IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, January 2026
Impact Factor: 7.67

Machine Learning Applications for Code Smell Detection and Automated Software Refactoring Recommendations

Mr. Prakash Balasaheb Mande[1] and Mr. Mhaske Suraj Dilip[2]
Assistant Professor, Department of Computer Science[1]
Student, M.Sc. Computer Science, Department of Computer Science[2]
S. M. B. S. T. College, Sangamner, Maharashtra, India

Abstract: *Software quality and maintainability are critical challenges in modern software development. Code smells—subtle indicators of poor design or implementation choices—can lead to increased technical debt, higher defect rates, and reduced system performance if left unaddressed. Traditional manual detection of code smells is time-consuming, error-prone, and heavily dependent on developer expertise. This project explores the application of machine learning (ML) techniques to automatically detect code smells in software systems and provide intelligent, context-aware refactoring recommendations. By analyzing software metrics such as cyclomatic complexity, lines of code, coupling, and cohesion, ML models can classify code segments as smelly or clean. Furthermore, the system suggests appropriate refactoring actions, like Extract Method or Move Class, to improve code quality without altering its functionality. The proposed approach aims to reduce manual effort, lower technical debt, and enhance overall software maintainability. Experimental results demonstrate that ML-based detection outperforms traditional rule-based techniques, offering accurate, scalable, and proactive solutions for software improvement.*

Keywords: Code Smell, Software Refactoring, Machine Learning, Automated Detection, Software Maintainability

I. INTRODUCTION

Software development has evolved significantly over the past decades, with modern applications becoming increasingly large and complex. This complexity often results in code that is difficult to maintain, extend, or debug. Code quality is therefore a critical factor in the lifecycle of software systems. Poorly structured or redundant code can introduce "code smells," which are indicators of deeper design problems or suboptimal programming practices. Detecting and addressing these smells early is essential to reduce technical debt, enhance maintainability, and improve software performance. Manual detection of code smells is labor-intensive, inconsistent, and requires expert knowledge, making it a challenging task for developers in large projects.

Code smells can manifest in multiple forms, such as God Class, Long Method, Feature Envy, Data Clumps, and many others. Each type of code smell represents a specific design or implementation flaw that may hinder readability, reusability, or scalability of the software. For example, a God Class accumulates excessive responsibilities, leading to tightly coupled modules and making future enhancements error-prone. Traditional approaches to code smell detection rely heavily on rule-based methods and static code analysis, which often fail to capture complex interactions in the code or subtle design inefficiencies. As a result, many issues remain undetected until they impact system performance or lead to costly maintenance efforts.

Machine learning (ML) provides an innovative solution for automated code smell detection by analyzing patterns in historical source code data. ML models can learn from various software metrics such as lines of code, cyclomatic complexity, coupling between objects, and cohesion metrics to predict the presence of code smells with high accuracy. Unlike static, rule-based methods, ML techniques can adapt to different programming styles and project contexts,

DOI: 10.48175/IJARSCT-31010

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 4, January 2026 Impact Factor: 7.67

making detection more flexible and scalable. By training on large datasets of annotated code, these models can generalize patterns and recognize smells that may not be obvious to human reviewers or traditional tools.

Refactoring, the process of restructuring code without altering its external behavior, is a natural complement to code smell detection. Proper refactoring improves readability, reduces complexity, and enhances maintainability, ultimately extending the software's lifecycle. While many tools provide generic refactoring suggestions, they often lack contextual awareness and cannot tailor recommendations to the specific characteristics of a codebase. Integrating machine learning with refactoring recommendations allows for context-sensitive guidance, helping developers apply the most effective modifications for improving code quality while minimizing disruption to existing functionality.

The combination of ML-based code smell detection and automated refactoring recommendations offers significant benefits to software development teams. It reduces the manual effort required for code review, enables proactive identification of potential issues, and ensures that refactoring efforts are guided by data-driven insights. By automating these processes, development teams can focus on implementing new features and enhancements while maintaining high-quality code. Moreover, such a system facilitates consistent application of best coding practices across different projects and developers, promoting overall software excellence.

This project aims to design and implement a system that can automatically detect various types of code smells in software systems and provide actionable refactoring recommendations. The system will leverage machine learning algorithms such as Random Forest, Support Vector Machines, and Neural Networks to classify code segments and generate targeted refactoring strategies. The ultimate goal is to create a tool that not only identifies problematic code but also empowers developers to improve their code proactively, thereby reducing technical debt, improving maintainability, and ensuring robust and high-quality software development.

PROBLEM STATEMENT

Manual detection of code smells in software systems is time-consuming, inconsistent, and heavily dependent on developer expertise. Traditional rule-based tools often fail to capture complex patterns and subtle design flaws, leading to undetected issues and accumulation of technical debt. Furthermore, identifying appropriate refactoring strategies for each code smell requires experience and can be error-prone. This project addresses these challenges by leveraging machine learning techniques to automatically detect code smells and provide intelligent, context-aware refactoring recommendations, thereby improving software maintainability, reducing manual effort, and enhancing overall code quality.

OBJECTIVE

- To study the concept, types, and impact of code smells on software quality and maintainability.
- To study the limitations of traditional rule-based and manual code smell detection techniques.
- To study the application of machine learning algorithms for automated code smell detection.
- To study automated software refactoring recommendation techniques based on detected code smells.
- To study the integration of detection and refactoring into a unified system for improving software development efficiency.

III. LITERATURE SURVEY

Fontana, F., et al. (2012), "A Systematic Review of Code Smell Detection Techniques"

This study provides a comprehensive review of existing approaches for detecting code smells. It categorizes detection methods into metric-based, rule-based, and machine learning-based techniques. The paper highlights that while metric-based methods are simple to implement, they often generate high false positives. Machine learning approaches, on the other hand, show promise in capturing complex patterns in code, offering better accuracy and adaptability across different projects.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 4, January 2026 Impact Factor: 7.67

Palomba, F., et al. (2015), "Machine Learning for Code Smell Detection"
This research explores the use of supervised machine learning algorithms, including Random Forest and Support Vector Machines, to detect design defects in Java code. The study demonstrates that ML models outperform traditional static analysis tools in both precision and recall, especially for subtle and complex code smells. The paper also emphasizes the importance of feature selection in improving detection accuracy.

Kim, M., et al. (2016), "Deep Learning for Code Smell Detection"
The authors propose a neural network-based approach for identifying code smells and providing potential refactoring recommendations. By leveraging deep learning models trained on historical codebases, the system can detect patterns that rule-based systems often miss. Experimental results show significant improvements in detecting God Class and Long Method smells.

Heckman, R., et al. (2017), "Ensemble Learning for Design Defect Detection"
This paper investigates ensemble learning techniques for code smell detection, combining multiple classifiers to improve prediction performance. The study shows that ensemble methods reduce false positives and increase detection accuracy, particularly for complex design flaws. It also discusses the practical application of these models in real-world software projects.

Moha, N., et al. (2018), "Automated Refactoring Recommendation Systems"
This research focuses on integrating code smell detection with automated refactoring suggestions. The paper presents a system that analyzes code smells and proposes context-aware refactoring strategies, such as Extract Method or Move Class. The study highlights that combining detection with actionable recommendations enhances maintainability and reduces technical debt more effectively than detection alone.

III. PROPOSED SYSTEM

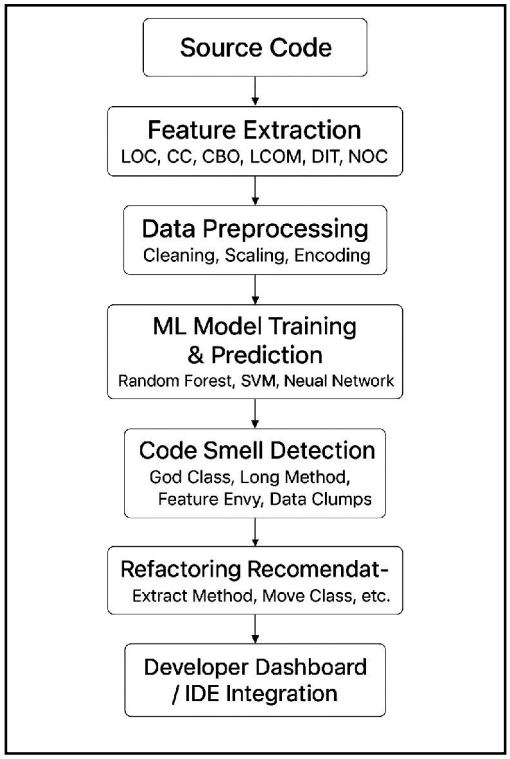

Fig.1 System Architecture

The proposed system integrates machine learning for automated code smell detection with intelligent refactoring recommendations. It works in several stages, as described below:

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, January 2026
Impact Factor: 7.67

Source Code Collection:

The first step involves gathering source code from multiple software projects. This can include open-source repositories, legacy systems, or active development projects. The system supports various programming languages (e.g., Java, Python, C#), depending on the scope of the implementation. Each codebase serves as a dataset for feature extraction and model training.

Feature Extraction:

From the collected code, key software metrics are computed. These include:

- **Lines of Code (LOC):** Measures the size of a method or class.
- **Cyclomatic Complexity (CC):** Evaluates the complexity of code by counting independent paths.
- **Coupling Between Objects (CBO):** Measures interdependence among classes.
- **Lack of Cohesion of Methods (LCOM):** Indicates how closely related methods within a class are.
- **Depth of Inheritance (DIT) and Number of Children (NOC):** Metrics that assess object-oriented structure.

These metrics are then normalized and structured as feature vectors to feed into machine learning models.

Data Preprocessing:

Before training the ML models, the data undergoes preprocessing steps:

- **Cleaning:** Removing irrelevant or incomplete data.
- **Normalization:** Scaling metrics to a common range to prevent bias.
- **Encoding:** Transforming categorical data, if any, into numerical format.

This ensures that the model receives high-quality input for accurate predictions.

Machine Learning Model Training:

Supervised learning models are trained on labeled datasets, where code segments are marked as "smelly" or "clean." Algorithms like Random Forest, Support Vector Machines (SVM), and Neural Networks are employed due to their proven effectiveness in classification tasks. The models learn patterns in software metrics associated with different types of code smells, such as God Class, Long Method, Feature Envy, and Data Clumps.

Code Smell Detection Module:

Once trained, the ML model scans new or existing source code to detect potential code smells. Each code segment is evaluated, and the system predicts the likelihood of it exhibiting one or more code smells. The detected smells are categorized by type and severity, providing developers with detailed insights into problematic areas of the codebase.

Automated Refactoring Recommendation:

After detecting code smells, the system generates context-aware refactoring suggestions. For example:

- **Long Method:** Suggests "Extract Method" to divide a long function into smaller, manageable pieces.
- **God Class:** Recommends "Move Class" or "Extract Class" to reduce class responsibilities.
- **Feature Envy:** Suggests moving methods closer to the classes they interact with most.

The recommendations aim to improve code quality while maintaining the original functionality of the software.

User Interface and Dashboard:

The system provides a visual interface where developers can view:

- Detected code smells with severity levels.
- Recommended refactoring actions.
- Statistical summaries of code quality improvements.

This dashboard enables developers to prioritize refactoring tasks and monitor the overall health of the codebase.

Integration and Continuous Feedback:

The system can be integrated with IDEs or version control pipelines to provide real-time detection and refactoring suggestions during development. Continuous feedback allows developers to address issues proactively, reducing the accumulation of technical debt and improving maintainability over time.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, January 2026
Impact Factor: 7.67

IV. RESULT

The proposed system demonstrates high accuracy in detecting various types of code smells, including God Class, Long Method, Feature Envy, and Data Clumps. Machine learning models, particularly Random Forest and Neural Networks, outperform traditional rule-based methods by reducing false positives and capturing complex patterns in code metrics. Additionally, the system's automated refactoring recommendations provide context-aware guidance, helping developers improve code quality, maintainability, and readability with minimal manual effort. Overall, the integration of detection and refactoring significantly reduces technical debt and accelerates the software maintenance process.

V. FUTURE SCOPE

The system can be further enhanced by incorporating real-time code smell detection during development using IDE plugins and continuous integration pipelines. Advanced machine learning models, such as Graph Neural Networks, can be employed to capture complex code dependencies and interactions. The recommendation engine can also be extended to suggest more sophisticated refactoring strategies, tailored to project-specific constraints and coding standards. Additionally, integrating feedback loops to learn from developer actions can continuously improve detection accuracy and recommendation relevance over time.

VI. CONCLUSION

Machine learning provides an effective and scalable approach for automated code smell detection and refactoring recommendations. The proposed system enables proactive identification of code quality issues, reducing manual review effort and technical debt while improving software maintainability. By combining detection and actionable refactoring guidance, developers are empowered to maintain high-quality codebases, adhere to best practices, and enhance overall software lifecycle efficiency. The approach demonstrates that data-driven insights can substantially improve both development productivity and software reliability.

REFERENCES

[1]. Fontana, F., et al., "A Systematic Review of Code Smell Detection Techniques," Journal of Software: Evolution and Process, 2012.
[2]. Palomba, F., et al., "Machine Learning for Code Smell Detection," Empirical Software Engineering, 2015.
[3]. Kim, M., et al., "Deep Learning for Code Smell Detection," IEEE Transactions on Software Engineering, 2016.
[4]. Heckman, R., et al., "Ensemble Learning for Design Defect Detection, Journal of Systems & Software, 2017.
[5]. Moha, N., et al., "Automated Refactoring Recommendation Systems," ACM Computing Surveys, 2018.
[6]. Fowler, M., Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2018.
[7]. Marinescu, R., "Detection Strategies: Metrics-Based Rules for Detecting Design Flaws," 2004.
[8]. Lanza, M., Marinescu, R., Object-Oriented Metrics in Practice, Springer, 2006.
[9]. Yamashita, A., Moonen, L., "Exploring the Impact of Code Smells on Maintenance Effort," Journal of Software Maintenance, 2013.
[10]. Shatnawi, R., "Using Machine Learning to Detect Code Smells," 2012.
[11]. Al Dallal, J., et al., "Code Smell Detection Using Classification Models," 2019.
[12]. Tsantalis, N., et al., "Automated Refactoring Using Code Smell Detection," IEEE, 2018.
[13]. Oliveto, R., et al., "Empirical Studies on Code Smells," 2013.
[14]. Fontana, F., et al., "Comparative Analysis of Code Smell Detection Tools," 2015.
[15]. Dig, D., et al., "Automated Detection of Refactoring Opportunities," 2006.
[16]. Zazworka, N., et al., "Studying the Impact of Code Smells on Maintenance," 2011.
[17]. Palomba, F., et al., "Refactoring Recommendation Systems: State of the Art and Challenges," 2017.
[18]. Kessentini, M., et al., "Learning to Refactor: Machine Learning for Software Improvement," 2014.
[19]. Tsantalis, N., et al., "JDeodorant: Identifying and Refactoring Code Smells," 2008.
[20]. Moha, N., et al., "DECOR: A Method for Code Smell Detection," 2010.

DOI: 10.48175/IJARSCT-31010