**Evaluating the Impact of Class Size on Software Maintainability: An Empirical Study**

***Submitted By***

**Pavan Kumar Gaddam**

**Sai Teja Singamaneni**

**Swamy Gorla**

**Yamini Dondla**

**Fall 2024-II**

**Abstract**

This study uses the Chidamber and Kemerer (CK) metrics suite to investigate the association between software maintainability and class size in open-source Java projects. To find out how class size affects maintainability aspects like complexity and cohesion, two metrics were examined: Weighted Methods per Class (WMC) and Lack of Cohesion in Methods (LCOM). Static analysis was used to collect data from five different Java programs. This study reveals trends that averages miss by mapping class size to these measures on a per-class basis, in contrast to studies that rely on averages. According to the findings, larger class sizes result in less cohesive and more complicated learning environments, which may make them less maintainable. The results direct developers to optimize structure and limit class size to create better maintainable systems.

*Keywords:* Software maintainability, CK metrics, Weighted Methods per Class (WMC), Lack of Cohesion in Methods (LCOM), Java, class size.

**1.Introduction**

The ease of expansion, modification, and updating of a software system is determined by its maintainability, a crucial quality attribute. System maintenance can become more expensive and time-consuming as systems get larger and more complex. Class size has a major effect on maintainability; the larger the class, the more difficult it may be to administrate. Class size has long been associated with software quality, although few empirical research specifically examines the relationship between maintainability and class size. By investigating the effects of class size on important maintainability factors including complexity and coherence in open-source Java projects, this study seeks to understand this link.

The Chidamber and Kemerer (C&K) metrics suite, which offers a collection of proven indicators for assessing software maintainability, is used in this study [6]. Weighted Methods per Class (WMC) and Lack of Cohesion in Methods (LCOM) are the two main measures employed in this study. While LCOM evaluates how effectively a class's methods relate to one another, WMC measures the complexity of a class based on the quantity and complexity of its methods. Because they provide information about the internal complexity and cohesiveness of classes—two factors that are essential to comprehending maintainability—these metrics were selected. This study aims to give a better understanding of how class size affects software maintainability by examining these metrics in relation to class size.

The importance of refactoring and clean code practices in software development cannot be understated. As highlighted by Fowler (2002), refactoring is a critical process for improving the design of existing code to enhance readability, reduce complexity, and facilitate maintainability [8]. This process helps developers avoid the pitfalls of large, complex, and less cohesive classes by promoting the idea of incremental improvements. Similarly, Martin (2008) advocates for maintaining clean code practices, ensuring that methods and classes are cohesive, maintainable, and easy to understand. He emphasizes that writing clean code should be a priority for software developers, as it directly impacts the ease of future modifications and the overall health of the system [9].

This study focuses on these principles by utilizing the Chidamber and Kemerer metrics to quantitatively assess the impact of class size on maintainability. By applying these metrics to real-world projects, this study aims to reveal how class design, based on size, complexity, and cohesion, can affect long-term software maintenanc

* 1. **Research Objectives**

Examining the connection between software maintainability and class size in open-source Java projects is the goal of this study. This study specifically aims to comprehend the effects of software class size, as determined by lines of code (LoC), on crucial maintainability elements like cohesion and complexity. To determine how these class size characteristics affect the overall maintainability of the software system, the study will use the Chidamber and Kemerer (C&K) metrics suite to examine important metrics like Weighted Methods per Class (WMC) and Lack of Cohesion in Methods (LCOM) [6]. Additionally, this study will investigate how greater class sizes lead to less cohesive and complex code, which can impact long-term maintainability, and offer developers practical suggestions for improving software structure

**1.2 Research Questions**

The following research questions will be investigated to have a better understanding of how class size affects software maintainability:

**a) How does class size, measured in lines of code (LoC), relate to the internal complexity and cohesion of Java classes, as measured by WMC and LCOM?**

This question investigates the connection between a class's internal complexity (WMC) and cohesiveness (LCOM) and its physical size. It aims to determine whether larger classes typically have less cohesiveness and more complicated methods, which could make maintainability more difficult

**b) What effects do variations in WMC and LCOM values have on the overall maintainability of Java software projects across small, medium, and large class sizes?**

The purpose of this inquiry is to investigate how varied WMC and LCOM values across various class sizes affect maintainability problems, such as difficulties with program refactoring, debugging, and extension. It will ascertain whether there are quantifiable decreases in maintainability because of larger classes, which have greater WMC and lower LCOM.

**1.3 Research Metrics**

The following metrics will be employed in this study to assess the connection between maintainability and class size:

**a) Lines of Code (LoC):** The main metric for class size is Lines of Code (LoC), which gives an impartial assessment of the physical size of the class. More lines of code are usually found in larger classes, which might increase complexity and maintenance work. LoC will assist in determining whether physically larger classes are associated with greater complexity and less maintainability.

**b) Weighted Methods per Class (WMC):** By tallying and evaluating the intricacy of a class's methods, the WMC measure determines how complex that class is. WMC values that are higher indicate greater internal complexity, making the class harder to understand, test, and maintain. This statistic will be used to assess the impact of class size on overall complexity and maintainability.

**c)Lack of Cohesion in Methods (LCOM):** LCOM assesses how cohesive a class's methods are. The methods in a class are less coherent when the LCOM score is larger, indicating that the class is poorly designed and challenging to maintain. This measure will assist in evaluating how class size affects the cohesiveness and internal organization of the class, which in turn affects maintainability.

With the use of these metrics, a thorough examination of the ways in which class size, complexity, and cohesiveness affect the general maintainability of Java software projects will be possible, giving developers advice on how to organize their code for improved long-term maintainability.

**2.Data Set Overview**

The Java projects selected for this empirical investigation, which aims to examine the connection between software maintainability and class size, are introduced in this section. To guarantee that the projects chosen to provide significant insights into the practical effects of class size on software quality, the following standards were set.

**2.1 Project Selection Criteria**

When choosing the Java projects, we used these guidelines to ensure a thorough and accurate analysis:

**Minimum Code Size**: To guarantee a suitably vast and sophisticated codebase, each chosen project must have at least 200,000 Lines of Code (LoC). This guarantees that a wide variety of maintainability problems pertaining to big software systems are captured.

**Project Age**: Projects must be at least eight years old for us to examine systems that have experienced substantial upkeep and evolution, providing a better understanding of the long-term effects of class size on maintainability.

**Contributor Engagement**: To represent the collaborative character of large-scale projects, which usually feature a range of coding styles and cooperation dynamics that impact maintainability, we selected projects with at least 200 contributors.

Based on these criteria, the chosen projects offer complicated and varied software systems that will shed light on the relationship between class size and software maintainability.

**2.2 Projects Chosen for Analysis**

The following five open-source Java-based projects were selected for this study based on the selection criteria:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project Name** | **Developers** | **Age (Years)** | **Size (LOC)** | **Description** |
| Apache Pulsar | 394 | 6 | 2,48,880 | A high-throughput, low-latency distributed messaging platform for real-time event processing. |
| TechEmpower Framework Benchmarks | 387 | 8 | 3,04,118 | A performance benchmark suite for evaluating the performance of various web frameworks. |
| Apache Druid | 370 | 10 | 2,45,187 | A high-performance real-time analytics database for OLAP queries on event data. |
| OpenHAB Add-ons | 368 | 7 | 2,90,837 | A set of add-ons for the open-source home automation platform, OpenHAB. |
| Teammates | 362 | 7 | 2,33,284 | An online peer evaluation system designed to improve feedback processes in educational settings. |

**2.3 In-Depth Look at the Selected Projects**

To set the scene for their inclusion in this study, the following is a thorough synopsis of each of the chosen projects:

**i)Apache Pulsar:** An open-source distributed messaging system called Pulsar was created to manage real-time event processing at high throughput and low latency. It is employed in the development of highly available and scalable applications that must process enormous volumes of data quickly [1]. Pulsar is a great choice to investigate how class size impacts maintainability in systems that need high scalability and fault tolerance because of its big codebase and the complexity of real-time messaging networks.

**ii)TechEmpower Framework Benchmarks:** The performance of several web frameworks is assessed using a set of benchmarks provided by this project [2]. From basic request-response systems to intricate web frameworks, it supports a broad variety of server architectures. Examining the effects of various class sizes and design decisions on the maintainability of performance-critical systems is made possible by this project.

**iii)Apache Druid:** A real-time analytics database designed for OLAP-style, fast searches on big datasets are called Druid [3]. It is essential for business intelligence and event-streaming use cases and is frequently used for real-time data processing. The project is an excellent choice for investigating the difficulties of overseeing sizable, intricate classes and how they affect maintainability because of its extensive data input and query processing methods.

**iv)OpenHAB Add-ons:** OpenHAB is a framework for creating smart home apps, and its extensions offer more features for integrating different IoT services and devices [4]. Examining how class size and dependencies impact the maintainability of a big, modular software system is made easier with the OpenHAB Add-ons project, which has a high number of contributors.

**v)Teammates:** An online peer evaluation tool called Teammates was created to make providing comments in educational settings more efficient[5]. The system presents an intriguing scenario for researching the maintainability of software that integrates various components with varying complexities because of its sizable codebase and the complicated interactions between administrative modules, assessment systems, and user interfaces

**3.Framework for Measurement**

The tools and metrics used in this study to examine the connection between software maintainability and class size in Java-based open-source projects are presented in this section. This study investigates the effects of internal complexity and software class structure on maintainability using key metrics from the Chidamber and Kemerer (C&K) suite.

**3.1 Analyzing Software Maintainability: The CKJM Tool**

The CKJM (Class-level Java Metrics) tool[6], an open-source software application made to calculate a range of object-oriented metrics that evaluate the maintainability of Java applications, serves as the main instrument for this study. This tool enables a thorough examination of a system's architecture, offering vital information about its internal organization and how certain factors—such as class size—affect software maintenance and extensibility. The results of CKJM are essential for comprehending the maintainability of big, intricate Java systems, where class design is crucial to the long-term health of the codebase.

**3.2 Using the C&K Suite for Metric Computation**

To produce a set of object-oriented metrics based on the Chidamber and Kemerer (C&K) suite, CKJM processes Java bytecode. We concentrate on two of these, Weighted Methods per Class (WMC) and Lack of Cohesion in Methods (LCOM), because they have a direct bearing on software maintainability and are hence very pertinent to this investigation. WMC and LCOM are given priority in this study because they offer more precise insights into how class size impacts the complexity and cohesiveness of software systems, even though CKJM also calculates other metrics including Depth of Inheritance Tree (DIT), Number of Children (NOC), and Coupling Between Objects (CBO).

**3.3 Data Extraction: Utilizing the Tool**

The source code of the five chosen Java projects was subjected to CKJM application, which was carried out in accordance with typical installation processes. After processing each project's bytecode, the program generated output with comprehensive stats for every class. By providing information about the classes' cohesiveness (LCOM) and complexity (WMC), these metrics enabled us to evaluate the structural and functional characteristics of every system. After the findings were arranged and ready for analysis, it became clearer how class size affects each project's maintainability.

**3.4 Core Metrics for Assessing Maintainability**

To assess how class size affects maintainability, this study focuses on two key metrics from the C&K suite:

* **Weighted Methods per Class (WMC):** WMC is a metric that calculates the class's overall difficulty by adding up the complexity of each method in the class. The complexity of a class rises with more methods, more arguments, and intricate interfaces with external systems. High WMC values frequently signify that a class has grown increasingly complex, making it more difficult to maintain, test, and comprehend. WMC values typically increase with class size, resulting in more maintenance work and maybe more challenging changes in the future.
* **Lack of Cohesion in Methods (LCOM):** LCOM gauges the degree of relationship between a class's methods. Poor design and decreased cohesiveness may be indicated by a high LCOM number, which indicates that the class's methods are not well connected. Low-cohesion classes are usually more difficult to comprehend and keep up with. Due to the addition of more varied techniques, larger classes frequently exhibit lesser cohesiveness, which raises LCOM values. This measure aids in determining whether classes get too complicated and challenging to oversee.

This study will give a comprehensive understanding of how class size leads to maintainability issues in Java-based software systems by concentrating on WMC and LCOM. These metrics provide specific information on how developers might enhance system quality by optimizing class design.

**4. Analysis and Results**

**i)Teammates Project Analysis:**

**Observation:** Smaller classes in Teammates generally show lower wmc and better lcom values, but medium and large classes have an uneven distribution of these metrics. Scatter plots indicate no clear threshold where lcom deteriorates sharply, but trends suggest medium-sized classes are optimal for maintainability.

**Research Question Supported**: "How does class size relate to internal complexity and cohesion?"

**Implication:** For future development, focus on enforcing cohesion through modular design in large classes.

A screen shot of a graph

Description automatically generated

A graph with green and orange dots

Description automatically generated

A graph with green and orange dots

Description automatically generated

A graph with a red and blue rectangle

Description automatically generatedA graph with green and orange dots

Description automatically generated

**ii)Apache Druid Analysis:**

**Observation:** Box plots of wmc indicate a wide range of complexity across medium and large classes, suggesting inconsistencies in design practices. Correlation heatmaps reveal a strong positive correlation between loc and wmc but a negative correlation between loc and lcom. The count of unmaintainable classes graph shows that medium and large classes contribute significantly to unmaintainable code.

**Research Question Supported:** "How does class size relate to internal complexity and cohesion?"

**Implication:** Standardizing complexity with consistent design patterns can improve maintainability. Refactoring efforts should focus on reducing unmaintainable classes in medium and large class categories.

**A graph with green dots

Description automatically generated**

**A graph with green and red dots

Description automatically generated**

**A graph with green dots

Description automatically generated**

**A screenshot of a graph

Description automatically generated**

**A graph with green dots

Description automatically generated**

**iii)Apache Pulsar Analysis:**

**Observation:** Larger classes in Pulsar show significantly higher wmc values, highlighting increased complexity. Scatter plots of loc vs. wmc reveal a steep upward trend, especially for classes with loc > 500. Additionally, lcom values indicate that cohesion is notably lower for these larger classes, confirming that larger sizes lead to less maintainable code. The count of unmaintainable classes graph shows a high number of classes exceeding the thresholds (wmc > 10 or lcom > 0.8), particularly in large class categories.

**Research Question Supported:** "How does class size relate to internal complexity and cohesion?"

**Implication:** Refactoring large, complex classes into smaller, cohesive units and prioritizing unmaintainable classes for redesign could significantly improve maintainability.

**A graph with green dots

Description automatically generated**

**A graph with green dots

Description automatically generated**

**A graph of green dots

Description automatically generated**

**A graph with a red and blue rectangle

Description automatically generated**

**A graph with green and orange dots

Description automatically generated**

**iv)TechEmpower Framework Benchmarks Analysis:**

**Observation:** Medium-sized classes (200 ≤ loc ≤ 500) provide the best balance of wmc (complexity) and lcom (cohesion). Small classes have low complexity but show occasional cohesion issues. Large classes exhibit both high complexity and low cohesion, contributing to a moderate count of unmaintainable classes in the larger size category, as shown in the graph.

**Research Question Supported:** "What effects do variations in wmc and lcom have on maintainability across different class sizes?"

**Implication:** Developers should aim to design classes within the medium size range to maximize maintainability and reduce the frequency of unmaintainable classes.

**A graph with blue dots and red dots

Description automatically generated**

**A graph with blue dots and black text

Description automatically generated**

**A graph with blue dots and numbers

Description automatically generated**

**A graph with a red and blue rectangle

Description automatically generated**

**A graph with blue dots

Description automatically generated**

**v)OpenHAB Add-ons Analysis:**

**Observation:** This project features several outliers with extremely high wmc values (>20) in the large class category, making them particularly challenging to maintain. Cohesion (lcom) is also poor for these classes, indicating functionality is distributed inefficiently. The count of unmaintainable classes graph highlights that a significant number of classes in the large size category exceed maintainability thresholds.

**Research Question Supported**: "What effects do variations in wmc and lcom have on maintainability across different class sizes?"

**Implication**: Focus should be placed on refactoring outlier classes with excessive complexity. Introducing modular designs to distribute functionality more effectively could reduce the count of unmaintainable classes.

**A graph with green dots

Description automatically generated**

**A graph with numbers and lines

Description automatically generated**

**A graph with green dots

Description automatically generated**

**A graph with a red and blue rectangle

Description automatically generated**

**A graph with green dots

Description automatically generated**

**5.0 Conclusion**

* **Influence of Class Size on Maintainability**: The study confirms that larger class sizes negatively affect software maintainability. Elevated wmc values indicate increased complexity, while higher lcom values reveal reduced cohesion. These trends highlight how larger classes hinder understandability, testing, and modification.
* **Impact on Complexity:** Larger classes encapsulate more functionality, leading to greater wmc values. This rise in complexity complicates debugging, increases the likelihood of errors during updates, and demands more extensive refactoring efforts.
* **Relationship Between Class Size and Cohesion**: The lcom metric demonstrates that larger classes tend to have lower cohesion, implying poorly integrated methods. This lack of cohesion makes the classes harder to maintain and prone to design flaws.
* **Consequences for Long-Term Maintenance:** The findings suggest that unchecked growth in class size could lead to higher maintenance costs and technical debt. Without proactive intervention, such as modularization and regular refactoring, systems become increasingly rigid and harder to evolve.

**5.1 Recommendations**

* **Optimize Class Size**: Refactor large classes into smaller, more focused ones using modular design principles and the Single Responsibility Principle (SRP).
* **Enhance Cohesion**: Regularly assess and improve lcom values by ensuring that methods within a class serve a unified purpose.
* **Monitor and Refactor**: Continuously analyze metrics like wmc and lcom to identify problematic classes. Regular refactoring can reduce complexity and improve maintainability.
* **Encourage Best Practices:** Utilize design patterns (e.g., facade or adapter) to manage dependencies and improve cohesion across large classes.

**6.2 Limitations**

* **Narrow Scope**: The analysis was limited to five Java projects, which may not generalize to all languages or software systems.
* **Focus on Static Metrics:** Only wmc and lcom were analyzed; other important metrics like coupling (CBO), inheritance depth, or runtime behavior were excluded.
* **Snapshot View:** The study considers a single point in time, omitting how maintainability evolves as projects undergo refactoring or expansion.

**REFERENCES**

[1] Apache Pulsar. (2024). *Apache Pulsar: A distributed messaging system designed for high-throughput, low-latency, real-time event processing*. Retrieved from <https://github.com/apache/pulsar>

[2] TechEmpower Framework Benchmarks. (2024). *A performance benchmark suite for evaluating the performance of various web frameworks*. Retrieved from <https://github.com/TechEmpower/FrameworkBenchmarks>

[3] Apache Druid. (2024). *Apache Druid: A high-performance, real-time analytics database for OLAP queries on event data*. Retrieved from <https://github.com/apache/druid>

[4] OpenHAB Add-ons. (2024). *OpenHAB Add-ons: A set of add-ons for the open-source home automation platform, OpenHAB*. Retrieved from <https://github.com/openhab/openhab-addons>

[5] Teammates. (2024). *Teammates: An online peer evaluation system designed to improve feedback processes in educational settings*. Retrieved from <https://github.com/TEAMMATES/teammates>

[6] CKJM Tool. (2024). *CKJM (Class-level Java Metrics) Tool: An open-source tool for calculating software metrics based on the Chidamber and Kemerer (C&K) suite*. Retrieved from <https://github.com/mauricioaniche/ck>

[7] Chidamber, S. R., & Kemerer, C. F. (1994). *A Metrics Suite for Object-Oriented Design*. *IEEE Transactions on Software Engineering, 20*(6), 476–493. DOI: 10.1109/32.295895

[8] Fowler, M. (2002). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional. ISBN: 978-0201485677.

[9] Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall. ISBN: 978-0132350884.