Journal of Theoretical and Applied Information Technology
15th December 2025. Vol.103. No.23
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# SOFTWARE REFACTORING OPPORTUNITIES PREDICTION ON COMMENT-BASED FEATURES USING MACHINE LEARNING CLASSIFIERS

**ARCHANA PATNAIK[1] , NEELAMADHAB PADHY[2], LOV KUMAR[3], RASMITA PANIGRAHI[4]**

[1]Assistant Professor, GIET University, Department of Computer Science and Engineering, India

[2] Professor, GIET University, Department of Computer Science and Engineering, India

[3]Assistant Professor, NIT Kurukshetra, Department of Computer Engineering, India

[41]Assistant Professor, GIET University, Department of Computer Science and Engineering, India

[1]archanapatnaik@giet.edu,[2]dr.neelamadhab@giet.edu,[3]lovkumar@nitkkr.ac.in,[4]rasmita@giet.edu

## ABSTRACT

**Context:** Software Refactoring focuses on internal improvement of the code base without altering its external behavior. For enhancing the quality of real-time open-source projects, it is highly essential to perform complexity analysis followed by code restructuring. **Objectives:** In this work, we have created a hybrid model by combining different machine learning classifiers to predict refactoring mechanisms on code comments. Quality assessment of a software project is performed on comment tags using machine learning models. **Methodology:** Initially, the comment-based code snippets are pre-processed to separate the clean and dirty code snippets. The implementation is done by using different classifiers like Naive Bayes, Random Forest, Decision Trees, Logistic Regression, K-Nearest Neighbor, and Support Vector Machine. **Result:** We have identified six comment tags for refactoring prediction, and TODO comments is high as compared to others. We have implemented six different classifiers, and it is observed that the performance of the Random Forest classifier is high with 95% accuracy, 0.95 precision, 0.94 recall, and 0.93 f-measure. **Conclusion:** It is observed that the performance of sample 3 is better than other samples, and the Random Forest classifier has having highest accuracy. In our future work, we will predict refactoring on Python code.

**Keywords**: *Refactoring, Code Comments, Machine Learning Classifiers, Software Quality.*

## 1. INTRODUCTION

Software quality maintenance is highly essential for the existing open-source real-time projects that is developed using different programming languages. The major issues that contribute towards the depreciation of software product qualities are improper code organization, lack of technical knowledge, lack of understanding, inappropriate management of time, and inadequate manpower. All the listed problems create a negative impact on the software development stage by increasing the code complexity. Code smell occurs due to the complexity issues arise during the development state which contains more amount of dirty code and less clean code. Refactoring targets to enhance the software quality by performing internal alterations of code without changing its external outputs. Code refactoring can be performed on software snippets, which are a subset of open source projects and can be predicted by using different parameters such as metrics, source code, and comments. Most commonly found code smells are large classes, long methods, dead code, god classes, and conditional complexity, for which suitable refactoring techniques used are inline class, extract class, extract method, move method, rename method, pull down method, and inline method. Metrics-based refactoring predictions deal with Lines of Code (LOC), Kilo Lines of Code (KLOC), Source Lines of Code (SLOC) , Weighted Metrics Class(WMC), Conditional Complexity (CC), Number of Issues (NOI) etc. Source code combines both executable and non-executable lines of code present in the software snippets.

Refactoring clones [1] are used for enhancing the product quality by reducing the development cost and risk of failures, considering different sections and sub-sections of code base. In order to deal with the issues [2-3] of dirty code analysis, move method refactoring techniques are applied on the code fragment, which is predicted by using different machine learning classifiers. Restructuring activities [4] can be promoted by building the statistical model on source code readability metrics on different open-source projects. The productivity [5] of existing software can be enhanced by using hybrid deep learning framework that plays a vital role complexity analysis. It is necessary to [6] determine the challenges faced due to the change in commit behavior at the structural level without modifying its external outputs. For better quality enhancement [7], code metrics and commit messages acts as input parameters by using multi-class classification. Generalization process [8] uses different optimization algorithms and Support Vector Machine (SVM) for small-sized software systems. Code developers uses new features to improve [9] software maintainability and quality for open-source systems with an average precision of applied refactoring. BERT-based [10] pre-trained model for performing operations on commits using scoring metrics with a low quality score. Trivial [11] refactoring plays a vital role in reducing the problems of technical debt and low structural code quality. The practical implementation [12-13] of code refactoring can be assessed by considering its software quality metrics, which detect the presence of code smell at very early stage by using various machine learning approaches. The move method [14] restructuring mechanism is applied to original and new class in order to overcome the existing limitations by providing better version of the code. Automated [15] refactoring lexically identifies all the expression present in the code snippets and extract the local variables. The major scope [16] of improving the existing open-source projects can be improved by using software metrics, and the prediction can also be performed by using boosting classifiers. Manual integration [17-18] on an existing project may not provide an accurate outcome, for which automated RefBots are used in real-time for performing various types of quality-related activities on code comments.

Move method [19] techniques prediction can be done by using deep learning and LLM on source datasets. Many researchers proposed some hybrid models by [20-21] using code comments as input metrics that contain maximum non-executable lines of code. Software code [22] snippets are developed in various languages that completely depend on the commit message for understanding the code structural design that creates complexity issues. Refactor clones are considered as [23] the automated advisor of software developers that suggests the requirement of code alterations. Researchers demonstrated the software development [24] tasks by using the code comment readability concept using machine learning concept. Recent survey and investigations highlights the benefits of code review that is [25] detected using deep learning algorithms. Defect prediction can be predicted on Abstract Syntax Tree (AST) for defective Java code snippets [26] for predefined features. Object-oriented quality attributes uses move method refactoring using single and multiple classes that is four thousand [27] classes with eight metrics. Update comment [28] request is processed 1221 comments for 384 sampled data using machine learning classifiers. For removing technical debt different improvement activities [29-30] are performed on open-source projects that is used to fix the bugs by implementing extract class. Restructuring mechanism can also be conducted by using pull request and updating [31-32] Just-in-Time comments that works on the improvement of software quality. The most commonly used code assessment project is JUnit [33] which contains different types of old and new features for 485 commits collected from GitHub. Automatic code comments review can be generated [34-35] by using a hybrid model, which is termed as CodeDoctor that uses an encoder and decoder concept to identify the best result baselines. Quality assessment of the real-time [36] open-source projects can be conducted by using different types of regression techniques. Refactoring focuses on the improvement of the internal quality of code by enhancing the project performance. The complexity issues can be detected by using the concept of code smell, which is detected at the initial level of code development. This work considers comments as input parameters for the hybrid model and predicts the scope of code restructuring by using different machine learning classifiers. The performance is evaluated by using different quality metrics that performs error analysis and code assessment. The research contributions are listed below:

- In this work we have predicted the refactoring by using comments as source parameters which is processed and relevant feature selection is performed by using Recursive Feature Elimination.

- Text embedding is done by using TF-IDF techniques which process both clean and dirty code snippets with non-executable lines of code.
- Refactoring prediction is performed by using different machine learning classifiers that is Naïve Bayes, Decision Trees, Random Forest, K-Nearest Neighbor, Support Vector Machines and Logistic Regression. Six different comment tags are considered for efficient detection of restructuring mechanism.

**RQ 1: What is the efficiency of code comments in predicting the need for refactoring compared to code-based features?**
Code snippets are the combination of both executable and non-executable lines of code. Comments are the line that is not executed while compiling the code. The efficient comparison is done between code snippets and comments for identifying the scope of refactoring understanding its labels that is 0 and 1 represents low and high scope respectively.

**RQ 2: What types of comments are most indicative towards identifying the potential refactoring opportunities?**
For identifying the refactoring opportunities six different comments like bug related comments, TODO comments, heavy comments, verbose comments, redundant comments and depreciation features are taken into consideration and the performance analysis is conducted using different result metrics.

Section 1 represents the introduction by highlighting the research objectives and contributions of the particular work. Section 2 reflects the extensive literature survey followed by tabular comparison and research motivation. In section 3 we have provided the research framework representing the algorithm details. Section 4 justifies the research question solution in the form of result analysis and comparisons. In section 5 represent threats to validity which provide a summarized version of internal, construct and external validity. This work is concluded with relevant research outcome which is followed by future scope of improvements which is represented in section 6.

## 2. LITERATURE SURVEY

Refactoring approach targets to improve the software project quality by altering the code structure internally without changing the external outputs. The code analysis is performed by used small pieces of java code snippets which is collect from various open-source real-time projects. Various categories of comments are considered for predicting the requirement of refactoring based on the labels. Zhai et al.,[1] implemented Bayesian network for code clone refactoring by using 640 samples C source code and achieved accuracy of 90% contributing towards quality enhancement. Cui et al.,[2] recommended move method refactoring based on the semantic and structural representation of code for which a comparative analysis of traditional approach with machine learning techniques. Yamei et al.,[3] selected 32 key features of Junit project for analyzing the program performance which is refactored using move method refactoring. DeepFM, HINNPerf, DeepPerf and CNN algorithms are implemented for identifying Feature Envy code smells. Sellitto et al.,[4] analyses the effects of software restructuring on program readability for 156 open-source projects by building statistical models for eight samples that quantifies different software metrics. For analyzing [5] the impact of software refactoring seven different projects are used and highest accuracy is obtained for Oryx and mapdb project that is 96.37%.Fraihatet al.,[6] focus of improving the internal code base by using the TF-IDF feature engineering followed by XGBoost algorithm with highest accuracy for 573 commits. Sagar et al.,[7] predicted the restructuring activities by comparing software metrics with commit messages and identified move method, rename method, inline method, pull up method, extract method and push down method for 800 java projects with 5004 commits with average accuracy of 75% for Random Forest Classifiers. Akour et al.,[8] conducted several studies to illustrate the correlation between test suite and code coverage for different open-source systems like MapDB, McMMO, Junit and Antlr4. After conducting the experimental analysis, the highest accuracy is obtained for McMMO project with 96% for Support Vector Machine (SVM).Nyamawe et al.,[9] recommended suitable refactoring techniques on particular feature request for improving the quality with precision of 73% for 43 open-source java projects. Jesse et al.,[10] uses RefactorScore for calibrating the performance of BERT model for 1,111,246 commits for automatic evaluation of metrics for source code.

Many recent works foster the metrics-based tree approach for identifying non-trivial and trivial refactoring by using decision [11] trees and random forest classifiers for improving code maintainability. Based on the systematic literature

survey the quality enhancement focuses on evaluation of software metrics, detection of code smell and prediction of refactoring. Agnihotri et al.,[12] conducted 68 articles, 31 conferences and 32 journals for identifying feature envy, long method and data class code smells by using large open-source datasets focusing on software maintenance properties. Noel et al.,[13] considers 199 python projects for identifying refactoring commits using machine learning frameworks. It is observed by using MLRefScanner highest precision value is obtained for 11 types of restructuring techniques. Kurbatova et al.,[14] proposed a model code2vec for representing path-based code by implementing move method refactoring automatically on synthetic datasets. It is observed that for the data used feature envy code smell is detected which is refactored using move method. Jiang et al.,[15] used automatic value extractor for extracting local variable from the code base for detecting 243 and 263 defects on Eclipse and IntelliJIDEA projects. For identifying [16] the refactoring scope few software metrics like LOC, SLOC, WMC, CCL and NOIare considered. By using Recursive Feature Elimination (RFE) 20 important features are selected out of 115 and highest accuracy is achieved for Extreme Gradient Boosting technique. Alizadeh et al.,[17] triggered merge action on pull requests for adding new features and fixing bugs on intelligent software refactoring bots automated tools.

Bots quantitatively and qualitatively evaluating the performance of the bot models that fixes the quality of software. Huang et al.,[18] proposes a learning-based model to detect consistency between outdated comments and code and named it as CoCC with precision 90% for open-source projects. Some recent [19] experimental analysis considers a model with 12,475 samples that is used to evaluate MoveRec dataset using GRU-LSTM-CNN deep learning models that is used for extracting text, semantic and textual features with 74% f1-score. Louis et al.,[20] concentrate on new comment-based dataset for identifying the refactoring opportunities and the model achieved 74% precision and 13% recall for enhancing software maintainability concept. Frommgen et al.,[21] describes the application that resolves code review comments using machine learning models to automate the positive feedback of streamlined process enhancing the google productivity. In order to study [22] the critical behavior of refactoring mechanism, RefT5 automated model is proposed for 110 projects which is executed using BiLSTM and CodeT5 algorithms with precision and recall of 51.61% and 52.9%.Sheneamaret al.,[23] proposes clone based refactoring software for improving the classification results for clone set that extract the relevant features from the source dataset that is more robust to outliers.

Niaziet al.,[24] study revealed 26.66% literal and 26.66% insufficient comments by adding many categories like profile, license, irrelevant, improper, commented and auto-generated code. Tufano et al.,[25] uses both open source and industrial projects for reviewing the comments using deep learning architecture for 16% code from the source dataset. The ultimate goal of the software developer to conduct efficient code review that contributes towards enhancement of quality. Shippey et al.,[26] uses Abstract Syntax Tree (AST)for identifying the defective features of the java code base that highly effects the performance of the prediction models. Dallal et al.,[27] aims to improve the quality of software projects by analyzing the impact of move method on 4K classes from seven open-source systems using logistic regression classification models. Sheikhaei et al.,[28] manually considered 1221 comments for overviewing the stack overflow on 377 javascript and 289 python projects. Lammarinoet al.,[29] performs self-admitted actions on technical debt removal using statistical likelihood function for refactoring actions by following Self-Admitted Technical Debt (SATD) approach. Sharif et al.,[30] developed a code analysis tool to predict extract class refactoring with 79% accuracy for 11 correct class out of 14 object-oriented metrics for software developers. Liu et al.,[31] performed the code change mechanism using automated just-in –time comment updating mechanism by eliminating bad comments present in the code base for which they have considered 108K samples. Coelho et al.,[32] considers 1845 Apache samples for processing Modern Code Review (MCR) based pull requests by manually examining 2096 review comments and detected 1891 refactoring scope. Soares et al.,[33] uses 17 new features of Junit projects for testing 13 different types of code smells for evaluating the 38 pull request for the code developed by 212 software developers. By using the LLM, data augmentation [34] is used for classifying the review comments that encodes the input into semantic tokens. In the decoding mechanism decoder captures the relevant information for multi-category data. Rani et al.,[35] conducted the systematic literature review for assessing code comment quality for inspecting TODO comments by conducting 47 studies for

specifying the program boundaries. Quality of different project is assessed by using different machine learning regression and the performance of Lasso regression is high and it is considered as best approach for predicting code alteration.

*Table 1. Comparison of Related Works*

| Authors | Paper Title | Material and Methods | Research Outcomes |
|---|---|---|---|
| Cui et al.,[2] | RMove: Recommending Move Method Refactoring Opportunities using Structural and Semantic Representations of Code | Move method refactoring technique is implemented on the code fragment which is termed as RMove. | The result is evaluated on publicly available dataset by using three automated refactoring tools JDeodorant, JMove and PathMove that performs class level refactoring. |
| Sellitto et al.,[4] | Toward Understanding the Impact of Refactoring on Program Comprehension | Statistical model is developed to analyze the readability concept of 156 open-source projects. | Semi-automated recommendation system is used to qualitative attributes of software system. |
| Fraihat et al.,[6] | Detecting refactoring type of software commit messages based on ensemble machine learning algorithms | Hyper parameter optimization is performed followed by feature engineering and TF-IDF algorithm is implemented. | After performing the text processing, XGBoost algorithm is implemented with 100% accuracy |
| Noel et al.,[13] | Detecting Refactoring Commits in Machine Learning Python Projects: A Machine Learning-Based Approach | Ensemble machine learning approach is used for refactoring prediction on python projects. | MLRefScanner model is created to predict different types of refactoring that ensures to improve software quality. |
| Louis et al.,[20] | Where should I comment my code? A dataset and model for predicting locations that need comments | Machine learning model is created that identify the requirement of comments in code base. | The model framed successfully identifies the location of comments and gives precision 74% and recall 13% |
| Sheneamer et al.,[23] | An Automatic Advisor for Refactoring Software Clones Based on Machine Learning | Proposed a model that supports unique learning of features extraction followed by code cloning | Experimental analysis provides a different approach of identifying the refactored class. |

Table 1 represents the comparative work of the related field that contributes towards complexity analysis and refactoring prediction using different machine learning algorithms. Cui et al.,[2] considers move method refactoring on JDEodrant, JMove and Path Move projects. Louis et al.,[20] implemented different machine learning models for identifying the location of comments used in code base. Sheneamer et al.,[23] created and automated advisor for detecting a clone based refactoring prediction model. Fraihat et al.,[6] detected the common types of refactoring on commit message using TF-IDF algorithms. Sellitto et al.,[4] created a semi-automatic machine model for improving the software quality by performing complexity analysis. Noel et al.,[13] created a model and named as MLRefScanner that considers python projects and predict refactoring.

## 3. PROPOSED RESEARCH FRAMEWORK

Refactoring approaches considers various input parameters to improve the quality of existing software projects with java code base. The sample dataset is the representation of java code snippets which combines both code base and comments. The entire workflow mechanism is further sub divided into eight different phases.

**Phase 1: Source Code Dataset Collection from Five Sample Snippets:**

In order to predict the refactoring, we have considered five different snippets from various open-source java projects. The sample snippets combines both executable and non- executable lines of code. We have considered 106 sample snippets which is collected from various sources. Table *2* represent the description of sample dataset that contains both executable and non-executable lines

of code that is comments for five different samples. Based on the given sample, it is observed that sample 3 contains maximum number executable and non-executable code that is 60 and10 respectively. Sample 2 consists of minimum number of codes with 27 executable and 3 non-executable lines of code.

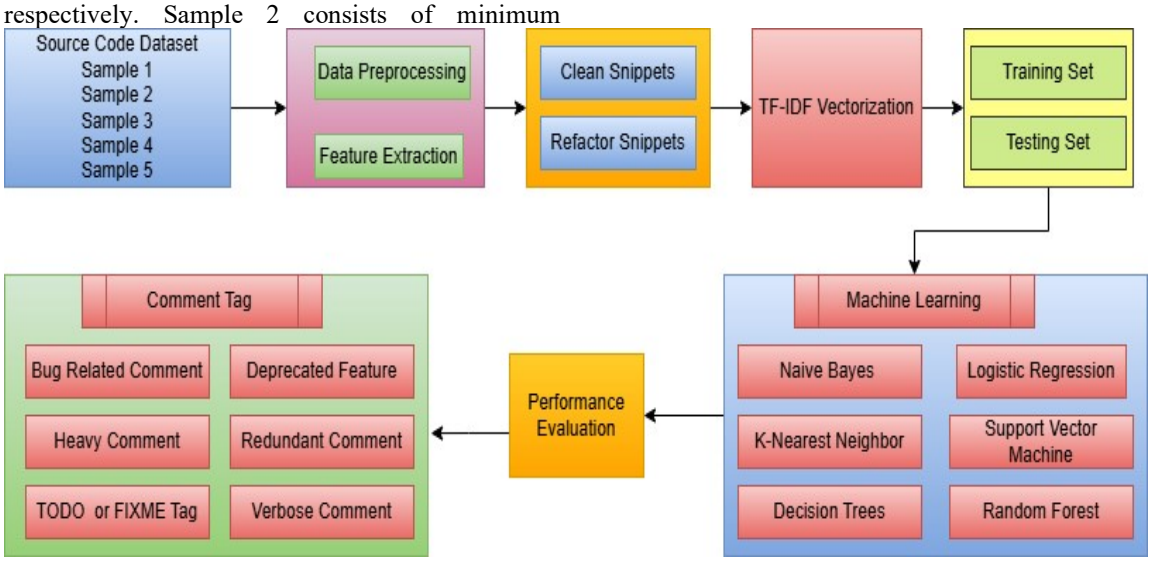


*Figure 1. Proposed Research Model*

*Table 2. Dataset Description*

| Sample Details | No of Lines | No of Executable Code | No of Non-Executable Code |
|---|---|---|---|
| Sample 1 | 50 | 45 | 5 |
| Sample 2 | 30 | 27 | 3 |
| Sample 3 | 70 | 60 | 10 |
| Sample 4 | 55 | 51 | 4 |
| Sample 5 | 48 | 44 | 4 |

**Phase 2: Data Preprocessing and Feature Extraction:**
The initial dataset collected is pre-processed to remove the outliers and noisy data. By applying the bin mean method we have handles the missing values and created a better version of dataset. In order to identify the relevant features present in the dataset, we have applied Recursive Feature Elimination (RFE) technique and synchronized the original data.

**Phase 3: Representation of Clean and Refactor Snippets:**
In this step we have identified the amount of clean code snippet present in the source dataset. The segregation of dirty and clean code snippets is performed manually by considering complexity as base parameter which many be space or time complexity that occurs due to more lines of code present in the snippets.

**Phase 4: TF-IDF (Term Frequency-Inverse Document Frequency) Vectorization:**
Comments are defined as unstructured text and by using TF-IDF technique we have converted those data into weighted features that describes potential of refactoring. Mathematically, it is represented as:

$$TF\text{-}IDF(t,d)=TF(t,d)\times IDF(t) \quad (1)$$

where t and d represent term data and document data. Multiple feature matrix are used for identifying the structural features of base snippets

**Phase 5: Training and Testing Datasets:**
Open-source projects data are collected from GitHub, Eclipse and Apache that contains multiple features. The data is divided into training and testing set. The step ensures the class balancing between refactoring and non-refactoring class.

**Phase 6: Implementation of Machine Learning Classifiers:**
In order to perform the comment based refactoring prediction six different machine learning classifiers like Naive Bayes, Decision Trees, K-Nearest Neighbor, Support Vector Machine, Logistic Regression and Random Forest algorithms are used.

**Phase 7: Performance Evaluation of Models:**
For evaluating the performance of the proposed model, different parameters like accuracy,

Journal of Theoretical and Applied Information Technology
15th December 2025. Vol.103. No.23
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

precision, recall and f-measure are calculated. The area under curve value is calculated for all the classifiers based on which the efficiency of the model.

**Phase 8: Identification of Comment Tags:** Six different comment tags are identified for identifying the scope of code internal alteration mechanism. It includes Verbose Comment, TODO Comments, Depreciated Features, Verbose Comments, Bug Related Comment and Redundant Comments.

## 4. RESULT ANALYSIS

Refactoring prediction can be performed by considering the source code snippet as input parameters that contained both comments and code. Based on the length of the code base, complexity analysis is performed by identifying the key features at different levels. The solution for the research question framed is represented below:

**RQ 1: What is the efficiency of code comments in predicting the need for refactoring compared to code-based features?**

For enhancement of software quality java code snippets are taken into consideration which contains comments and other executable lines of code based on which the performance evaluation can be done efficiently. In order to perform comment based refactoring prediction we have used six different classifiers and used six different comment tags for identifying the scope of restructuring mechanism.

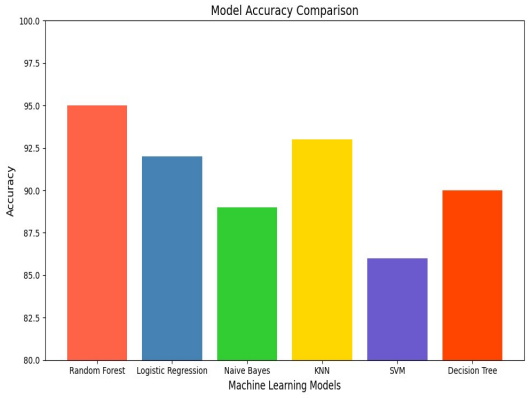


*Figure 2.1. Model Performance Evaluation*

Figure 2.1 represents the performance of different classifiers on comment-based dataset which in represented in terms of accuracy. For efficient tag analysis Random Forest and Support Vector Machines are having highest and lowest performance respectively. The model performance varies from 85 % to 95 % and K-Nearest Neighbor classifiers is considered as second highest approach for ensuring the project quality.

*Table 3. Performance Evaluation of Classifiers*

| Algorithms | Accuracy | Precision | Recall | F-Measure |
|---|---|---|---|---|
| Random Forest | 95% | 0.95 | 0.94 | 0.93 |
| Logistic Regression | 92% | 0.92 | 0.91 | 0.91 |
| Naïve Bayes | 89% | 0.89 | 0.89 | 0.88 |
| SVM | 93% | 0.85 | 0.85 | 0.85 |
| Decision Trees | 86% | 0.90 | 0.90 | 0.90 |
| KNN | 90% | 0.92 | 0.92 | 0.92 |

Table 3 represent the performance of different classifiers used for refactoring prediction which is represented using accuracy, precision, recall and f-measure parameters. The efficiency of the hybrid model is evaluated by using the performance metrics which gives best , average and worst result for all the classifiers used.

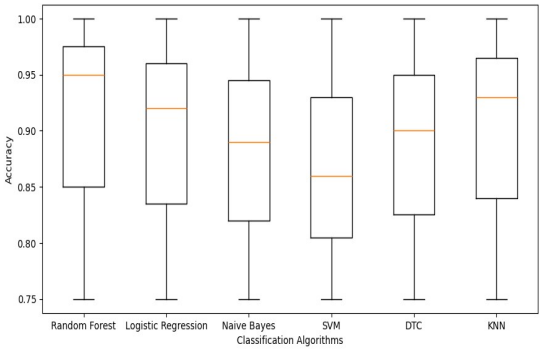


*Figure. 2.2 Accuracy*

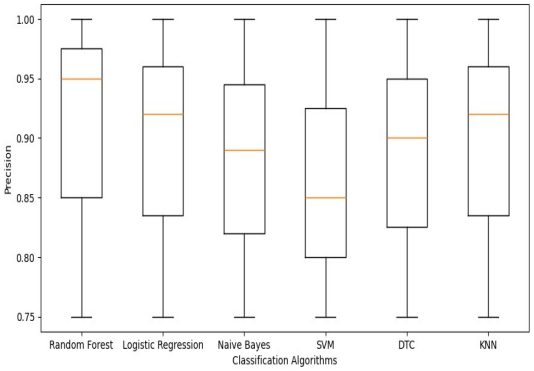


*Figure 2.3. Precision*

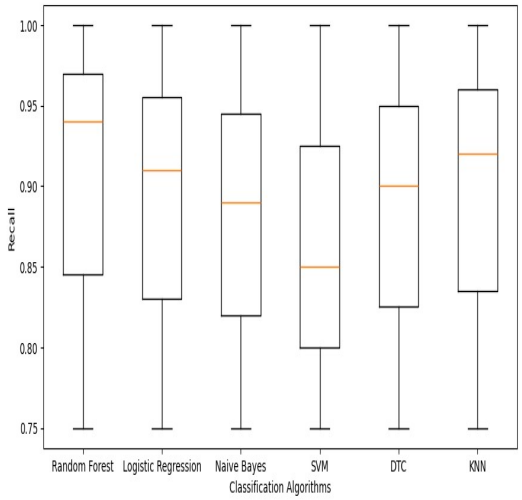


*Figure 2.4. Recall*

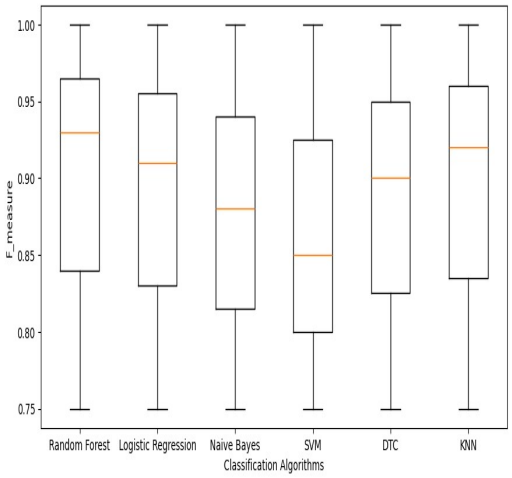


*Figure 2.5. F-Measure*

Figure 2.2 represent the performance in terms of accuracy which is maximum for Random Forest that is 95% Similarly for Logistic Regression, Naïve Bayes, SVM, Decision Tree and KNN the accuracies are 92%,89%,93%, 86% and 90 % respectively. Figure 2.3 represents the precision that varies within the range of 0.85 to 0.95 and is highest for Random Forest and is lowest for Support Vector Machine. Figure 2.4 illustrates the recall performance metrics that gives average performance for logistic regression , decision trees and Naïve Bayes classifier with values 0.91,0.90 and 0.89 respectively. Figure 2.5 shows the f-measure value of ensemble-based classifier is highest that is 0.93 and lowest for Support Vector Classifiers that is 0.85.

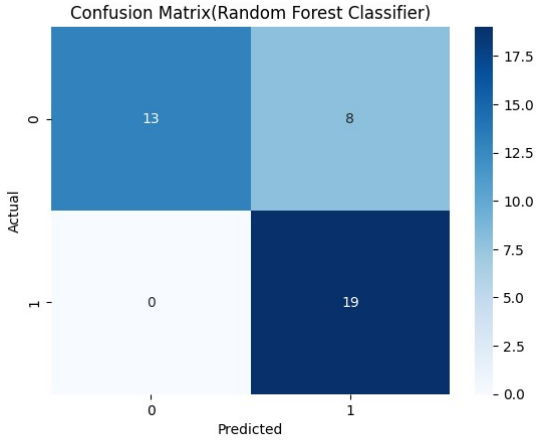


*Figure 3. Confusion Matrix for Random Forest Classifiers.*

Figure 3 represents the confusion matrix for all t classifiers used by considering the True Positive Rate, True Negative Rate, False Positive and Negative rate. It generally highlights the gap between actual and predicted values. For dealing with quality related issues the matrix is considered as a source parameter that efficiently detects whether refactoring is required or not by using 0 and 1 labels

*Table 4. AUC- ROC Curve*

| Machine Learning Classifiers | AUC Value |
|---|---|
| Random Forest | 1.00 |
| Logistic Regression | 0.99 |
| Naïve Bayes | 0.99 |
| SVM | 0.99 |
| Decision Tree Classifier | 0.99 |
| KNN | 1.00 |

Table 4 represent the Area Under Curve value for ROC curve plotted for each classifier used for evaluating the efficiency of the hybrid model and it is observed that AUC value is maximum for Random Forest and KNN classifiers that is 1.00 and for other classification model gives an average value 0.99 is obtained.

Journal of Theoretical and Applied Information Technology
15th December 2025. Vol.103. No.23
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

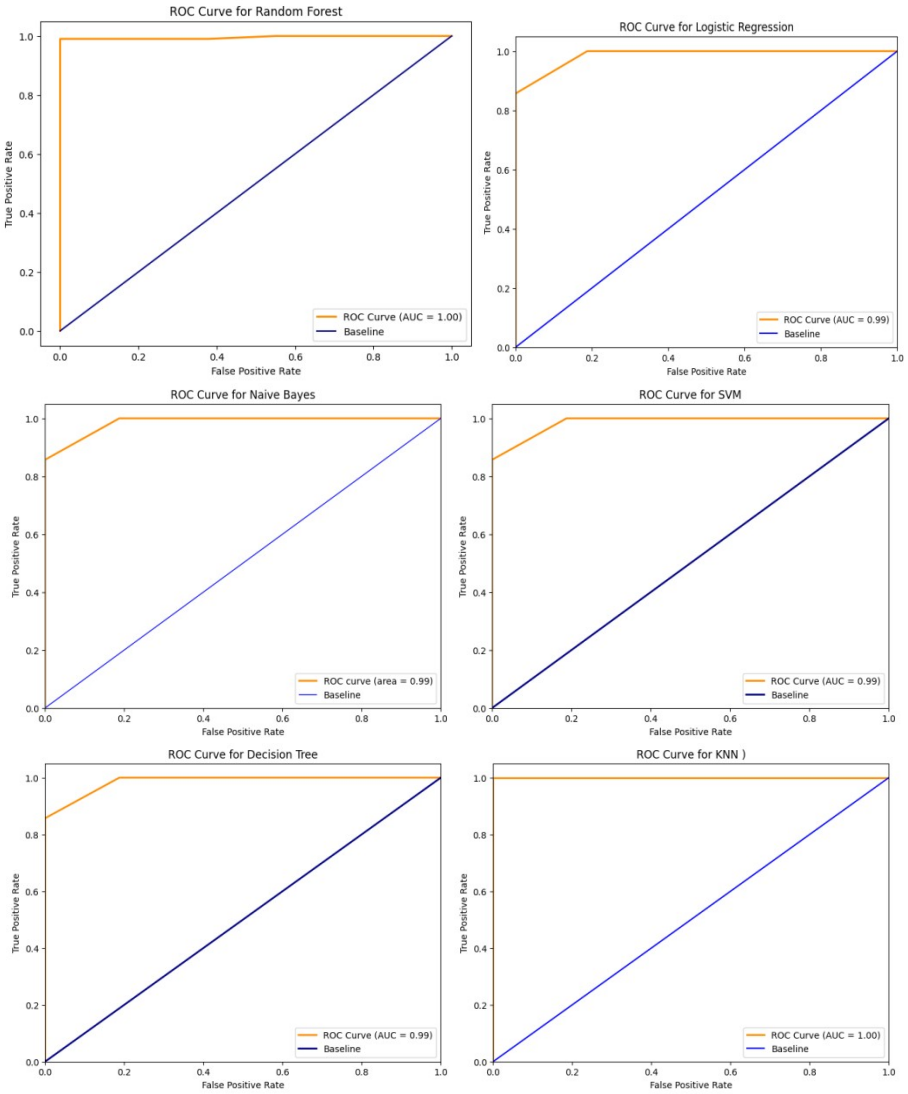


*Figure 4. AUC -ROC for Machine Learning Classifiers*

Figure 4 represents the ROC curve for all the classifier with AUC values that represents the overall performance of each classification model. In order to predict refactoring using comments, AUC value is considered as one of the valid parameters which is represented in Receiver Operating Characteristic Curve. The ensemble classification model and K-Nearest Neighbor model provides highest AUC value because the probabilistic ability increases the ranking ability by providing highest accuracy.

**RQ 2: What types of comments are most indicative towards identifying the potential refactoring opportunities?**

Basically metrics, code base, comments are considered as input pattern for predicting the need of code alteration mechanism. For identifying the potential of refactoring opportunities comments are considered as base parameters for which the details are numerical values are presented in the table 5 that describes different types of comments and labels providing different refactoring ability. It is observed that refactor label is high for extract class, rename variables and improves readability.

Journal of Theoretical and Applied Information Technology
15th December 2025. Vol.103. No.23
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

*Table 5. Refactoring Scope Identification*

| Comments | Refactoring Labels |
|---|---|
| Refactored method to improve readability | 1 |
| Fixed bug in login module | 0 |
| Extracted common code to new method | 1 |
| Updated documentation and comments | 0 |
| Renamed variables | 1 |
| Added new feature for user authentication | 0 |

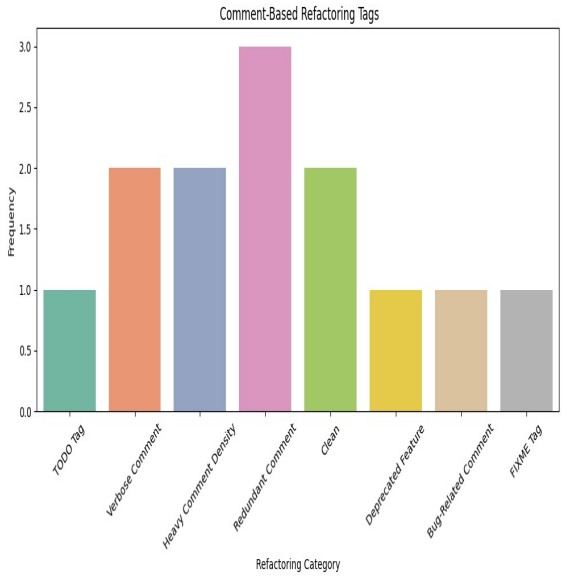


*Figure 5. Comment Based Refactoring Tags*

Figure 5 represent the comment based refactoring tags based on the frequency range and Redundant Comment is having highest performance as compared to other comments. However, the lowest scope is obtained for Depreciated Features, FIXME tag and Bug related comment tags. The importance of restructuring mechanism is represented in above table 5 that represent 0 and 1 for low and high score of refactoring respectively. Figure 6 represents co-occurrence of comment tags that represents the clean code and the code base contains bug related comments, heavy comment density, redundant comments, verbose comments, TODO or FIXME comments. The heatmap contains highest value that is 3 for redundant comments. The amount of clean and deprecated feature is very minimum as compared to other comments.

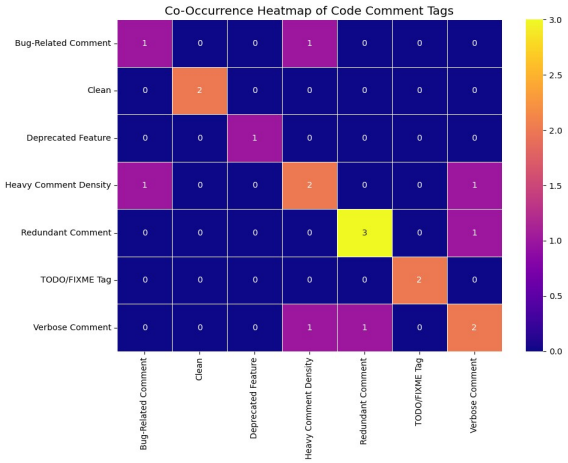


*Figure 6. Co-Occurrence of Comment Tags*

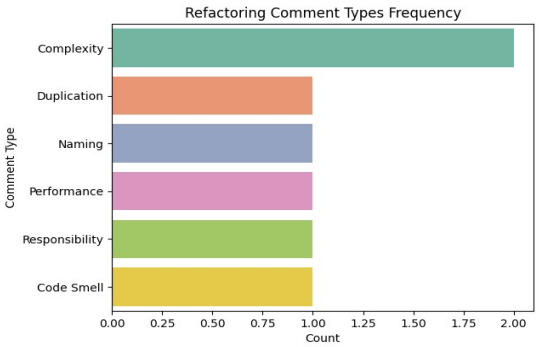


*Figure 7. Frequency Evaluation of Comment Tags*

In order to perform the complexity analysis frequency evaluation is conducted for all the comment tags which is represented in figure 7. The x-axis represents the frequency count which varies within a range of 0 to 2 and it is observed that complexity is very high. Similarly, the other parameters like duplication, naming, performance , responsibility and code smell values are 1,00.

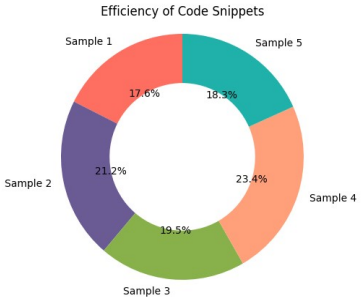


*Figure 8. Efficiency of Code Snippets*

Journal of Theoretical and Applied Information Technology
15th December 2025. Vol.103. No.23
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 8 shows the efficiency of five different code snippets which is highest for sample 4 that is 23.4%. The values of sample 1, sample 2, sample 3 and sample 5 are 17.6%, 21.3%.19.5% and 18.3% respectively. Based on the implementation conducted we can assume that refactoring prediction can be efficiently performed on snippet 4.

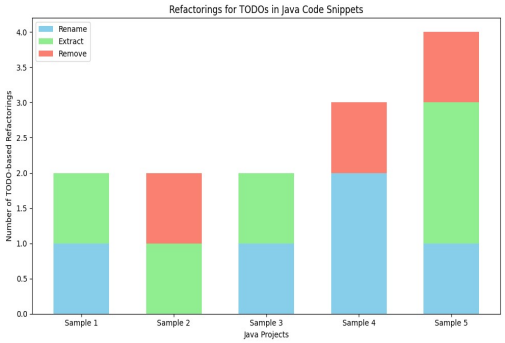


*Figure 9. Refactoring Prediction for Todo Comments*

We have conducted the experimental analysis for all the type of comments by considering different parameters and represents the detailed analysis for TODO comments in figure 9. In order to identify the amount of refactoring we have used three refactoring techniques which includes rename method, remove method and extract method.

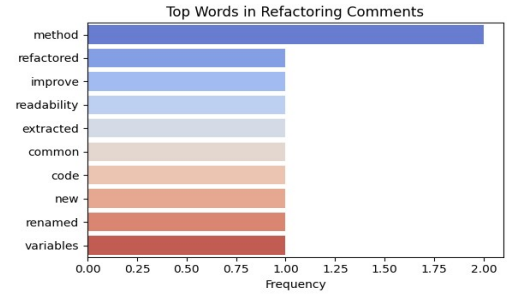


*Figure 10.1. Words Embedding in Refactoring*

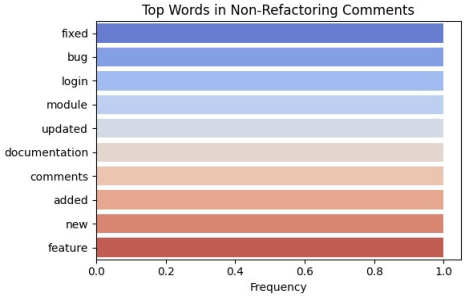


*Figure 10.2. Words embedding in non- refactoring*

Figure 10.1 and 10.2 represents the word embedding techniques for both refactoring and non-refactoring mechanism with varying frequency of 0 to 2. The top words in refactoring comments are refactored, new, renamed, method, code, variables, common, improve and extracted. For non-refactoring comments top words are fixed, bug, login, module, updated, documentation, comments, added, new and features.

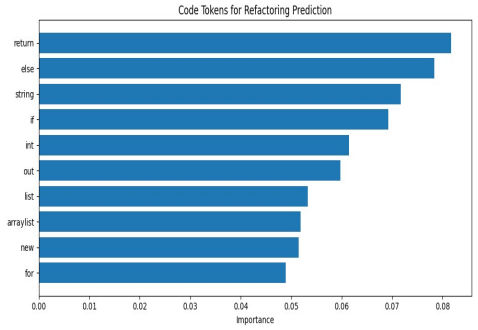


*Figure 11.1. Code Sample 1*

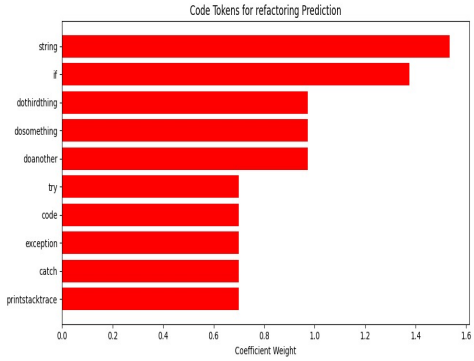


*Figure 11.2. Code Sample 2*

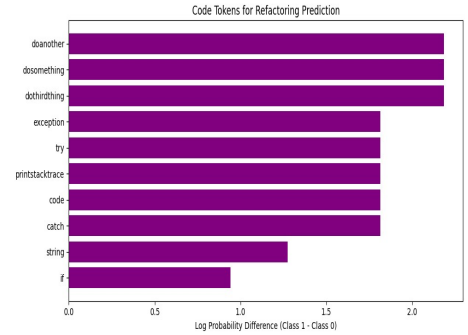


*Figure 11.3. Code Sample 3*

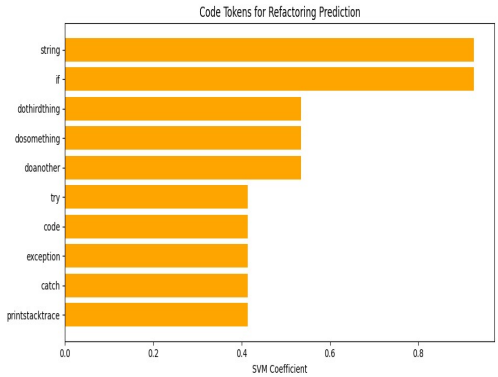


*Figure 11.4. Code Sample 4*

*Figure 11. Refactoring Prediction for Sample Source Snippets*

Figure 11 illustrates refactoring mechanism for sample snippets for different technical parameters present in the code base. Figure 11.1 represents first sub sample 1 feature importance is evaluated for like return, else, string, if, int, out, list, array list, new and for keywords. Similarly figure 11.2 shows sub sample snippet 2 string, if , dothirdthing, dosomething, doanother ,try, code, exception, catch and printstacktrace. Figure 11.3 represent sample snippet 3 with log probability difference for code token that is doanother, dosomething, dothirdthing, exception, try, printstack, code, catch, string, and if keywords. For sample snippets 4, the coefficients performs well for top words, which is already extracted in other sample snippets and is represented in Figure 11.4. To illustrate the practical implementation of this work to the real-time industry, comment-based restructuring prediction is highly efficient, which identifies the scope of improvement for the existing source code snippets.

## 5. THREATS TO VALIDITY

Source code software metrics used for refactoring prediction are collected from the tera-PROMISE repository, GitHub and other real-time sources. Although we believe the data has been obtained consistently, we cannot give the surety for the dataset collected form open-source projects is 100% accurate. We can provide the balanced and processed data after using RFE feature selection techniques but cannot assured that is the exact version of the original dataset collected directly. The study analyses the prediction of refactoring on comments using classification model and evaluated the performance parameters like accuracy, precision , f-measure and recall which is implemented for Machine Learning classification algorithms. Area Under Curve (AUC) value is also evaluated for plotting the ROC curve and the performance is represented in terms of confusion matrix. Therefore, we are confident that the construct validity of risk is eliminated from this work. We have considered the source code metrics for java projects but the outcome may be different for other projects based on different programming languages. As a result of which there exists the risk of external validity for this work.

## 6. CONCLUSION

Software refactoring assures the quality of the real-time projects by performing the complexity analysis of source code based collected from large java projects. Our work targets to enhance the software product quality by reorganizing the code structure developed by the software developers.

- In this work we have considered the source code base collected from various open-source projects that consists of both executable and non-executable line of code. For identifying the relevant features, we have used Recursive Feature Elimination Techniques (RFE) and segregated the clean and dirty sample snippets.
- For efficient refactoring prediction on comment-based datasets we have implement six different machine learning classifiers and Random Forest ensemble classifiers provides highest accuracy of 95% with precision 0.95, recall 0.94 and f-measure 0.93 respectively. It is observed that for sample 4.
- Different types of comment tag used are Verbose Comments, Redundant Comments, TODO comments, Deprecated Features , Heavy Comments and Bug Related Comments. TODO comment analysis is represented for sample snippets by using rename, extract and remove refactoring methods.

Recent works considers java code as base input parameter by performing different activities like software metrics extraction, code smell detection, complexity analysis, code spinning, code optimization and code internal alteration prediction. In the future work complex python code snippets will consider for predicting the requirement of code alteration by using some advance machine learning and deep learning models.

## REFERENCES:

[1] Zhai, Y., Liu, D., Wu, C. and She, R., 2020. A Recommendation Approach Based on Bayesian Networks for Clone Refactor. Computers, Materials & Continua, 64(3).

[2] Cui, D., Wang, S., Luo, Y., Li, X., Dai, J., Wang, L. and Li, Q., 2022, October. Rmove: Recommending move method refactoring opportunities using structural and semantic representations of code. In 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 281-292). IEEE.

[3] Yu, Y., Lu, Y., Liang, S., Zhang, X., Zhang, L., Bai, Y. and Zhang, Y., 2025. Predicting a Program's Execution Time After Move Method Refactoring Based on Deep Learning and Feature Interaction. Applied Sciences (2076-3417), 15(8).

[4] Sellitto, G., Iannone, E., Codabux, Z., Lenarduzzi, V., De Lucia, A., Palomba, F. and Ferrucci, F., 2022, March. Toward understanding the impact of refactoring on program comprehension. In 2022 IEEE international conference on software analysis, evolution and reengineering (SANER) (pp. 731-742). IEEE.

[5] Pandiyavathi, T. and Sivakumar, B., 2025. Software Refactoring Network: An Improved Software Refactoring Prediction Framework Using Hybrid Networking-Based Deep Learning Approach. Journal of Software: Evolution and Process, 37(2), p.e2734.

[6] Al-Fraihat, D., Sharrab, Y., Al-Ghuwairi, A.R., Sbaih, N. and Qahmash, A., 2024. Detecting refactoring type of software commit messages based on ensemble machine learning algorithms. Scientific Reports, 14(1), p.21367.

[7] Sagar, P.S., AlOmar, E.A., Mkaouer, M.W., Ouni, A. and Newman, C.D., 2021. Comparing commit messages and source code metrics for the prediction refactoring activities. Algorithms, 14(10), p.289.

[8] Akour, M., Alenezi, M. and Alsghaier, H., 2022. Software refactoring prediction using SVM and optimization algorithms. Processes, 10(8), p.1611.

[9] Nyamawe, A.S., Liu, H., Niu, N., Umer, Q. and Niu, Z., 2019, September. Automated recommendation of software refactorings based on feature requests. In 2019 IEEE 27th International Requirements Engineering Conference (RE) (pp. 187-198). IEEE.

[10] Jesse, K., Kuhmuench, C. and Sawant, A., 2023. RefactorScore: Evaluating refactor prone code. IEEE Transactions on Software Engineering, 49(11), pp.5008-5026.

[11] Pinheiro, D., Bezerra, C.I.M. and Uchôa, A., 2022, October. How do Trivial Refactorings Affect Classification Prediction Models?. In Proceedings of the 16th Brazilian Symposium on Software Components, Architectures, and Reuse (pp. 81-90).

[12] Agnihotri, M. and Chug, A., 2020. A systematic literature survey of software metrics, code smells and refactoring techniques. Journal of Information Processing Systems, 16(4), pp.915-934.

[13] Noei, Shayan, Heng Li, and Ying Zou. "Detecting Refactoring Commits in Machine Learning Python Projects: A Machine Learning-Based Approach." ACM Transactions on Software Engineering and Methodology 34, no. 3 (2025): 1-25.

[14] Kurbatova, Z., Veselov, I., Golubev, Y. and Bryksin, T., 2020, June. Recommendation of move method refactoring using path-based representation of code. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (pp. 315-322).

[15] Jiang, Y., Chi, X., Zhang, Y., Ji, W., Li, G., Wang, W., Xia, Y., Zhang, L. and Liu, H., 2025. Automated Recommendation of Extracting Local Variable Refactorings. ACM Transactions on Software Engineering and Methodology.

[16] Patnaik, A., Padhy, N., Kumar, L. and Panigrahi, R., 2025, February. Metrics-Based Refactoring Prediction Using Boosting Algorithm. In 2025 International Conference on Emerging Systems and Intelligent Computing (ESIC) (pp. 459-464). IEEE.

[17] Alizadeh, V., Ouali, M.A., Kessentini, M. and Chater, M., 2019, November. RefBot: Intelligent software refactoring bot. In 2019 34th IEEE/ACM international conference on automated software engineering (ASE) (pp. 823-834). IEEE.

[18] Huang, Y., Chen, Y., Chen, X. and Zhou, X., 2025. Are your comments outdated? Toward automatically detecting code-comment consistency. Journal of Software: Evolution and Process, 37(1), p.e2718.

[19] Zhang, Yang, Yanlei Li, Grant Meredith, Kun Zheng, and Xiaobin Li. "Move method refactoring recommendation based on deep learning and LLM-generated information." Information Sciences 697 (2025): 121753.

[20] Louis, A., Dash, S.K., Barr, E.T., Ernst, M.D. and Sutton, C., 2020, June. Where should I comment my code? A dataset and model for predicting locations that need comments. In Proceedings of the ACM/IEEE 42nd international conference on software engineering: new ideas and emerging results (pp. 21-24).

[21] Frömmgen, Alexander, Jacob Austin, Peter Choy, Nimesh Ghelani, Lera Kharatyan, Gabriela Surita, Elena Khrapko et al. "Resolving code review comments with machine learning." In Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, pp. 204-215. 2024.

[22] Li, T. and Zhang, Y., 2024. Multilingual code refactoring detection based on deep

learning. Expert Systems with Applications, 258, p.125164.
[23] Sheneamer, A.M., 2020. An automatic advisor for refactoring software clones based on machine learning. IEEE Access, 8, pp.124978-124988.
[24] Niazi, T., Das, T., Ahmed, G., Waqas, S.M., Khan, S., Khan, S., Abdelatif, A.A. and Wasi, S., 2023. Investigating novice developers' code commenting trends using machine learning techniques. Algorithms, 16(1), p.53.
[25] Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D. and Bavota, G., 2021, May. Towards automating code review activities. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 163-174). IEEE.
[26] Shippey, T., Bowes, D. and Hall, T., 2019. Automatically identifying code features for software defect prediction: Using AST N-grams. Information and software technology, 106, pp.142-160.
[27] Al Dallal, J., Abdulsalam, H., AlMarzouq, M. and Selamat, A., 2024. Machine learning-based exploration of the impact of move method refactoring on object-oriented software quality attributes. Arabian Journal for Science and Engineering, 49(3), pp.3867-3885.
[28] Sheikhaei, M.S., Tian, Y. and Wang, S., 2023. A study of update request comments in Stack Overflow answer posts. Journal of Systems and Software, 198, p.111590.
[29] Iammarino, M., Zampetti, F., Aversano, L. and Di Penta, M., 2021. An empirical study on the co-occurrence between refactoring actions and self-admitted technical debt removal. Journal of Systems and Software, 178, p.110976.
[30] Sharif, K.Y., Muftah, M.G., Osman, M.H. and Zakaria, M.I., 2021. Code analysis tool to detect extract class refactoring activity in Vb. Net classes. Turkish Journal of Computer and Mathematics Education, 12(3), pp.2172-2177.
[31] Liu, Z., Xia, X., Yan, M. and Li, S., 2020, December. Automating just-in-time comment updating. In Proceedings of the 35th IEEE/ACM International conference on automated software engineering (pp. 585-597).
[32] Coelho, F., Tsantalis, N., Massoni, T. and Alves, E.L., 2021, October. An empirical study on refactoring-inducing pull requests. In Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 1-12).
[33] Soares, E., Ribeiro, M., Gheyi, R., Amaral, G. and Santos, A., 2022. Refactoring test smells with junit 5: Why should developers keep up-to-date?. IEEE Transactions on Software Engineering, 49(3), pp.1152-1170.
[34] Li, Y., Wu, Y., Wang, Z.A., Huang, L., Wang, J., Li, J. and Huang, M., 2025. CodeDoctor: multi-category code review comment generation. Automated Software Engineering, 32(1), p.25.
[35] Rani, P., Blasi, A., Stulova, N., Panichella, S., Gorla, A. and Nierstrasz, O., 2023. A decade of code comment quality assessment: A systematic literature review. Journal of Systems and Software, 195, p.111515.
[36] Patnaik, A., Padhy, N., Kumar, L. and Panigrahi, R., 2024, February. Quality Assessment of Software Projects Using Machine Learning. In National conference on Intelligent Systems, IoT, and Wireless Communication for the Society (pp. 131-140). Singapore: Springer Nature Singapore.