success.

In addition to the main objective, the research aims to automate the classification of code quality by determining and confirming important software quality measures like Lines of Code (LOC), Coupling Between Objects (CBO), and Depth of Inheritance Tree (DIT). Second, it builds and trains a machine learning model based on these metrics, comparing algorithms like XGBoost, Random Forest, and SVM to determine code quality. Lastly, the model is thoroughly tested based on the typical metrics: Accuracy, Precision, Recall, and F1 Score, on a test set of 46,494 observations, which makes it applicable in a real-life OSS development.

2 Literature Review

The quality of code within Java-based open-source software projects stands as the essential factor to guarantee software reliability and maintainability in collaborative development environments. Java has gained popularity in OSS development because of its ability to operate across platforms and its large library collection which attracts projects such as Apache Hadoop and Eclipse [8]. OSS code quality becomes inconsistent because the project contains contributors at various skill levels while managing inconsistent coding practices [9].The object-oriented Java features improve modularity according to research findings but quality maintenance becomes harder to manage when these features remain uncontrolled especially during large projects with multiple contributors [10]. Java-based OSS code quality assessment depends on software quality metrics which measure object-oriented paradigms through Coupling Between Objects (CBO) and Depth of Inheritance Tree (DIT) and Lines of Code (LOC) [11]. The metrics serve to quantify attributes regarding complexity and maintainability which form a basis for quality assessment. Research indicates that elevated CBO values in Java projects lead to higher defect rates because they demonstrate how inter-class dependencies affect quality [12]. Saklani et al.'s [11] review demonstrates that these metrics when utilized with machine learning effectively predict quality issues in Java OSS contexts according to research findings.

The application of ML techniques to Java-based OSS quality prediction encounters difficulties in both interpretability and generalizability. The black-box nature of SVM and Neural Networks (NN) makes it difficult to understand quality outcomes despite their high predictive power according to Sheshasaayee and Jose [13].

The assessment of code quality in Java-based open-source software projects heavily depends on source code metrics which measure structural aspects that include complexity and maintainability [11]. Java codebases receive evaluation through three metrics CBO and DIT and LOC which help identify defect-proneness and other quality issues [12]. The

combination of these metrics with machine learning algorithms enhances OSS quality prediction accuracy according to S. Saklani et al. [11] as described in their IJARCS downloadable review. The object-oriented characteristics of Java make these interdependencies vital for determining quality.

A. Sheshasaayee and R. Jose [14] demonstrate that although Support Vector Machines (SVM) and Neural Networks (NN) deliver accurate predictions, they remain difficult to interpret for understanding quality predictions in open-source software contexts. The lack of transparency in these models creates issues for projects that prioritize explainable methods in Java development.

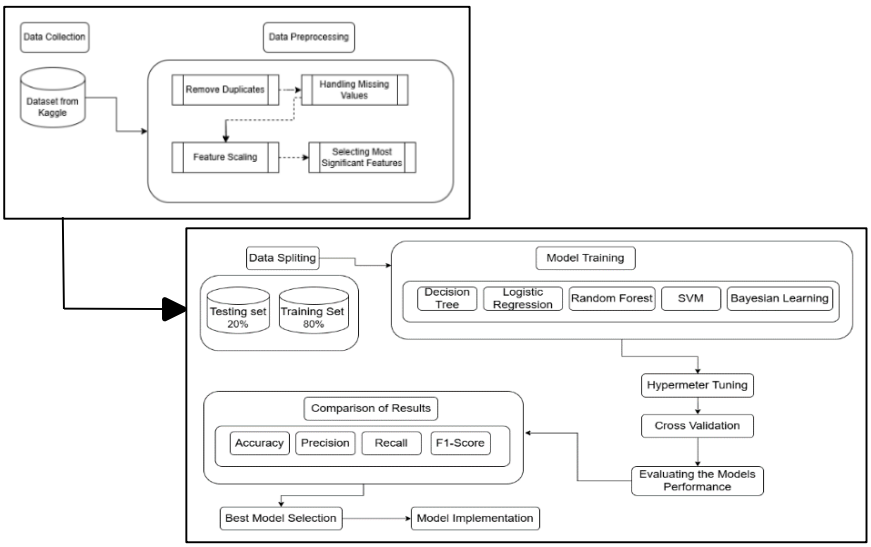

Fig. 1. Research Design

The combination of software quality metrics CBO and LOC and DIT with contribution-specific data creates important opportunities to enhance Java-based open-source software quality classification. Property-based software quality measures CBO and LOC gain improved ability for classifying OSS defects when used alongside developer variables such as contributor frequency and experience levels as proposed by S. Saklani et al. [11]. These

quality metrics enhance Java OSS quality classification precision by linking them to contributor activity according to the authors. The use of metrics that consider contribution levels provides better justification than analyzing code independently.

The literature review demonstrates that software quality metrics and machine learning techniques serve as essential tools for Java-based open-source project code quality evaluation through the use of CBO, DIT, and LOC metrics for structural analysis and Random Forest and Decision Trees algorithms for quality prediction.

2.1 Research Design

The research implemented a quantitative empirical method to create and test an ML-based system which classifies code quality into three levels of high, medium and low. The classification thresholds were defined based on the model's output probability: code quality is classified as Low if the probability is <0.5 and requires refactoring; Medium if the probability is <0.7 and may need refactoring; and High if the probability is >0.7 and does not require refactoring. The research method used software quality metrics extracted from Java source code to measure developer contributions (shown in Fig. 1).

The research implemented seven supervised ML algorithms, including Logistic Regression, Decision Trees, Random Forest, Support Vector Machine (SVM), Bayesian Learning, XGBoost and Multi-Layer Perceptron. The methodology integrated theoretical ML applications with practical OSSD needs to develop a system which enables developers to submit Java files for immediate quality assessment.

2.2 Data Collection and Data Preprocessing

The study utilized the Code Metrics Dataset for Software Project Structure available at Kaggle, which contains the OnlyTrivial_dt.csv file. A total of 232,468 entries compiled from 53 columns that offered various software quality metrics and a refactoring decision column extracted from multiple open-source Java projects. The refactoring decision column was used to derive the quality categories, serving as ground truth for model training and evaluation.

The data quality needed initial preprocessing steps to prepare it for analysis. Model training bias was prevented through duplicate entry removal since duplicated data points would skew the classification outcomes. The median value of each column served as the imputation method for handling missing values in numerical columns lcom*, tcc and lcc because it showed resilience to outliers in software metrics. The dataset maintained its integrity by dropping rows with many missing values, yet the data size slightly decreased

to preserve reliability. The cleaning process allowed the dataset to maintain a considerable number of observations (232,468), which provided adequate size and diversity to train and evaluate ML models effectively.

2.3 Feature Extraction

Feature selection was performed to enhance model performance and reduce computational complexity, focusing on the most predictive metrics for code quality classification. From the available metrics from the dataset, a subset was selected based on their relevance to quality assessment and prior studies. The chosen features include LOC, CBO, RFC, WMC, LCOM, DIT, and NOC as these metrics effectively capture complexity, coupling, cohesion, and inheritance patterns in Java code.

Correlation analysis was conducted to identify and exclude highly correlated features, minimizing redundancy (e.g., using Pearson correlation to check for multicollinearity among metrics like cbo and cboModified). The target variable was prepared for multi-class classification by encoding the high, medium, and low categories into numerical labels, ensuring compatibility with ML algorithms. This refined feature set and preprocessed target variable ensure that the ML models focus on the most informative attributes, improving classification accuracy and efficiency for the subsequent training phase.

2.4 ML Model Development

The machine learning (ML) model development phase trained seven supervised ML algorithms to identify code quality levels in Java-based open-source software (OSS) projects through high, medium, and low classifications. The following algorithms were selected based on their proven effectiveness in software quality prediction and multi-class classification tasks:

Logistic Regression models class probabilities by the logistic function. Decision Tree: Tree classification based on decision rules, and Random Forest: A combination of multiple trees for better accuracy. SVM uses an optimal hyperplane for class separation, and Bayesian Learning makes inferences based on Bayes' theorem. MLP is a multi-layer feedforward neural network for pattern recognition, and XGBoost is an efficient gradient boosting with regularization.

Hyperparameter tuning was performed for each model using GridSearchCV with 10-fold cross-validation to optimize performance. Key tuned parameters included: learning rate and max_depth for XGBoost; C and kernel for SVM; number of trees and depth for Random Forest; and solver and regularization for Logistic Regression and MLP.

The training set included 80% of the preprocessed data, totaling 185,974 observations, while the testing set contained 20% of the data with 46,494 observations through train_test_split from scikit-learn using random_state=42 for reproducibility. This data split provides a strong evaluation of model performance on new data and optimizes the amount of training data available for detecting patterns within software quality metrics.

The assessment of machine learning (ML) models constituted an essential process to evaluate their capability for classifying code quality as high, medium or low within Java-based open-source software (OSS) projects. The evaluation employed four standard metrics that include Accuracy, Precision, Recall and F1-Score to assess multi-class classification tasks. Each of the seven ML models was evaluated using the test set, and their performance scores were compared to identify the best algorithm. The evaluation process involved predicting the quality class for each test instance, followed by calculating the metrics for each model.

2.5 Model Implementation

The XGBoost model was integrated into an operational system for Java-based open-source software project code classification, achieving competitive Precision, Recall, and F1-Score values. The system, constructed using scikit-learn for the XGBoost model and pandas for data management, enables developers to upload Java files and extract software quality metrics. The extracted metrics are preprocessed using MinMaxScaler, and the trained XGBoost model generates quality class predictions, offering an automated, user-friendly assessment solution.

3 Results and Discussion

The seven ML models conducted evaluations on the test data through Accuracy and Precision and Recall and F1-Score metrics that operated within each quality classes. Fig. 2. presents the performance results.

Accuracy measures the proportion of correct predictions among the total predictions made. Precision represents the proportion of true positive predictions among all positive predictions. Recall measures the proportion of actual positives correctly identified, and the F1-Score provides the harmonic mean of Precision and Recall.

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.68	0.661	0.672	0.666
Decision Trees	0.78	0.762	0.771	0.766
Random Forest	0.802	0.814	0.792	0.803
SVM	0.701	0.682	0.691	0.686
Bayesian Learning	0.62	0.601	0.612	0.606
MLP	0.75	0.741	0.732	0.736
XGBoost	0.82	0.832	0.811	0.821

Fig. 2 Comparison between Models

For understanding the XGBoost model's individual category performance, Table 1 shows the confusion matrix and the per-class evaluation metrics. It shows the disintegration of predicted values for the 46,494 samples in the testing set, divided into three quality classes. The steady results in every class confirm the equal efficiency of the model, ensuring that rigorous verification has been reached because the model does not favor any category.

The research developed an ML-powered system for Java-based open-source software project code quality assessment, addressing developer inconsistency issues. It established essential quality metrics like LOC, CBO, RFC, WMC, LCOM, DIT, and NOC, using seven supervised ML algorithms for classification and assigning categories as high, medium, and low.

Table 3. Confusion Matrix and Per-Class Performance for XGBoost

Predicted Class			
Actual Class	**Low**	**Medium**	**High**
Low	12800	1600	1098
Medium	1200	12650	1648
High	900	1923	12675
Per-Class Performance Metrics			
Class	Precision	Recall	F1-Score
Low	0.859	0.826	0.842
Medium	0.782	0.816	0.799
High	0.822	0.818	0.820

This research demonstrates the effectiveness of XGBoost as a machine learning technique for Java-based OSS project code quality classification. With an 82% accuracy rate, it effectively handles OSSD complexity due to contributor knowledge levels and quick

code modifications. The system's practical deployment confirms its value for Java OSSD applications by enabling Java file uploads from developers to generate automatic quality assessments (Fig. 3).

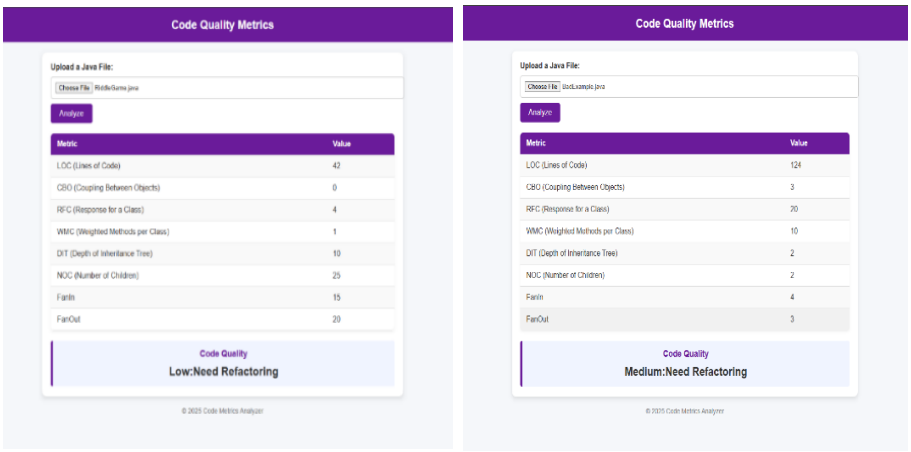

Fig. 3. System output examples for various java file uploads

3.1 Future Works

This research successfully created and deployed a code quality classification system for Java-based open-source software projects, yet several upcoming research areas would strengthen its impact. Advanced ML techniques should be investigated for future work to extend beyond Random Forest methods and neural networks, since potential research demonstrates that these may elevate classification precision above the current 82%. The methods could process complex non-linear patterns in software metrics, yet they should maintain interpretability, which XGBoost provided in this study. Expanding the system to incorporate Python and C++ programming languages would enhance its impact on the OSS ecosystem while filling the research gap for multi-language quality assessment tools.

4 Conclusions

The research developed an automated code quality classification system for Java-based open-source software projects using machine learning techniques. XGBoost outperformed other models with 82% accuracy in classifying code quality into high, medium, and low categories. The system uses seven software quality metrics to assess code quality based on structural characteristics and developer contributions. The study highlights the importance of combining contribution analysis with quality metrics to understand quality evolution in collaborative OSS contexts.

References

1. J. W. Castro Llanos and S. T. Acuña Castillo, Differences between traditional and open source development activities, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7343 LNCS, pp. 131–144, Jun. 2012, doi: 10.1007/978-3-642-31063-8_11.
2. S. Rusovan, M. Lawford, and D. L. Parnas, Open Source Software Development, Geographic information science & technology body of knowledge, pp. 107–122, Apr. 2021, doi: 10.7551/MITPRESS/5326.003.0011.
3. Compare B2B Software, Download, & Develop Open Source & Business Software - SourceForge. Accessed: Apr. 03, 2025. [Online]. Available: https://sourceforge.net/
4. S. Kaur and S. Singh, Improving the Quality of Open Source Software, Agile Software Development: Trends, Challenges and Applications, pp. 309–323, Jan. 2023, doi: 10.1002/9781119896838.CH16.
5. J. P. Meher and R. Mall, Machine Learning-based Software Bug Classification through Mining Open Source Repositories, Proceedings ICIT 2023 - 21st International Conference on Information Technology, pp. 17–22, 2023, doi: 10.1109/OCIT59427.2023.10431348.
6. H. P. Putro, U. L. Yuhana, E. M. Yuniarno, and M. H. Purnomo, Source Code Statement Classification using ANTLR and Random Forest, Proceedings of International Seminar on Intelligent Technology and Its Applications: Leveraging Intelligent Systems to Achieve Sustainable Development Goals, ISITIA 2023, pp. 60–65, 2023, doi: 10.1109/ISITIA59021.2023.10220999.
7. A. Khan, R. R. Mekuria, and R. Isaev, Applying Machine Learning Analysis for Software Quality Test, May 2023, doi: 10.1109/ICCQ57276.2023.10114664.
8. Cristina Gacek and Budi Arief, The Many Meanings of Open Source, 2004. [Online]. Available: www.opensource.org/docs/definition.

9. S. Haider, W. Khalil, A. S. Al-Shamayleh, A. Akhunzada, and A. Gani, Risk Factors and Practices for the Development of Open Source Software from Developers' Perspective, IEEE Access, 11, pp. 63333–63350, 2023, doi: 10.1109/ACCESS.2023.3267048.
10. Vinay Tiwari, Software Engineering Issues in Development Models of Open Source Software, 2, 2011.
11. S. Saklani, A. Kalia, S. Sood, and K. Kumari, Software Quality Prediction Using Machine Learning Techniques and Source Code Metrics: A Review, International Journal of Advanced Research in Computer Science, 13(6), 2022, doi: 10.26483/ijarcs.v13i6.6918.
12. V. Antinyan, M. Staron, and A. Sandberg, Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time, 22, pp. 3057–3087, 2017, doi: 10.1007/s10664-017-9508-2.
13. J. Cao, Z. Chen, J. Wu, S. Cheung, and C. Xu, JavaBench: A Benchmark of Object-Oriented Code Generation for Evaluating Large Language Models, Jun. 2024, [Online]. Available: http://arxiv.org/abs/2406.12902
14. A. Sheshasaayee and R. Jose, Prospects and Challenges of using Machine Learning Algorithms for Software Quality Assessment and Prediction, Int. J. Comput Appl, 129(7), pp. 33–35, Nov. 2015, doi: 10.5120/IJCA2015906887.

.