3rd International Engineering Research Symposium 2025 (IERS 2025)
Colombo, Sri Lanka

Enhancing Software Quality through Comparative Analysis of Machine Learning Techniques for Test Case Prioritization using Object-Oriented Metrics

M. K. M. N. Kumari[1] and W. V. S. K. Wasalthilaka[2]

[1]Department of Computing and Information Systems, Faculty of Computing, Sabaragamuwa University of Sri Lanka, Sri Lanka.
[2]Department of Software Engineering, Faculty of Computing, Sabaragamuwa University of Sri Lanka, Sri Lanka.
nadeeshamayuri15@gmail.com

Abstract. Software quality plays an important role in software engineering. Quality software depends heavily on software testing. Regression testing is essential to this process and ensures that recent changes do not affect existing features. However, regression testing is often time-consuming and resource-intensive. Test Case Prioritization (TCP) techniques can be used to optimize the regression testing process. This research focuses on leveraging machine learning techniques to improve TCP using object-oriented metrics like Coupling Between Objects (CBO), Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Response for a Class (RFC), Lack of Cohesion of Methods (LCOM), fan-in, and fan-out. The dataset is divided into 70% of training and 30% of testing sets by using data preprocessing and feature selection methods. Then the trained models were evaluated using performance metrics such as accuracy, precision, recall, and f1-score. Different machine learning algorithms such as Decision Tree, Random Forest, Neural Networks, K-Nearest Neighbor (KNN), and Logistic Regression were compared using object-oriented metrics to identify the best approach. Among these algorithms, the decision tree algorithm outperformed other algorithms. Because of its ability to handle complex decision boundaries with minimal overfitting and achieved 71% accuracy. Finally developed the TCP framework using the Decision Tree algorithm. This framework automatically prioritizes test cases by code metric values to optimize testing efforts, detecting critical defects early while saving time and resources. This result highlighted their potential for greatly improving regression testing efficiency and software quality.

Keywords: Machine Learning, Object-Oriented Metrics, Software Quality, Test Case Prioritization

1 Introduction

1.1 Background

Software testing is an important process in the software development life cycle. Here, it is verified and validated that the software application is free from errors, meets the expected requirements, and efficiently and effectively satisfies the user's requirements. Effective testing increases the quality of the software and contributes to delivering high-quality software [1].

There are different types of software testing. Each of those software testing types has its own unique goals. Some of the main testing types are unit testing, integration testing, system testing, acceptance testing, regression testing, smoke testing, sanity testing, fuzz testing, black box testing, white box testing, grey box testing, etc. [2].

Among these available testing types, this research focuses on regression testing. It is an essential part of quality assurance in software to perform regression testing to make sure that the modifications made to the code haven't impacted already existing functionalities. In most cases, regression test suites increase with increased quality assurance tasks done or the program being improved. In such a way, the creation of more test cases will make a test suite effective. However, running a large test suite is expensive, and sometimes it runs for hours or days [3].

One of the most promising approaches to improving test case prioritization involves the use of object-oriented metrics integrated with machine learning techniques. Concepts in OOP, like encapsulation, inheritance, polymorphism, and abstraction, are meant to enhance the modularity and maintainability of the code; however, they add additional levels of complexity that might further deteriorate the quality of software [4].

Object-oriented metrics inform about the structural quality of software. Coupling Between Objects (CBO), Response for a Class (RFC), Weighted Methods Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), and Lack of Cohesion of Methods (LCOM), etc., are some of the measures used in estimating complexity, maintainability, and probably fault-prone areas for different software modules [5].

This research aims to improve the quality, maintainability, and reusability of software by finding bugs, ambiguities, and bad smells using machine learning techniques. Various methods of software defect prediction are applied for the prediction of software defects using statistical techniques. However, machine learning techniques have an important role in software bug detection [6]. ML algorithms provide very powerful tools for the analysis of historical data and for making predictions [7]. It can analyze the trend of historical test data to predict the error-prone nature of code modules in software testing using machine

learning. These predictive capabilities are then used to schedule test cases that are more likely to catch errors, hence increasing the efficiency of testing.

This research focuses on integrating object-oriented metrics such as CBO, WMC, DIT, NOC, RFC, LCOM, FIN, and FOUT with different machine learning techniques like Decision Tree, Random Forest, Neural Networks, K-Nearest Neighbor (KNN), and Logistic Regression into test case prioritization. Decision Tree makes simple hierarchical rules (critical parameter: max_depth), and Random Forest votes from multiple trees (n_estimators). Neural Networks learn nonlinear relationships (hidden layers, activation functions), KNN classifies objects based on similarity (k value, distance metric), and Logistic Regression builds a linear probabilistic model (regularization parameter C). [6], [7]. Using these metrics as ML features enables ranking test cases by criticality, improving regression testing efficiency, especially in CI/CD environments.

This research proposes a test case prioritization framework using the best-performing machine learning algorithm. This framework automatically prioritizes test cases by code metric values to optimize testing efforts, detecting critical defects early while saving time and resources. This result highlighted their potential for greatly improving regression testing efficiency and software quality.

1.2 Problem Statement and Research Objectives

Regression testing is very important to maintain software quality in software development. However, software companies struggle due to complex code and frequent changes. This makes it hard to identify and prioritize test cases effectively, leading to missed defects, lower software quality, higher maintenance costs, and delayed product releases.

Indeed, traditional test case prioritization methods such as coverage-based, history-based, and risk-based fail to effectively address the complexities of object-oriented software design. Therefore, they are not able to detect and prioritize test cases with a high likelihood of revealing critical faults early in the testing process. This leads to lengthy testing cycles, undetected defects, decreased software quality, and higher maintenance and delayed release expenses.

This research addresses the need for an improved approach to test case prioritization that stimulates machine learning to improve the efficiency and effectiveness of regression testing using an object-oriented matrix. To meet the problem statement, the research questions that have been formulated are as follows.

The main objective of this research is to develop an ML-driven framework for automated TCP by identifying influential OO metrics and optimizing regression testing efficiency. The sub-objectives of this research are to compare and evaluate the performance of five machine-learning algorithms for TCP, investigate the correlation between object-

oriented metrics and test case failure probability, and formulate, design, and implement a fully automated test case prioritization framework with the best-performing machine learning algorithm.

1.3 Significance of the Study

Regression testing is critical to software quality but generally ineffective due to poorly prioritized test cases. While there are traditional Test Case Prioritization (TCP) practices, they fail because of the complexity of modern software [8]. This research introduces a machine learning (ML) approach with object-oriented metrics to revolutionize TCP by addressing the following critical gaps in TCP.

- Prior research focuses on single ML models with no rigorous comparison. This study compared five algorithms, namely, the decision tree, random forest, neural networks, K-nearest neighbor (KNN), and logistic regression, to find the best for TCP.
- Previous work utilizes raw object-oriented metrics without considering interactions. In this research, derived features are suggested to achieve improved prediction accuracy.
- Existing ML-based TCP solutions are primarily in the form of experimental prototypes or conceptual designs. This work bridges theory and practice with the development of an operational framework that is specifically designed to be easily integrated into today's development practices.
- Existing research is on small datasets (<10K test cases). This framework is verified on 232K+ test cases on different OO projects.

2 Literature Review

Software testing is a very important process in the software development life cycle. It depends on the ability to detect errors early in the software development life cycle. Early detection of defects can greatly reduce costs and improve software quality. There are different types of software testing. Each of those software testing types has its own unique goals. This research focuses on regression testing.

V. Garousi and J. Zhi [1] analyzed that regression testing is a critical part of software engineering. It ensures that new code changes do not affect existing functionality. However, they showed that regression testing is often time-consuming and resource-intensive. S. Elbaum et al. [9] conducted a study for Test Case Prioritization (TCP) techniques can be used to optimize the regression testing process so that the most important test cases can

be selected and executed first, the testing time can be reduced, and resource usage can be optimized.

Z. Li, M. Harman and R. M. Hierons [9] explored the traditional TCP methods, such as coverage-based and history-based methods. They found that these methods are insufficient to handle the complexity of modern object-oriented systems. Rongqi Pan et al. [10] highlighted that recent advances in Machine Learning (ML) offer encouraging solutions by leveraging object-oriented metrics (Ex: CBO, WMC) to predict high-risk test cases.

M. Z. Khan [11] evaluated that Machine learning algorithms have been very promising in TCP because they learn from labeled test execution data to forecast test case priority. They showed that Decision Trees (DT) have been very popular among supervised learning methods due to interpretability, where the cause of prioritization can be interpreted by testers in comprehensible decision rules. Also, they discussed that Random Forest (RF) algorithms utilize a set of decision trees, achieving high accuracy at the expense of increased computational complexity. O. F. Arar and K. Ayan [12] discussed that Neural networks (NN) have been able to model nonlinear relationships and interaction effects between test case features. R. Malhotra and R. Raje [6] discussed that Logistic Regression (LR) offers a probabilistic model which works well when priorities and test features relationships are linear. Also, they introduced that the K-Nearest Neighbors (KNN) gives priority to the test cases according to their resemblance to the already labeled examples and is a simple though frequently effective method.

Prykhodko et al. [13] showed that higher CBO values are associated with 30-40% more faults in Java programs. M. Khatibsyarbini et al. [14] showed that Weighted Methods per Class (WMC), computes method complexity in a class by adding cyclomatic complexity, with larger values of WMC consistently correlated with larger defect rates. Also, they showed that Response for a Class (RFC), which provides the number of possible methods calls, helps to identify classes with complex interaction patterns frequently requiring extensive testing. These metrics have gained such popularity that they are now part of most static analysis tools, for example, Eclipse Metrics and SonarQube.

The evolution of TCP techniques has been critically analyzed in this study, finding developments from traditional coverage-based methods to recent machine learning-based approaches incorporating object-oriented metrics. Although ML-based approaches demonstrate better fault detection, significant gaps still exist. Such constraints underscore the need for detailed empirical evaluation and applicable models that span theoretical research and industry practice. This research endeavors to address this through its juxtaposition of machine learning methods, extended feature engineering, and continuous implementation against large test scenarios.

3 Methodology

The proposed approach consists of several steps as follows.

1. **Data collection** - The dataset was obtained from Kaggle under the "Code Metrics Dataset for Software Project Structure." It contains object-oriented code metrics and corresponding test execution data collected from multiple open-source Java projects, totaling 232,468 instances with 53 features, which were used for training and testing.
2. **Data preprocessing** - The dataset was preprocessed through duplicate removal and handling missing values (median imputation). To resolve class imbalance, SMOTE was applied. The features were normalized through Min-Max scaling, and highly correlated metrics were reduced through feature selection (RFE and correlation analysis). A few interaction features (e.g., CBO × WMC) were also created for improving predictability.

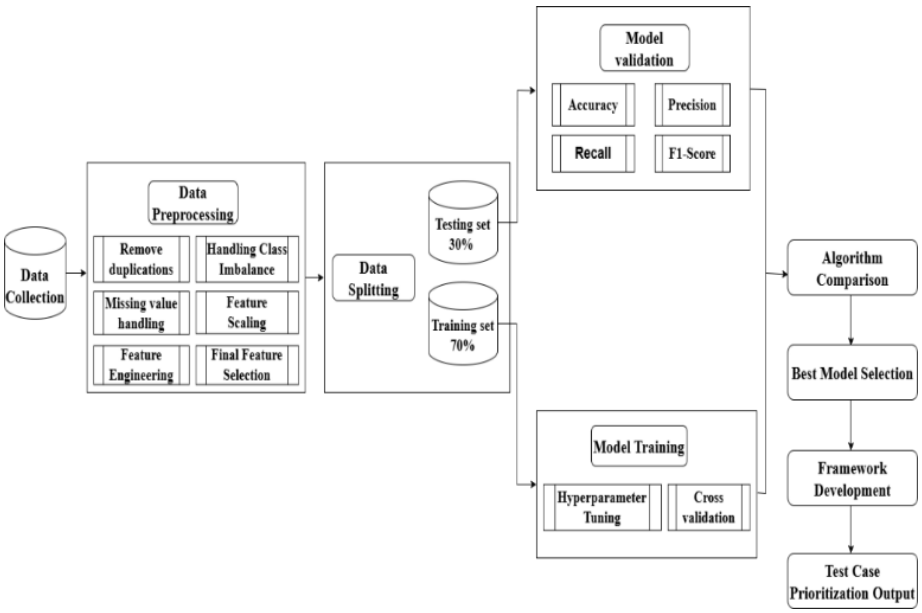

Fig. 1. Research Methodology

3. **Data splitting** - The preprocessed data were split into 70% training and 30% test sets, enabling robust model evaluation.

4. **Model training -** Five machine learning algorithms, such as Decision Tree, Random Forest, Neural Networks, K-Nearest Neighbor (KNN), and Logistic Regression, were trained in this research using the training dataset. Hyperparameter optimization (via grid search) and 5-fold cross-validation were performed to ensure generalization. The models were then evaluated using standard performance metrics (accuracy, precision, recall, F1-score) computed with scikit-learn evaluation functions.
5. **Model evaluation -** Then the trained models were evaluated using performance metrics such as accuracy, precision, recall, and F1-score [15].
6. **Algorithm comparison -** Each model is compared using the evaluated results to get the best-performing model. Among them, the Decision Tree outperforms the other models.
7. **TCP framework development -** Finally, using the best-performing machine learning algorithm (decision tree), developed the TCP framework. This framework automatically prioritizes test cases by code metric values to optimize testing efforts, detecting critical defects early while saving time and resources.

4 Results and Findings

4.1 Performance Comparison of Machine Learning Algorithms

To identify the best machine learning approach for Test Case Prioritization (TCP), the study carried out an extensive comparative analysis of five prominent supervised learning algorithms. They were tested for four key classification measures: accuracy, precision, recall, and F1-score to thoroughly understand their performance in test case prioritization. The comparative results are provided in Table 1 and also plotted graphically in Fig. 2.

Table 1. Performance metrics of machine learning algorithms.

Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Decision Tree	71.55	69.42	80.11	74.38
Random Forest	64.21	66.09	58.51	62.07
Neural Networks	61.42	59.98	68.86	64.11
K-Nearest Neighbors	62.94	61.94	67.31	64.51
Logistic Regression	51.81	51.39	68.66	58.78

A one-way ANOVA test indicated that the Decision Tree's superior performance was statistically significant compared to other models ($p < 0.05$). To enhance clarity, Table 1 results were also visualized in a bar chart (Fig. 2).

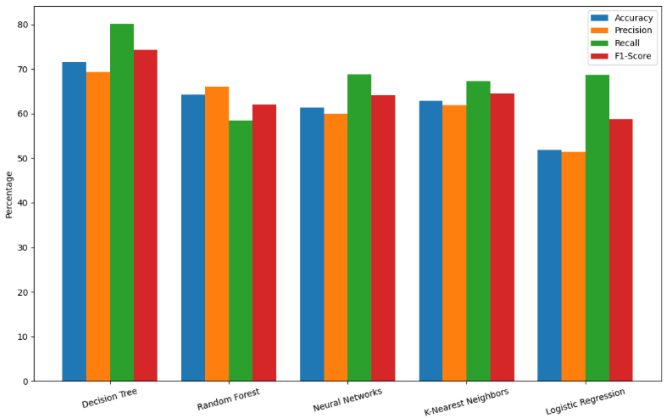

Fig. 2. Comparative performance of ML algorithms (graphical representation of Table 1)

Decision Tree performed the highest accuracy with 71.55%, significantly outperforming other approaches. Also, it reflects its competence in managing non-linear relationships among object-oriented metrics and test case priority. The Random Forest model was ranked close but with lower recall, which suggests a sensitivity vs. precision trade-off. Neural Networks and KNN were both ranked as medium performers. Neural Networks underperformed due to overfitting on the imbalanced data, while Logistic Regression underperformed due to non-linear feature interactions inherent in object-oriented metrics. These results affirm that Decision Trees are most appropriate for TCP because they are interpretable and demonstrate well-balanced performance.

4.2 Correlation Analysis of Object-Oriented Metrics

The study analyzed the predictive power of eight object-oriented metrics, including CBO, WMC, DIT, NOC, RFC, LCOM, fan-in, and fan-out.

Feature importance rankings

As shown in Fig. 3, fan-in and WMC emerged as the most influential metrics for predicting test case priority, followed by fan-out and DIT. The metrics of CBO and NOC demonstrated moderate importance levels. The analysis revealed that RFC, together with LCOM, exhibited minimal impact on test case criticality rates.

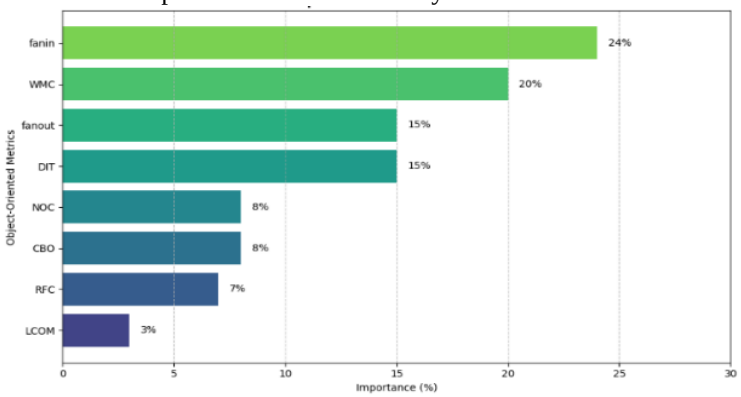

Fig. 3. Feature importance of OO metrics for test case prioritization

Correlation Heatmap Analysis

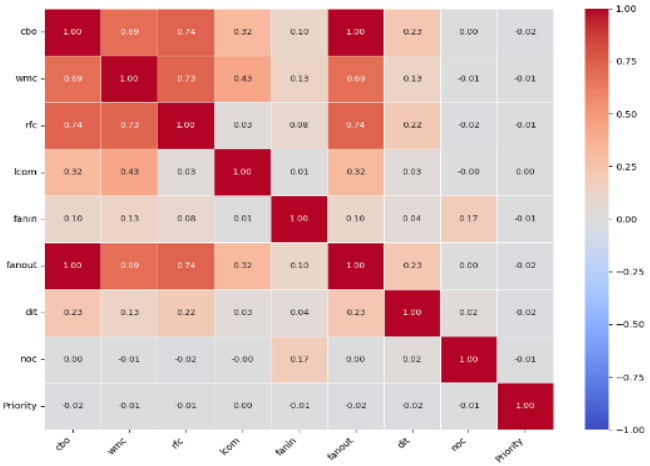

Fig. 4. Correlation heatmap of OO metrics

Fig. 4 illustrates the correlation heatmap, where fan-out and CBO are perfectly correlated (1.00), while RFC shows strong correlation with both CBO and WMC. Inheritance metrics (DIT and NOC) have very few correlations with the other metrics, and LCOM is fairly isolated, with only moderate correlation with WMC (0.43).

4.3 Framework Validation Using APFD

The efficiency of the framework was validated by both quantitative APFD scores and qualitative interface inspection. The employed system provides:

- **User Interface Layer:** Interactive web-based interface for both single test case analysis and batch processing. It includes input boxes for the major object-oriented metrics, a file upload option for batch processing (CSV format), and visualization of the decision process.
- **Machine Learning Core:** The model was trained on 232,468 test cases with object-oriented 33 metrics. The model produces probability scores (0-1) that numerically quantify the priority of each test case, allowing for effective high/low priority distinction for regression testing optimization.
- **Decision Visualization:** The framework supports interactive decision tree visualization, and the users can navigate 1-5 levels deep.

According to the APFD computation the 88% of errors emerge early during test suite execution. This demonstrates that the framework can optimally execute tests for earlier fault detection during testing [16]. For example, in a Java project module having high WMC and fan-in values, the framework prioritized critical test cases that detected three major defects early in the iterations, which reduced debugging effort.

5 Discussion and Conclusions

This study bridges the practice-theory gap between the needs of software testing and machine learning theory and presents a practical contribution within the field of Test Case Prioritization (TCP). Through a comparative analysis on a large scale of five mature machine learning algorithms, such as Decision Tree, Random Forest, Neural Networks, K-Nearest Neighbors, and Logistic Regression, the study identifies Decision Trees as the most appropriate model for ranking test cases in object-oriented software systems. It performed better with 71.55% accuracy.

One of the greatest contributions of this study is demonstrating how interaction terms among object-oriented metrics (Ex: CBO × WMC, DIT × RFC) can greatly improve the

predictive power of machine learning models. These results provide a new direction for feature engineering for test optimization research and practice.

The framework obtained a high APFD of 0.88, which further confirms its efficiency at detecting faults earlier during the test process. The metric provides a reproducible baseline and adds a solid empirical component to future studies on regression test optimization. As validated in Section 4.3 with APFD score (0.88), this provides strong empirical evidence for early defect detection.

In conclusion, this study proposes a scalable and effective ML-based TCP method that extends the frontier of software quality assurance theory and practice. By applying machine learning and feature engineering, it enhances the effectiveness of regression testing and provides a basis for its future use in CI/CD and agile environments.

References

1. V. Garousi and J. Zhi, A survey of software testing practices in Canada, *Journal of Systems and Software*, 86(5), pp. 1354–1376, 2013, doi: 10.1016/j.jss.2012.12.051.
2. I. Hooda, R. Scholar, and R. Singh Chhillar, "Software Test Process, Testing Types and Techniques," 2015.
3. D. K. Yadav and S. Dutta, "Regression test case selection and prioritization for object oriented software," *Microsystem Technologies*, vol. 26, no. 5, pp. 1463–1477, May 2020, doi: 10.1007/s00542-019-04679-7.
4. R. A. Khan, K. Mustafa, and S. I. Ahson, "An Empirical Validation of Object Oriented Design Quality Metrics," *Journal of King Saud University - Computer and Information Sciences*, vol. 19, pp. 1–16, 2007, doi: 10.1016/s1319-1578(07)80001-2.
5. Josiane. Mothe, L. Hoang. Son, and T. Q. Vinh. Nguyen, *Proceedings of 2019 11th International Conference On Knowledge And Systems Engineering : KSE 2019 : October 24-26, 2019, Da Nang, Vietnam*. IEEE, 2019.
6. R. Malhotra and R. Raje, "An Empirical Comparison of Machine Learning Techniques for Software Defect Prediction," 2014. [Online]. Available: http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html
7. Y. Zhao, D. Hao, and L. Zhang, "Revisiting Machine Learning based Test Case Prioritization for Continuous Integration," Nov. 2023, [Online]. Available: http://arxiv.org/abs/2311.13413
8. R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review," Oct. 2021, doi: 10.1007/s10664-021-10066-6.
9. Z. Li, M. Harman, and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization."

10. Rongqi Pan, Mojtaba Bagherzadeh, Taher A. Ghaleb, and Lionel Briand, “Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review,” 2021.
11. R. Kizito, P. Scruggs, X. Li, R. Kress, M. Devinney, and T. Berg, “The Application of Random Forest to Predictive Maintenance,” 2018.
12. Ö. F. Arar and K. Ayan, “Software defect prediction using cost-sensitive neural network,” *Applied Soft Computing Journal*, vol. 33, pp. 263–277, Apr. 2015, doi: 10.1016/j.asoc.2015.04.045.
13. S. B. Prykhodko, K. S. Prykhodko, and T. G. Smykodub, “A statistical estimation of the coupling between objectmetric for open-source appsdeveloped in Java,” *Herald of Advanced Information Technology*, vol. 5, no. 3, pp. 175–184, Nov. 2022, doi: 10.15276/hait.05.2022.13.
14. M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, “Test case prioritization approaches in regression testing: A systematic literature review,” Jan. 01, 2018, *Elsevier B.V.* doi: 10.1016/j.infsof.2017.08.014.
15. A. Salvadorrgarcíaa, M. R. Pratii, and B. Franciscooherrera, “Learning from Imbalanced Data Sets.”
16. H. Do and G. Rothermel, “A controlled experiment assessing test case prioritization techniques via mutation faults,” in *IEEE International Conference on Software Maintenance, ICSM*, 2005, pp. 411–420. doi: 10.1109/ICSM.2005.9.