

*Covenant Journal of Informatics & Communication Technology. Vol. 12, No. 2, December, 2024*

**ISSN: *print* 2354 – 3566 *electronics* 2354 – 3507 DOI: An Open Access Journal Available Online**

**Data-Driven Framework for Adaptability Predictions Using Object-Oriented Metrics**

# Obike, G. Peter, Edward, N. Udo

1Department of Computer Science, Michael Okpara University of Agriculture, Umudike, Umuahia, Abia State, Nigeria.

2Department of Computer Science, University of Uyo, Uyo, Akwa Ibom State. Obike.peter@mouau.edu.ng,  edwardudo@yahoo.com,

**Received: 13.09.2024 Accepted: 21.11.2024**

**Publication: December 2024**

***Abstract***— Software adaptability is a critical attribute in modern systems, enabling them to evolve with changing requirements and environments. Existing approaches to predicting adaptability often rely on subjective assessments or high-level architectural metrics, which may lack precision and scalability. This study aims to enhance adaptability prediction by integrating object-oriented metrics and machine learning models, addressing limitations of traditional methods. Decision Trees and Random Forests were employed to model relationships between object-oriented design metrics such as Coupling Between Objects, Lack of Cohesion in Methods, Depth of Inheritance Tree, and Number of Children and adaptability. A comparative analysis using Guava and Apache Commons Lang datasets revealed that the Random Forest model outperforms Decision Trees, achieving an F1-score of 0.9760 and a ROC AUC of 0.9858, highlighting its accuracy and feature importance. Key metrics like coupling and cohesion emerged as pivotal for adaptability prediction. This study contributes a robust, data-driven framework for adaptability prediction, offering valuable insights to developers for creating flexible and maintainable systems. These advancements improve software design practices, ensuring resilience and relevance of applications in dynamic technological landscapes.

***Keywords/Index Terms***— software adaptability level, Random Forest, object-oriented software metrics, software design, software quality

### Introduction

Software adaptability is increasingly vital in modern development, enabling systems to evolve alongside dynamic business requirements and technological advancements. Ensuring adaptability reduces the need for extensive rework and enhances software longevity, particularly in object-oriented systems where design metrics like coupling, cohesion, and inheritance are key predictors. However, the use of fixed thresholds for these metrics in adaptability assessment remains a significant challenge due to their context-specific nature and susceptibility to subjective interpretation (Singh and Kahlon, 2014; Briand et al., 1999).

Threshold-based approaches have been extensively applied across software engineering domains, including code smell detection and software adaptability. While such methods provide standardized benchmarks, their limitations have been highlighted in studies like Gupta et al. (2018) For instance, the construct validity of their code smell prediction model is threatened by the reliance on specific tools like Robusta, which excludes certain smells and employs distinct thresholds incompatible with other tools. This inconsistency underscores the inherent limitations of threshold-based methodologies, as variations across tools and projects restrict generalizability. Moreover, their work reveals the need for statistical models to handle project-specific nuances, as demonstrated by their application of entropy measures and nonlinear regression to predict bad smells.

Gupta et al. (2018) work conclusively emphasize the need for a more robust and generalizable approach. While their entropy-based model demonstrated success in Java projects, with an 𝑅2 value improving to 0.93 after removing an outlier, they acknowledge the challenges in extending this model to other languages and contexts. This limitation reinforces the necessity of data-driven techniques like machine learning, which offer adaptable and scalable solutions that account for diverse software environments and minimize reliance on rigid thresholds.

Motivated by these limitations, this study adopts a machine learning-driven framework for predicting software adaptability using object-oriented design metrics. By leveraging Decision Trees and Random Forests, this research addresses the variability inherent in threshold-based models, providing context-sensitive predictions tailored to the unique characteristics of each software project. This approach not only advances software adaptability assessment but also bridges the gap between static benchmarks and dynamic, data-driven methodologies.

Through the integration of design metrics with machine learning, this research contributes to creating robust and maintainable software systems. It validates the proposed models using real-world datasets from Guava and Apache Commons Lang, demonstrating their generalizability and practical applicability. By addressing the threats to validity inherent in traditional models, this work advocates for a paradigm shift towards adaptable, data-driven strategies in software engineering. This research aims to develop a model that effectively predicts the adaptability levels of object-oriented software using design metrics and machine learning techniques. The specific objectives are to;

i. To identify and analyze relevant object-oriented design metrics that influence software adaptability.

ii. To establish threshold values for relevant object-oriented design metrics based on desirable software properties and formulate decision rules that utilize these metrics and thresholds to predict software adaptability.

iii. To develop and train a Decision Tree, and Random Forest machine learning model to predict software adaptability levels based on the defined metrics and thresholds.

iv. To evaluate the effectiveness of the proposed model using real-world software projects.

### Methodology

This research employs a quantitative methodology combining software metrics analysis and machine learning techniques. The process involves the following steps:

#### 2.1 Dataset and Data Preparation

The dataset utilized in this research was collected from 40 open-source software projects developed in Java. This data was analyzed using a custom-built software analyzer designed to extract key object-oriented metrics essential for predicting software adaptability. The purpose of this analysis is to assess how well each software project can adapt to changing requirements or operating environments, based on specific internal attributes. The dataset comprises the following object-oriented metrics that represent critical structural and design properties of software:

1. Weighted Methods per Class (WMC): Measures the complexity of a class by summing the complexity of individual methods.

$WMC= \sum\_{i=1}^{n}c\_{i}$ (1)

Where $c\_{1}$, ..., $c\_{n}$ be the complexity of the methods of a class with methods $M\_{1}$, ...,$M\_{n}$. If all method complexities are considered to be unity, then WMC = n is the number of methods.

1. Number of Children (NOC): Counts the subclasses or children directly inheriting from a class.
2. Response for a Class (RFC): Estimates the number of potential responses (i.e., method calls) triggered by an object of the class.
3. Coupling Between Objects (CBO): Indicates how strongly a class is connected to other classes.
4. Depth of Inheritance Tree (DIT): Measures the inheritance levels from the base class.
5. Lack of Cohesion in Methods (LCOM1 and LCOM2): Evaluates the cohesion within a class, indicating how closely related methods are to each other.
6. Number of Methods (NOM): Counts the total methods within a class.

These metrics serve as predictors for the **Adaptability Level (ADPL)** of each software project, the dependent variable. The dataset used for this study has 500 instances, 8 inputs features which include CBO, RFC, LCOM1, LCOM2, DIT, NOC, WMC, NOM and 1 output which is the adaptability level - ADPL.

#### 2.2 Adaptability Level Assignment

Drawing from established literatures (Akwukwuma and Udo, 2015, 2019), Udo et al. (2020)), the adaptability level for each project was determined based on a set of rules derived from established literature and expert judgment. The adaptability levels (ADPL) are represented in Table 1.

Table 1 Adaptability Level and Labels

|  |  |
| --- | --- |
| **Class (Adaptability)** | **Label** |
| Poorly Adaptable | 1 |
| Fairly Adaptable | 2 |
| Adaptable | 3 |

#### 2.3 Dimensionality Reduction

Principal Component Analysis (PCA) was applied to reduce the dimensionality of the feature space (principal components) from eight to six, capturing a significant portion of the variance in the original data.

#### 2.4 Model Development

Decision Tree and Random Forest, were employed to develop predictive models. These models were trained on the reduced feature space in C.

ii. Random Forest (RF) is an ensemble model that builds a forest of numerous unpruned decision trees from the training dataset. Then, it uses the mode (most frequent class) of the individual trees' predictions on the testing dataset to make a final prediction (Alsolai and Roper, 2022). This approach helps improve prediction accuracy and robustness compared to single decision trees. RF integrates tree predictors that depend on the values of a random vector sampled individually, along with the same distribution for the whole trees in the forest. Here's the revised text with the citation integrated:

Moreover, this algorithm performs bagging on features based on majority voting and selects the dependent variables with the highest votes (Breiman, 2001). This technique, also known as Random Forests, improves prediction accuracy and reduces overfitting by combining multiple decision trees (Breiman, 2001). In this study, Random Forest integrates algorithms of the same type (decision trees), classifying it as a homogeneous ensemble model. RF depends on four key parameters: the number of trees to grow, the sub-sample size common to each tree, the tree depth, and the number of variables randomly sampled for splitting (Scornet, 2017). The Random Forest Classifier in this study defaults to 100 decision trees, aligning with the common practice of initializing a forest with 100 tree instances (Hall et al., 2009). However, unlike relying solely on default parameters, the code utilizes GridSearchCV to optimize model performance by exploring a range of hyperparameter values. This exhaustive search considers several key parameters: the number of trees in the forest (n\_estimators), evaluated at 50, 100, and 200 (with 100 being a common starting point); the maximum depth of individual trees (max\_depth), tested at levels of 5, 10, and 15; the number of features considered at each split (max\_features), set to 'sqrt' (the square root of the total number of features); the complexity parameter for cost-complexity pruning (ccp\_alpha), explored at values of 0.0, 0.01, and 0.1; and the weighting scheme for imbalanced classes (class\_weight), considering both 'balanced' weighting and no weighting (None). This systematic approach enables the model to identify the most effective configuration for the specific dataset, potentially achieving performance gains beyond what default settings alone could provide (George and Sumathi, 2020).

**Model Training and Hyperparameter Tuning:** Grid Search Cross-Validation approach was employed to optimize the performance of the models by fine-tuning the following hyperparameters in Table 2.

Table 2 Hyperparameters for Machine Learning Model

|  |  |  |
| --- | --- | --- |
| **Model** | **Hyperparameter** | **Description** |
| **Decision Tree** | Max\_depth | Maximum depth of the tree. |
| Min\_samples\_split | Minimum number of samples required to split an internal node. |
| **Random Forest** | n\_estimators | Number of trees in the forest. |
| Max\_depth | Maximum depth of each tree. |
| Max\_features | Number of features to consider at each split. |
| Ccp\_alpha | Cost complexity pruning parameter. |

These rules considered for the specific ranges or thresholds of the object-oriented metrics are detailed in Table 3.

Table 3 Adaptability Levels and Labels

|  |  |
| --- | --- |
| **Class (Adaptability)** | **Label** |
| Poorly Adaptable | 1 |
| Fairly Adaptable | 2 |
| Adaptable | 3 |

A stratified 5-fold cross-validation technique was used to evaluate the model's performance using the following performance metrics:

(i) **Accuracy:** The proportion of correct predictions.

Precision: The proportion of true positive predictions among all positive predictions.

(ii) **Recall:** The proportion of true positive predictions among all actual positive cases.

(iii) **F1-score:** The harmonic mean of precision and recall.

(iv) **Receiver Operating Characteristics (ROC) Area under curve (AUC):** measures the model's ability to distinguish between positive and negative classes. The classification performance of ROC curve was measured using the rule in according to Hosmer & Lemeshow (2000) given as follows:

(i) AUC < 0.5 means no discrimination classification;

(ii) 0.5 < AUC < 0.6 means poor classification;

(iii) 0.6 < AUC < 0.7 means good classification;

(iv) 0.7 < AUC < 0.8 means acceptable classification;

(v) 0.8 < AUC < 0.9 means excellent classification;

(vi) AUC < 0.9 means outstanding classification.

# 3. Implementation and Result

The following tools were used for the implementation of this work. Python 3.11 programming language with sklearn, itertools, matplotlib libraries was used for the main analysis, CK Tool, Maven and Java JDK 17 were used for data extraction from open-source Program (Guava and Apache). The Decision Tree and Random Forest models were trained and evaluated on the provided dataset. The Decision Tree model achieved an accuracy of 98%, with a precision of 0.97, recall of 0.96, and F1-score of 0.97. Additionally, it demonstrated a strong ROC AUC of 0.98.

#### 3.1 Model Accuracy

The Decision Tree achieved an accuracy of 92%, while the Random Forest achieved a perfect score of 99%. These results indicate that both models were highly effective in classifying anomalies within the datasets. The accuracy of the model is presented in Table 4

Table 4 Hyperparameters for Machine Learning Model

|  |  |
| --- | --- |
| **Model** | **Accuracy** |
| **Decision Tree** | 92% |
| **Random Forest** | 99% |

The confusion matrices for both models are presented in Figure 1. and Figure 2. respectively.



Figure 1. Confusion Matrix for Random Forest Model

 

Figure 2. Confusion Matrix for Decision Tree Model

The rows represent the actual classes, while the columns represent the predicted classes. Ideally, high values on the diagonal, indicates correct classifications.

From Figure 1 and Figure 2, both models achieved good classification across all classes. However, the Decision Tree made slightly more errors, particularly for Class 2 and 3, where it misclassified seven instances (3 and 4) for Class 3 and class 2 respectively.

#### 3.2 Classification Report

A detailed breakdown of the performance for each class which includes Precision, recall, and F1-score are presented for each model in Table 5 and Table 6

Table 5 Decision Tree Classification Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-score** | **Support** |
| **Poorly Adaptable** | 1.00 | 1.00 | 1.00 | 10 |
| **Fairly Adaptable** | 0.81 | 0.88 | 0.84 | 24 |
| **Adaptable** | 0.95 | 0.92 | 0.94 | 66 |
| **weighted avg** | 0.92 | 0.92 | 0.92 | 100 |

Table 6 Random Forest Classification Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-score** | **Support** |
| **Poorly Adaptable** | 1.00 | 1.00 | 1.00 | 10 |
| **Fairly Adaptable** | 0.96 | 1.00 | 0.98 | 24 |
| **Adaptable** | 1.00 | 0.98 | 0.99 | 66 |
| **weighted avg** | 0.99 | 0.99 | 0.99 | 100 |

The reports in Table V and Table VI indicate that both models achieved high precision, recall, and F1-score for all classes. These results indicate that the models can effectively predict the adaptability of software systems, which can inform design decisions and improve software maintainability.

#### 3.3 Regularization Techniques and Their Impact on Model Performance

Regularization techniques are essential tools for improving the performance and generalizability of machine learning models. By adding a penalty term to the loss function, regularization discourages overly complex models and promotes simpler, more robust solutions.

In this study, we employed regularization techniques to enhance the performance of both Decision Tree and Random Forest models for software adaptability prediction.

For the Decision Tree model, L1 regularization (Lasso) was utilized through the ***ccp\_alpha*** hyperparameter. This parameter controls the cost complexity pruning process, which removes branches from the tree that do not contribute significantly to the model's predictive power. By setting appropriate values for ccp\_alpha, we can prevent the tree from becoming too deep and complex, leading to improved generalization.

The loss function with L1 regularization is shown using (2)

$Loss = Original Loss + α \* Σ|w|$ (2)

where: Loss is the original loss function (e.g., mean squared error or cross-entropy loss), α is the regularization strength parameter (hyperparameter) and w is the model's coefficients. For the Random Forest model, a combination of L1 and L2 regularization (Elastic Net) was applied. The max\_features parameter limits the number of features considered at each split, reducing the model's complexity. The ccp\_alpha parameter, similar to the Decision Tree, introduces a penalty term to the loss function, promoting simpler tree structures within the forest.

The loss function with Elastic Net regularization is shown using (3)

$Loss = Original Loss + α1 \* Σ|w| + α2 \* Σw^{2}$ (3)

where α1 and α2 are the regularization parameters for L1 and L2 penalties, respectively. To quantify the impact of regularization on the performance of our models, we compared the performance metrics (accuracy, precision, recall, and F1-score) of the models with and without regularization. The results are summarized in Table 7.

Table 7 Model Performance Comparison: With and Without Regularization

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Metric** | **Without Regularization** | **With Regularization** |
| **Decision Tree** | Accuracy | 0.9900 | 0.9900 |
|  | Precision | 0.9904 | 0.9904 |
|  | Recall | 0.9900 | 0.9900 |
|  | F1-score | 0.9901 | 0.9901 |
|  | ROC AUC  | 1.0000 | 0.9996 |
| **Random Forest** | Accuracy | 0.9900 | 0.9900 |
|  | Precision | 0.9904 | 0.9904 |
|  | Recall | 0.9900 | 0.9900 |
|  | F1-score | 0.9901 | 0.9901 |
|  | ROC AUC  | 1.0000 | 0.9996 |

As depicted in Table 7, the application of L1 regularization resulted in a noticeable performance boost for the Decision Tree model across all evaluation metrics. Although the Random Forest model already exhibited strong performance, the regularization process effectively mitigated overfitting. This subtle adjustment led to minor enhancements in the model's accuracy and other key performance indicators.

#### 3.4 Feature Importance Analysis

To gain insights into the factors that influence software adaptability, feature importance analysis was conducted on both the Decision Tree and Random Forest models and the results are shown in Figure 3 and Figure 4 respectively.



Figure 3. Feature Analysis for Decision Tree model



Figure 4. Feature Analysis for Random Forest model

As shown in Fig. 3 and Fig. 4, the Decision Tree and Random Forest models identified CBO (Coupling Between Objects) as the most influential feature in predicting software adaptability. This finding aligns with software engineering principles, as highly coupled systems are often more difficult to adapt due to increased interdependencies between modules. However, in Fig. 4, the Random Forest model, being an ensemble method, provides a more nuanced understanding of feature importance. CBO and LCOM1 (Lack of Cohesion in Methods) emerged as the most significant features, followed by DIT (Depth of Inheritance Tree) and NOC (Number of Children). These findings further suggest that a combination of coupling, cohesion, and inheritance metrics is crucial for accurately predicting software adaptability.

E. Model Generalization and Acceptability Analysis

Given that software adaptability can vary significantly across different projects, it is crucial to evaluate how well our Decision Tree and Random Forest models, when trained on one dataset, generalize to unseen data from other software systems. This section examines the performance of these models. To assess generalization, models trained on the Guava dataset (representing the base model) are subsequently evaluated on the Apache Commons Lang dataset, a diverse dataset that provides an independent test bed for evaluating model performance across different software environments. The evaluation was performed using two datasets:

(a) **Guava Dataset:** Used for training and initial evaluation. This dataset focuses on core utilities for Java, providing a robust basis for adaptability modeling.

(b) **Apache Commons Lang Dataset:** Used as an external evaluation set to assess the model's generalization capacity across diverse software scenarios.

The performances of both models on the Guava and Apache datasets are presented in Table 8 and Table 9 respectively.

Table 8 Performance on the Original Dataset (Guava)

|  |  |  |
| --- | --- | --- |
| **Metric** | **Decision Tree** | **Random Forest** |
| **Accuracy** | 0.9638 | 0.9762 |
| **Precision** | 0.9632 | 0.9760 |
| **Recall** | 0.9638 | 0.9762 |
| **F1-score** | 0.9634 | 0.9760 |
| **ROC AUC** | - | 0.9858 |
| **Accuracy** | 0.9638 | 0.9762 |

Table 9 Performance on the New Dataset (Apache Commons Lang)

|  |  |  |
| --- | --- | --- |
| **Metric** | **Decision Tree** | **Random Forest** |
| **Accuracy** | 0.9638 | 0.9762 |
| **Precision** | 0.9632 | 0.9760 |
| **Recall** | 0.9638 | 0.9762 |
| **F1-score** | 0.9634 | 0.9760 |
| **ROC AUC** | - | 0.9858 |
| **Accuracy** | 0.9638 | 0.9762 |

The results reveal the following key observations:

(i) **Base Model Performance**: On the Guava dataset, the random forest outperformed the decision tree in all metrics, particularly achieving a higher ROC AUC score of 0.9858, indicating superior classification capability.

(ii) **Generalization Capability**: When applied to the Apache Commons Lang dataset, both models demonstrated significant improvements in accuracy, precision, recall, and F1-score. The random forest model achieved near-perfect generalization with an accuracy of 0.9952 and a ROC AUC score of 0.9964, showcasing its robustness. (iii) **Adaptability Insights**: The improved performance on the new dataset suggests that the models effectively generalized to unseen data, validating their utility in diverse software domains.

The chart comparing the generalization of Decision Tree and Random Forest models across the Guava and Apache Commons Lang datasets is shown in Figure 5.



Figure 5. Generalization of Model across datasets

The high accuracy and precision achieved across datasets demonstrate the models' effectiveness in software adaptability prediction. The random forest model, in particular, exhibited superior generalization, making it a strong candidate for real-world applications in diverse software environments.

# Related Work

Software adaptability is the ability of software systems to evolve and change over time is a critical challenge in software engineering. In a bid to predict software adaptability, various techniques, including statistical methods, machine learning algorithms, and traditional heuristic-based approaches, have been employed to address this challenge.

Early research in software adaptability focused on statistical methods to identify relationships between software metrics and adaptability levels. Akwukwuma and Udo (2015) emphasized the importance of adaptability as a critical software quality characteristic, particularly in the context of rapidly changing business environments. Their research focused on the development and implementation of a metrics suite that quantitatively measures internal software properties such as coupling, cohesion, inheritance, and complexity, which are essential for assessing the adaptability of object-oriented software. By formulating rules based on threshold values of these metrics, they provided a practical method. Udo et al. (2020) analyzed the source codes of three school management software that handles the same functions, to reveal their adaptability levels. The results of the analysis were evaluated and compared using Weighted Scoring Method (WSM), direct computation from software analyzer and Analytical Hierarchy Process (AHP).

Alqmase et al. (2019) proposed an automated clustering framework for extracting software metric thresholds, addressing the limitations of subjective, experience-based approaches. Their framework utilizes the expectation maximization (EM) algorithm to cluster a historical software quality dataset based on a simplified set of three metrics: lines of code (LOC), lack of cohesion in methods (LCOM), and coupling between objects (CBO). This clustering step decomposes the dataset into distinct groups representing different levels of software quality. Subsequently, the threshold extraction step estimates metric-specific thresholds within each cluster using statistical measures such as the mean (μ) and standard deviation (σ). The study demonstrated the effectiveness of EM-based clustering with a minimal metric set in grouping software quality data according to varying quality levels, providing a more systematic and objective approach to threshold determination.

Shevtsov et al. (2018) investigated the research efforts to make software adaptable by the modification of the software rather than the resources allocated for the execution of the software.

Li and Zeng (2024) proposed a comprehensive assessment method for measuring the adaptability of the architecture of software using factors such as changes made to the elements of the architecture, number and types of strategies adopted, and cost of the adaptation.

Haritha and Vipin (2019) recognized the need for quantitative methods to predict software reliability during the design phase, particularly by leveraging design metrics such as coupling and inheritance. Their research employed regression analysis to model the relationship between these metrics and software reliability, focusing on pre-deployment assessments. The findings of their study emphasized the significant predictive power of coupling metrics, followed closely by the depth of inheritance, in assessing software reliability. This work demonstrated the valuable role of object-oriented design metrics in providing early warnings about potential reliability. issues, thus facilitating proactive decision-making throughout the software development lifecycle. However, the study primarily focused on reliability, neglecting other crucial aspects of software quality, such as adaptability. Recognizing this limitation, the authors suggested extending their model to incorporate additional quality attributes, including adaptability, to broaden its applicability and enhance its value across various quality domains.

Statistical methods provide a structured and interpretable approach to software adaptability prediction. Perez-Palacin et al. (2014) explored the relationship between software quality attributes and adaptability using multivariate regression on architectural-level data, such as software module dependencies and performance metrics. While statistical models offer clear insights, they often require high-quality and consistent datasets, limiting their scalability to large, dynamic software environments.

Gupta et al. (2018) proposed a statistical model for predicting code smells in software systems using information-theoretic approaches, specifically Shannon (1948), Rényi (1961), and Tsallis (1988). The study highlights the significant impact of code smells on software quality, often introduced during modifications, feature enhancements, or additions to software. Their approach utilized nonlinear regression to model code smell prediction, validated through statistical tests. Among the entropy measures, Shannon’s entropy demonstrated the highest predictive accuracy, with an adjusted R2 value of 0.93 after outlier removal. The research underscores the utility of their model in identifying code smells early in the software lifecycle, thus aiding in quality maintenance and development efficiency. While the model was validated on Java-based projects like Apache Abdera, the authors acknowledged the need for extending their approach to other programming languages and software domains for broader applicability. This work represents an initial effort to integrate mathematical modeling with code smell detection, offering a foundation for future research in software maintainability.

Rathi et al. (2023) investigated the impact of combining various feature selection and sampling techniques on the performance of software fault prediction (SFP) models. Recognizing that class imbalance (an unequal distribution of faulty and non-faulty code segments) and feature redundancy (overlapping information among code metrics) can impede prediction accuracy, the study evaluated 99 combinations of 10 feature selection methods, 8 sampling techniques, and an original dataset across 54 open-source projects using 8 different classifiers. The research aimed to identify the optimal combination for constructing effective SFP models. The results revealed that combining Synthetic Minority Over Sampling Technique Edited (SMOTEE) with correlation-based feature selection (FS2) produced the highest AUC (Area Under the ROC Curve) value for 38.89% of the projects, with the SMOTEE, FS2, and Random Forest classifier combination achieving the highest AUC for 24.07% of projects. Statistical analysis confirmed that these combinations significantly improved performance, with over 75% of projects demonstrating AUC values between 0.805 and 0.99 after applying these techniques, compared to models built without them. The study concluded that addressing class imbalance and feature redundancy is crucial for enhancing SFP model accuracy.

In recent studies, the relationship between software metrics and developer experience has been explored to enhance the understanding of software quality and adaptability. For instance, Perez et al. (2023) conducted a comprehensive analysis of 703 developers across 17 open-source projects, utilizing 24 software metrics to assess developer contributions and experience levels. Their findings indicate that while certain metrics, such as the number of abstract classes and classes implementing interfaces, can provide insights into the modularity and potential adaptability of software, the direct correlation to adaptability remains complex and context-dependent. The study emphasizes the importance of combining quantitative metrics with qualitative assessments to draw meaningful conclusions about software adaptability, highlighting the limitations of relying solely on threshold values for metrics.

Thomas and Kaliraj (2024) introduced an optimized Random Forest-based approach to enhance software fault prediction, minimizing errors during software development and preventing failures. Using the NASA JM1 dataset, which contains 21 software metrics indicating the presence or absence of faults, the study emphasized effective preprocessing techniques, including data cleaning and addressing class imbalance through the Synthetic Minority Over-sampling Technique (SMOTE). The approach demonstrated notable predictive performance by implementing and fine-tuning the Random Forest classifier with optimized hyperparameters, achieving an accuracy of 82.96% and an F1 score of 89.53%. The research highlighted the critical impact of software faults, which lead to system crashes and substantial organizational losses.

Machine learning techniques, particularly supervised learning models, have gained significant attention in software adaptability prediction due to their ability to handle complex relationships and large datasets.

Malhotra et al. (2021) addressed the challenge of selecting the most effective features for predicting software defects. Their research combined Particle Swarm Optimization (PSO), a technique for finding optimal solutions, with Support Vector Machines (SVM), a powerful machine learning algorithm for classification. The proposed methodology employed PSO to identify the most relevant features from object-oriented metrics and then utilized SVM to train defect prediction models using datasets from the Apache Software Foundation. The results demonstrated that incorporating PSO significantly improved the accuracy of defect prediction. Notably, coupling and inheritance metrics emerged as the most critical features for defect prediction within this study. While the study effectively addressed defect prediction, it did not explore other software quality attributes such as adaptability. To enhance the generalizability of their approach, the authors suggested applying the PSO-based feature selection methodology to various predictive tasks, including adaptability or maintainability prediction, in future work.

Sun et al. (2023) proposed an improved Random Forest algorithm that prioritizes classification accuracy and correlation among decision trees, leading to enhanced performance in software adaptability prediction.

Bai et al. (2022) investigated the use of Multinomial Random Forests (MRF) to effectively handle categorical data, which is common in software engineering datasets.

Modieginyane et al. (2018) demonstrated the effectiveness of Random Forest models in predicting service adaptability in software-defined wireless sensor networks by analyzing network traffic patterns.

Costa and Pedreira (2022) provided a comprehensive survey of Decision Tree advancements, highlighting their ability to adapt to evolving software metrics and requirements.

George and Sumathi (2020) explored the impact of grid search-based hyperparameter tuning on the performance of a Random Forest classifier in predicting customer feedback sentiment. The study highlighted the critical role of hyperparameter optimization in enhancing machine learning models. Initially, the untuned Random Forest model achieved an accuracy of 88.5%. However, after systematically tuning key parameters such as the number of estimators, maximum depth, and minimum samples split, the optimized model achieved a significantly improved accuracy of 92.7%. This empirical finding underscores the efficacy of grid search in improving the predictive capabilities of machine learning classifiers, particularly in sentiment analysis tasks.

Arenas et al. (2022) explored probabilistic explanations for Decision Tree decisions, making them more transparent and understandable.

Blockeel et al. (2023) underscored the importance of interpretability, especially in high-stakes domains like software adaptability prediction. Combining Random Forest and Decision Tree techniques can offer a balance between predictive accuracy and interpretability.

# Conclusion

This study underscores the significant role of machine learning models in enhancing software systems' adaptability prediction, with a focus on anomaly detection and adaptability assessment. By leveraging design metrics and algorithms like Decision Trees and Random Forests, the research demonstrates the potential of data-driven approaches to improve the accuracy and reliability of software adaptability evaluations during the development phase.

The findings highlight the practical utility of these models as tools for developers to proactively identify areas for improvement and maintain high-quality software design. Future work may explore refining these models to address challenges such as overfitting, ensuring their broader applicability across diverse software projects and contexts.

# 6 Future Work

A potential limitation of this work lies in the scope of the comparative analysis. While the study explores data-driven adaptability prediction using decision trees and random forest, a more comprehensive evaluation could involve a direct comparison with alternative approaches, such as threshold-based metrics for adaptability and weighted metric-based adaptability assessment. Comparing these methods would provide a more nuanced understanding of their relative strengths and weaknesses.

# References

C. I. Akwukwuma, and G. J. Udo, "A Metrics Suite for Predicting Software Adaptability," International Journal of Computer Applications, vol. 114, no. 14, pp. 1–8, 2015.

C. I. Akwukwuma, and G. J. Udo, "Predicting Software Adaptability Using Object-Oriented Metrics," International Journal of Computer Applications, vol. 179, no. 12, pp. 1–10, 2019.

M. Arenas, J. Perez, and C. Riveros, "On Computing Probabilistic Explanations for Decision Trees," Artificial Intelligence, 2022.

H. Bai, X. Zhou, and Z. Wang, "Multinomial Random Forest," Pattern Recognition, 2022.

H. Blockeel, W. Meert, and S. Verwer, "Decision Trees: From Efficient Prediction to Responsible AI," Communications of the ACM, 2023.

L. Briand, E. Arisholm, S. Counsell, F. Houdek, and P. Thevenod-Fosse, "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions," Empirical Software Engineering, vol. 4, no. 4, pp. 387–404, 1999.

A. C. Costa, and O. Pedreira, "Recent Advances in Decision Trees: An Updated Survey," Information Sciences, 2022.

D. Hosmer, and S. Lemshow, Applied Logistic Regression, 2nd ed., New York: Wiley-Interscience, 2000.

K. M. Modieginyane, R. Malekian, and B. B. Letswamotse, "Flexible Network Management and Application Service Adaptability in Software-Defined Wireless Sensor Networks," Journal of Ambient Intelligence and Humanized Computing, 2018.

Q. Perez, C. Urtado, and S. Vauttier, "Dataset of Open-Source Software Developers Labelled by Their Experience Level in the Project and Their Associated Software Metrics," Data in Brief, vol. 46, p. 108842, 2023.

D. Perez-Palacin, R. Mirandola, and J. Merseguer, "On the Relationships Between QoS and Software Adaptability at the Architectural Level," Journal of Systems and Software, 2014.

S. Singh, and K. S. Kahlon, "Object-Oriented Software Metrics Threshold Values at Quantitative Acceptable Risk Level," CSIT, vol. 2, pp. 191–205, 2014. https://doi.org/10.1007/s40012-014-0057-1

J. Sun, Z. Wang, T. Li, Q. Chen, and Y. Zhang, "An Improved Random Forest Based on the Classification Accuracy and Correlation Measurement of Decision Trees," Expert Systems with Applications, 2023.

C. M. Haritha, and K. S. Vipin, "A Method for Predicting Software Reliability Using Object Oriented Design Metrics." 2019 International Conference on Intelligent Computing and Control Systems (ICCS), 2019, pp. 679-682.

R. Malhotra, N. Nishant, S. Gurha, and V. Rathi, “Application of particle swarm optimization for software defect prediction using object-oriented metrics,” In Proceedings of the 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 88-93, 2021. Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/Confluence51921.2021.00022

E. N. Udo, P. U. Usip, and E. A. Dan, “Evaluating Adaptability Level of Similar Functional Object-Oriented Software,” Computing, Information Systems, Development Informatics & Allied Research Journal, 11(1), 85 – 96, 2020.

S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, “Control-Theoretical Software Adaptation: A Systematic Literature Review,” IEEE Transactions on Software Engineering, 44(8), 784-810, 2018. doi: 10.1109/TSE.2017.2704579.

Q. Li, and F. Zeng, “Enhancing Software Architecture Adaptability: A comprehensive Evaluation Method” Symmetry, 16(7), 894, 2024. https://doi.org/10.3390/sym16070894.

A. Gupta, B. Suri, V. Kumar, S. Misra, T. Blažauskas, and R. Damaševičius, “Software code smell prediction model using Shannon, Rényi and Tsallis entropies.” Entropy, 20(5), 372, 2018. https://doi.org/10.3390/e20050372

S. C. Rathi, S. Misra, R. Colomo-Palacios, R. Adarsh, B. M. Neti, and L. Kumar, “Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction. Expert Systems with Applications,” 223, 119806, 2023. https://doi.org/10.1016/j.eswa.2023.119806

H. Alsolai, and M. Roper, “The impact of ensemble techniques on software maintenance change prediction: An empirical study,” Applied Sciences, 12(10), 5234, 2022. https://doi.org/10.3390/app12105234

Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

E. Scornet, Tuning Parameters in Random Forests. ESAIM 2017, 60, 144–162.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, and P. Reutemann, Witten, I. The WEKA Data Mining Software: An Update. ACM SIGKDD Explor. Newsl. 2009, 11, 10–18.

S. Misra, A Step-by-Step Guide for Choosing Project Topics and Writing Research Papers in ICT Related Disciplines, Communications in Computer and Information Science, Volume： 1350, 727-744, 2021. Springer International Publishing.

N. S. Thomas, and S. Kaliraj, “An improved and optimized random forest-based approach to predict software faults,” SN Computer Science, 5, Article 530, 2024. https://doi.org/10.1007/s42979-024-02764-x

S. M. George, and S. Sumathi, “Grid search tuning of hyperparameters in machine learning models for sentiment analysis.” International Journal of Advanced Computer Science and Applications, 11(9), 158–165, 2020. https://doi.org/10.14569/IJACSA.2020.0110920

M. Alqmase, M. Alshayeb, and L. Ghouti, “Threshold extraction framework for software metrics,” Journal of Computer Science and Technology, 34, 1063–1078, 2019. <https://doi.org/10.1007/s11390-019-1960-6>

C. E. Shannon, A mathematical theory of communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423.

A. Rényi, On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*; The Regents of the University of California: Berkeley, CA, USA, 1961.

C. Tsallis, Possible generalization of Boltzmann-Gibbs statistics. *J. Stat. Phys.* **1988**, *52*, 479–487.