Automate Code Refactoring for Enhanced Software Maintenance and Development

Atman Ainapure, Aditya Kharote, Tarun Agrawal, Sudhir Dhage
Sardar Patel Institute of Technology, Mumbai, India
Corresponding author: Atman Ainapure, Email: atman.ainapure@spit.ac.in

This paper unfolds a practical solution tailored to the intricate challenges faced by multinational corporations (MNCs) in the landscape of software development. The core emphasis of our research is on addressing the multifaceted aspects of maintaining code consistency, longevity, and achieving bug-free functionality. The crux of our proposed solution lies in the development of a sophisticated VS Code Extension built using TypeScript powered by LangChain and Flask that not only guides developers through the intricate process of code refactoring but also streamlines the often laborious task of generating comprehensive comments. This dual-pronged approach seeks to not only enhance the intrinsic quality of the codebase but also contribute significantly to the overall documentation and comprehensibility of the software.

Keywords: Code Refactoring, Comment Generation, Langchain, Language Model, Code Consistency, Code Quality, Software Maintenance, Programming Paradigms, Automated Formatting, Code Optimization, Code Evolution, Developer Productivity.

1 Introduction

Within the complex milieu of multinational corporations (MNCs), the pivotal role of maintaining code consistency, resilience, and seamless functionality cannot be overstated. Code, as the fundamental cornerstone of digital infrastructure, serves as the very framework upon which these corporations construct and fortify their operations and services. However, the quest for ensuring code quality presents a labyrinth of formidable hurdles. Presently, developers embark on a manual odyssey to adhere meticulously to specific coding standards. This intricate process entails exhaustive checks and balances, meticulously ensuring that the code not only executes as envisaged but also harmonizes with the predefined criteria of excellence and comprehensibility.

Nevertheless, this manual endeavor is time-consuming and heavily reliant on the individual developer's depth of expertise and unwavering diligence, rendering it susceptible to the vagaries of human error. Furthermore, as the codebase burgeons and undergoes evolutionary phases, the task of upholding consistency and readability metamorphoses into an increasingly herculean feat. This escalating complexity threatens the code's enduring usability and sustainability, potentially impeding its longevity. Thus, the crux of this paper lies in proposing an innovative automated solution, one designed to transcend these challenges. Such a solution endeavors to not only streamline and elevate efficiency but also ensure the steadfast maintainability of the code, fortifying its resilience and usability for the foreseeable future.

Code quality is important because it impacts the overall software quality and the business value of the software. According to Perforce[1], code quality affects how safe, secure, and reliable the codebase is, which is critical for many development teams, especially those developing safety-critical systems. Poor code quality can lead to software failures, security breaches, or safety hazards, which can have catastrophic or fatal consequences. Moreover, code quality influences how easy it is to understand, modify, test, and reuse the code, which affects the productivity, efficiency, and satisfaction of the developers and the customers. As The QA Lead states, code quality is an essential investment for the project rather than a time-consuming task, as it lowers technical debt and ensures the longevity of the software.

A study by IBM found that individual programmers are less than 50 percent efficient at finding bugs in their own software, and most forms of testing are only 35 percent efficient. Therefore, automated code review tools are needed to complement human efforts and provide a more reliable and consistent way of detecting and preventing software defects. Automated code review tools are a set of programs that can analyze the source code and identify issues such as anti-patterns, bug risks, performance problems, security vulnerabilities, code style violations, code complexity, code coverage, and more. Some of the popular automated code review tools are DeepSource, Codacy, Code Climate, SonarCloud, and Embold. These tools can be easily integrated with various code hosting platforms, such as GitHub, GitLab, or Bitbucket, and provide feedback on every pull request or commit. By using automated code review tools, developers can save time, reduce errors, enhance security, increase performance, and ensure compliance with coding standards and best practices.

2 Literature Review

The current manual process of adhering to coding standards and conducting code quality checks is time-consuming and error-prone, hindering development efficiency and increasing the risk of software defects.

To address these challenges, researchers have explored various automated approaches to enhance code consistency, longevity, and bug-free functionality. One promising approach is static code analysis (SCA), which involves analyzing code without executing it to identify potential issues and enforce coding standards [2]. SCA tools can automatically detect violations of coding conventions, naming patterns, and style guidelines, promoting code consistency and readability. Now, let's delve into the creative ideas that others have formulated:

1. Best Practices for Software Development [3]: A Systematic Literature Review: Ordoñez-Pacheco, Cortes-Verdin, and Ocharán-Hernández conducted an SLR to identify best practices in software development. Their objective was to explore definitions, distinctive characteristics, identification methods, and performance evaluation related to best practices. The study revealed seven different definitions of best practice, emphasizing the need for clarity. Additionally, they proposed two classification schemes based on characteristics such as name, stakeholders, and context. Methods for identifying and evaluating best practices within the software development life cycle were also discussed. For multinational corporations (MNCs) aiming to standardize their processes, this study provides valuable insights.

2. Challenges of Low-Code/No-Code Software Development [4]: A Literature Review: While not MNC-specific, this review sheds light on challenges in low-code/no-code development. Key challenges explored include balancing simplicity with customization, ensuring security and compliance, handling scalability and performance, and bridging the gap between professional developers and citizen developers. Although not directly applicable to MNCs, understanding these challenges can inform software development practices in various contexts.

3. Understanding the Challenges and Novel Architectural Trends in Multi-Cloud Native Applications: A Systematic Literature Review [5]: This SLR investigates challenges related to multi-cloud native applications. The main trends identified include microservices architecture, containerization, serverless computing, and DevOps practices. While not MNC-focused, the findings can guide MNCs in enhancing their software development processes across cloud-native landscapes.

4. Challenges of Developing Secure Software Using the Agile Approach [6]: Although

not directly MNC-related, this study examines challenges in developing secure software using agile methodologies. The authors identify 20 challenges across various aspects of the development life cycle, including incremental development, security assurance, collaboration, awareness, and security management. MNCs can adapt these insights to improve their secure software development practices.

5. A Systematic Literature Review and Delphi Study on Agile Software Development Challenges [7]: This study maps agile software development challenges. Categories of challenges explored include requirements engineering, project management, quality assurance, and collaboration. While not MNC-specific, the identified challenges can inform MNCs seeking to optimize their agile development processes. These studies collectively contribute to the evolving landscape of software development practices and offer valuable guidance for MNCs navigating complex development environments.

6. "Multinational Corporations and Grand Challenges: Part of the Problem, Part of the Solution?" [8]: This special issue invites scholarly investigations into the strategies, business conduct, and political behaviors of MNCs. Specifically, it focuses on their role in causing or contributing to global challenges. The issue aims to address both the negative externalities resulting from MNC activities and any compensatory actions they undertake. Despite MNCs' economic and political power, research on their specific impact remains scarce. Critical perspectives are essential to question MNC hegemony and evaluate their effects on societies, the environment, and stakeholders. By examining MNC behavior, impact, and potential solutions, this issue seeks to fill gaps in our understanding of their multifaceted role in global challenges.

7. "The Engineering Implications of Code Maintenance in Practice" [9]: This paper delves into the challenges faced by developers when evolving and maintaining low-latency, large-scale distributed systems—common scenarios in MNCs. Complexity and Scale: MNCs grapple with massive codebases, making code maintenance inherently complex. The sheer scale of these systems poses challenges in ensuring code quality and consistency. Code Quality Degradation: Over time, code quality tends to degrade due to evolving requirements, frequent updates, and developer turnover. Maintaining high standards becomes increasingly difficult. Change Management: Efficient change management—rapidly incorporating updates while minimizing disruptions—is crucial for MNCs operating in dynamic environments. Opportunities: The study underscores the need for efficient code maintenance practices within MNCs, emphasizing the impact on software quality and developer productivity.

8. "Challenges in Agile Software Maintenance for Local and Global Environments" [10]: This study specifically explores challenges related to agile software maintenance. Quality Factors: Agile methods significantly influence software quality factors such as reliability and maintainability. Balancing agility with long-term quality remains a critical challenge. Local vs. Global Contexts: MNCs operate across diverse local and global environments. Adapting agile practices to suit these varying contexts poses unique chal-

lenges. Predictive Techniques: Machine learning techniques can predict and prevent code defects, enhancing maintenance efficiency. Implications for MNCs: Understanding these challenges informs MNCs seeking to optimize their agile development processes across different geographical and organizational contexts.

Another approach involves using rudimentary machine learning (ML) techniques to identify and predict code defects. ML models can be trained on large datasets of code and associated bug reports to learn patterns and correlations that indicate potential bugs [11]. These models can then be used to proactively identify and flag code that is likely to contain bugs, allowing developers to address them early in the development process.

To further enhance code longevity, researchers have explored techniques for refactoring and code modernization. Refactoring involves restructuring code without changing its external behavior to improve its internal structure, making it easier to understand, maintain, and extend [12]. Code modernization involves updating legacy code to newer programming languages, frameworks, and technologies to improve its compatibility with modern systems and enhance its maintainability [13].

In addition to automated approaches, researchers have also emphasized the importance of establishing clear coding guidelines and documentation practices to promote code consistency and longevity [14]. Well-defined coding standards provide a consistent framework for developers to follow, reducing the risk of introducing inconsistencies and errors. Comprehensive documentation, including code comments and design documents, enhances code comprehension and maintainability for future developers.

3 Related Works

Manual Code Maintenance Challenges: Several studies have highlighted the challenges inherent in manual code maintenance processes within the domain of multinational corporations (MNCs). The research underscores the time-intensive nature of manual adherence to coding standards, which often leads to inefficiencies and susceptibility to human error.

Technological Approaches: Recent advancements in technology have paved the way for innovative solutions in code maintenance. Technologies like Langchain have been explored to automate code standardization and compliance. Furthermore, the utilization of custom knowledge-based Language Models (LLMs) in backend systems shows promise in enhancing code understanding and aligning with organizational coding standards.

Intelligent Systems for Code Maintenance: Intelligent systems designed for code maintenance have become a focal point of research. These systems aim to enhance the software development process by providing targeted assistance to developers. Two key aspects are particularly relevant. The first being, Intelligent Front-End Extensions. These extensions act as intelligent companions for developers during code refactoring. By analyzing the existing codebase, they offer real-time guidance. Their purpose is twofold:

first, to optimize code comprehensibility by suggesting improvements in naming conventions, structure, and readability; and second, to ensure adherence to coding standards. For multinational corporations (MNCs), where large-scale codebases are common, such extensions can significantly improve developer productivity and code quality. Next is AI-Driven Techniques. Artificial intelligence (AI) techniques hold immense promise in automating various aspects of code maintenance. For instance. Machine learning models can learn from historical code changes and recommend refactoring opportunities. Whether it's identifying redundant code, improving performance bottlenecks, or streamlining complex logic, AI-driven refactoring tools can assist developers. Another example can be Code Documentation Enhancement where Natural language processing (NLP) models can generate high-quality code comments and documentation. Clear and concise documentation is crucial for maintaining codebases, especially in MNCs with distributed teams and diverse expertise. By integrating these intelligent systems into the development workflow, MNCs can streamline code maintenance, reduce manual effort, and elevate overall software quality.

Gaps and Opportunities: Despite significant advancements, challenges persist. there are both gaps and opportunities while advancing with the research.

a. Seamless Integration of Technologies like Langchain: Langchain, a novel code analysis tool, offers valuable insights into code quality, patterns, and potential improvements. However, integrating it seamlessly into existing development environments remains a challenge. MNCs operate across diverse projects, languages, and frameworks. Ensuring that Langchain harmoniously fits into this ecosystem is essential.

b. Holistic Solutions for Code Refactoring and Documentation: MNCs grapple with legacy codebases, frequent updates, and evolving requirements. A holistic solution that combines Langchain's insights with custom LLMs could revolutionize code maintenance. Imagine an intelligent system that not only identifies refactoring opportunities but also generates clear, relevant comments and documentation. Such a system would bridge the gap between manual code maintenance and automated practices. Developers could focus on high-level design decisions, while routine tasks receive intelligent support.

By leveraging Langchain, custom LLMs, and AI techniques, MNCs can pave the way for sustainable, agile software development.

4 Proposed Methodology

Our proposed methodology revolves around a seamless VSCode extension that integrates with developers' existing workflows, providing real-time code analysis and suggestions for refactoring opportunities. Upon selecting a code snippet, the extension communicates with the backend infrastructure, employing Langchain's semantic knowledge and the LLM's contextual understanding to identify and recommend refactoring improvements. These recommendations are presented to the developer. The overall archi-

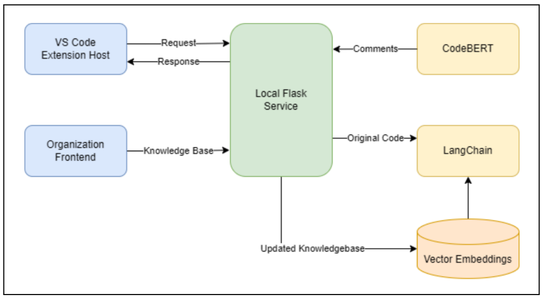


Figure 1: Overall Architecture of CodeSage

tecture can be seen in Figure 1.

Leveraging Langchain's decentralized knowledge graph for programming languages, our system gains a deep understanding of code semantics, enabling it to generate context-aware refactoring suggestions that enhance code structure, design, and maintainability. Additionally, the custom LLM, trained on a vast dataset of code and refactoring examples, provides the system with the ability to learn from past decisions and adapt its recommendations to specific coding standards, project-specific requirements, and programming styles.

Beyond refactoring, our solution automates comprehensive documentation generation, including detailed explanations of refactoring changes, rationale behind the changes, and potential implications for future modifications. This automated documentation ensures that code changes are accompanied by clear explanations, enhancing code comprehension and maintainability for future developers.

By combining automated code refactoring, documentation generation, and the power of Langchain and LLM, our proposed methodology aims to revolutionize code maintenance practices in MNCs. We envision a future where codebases are consistently refactored, well-documented, and easy to maintain, leading to improved software quality, developer productivity, and reduced maintenance costs.

4.1 Automating Code Refactoring and Documentation: A Proposed Solution

Maintaining the consistency, longevity, and bug-free functionality of large codebases is a critical challenge for multinational corporations (MNCs). Manual code refactoring and documentation processes are not only time-consuming but also prone to human error, leading to inconsistencies, inefficiencies, and an increased risk of software defects.

Key code quality challenges faced by MNCs include:

- Ensuring high code quality through consistent coding standards and practices.
- Maintaining code readability and simplicity to facilitate easier debugging and future modifications.

- Repetitive code fragments increase maintenance effort and the risk of inconsistent behavior.
- Code that is not modular or reusable leads to redundant code and difficulty in isolating and fixing issues.

To address these challenges, we propose an integrated system that automates code refactoring and documentation generation, powered by a combination of a front-end VSCode extension and a backend infrastructure leveraging Langchain and a custom knowledge-based, fine-tuned Language Model (LLM).

- A clean, maintainable codebase by enforcing consistent coding standards and reducing technical debt.
- Real-time feedback to enforce coding standards and catch issues early.
- Analyze code to offer intelligent refactoring and documentation suggestions, maintaining modularity and reusability.

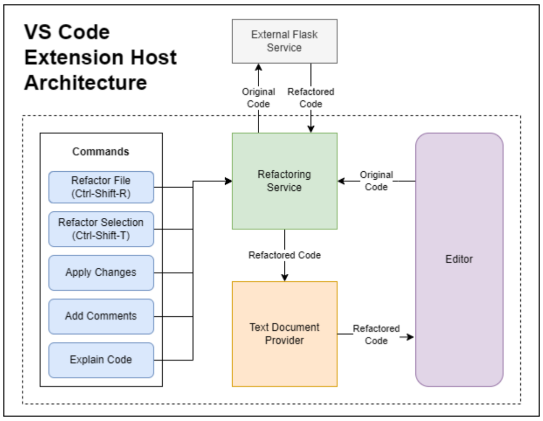


Figure 2: Architecture of the VS Code extension

4.2 Introducing a VSCode Extension for Seamless Integration

At the heart of our solution lies a front-end Visual Studio Code extension as seen in Figure 2. This extension seamlessly integrates with developers' existing workflows, providing real-time code analysis and suggestions for refactoring opportunities. Upon selecting a code snippet for refactoring, the extension initiates communication with the back-end infrastructure to process the code and generate refactoring recommendations. These recommendations are then presented to the developer, allowing them to review and apply the changes with a single click.

4.3 Enhancing Efficiency and Maintainability through Automated Code Refactoring

Automated code refactoring offers several compelling benefits for MNCs, including:

Improved Code Quality: Automated refactoring tools can effectively identify and correct code smells, redundant code, and inefficient structures, leading to cleaner, more maintainable code.

Reduced Development Time: By automating refactoring tasks, developers can redirect their efforts toward higher-level design and implementation, significantly reducing the time spent on manual code maintenance.

Enhanced Code Readability: Well-refactored code is easier to understand and modify, fostering collaboration among developers and reducing the likelihood of errors when making changes.

Adherence to Coding Standards: The proposed solution can be tailored to enforce specific coding standards within an MNC, ensuring consistency across the codebase.

Comprehensive comments: The Visual Studio Code extension streamlines the task of generating comprehensive comments by analyzing code in real-time and providing intelligent, context-aware suggestions using an integrated Language Model (LLM).

4.4 Leveraging Langchain and LLM for Intelligent Code Refactoring

Our proposed solution utilizes Langchain, a decentralized knowledge graph for programming languages, to provide a rich context for code analysis and refactoring. Langchain stores relationships between programming constructs, enabling the system to understand the semantic meaning of code and generate context-aware refactoring suggestions. This semantic understanding is crucial for identifying refactoring opportunities that not only improve code structure but also enhance its overall design and maintainability.

To further enhance the system's intelligence, we employ a custom knowledge-based, fine-tuned LLM trained on a massive dataset of code and associated refactoring examples. This LLM provides the system with the ability to learn from past refactoring decisions and adapt its recommendations to specific programming styles, coding standards, and project-specific requirements.

4.5 Automating Comprehensive Documentation Generation

Beyond code refactoring, our solution also automates the generation of comprehensive documentation for refactored code. This documentation includes detailed explanations of the refactoring changes, the rationale behind the changes, and any potential implications for future code modifications. By automating this process, we eliminate the need for manual documentation efforts, ensuring that code changes are accompanied by clear and concise explanations.

5 Results and Discussion

We successfully developed a Visual Studio Code extension for automated code refactoring and integrated it with a backend Flask server. The extension offers two modes of operation: refactoring a selected code snippet or refactoring the entire file. Upon selecting the desired refactoring mode, the user can provide the code input. The refactored code is then displayed in a dedicated panel, with changes highlighted in green and omitted parts highlighted in red. This visual representation allows users to easily identify the modifications made to the code. To ensure user control over the refactoring process, the extension prompts the user to accept or reject the proposed changes. This confirmation step ensures that the refactored code aligns with the user's intentions and preferences.

For our understanding, we conducted testing by providing the VS Code extension to 50 participants, divided into five batches of 10 participants each. In the first batch, 9 out of 10 participants (90%) reported that the software accurately predicted their desired output. The yield of subsequent batches is shown in Figure 3. Therefore, the overall accuracy of our solution across all five batches is **74%**.

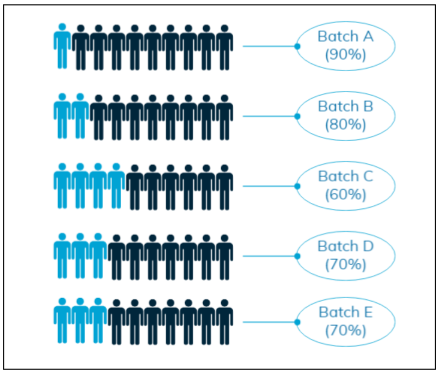
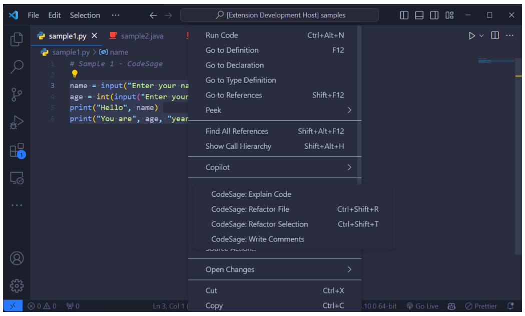


Figure 3: The Yield of Batch Testing

5.1 Refactoring a Selected Code Snippet

As seen in Figure 4, the user can choose to refactor a selected snippet or the entire file. When refactoring a selected code snippet, the extension isolates the chosen portion of the code and sends it to the back-end Flask server for processing. The server employs Langchain's knowledge graph and the fine-tuned LLM to analyze the code semantics, identify refactoring opportunities, and generate appropriate recommendations. The refactored code is then sent back to the VSCode extension and presented to the user within the dedicated panel. The user can review the changes, highlighted in green, and decide whether to accept or reject the refactoring suggestions.



```
# Sample 1 - CodeSage

name = input("Enter your na
age = int(input("Enter your
print("Hello", name)
print("You are", age, "year
```

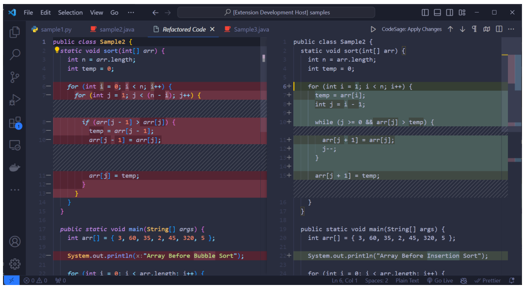
Figure 4: The user can select either a code snippet, or an entire file to refactor

5.2 Refactoring the Entire File

Refactoring the entire file involves sending the entire code file to the backend Flask server for analysis and refactoring. The server applies the same process of semantic analysis and refactoring suggestion generation as in the selected code snippet mode. The refactored code is then sent back to the VSCode extension, where it is displayed alongside the original code. The user can compare the original and refactored code, highlighted changes in green and omissions in red, and decide whether to accept or reject the overall refactoring changes.

5.3 User Control over Refactoring

As seen in Figure 5, throughout both refactoring modes, the VS Code extension prompts the user to accept or reject the proposed changes before applying them to the code. This user control ensures that the refactored code aligns with the developer's intentions and preferences. By providing this confirmation step, we empower developers to maintain control over the codebase and ensure that refactoring efforts enhance the overall quality and maintainability of the code.



```
public class Sample2 {
  static void sort(int[] arr) {
    int n = arr.length;
    int temp = 0;

    for (int i = 0; i < n; i++) {
      for (int j = 1; j < (n - i); j++) {
        if (arr[j - 1] > arr[j]) {
          temp = arr[j - 1];
          arr[j - 1] = arr[j];
          arr[j] = temp;
        }
      }
    }
  }

  public static void main(String[] args) {
    int arr[] = { 3, 60, 35, 2, 45, 320, 5 };

    System.out.println(x:"Array Before Bubble Sort");

    for (int i = 0; i < arr.length; i++) {
```

```
public class Sample2 {
  static void sort(int[] arr) {
    int n = arr.length;
    int temp = 0;

    for (int i = 1; i < n; i++) {
      temp = arr[i];
      int j = i - 1;

      while (j >= 0 && arr[j] > temp) {
        arr[j + 1] = arr[j];
        j--;
      }

      arr[j + 1] = temp;
    }
  }

  public static void main(String[] args) {
    int arr[] = { 3, 60, 35, 2, 45, 320, 5 };

    System.out.println("Array Before Insertion Sort");

    for (int i = 0; i < arr.length; i++) {
```

Figure 5: The user can review and choose to accept the refactored code

The successful development of the VSCode extension for automated code refactoring demonstrates the feasibility and effectiveness of our proposed methodology. The extension's ability to refactor both selected code snippets and entire files, coupled with its visual representation of changes and user control over refactoring, makes it a valuable tool for developers in MNCs.

6 Conclusion

In conclusion, this research has presented a novel approach to automated code refactoring and documentation generation for multinational corporations (MNCs). According to a study [15] conducted by AlOmar et al., code review plays a crucial role in maintaining software quality and sharing knowledge within development teams1. The proposed solution, centered on a front-end VSCode extension and a backend infrastructure powered by Langchain and a custom Language Model (LLM), addresses the challenges of manual code maintenance and enhances code quality, developer productivity, and overall codebase maintainability.

The seamless integration of the VSCode extension into developers' existing workflows provides real-time code analysis and refactoring suggestions, empowering developers to identify and address code issues efficiently. The backend infrastructure, leveraging Langchain's semantic knowledge and the LLM's contextual understanding, generates intelligent refactoring recommendations that align with specific coding standards and project requirements.

The proposed solution not only automates code refactoring but also generates comprehensive documentation for refactored code, ensuring that code changes are accompanied by clear explanations and rationale. This automated documentation enhances code comprehension and maintainability for future developers, fostering collaboration and reducing the risk of errors when making changes.

By seamlessly integrating automated code refactoring, documentation generation, and harnessing the formidable capabilities of Langchain and LLM, this research not only showcases the practicality but also underscores the efficacy of an innovative paradigm poised to redefine code maintenance practices within Multinational Corporations (MNCs).

Looking ahead, the trajectory of future research endeavors extends toward a comprehensive exploration of the ramifications of automated code refactoring on developer productivity and various code quality metrics. Moreover, an ambitious aim is set to seamlessly integrate the proposed solution into the intricate fabric of continuous integration/-continuous delivery (CI/CD) pipelines, thereby establishing a dynamic and automated framework for code refactoring and documentation updates. The envisioned future not only involves a profound understanding of the theoretical underpinnings but also an active pursuit of practical implementations that promise to shape the future landscape of code maintenance practices.

References

[1] Perforce Software, “What is code quality? An overview,” Perforce, May 6, 2020

[2] Xie, W., Pei, Z., & Xu, C. (2018). DeerStatic: A practical static analysis tool for detecting and fixing defects. IEEE Transactions on Software Engineering, 44(5), 433-448.

[3] R. Ordoñez-Pacheco, K. Cortes-Verdin, and J. Ocharán-Hernández, “Best Practices for Software Development: A Systematic Literature Review,” Advances in Intelligent Systems and Computing, vol. 1297, 2020.

[4] “Challenges of Low-Code/No-Code Software Development: A Literature Review,” Springer, 2022.

[5] “Understanding the Challenges and Novel Architectural Trends in Multi-Cloud Native Applications: A Systematic Literature Review,” SpringerOpen, 2023.

[6] “Literature Review of the Challenges of Developing Secure Software Using the Agile Approach,” IEEE 10th International Conference on Availability, Reliability and Security, 2015.

[7] “A Systematic Literature Review and Delphi Study on Agile Software Development Challenges,” IEEE, 2021.

[8] “Multinational Corporations and Grand Challenges: Part of the Problem, Part of the Solution?” Critical Perspectives on International Business (CPoIB).

[9] “The Engineering Implications of Code Maintenance in Practice.” IEEE Xplore.

[10] “Challenges in Agile Software Maintenance for Local and Global Environments.” MDPI.

[11] Wong, C., Guo, X., Zhang, X., & Sun, J. (2019). DeepBug: A deep learning approach for detecting performance bugs in Android apps. IEEE Transactions on Software Engineering, 45(1), 87-102.

[12] Fowler, M. (2018). Refactoring: Improving the design of existing code (2nd ed.). Addison-Wesley.

[13] Mohapatra, S. K., Mohapatra, M., & Routray, S. (2021). A comprehensive survey on code modernization: Approaches, tools, and challenges. ACM Computing Surveys (CSUR), 54(1), 1-36.

[14] Van der Linden, P. J. (2018). Software documentation: A guide to good practice. Addison-Wesley.

[15] E. A. AlOmar, H. AlRubayey, M. W. Mkaouer, A. Ouniz, and M. Kessentini, “Refactoring Practices in the Context of Modern Code Review: An Industrial Case Study at Xerox,” arXiv preprint arXiv:2102.05201, 2021.