Investigating the Relationship Between Code Quality Metrics and Software Maintainability: An Empirical Study

Mr. Chinmay Chandrakant Salvi ,
Mr. Dnyaneshwar Balu Lokhande (Assistant Professor*)*
Department of Master of Computer Application,
JCEI's Jaihind Institute of Management and Research, Vadgaon Sahani
Email : chinmaysalvi1999@gmail.com,Lokhande.dyan@gmail.com,

Abstract: Software maintainability is an important aspect of software quality because software systems frequently require modifications, corrections, and enhancements throughout their lifecycle. Poorly structured code, excessive complexity, and duplicated implementations can increase maintenance effort and make software changes more difficult. Therefore, identifying code quality characteristics associated with maintainability is essential for developing reliable and maintainable software systems. This study empirically examines the relationship between selected code quality metrics and software maintainability. The metrics considered include Lines of Code (LOC), cyclomatic complexity, code duplication, coupling, code smells, and the Maintainability Index (MI). Static code analysis is used to extract these metrics from the source code of multiple software projects. The collected measurements are then analyzed to investigate potential relationships between code characteristics and maintainability. The main objective is to determine which code quality metrics are associated with software maintainability and whether these metrics can help identify potential maintenance challenges. The study aims to provide insights that may assist software developers and engineers in recognizing problematic code characteristics and prioritizing improvements. The findings may also contribute to future research on software maintainability assessment and the development of effective code quality evaluation techniques.

Key Words: Software Maintainability, Software Quality Metrics, Static Code Analysis, Cyclomatic Complexity, Code Smells, Code Duplication, Maintainability Index, Software Measurement, Empirical Software Engineering.

1. Introduction:

Software systems support a wide range of activities across industries, including banking, healthcare, education, business, and communication. As organizational needs and user expectations continue to evolve, these systems require regular updates, corrections, and enhancements. Consequently, software development involves not only implementing functional requirements but also ensuring that software remains understandable, adaptable, and manageable throughout its lifecycle. Software maintainability is an important software quality attribute that describes the extent to which a software system can be modified to correct defects, improve performance, adapt to changing requirements, or introduce new functionality. As software systems evolve, maintenance activities may become increasingly challenging because of growing system complexity, architectural limitations, and poorly structured source code. These challenges can increase the effort required to understand existing implementations, identify defects, and introduce changes without affecting other parts of the system. Good code quality makes software easier to understand, test, review, and modify.. Several characteristics of source code, including its size, structural complexity, duplication, and coupling, may influence the effort required for maintenance. Additionally, code smells can indicate potential weaknesses in software design or implementation. Although these characteristics do not necessarily cause immediate functional failures, they may increase the difficulty of future modifications and contribute to maintenance challenges. Therefore, evaluating code quality metrics can help developers identify problematic code structures and determine areas that may require improvement.

In this context, the present study investigates the relationship between selected code quality metrics and software maintainability through an empirical analysis of software projects. Static code analysis is employed to measure characteristics such as Lines of Code (LOC), cyclomatic complexity, code duplication, coupling, and code smells. The Maintainability Index (MI) is also considered as a quantitative indicator of maintainability. The collected measurements are analyzed to explore associations between these code characteristics and maintainability outcomes. The study aims to identify which metrics are most strongly associated with maintainability and assess their potential usefulness in recognizing code that may require additional maintenance effort. The results are expected to contribute to a better understanding of maintainability assessment and support more informed decisions regarding software quality improvement.

1.1 Problem statement:

As software systems grow in size and complexity, maintaining them becomes increasingly difficult. High code complexity, duplication, and coupling may increase maintenance effort. Although several code quality metrics are available, their relationship with software maintainability requires further investigation.

This study examines six metrics: Lines of Code (LOC), cyclomatic complexity, code duplication, coupling, code smells, and Maintainability Index (MI). Static code analysis will be used to collect and analyze data from selected software projects to identify relationships between code quality and maintainability.

1.2 Objectives:

1. To identify and examine established code quality metrics relevant to the assessment of software quality and maintainability.

2. To investigate the relationship between selected code quality metrics and software maintainability through empirical analysis.

3. To measure key code characteristics, including Lines of Code (LOC), cyclomatic complexity, code duplication, coupling, and code smells, using static code analysis techniques.

4. To assess software maintainability using the selected metrics and static analysis results, and to identify source-code components that may require maintenance or quality improvements.

1.3 Contributions:

This study examines the relationship between source code quality and software maintainability through the empirical analysis of measurable code characteristics. By applying static code analysis techniques, the research seeks to understand how selected code quality metrics relate to maintainability and help identify potential maintenance challenges.

- **Empirical Assessment of Maintainability:** The study evaluates software maintainability using measurable source code characteristics and selected code quality metrics, providing an empirical perspective on the research problem.

- **Analysis of Code Quality Relationships:** The research investigates the relationships between selected code quality indicators, including Lines of Code (LOC), cyclomatic complexity, code duplication, coupling, and code smells, and the chosen measure of maintainability.

- **Identification of Potential Maintenance Challenges:** The analysis aims to identify code characteristics that may be associated with increased maintenance difficulty and may require further investigation or improvement.

- **Practical Implications for Software Development:** The findings may help software developers and maintenance teams prioritize code review activities and identify areas where code quality improvements could support easier maintenance.

- **Contribution to Further Research:** The study may provide a basis for future investigations into software quality metrics, maintainability assessment, and the use of static code analysis in empirical software engineering.

2. Literature Review:

The ISO/IEC 25010 software quality model identifies maintainability as an important quality characteristic of software products and systems. It concerns the effectiveness and efficiency with which software can be modified by its intended maintainers. Such modifications may involve correcting defects, improving performance, adapting the software to changing environments, or implementing new requirements. This perspective highlights the importance of designing software that can be understood and modified without excessive effort. Research on software metrics has also investigated the relationships between source code characteristics and maintainability. A systematic review of object-oriented software metrics examined the relevance of measures such as Lines of Code (LOC), coupling, complexity, and cohesion to software quality and maintainability. The reported evidence indicates that these metrics may provide useful information about software characteristics, although the strength and direction of their relationships can vary across studies. Differences in measurement techniques, project contexts, and definitions of maintainability also complicate comparisons between findings. These limitations highlight the need for further empirical investigation into how individual code quality metrics relate to software maintainability.

2.1 Current Challenges:

Research on code-smell detection has expanded beyond conventional rule-based and metric-based techniques to include machine-learning and deep-learning approaches. A systematic review published in 2026 examined 86 studies on code-smell detection published between 2014 and 2024, covering tool-based detection methods, data-driven techniques, and challenges associated with imbalanced datasets. The review also highlighted limitations in existing detection approaches and identified opportunities for further research (Thakur et al., 2026). These findings suggest that detecting a code smell should not, by itself, be treated as conclusive evidence of reduced maintainability or increased maintenance effort. The implications of code smells may vary according to project characteristics, software context, and the metrics used for evaluation. Therefore, empirical analysis is necessary to investigate the relationships between code quality indicators and software maintainability within a defined project context.

2.2 Techniques, Strengths and Limitations:

- **Lines of Code (LOC):** Measures software size and helps estimate code complexity, but does not directly determine maintainability.
- **Cyclomatic Complexity:** Measures independent execution paths and helps identify complex code that may be difficult to test and maintain.
- **Maintainability Index (MI):** Estimates software maintainability using selected code metrics and supports comparisons between software components.
- **Code Smells**: Identifies potential weaknesses in code design that may make future maintenance more difficult.
- **Code Duplication:** Detects repeated code fragments that may increase maintenance effort and create inconsistencies.
- **Coupling:** Measures dependencies between software components. High coupling may make modifications and testing more difficult.

- **Static Code Analysis:** Examines source code without executing it to identify complexity, code smells, and other potential quality issues.
- **Repository Analysis:** Examines version-control history to understand code changes, frequently modified components, and maintenance patterns.
- **Expert Evaluation:** Uses experienced developers' judgments to assess maintainability and identify issues that automated metrics may overlook.

3 Research Methodology:

The present study adopts an empirical case study approach to examine how different code quality characteristics are associated with software maintainability. It focuses on identifying measurable attributes from software source code and evaluating their relationships with maintainability. The selected code metrics will be extracted and examined to identify patterns and correlations that may help explain how code structure and complexity influence the effort required to maintain software. Through this analysis, the study aims to develop a clearer understanding of the role of code quality metrics in assessing maintainability and identifying potential maintenance challenges.

3.1 Research Approach:

The present study will employ a quantitative empirical research approach to investigate software code quality and maintainability. Data will be collected by applying static code analysis techniques to selected software products or components. The resulting measurements will be analyzed to evaluate code quality characteristics and examine their relationship with software maintainability. This approach will help identify potential maintenance challenges and provide quantitative evidence regarding the usefulness of selected code metrics in assessing software quality and maintainability.

3.2 Data Collection:

The source code used in this study will be obtained from selected software projects, with preference given to open-source projects that provide publicly accessible code and relevant project information. Project selection will be guided by criteria such as programming language, source code availability, project size, and suitability for static code analysis. These criteria will help ensure that the selected projects are appropriate for investigating software quality and maintainability.

After selecting the projects, the source code will be examined using static code analysis tools to extract and measure the specified code quality metrics. The collected data will then be processed and organized for further analysis. The same methods will be used for all projects to ensure consistent measurements and reliable comparisons of code quality and software maintainability.

3.3 Metrics Selected for the Study:

The following metrics will be considered in the research:

- **Lines of Code (LOC):** This metric measures the size of the source code and provides an indication of the amount of code within a software component or project.
- **Cyclomatic Complexity:** This metric measures the complexity of a program's control flow by determining the number of linearly independent execution paths.
- **Code Duplication:** This metric identifies repeated or substantially similar code fragments that may increase maintenance effort and create difficulties when changes need to be applied consistently.

- **Coupling:** This metric assesses the degree of dependency between software modules and helps identify relationships that may affect the ease of modifying and maintaining individual components.
- **Coupling:** This metric assesses the degree of dependency between software modules and helps identify relationships that may affect the ease of modifying and maintaining individual components.
- **Maintainability Index (MI):** This metric provides an overall estimate of source code maintainability based on a combination of selected code characteristics, depending on the formula and analysis tool employed.

3.4 Static Code Analysis:

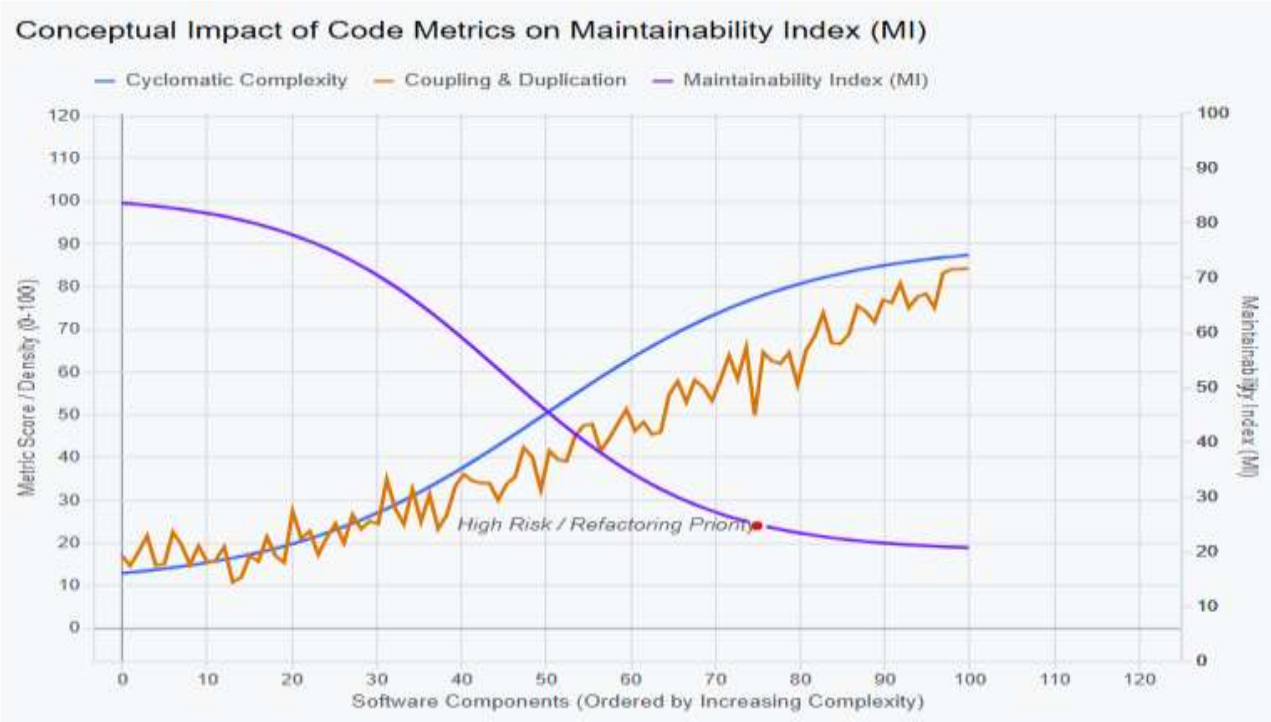
A static code analysis will be performed to assess the source code without actually executing the software in question. An appropriate analysis tool will be used to calculate the chosen values and find potential quality issues.
The obtained results will be documented and formed into a data set that will become the basis for the subsequent process of the project's empirical analysis.

3.5 Data Analysis:

After obtaining the metric results, it will be necessary to analyze the information to identify key patterns and draw conclusions. Firstly, the descriptive and statistical analysis will enable understanding the attributes of the selected software projects.
The next step will be to perform a correlation analysis of the obtained results to evaluate how the separate code quality indicators impact the code maintainability. For instance, the study will assess whether there is a link between the high Cyclomatic Complexity, code duplication, coupling, and code maintainability

3.6 Results:

1. The outcomes of this research will be based on the benchmarking of particular software projects applying code quality and maintainability metrics. By analyzing the data collected from the source code, it will be possible to determine if there is a connection between the changes in code quality indicators and software maintainability. The code quality of the selected projects will be benchmarked using static code analysis. Thus, the collected values of Lines of Code (LOC), Cyclomatic Complexity, Code Duplication, Coupling, Code Smells, and Maintainability Index will be compared and organized.
2. The initial stage of code benchmarking will involve the determination of the values of the code quality indicators mentioned above.
3. In order to assess the maintainability of the selected components, the Maintainability Index will be applied. According to the determined index, the software elements will be prioritized as well. Thus, components with a higher maintainability indicator will have a lower priority for maintenance and subsequent code refactoring



Points Identified on the Framework:

• Low risk/little or no maintenance (on the left): Fewer independent paths (low cyclomatic complexity) and less coupling existing between the modules is positioned in a higher region (green) of the Maintainability Index.

• Inversion crossover (in the middle): As you start to have more duplicate code and more modules coupled heavily the maintenance takes off exponentially and lowers the Maintainability Index.

• Highest risk/requires a lot of maintenance (on the right): Your highest points in the independent variables (complexity, smells, duplication) are actually at the lowest point (valley) of the Maintainability Index.Based on your method, these areas are the ones that should be fixed first through code refactoring.

4. Discussion:

The main objective of this study is to investigate the relationship between software quality characteristics and software maintainability. Maintainability is influenced by several aspects of source code, and no single metric can fully represent all the factors that affect the ease of modifying and improving a software system. Therefore, this study considers multiple code characteristics, including Lines of Code (LOC), cyclomatic complexity, code duplication, coupling, code smells, and the Maintainability Index (MI). These metrics provide different perspectives on software structure and can help identify characteristics that may influence maintenance effort.

Lines of Code (LOC):

Lines of Code It provides a simple way to compare the size of software components and may offer an indication of the amount of code that developers need to understand and maintain. Generally, larger software systems may require greater maintenance effort because they contain more code to review, test, and modify. However, LOC alone cannot determine software quality or maintainability, as these also depend on factors such as code organization, complexity, and design.

Cyclomatic Complexity:

Cyclomatic Complexity measures the number of linearly independent paths through a program's control-flow structure. As this value increases, the program may become more difficult to understand, test, and modify because additional decision paths require consideration. Nevertheless, complexity should not be considered in isolation, since the maintainability of a software system also depends on its design, modularity, and other code quality characteristics.

In this study, LOC and Cyclomatic Complexity will be examined alongside the other selected metrics to investigate their relationships with software maintainability. The analysis will help determine whether these measurements provide useful indications of potential maintenance difficulties in the selected software projects.

The study did a thorough review of existing research on software maintainability metrics and the tools used for analysis. It found out the commonly used software maintainability metrics and the issues with current methods. Additionally, McCabe introduced Cyclomatic complexity, a metric that helps measure the complexity of a program's source code. The referenced studies offer important information about the features of source code that are important for analysing software maintainability.

5. Conclusion:

Software maintainability is an important aspect of software quality because software systems require regular updates, corrections, and improvements throughout their life cycle. Poorly organized source code can make it difficult for developers to identify and fix errors, modify existing functions, perform testing, and introduce new features. Therefore, understanding how code quality characteristics relate to maintainability is essential for developing software that can be maintained efficiently over time. This study investigates the relationship between software maintainability and selected code quality metrics, including Lines of Code (LOC), Cyclomatic Complexity, Code Duplication, Coupling, Code Smells, and the Maintainability Index (MI). Each metric represents a different aspect of source code and provides information that can help identify potential maintenance challenges. By examining these characteristics, the study aims to understand how different code properties are associated with the ease of maintaining software systems.

The study recognizes that software maintainability cannot be fully evaluated through a single code quality metric. Cyclomatic Complexity, code duplication, and coupling can help identify structural characteristics that may increase maintenance effort, while code smells can highlight areas requiring further investigation. The Maintainability Index provides a combined estimate based on selected code characteristics, although its effectiveness depends on the calculation method and the factors included. Considering these metrics together can provide a broader understanding of software maintainability and support more informed decisions about code quality improvement.

6. Future Work:

1. **Expanding the Dataset:** Future research could examine a larger and more diverse collection of software projects developed using different programming languages.

2. **Applying Machine Learning Techniques:** Machine learning methods could be explored to predict potential maintainability problems using code quality metrics. Training models on historical project data may help identify code components that are more likely to experience maintenance difficulties.

3. **Analyzing Software Repository History**: Further studies could investigate commit records, bug reports, and code changes to understand how code quality characteristics relate to actual maintenance activities and software evolution.

4. **Developing Automated Assessment Systems**: Future work could focus on developing an automated tool that analyzes source code, evaluates selected quality metrics, and highlights components that may require maintenance or improvement.

5. **Comparing Static Analysis Tools:** Different static code analysis tools could be evaluated to determine whether variations in their metric calculations and detection capabilities influence maintainability assessments.

6. **Conducting Longitudinal Studies:** Researchers could examine multiple versions of the same software project over time to investigate how code quality metrics change and how these changes relate to maintainability throughout the software lifecycle.

7. References:

1. ISO/IEC, ISO/IEC 25010:2023 — Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model, International Organization for Standardization, 2023.

2. L. Ardito, R. Coppola, L. Barbato, and D. Verga, "A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review," Scientific Programming, vol. 2020, Article ID 8840389, 2020, doi: 10.1155/2020/8840389.

3. N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2 nd ed. Boston, MA, USA: PWS Publishing, 1997.

4. T. J. 1.McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, 1976.

5. M. H. Halstead, Elements of Software Science. New York, NY, USA: Elsevier North-Holland, 1977.

6. S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, 1994.

7. M. Sjøberg, D. I. K. S. Yamashita, A. Aranda, D. I. S. Mockus, and M. A. D. A. 2.Jørgensen, "Code Smells as System-Level Indicators of Maintainability: An Empirical Study," Journal of Systems and Software, vol. 86, no. 10, pp. 2639-2653, 2013, doi: 10.1016/j.jss.2013.05.007.

8. L. O. B. Oliveira, M. A. G. Maia, and M. A. C. S. 3.Machado, "An empirical study on the relationship between code smells and software maintainability," Information and Software Technology , 2018.

9. K. Hotta, Y. Sasaki, Y. Sano, Y. Higo, in 2012, Art. no. 938296, 2012.