

Empirical Analysis of Class Size and Its Impact on Software Maintainability Using the C&K Metrics

Group Assignment 1



April 14, 2024

Tharun Boyapati

ROHITH PUPPALA

# Section 1: Objectives, Questions, and Metrics

## 1.1 Objectives

The primary objective of this study is to empirically evaluate how the size of classes in Java software impacts their maintainability. This research aims to identify patterns and relationships between class size and maintainability attributes, providing insights that could guide software development practices towards creating more maintainable software systems.

## 1.2 Research Question

Based on the overarching goal of understanding the impact of class size on software maintainability, the study focuses on the following research question:

**How does the size of classes, measured in lines of code, affect the maintainability of software systems in Java projects?**

## 1.3 Metrics

To answer the research question, we will utilize two specific metrics from the Chidamber and Kemerer (C&K) suite, which are particularly relevant for assessing software maintainability:

* **Coupling Between Object classes (CBO):** This metric measures the number of classes coupled to a given class (either through method calls or variable accesses). Higher values of CBO indicate higher coupling, which can reduce maintainability because the class is less modular and changes in one class might affect multiple other classes.
* **Weighted Methods per Class (WMC):** This metric sums the complexities of all methods in a class. It acts as an indicator of how much responsibility a class is carrying; more methods or more complex methods can make the class harder to understand and maintain.

Both metrics, CBO and WMC, provide a quantitative foundation for evaluating and understanding the maintainability of software components based on class design and interaction within the software architecture. By analyzing these metrics across different classes and correlating them with class size, we can derive empirical evidence about the effects of class size on maintainability.

These metrics will be computed using the CK-Code metric tool, which performs static analysis of Java code to extract the required data. The choice of these metrics is grounded in their established reliability and relevance in numerous empirical studies in software engineering research, focusing on maintainability and quality assessment of object-oriented software systems.

# Section 2: Description of Subject Programs (Data Set)

In this study, we selected five Java projects from GitHub to analyze the impact of class size on software maintainability using the C&K metrics. The projects were chosen based on their popularity, diversity in application domains, and compliance with our pre-defined criteria (i.e., size, age, and developer engagement). Below is a detailed description of each project, along with a table summarizing their main attributes.

## 2.1 Bytecode Viewer

**Bytecode Viewer** is an advanced yet user-friendly Java bytecode viewer, decompiler, and editor. It integrates multiple decompilers and bytecode editors, facilitating a wide range of actions on Java .class files for analysis and modification purposes.

## 2.2 Mantis

**Mantis** by Netflix is a platform for building real-time, cost-effective, operations-focused applications. It is used to manage and analyze live workflows, providing insights that help in operational decision-making across various services.

## 2.3 Mockito

**Mockito** is a popular mocking framework for unit tests in Java. It is used to create and configure mock objects. Utilizing Mockito significantly simplifies the development of tests by replacing dependencies with mock objects.

## 2.4 OpenNLP

**OpenNLP** is a machine learning-based toolkit for processing natural language text. It supports common NLP tasks such as tokenization, sentence segmentation, part of speech tagging, named entity extraction, chunking, parsing, and coreference resolution.

## 2.5 Priam

**Priam** is an open-source project by Netflix that provides automated backup and recovery support for Amazon's Cassandra. It also performs configuration and maintenance tasks for Cassandra instances, thus easing the management of large Cassandra clusters.

2.6 Summary Table of Studied Programs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Project** | **URL** | **Description** | **Size (LoC)** | **Age (Years)** | **Developers Count** |
| Bytecode Viewer | <https://github.com/Konloch/bytecode-viewer> | Java bytecode viewer and editor | 19,265 | 10 | 42 |
| Mantis | <https://github.com/Netflix/mantis> | Real-time operations platform | 93,605 | 5 | 29 |
| Mockito | <https://github.com/mockito/mockito> | Mocking framework for unit tests in Java | 51,494 | 9 | 288 |
| OpenNLP | <https://github.com/apache/opennlp> | Toolkit for processing natural language text | 61,929 | 6 | 50 |
| Priam | <https://github.com/Netflix/Priam> | Cassandra management tool from Netflix | 18,941 | 7 | 51 |

# Section 3: Description of the Analysis Tool

## 3.1 Tool Overview

For the purpose of extracting the Coupling Between Object classes (CBO) and Weighted Methods per Class (WMC) metrics, we utilized the CK tool, a Java software metrics collection utility. This tool is specifically designed to perform static analysis on Java codebases to compute a suite of object-oriented metrics proposed by Chidamber and Kemerer, commonly known as C&K metrics.

## 3.2 Tool Capabilities

The CK tool is capable of analyzing Java projects to extract numerous metrics including, but not limited to, CBO, WMC, Number of Methods (NOM), Depth of Inheritance Tree (DIT), and Number of Children (NOC). For this study, our focus was particularly on CBO and WMC due to their relevance to assessing software maintainability as discussed earlier.

## 3.3 Usage of the Tool

The CK tool requires Java and Maven to be installed on the system where the analysis is conducted. It operates by cloning a Java project from a repository, compiling the project, and then running the analysis through a command-line interface. The output is generally in the form of a CSV file, which lists the calculated metrics for each class in the analyzed software. This output format facilitates easy integration with data analysis tools for further examination.

## 3.4 Tool Application

In our study, the CK tool was systematically applied to the five selected Java projects. For each project, the tool was executed to extract the metrics values for each class within the projects. The tool's efficiency in handling projects of varying sizes and complexities was crucial, as it allowed for a consistent methodology across different codebases.

# Section 4: Analysis of the projects

**CBO (Coupling Between Object Classes):** The CBO metric reflects the level of interdependence between classes. A high CBO value suggests that a class is highly dependent on other classes, which could indicate a design that is less modular and might be more difficult to maintain. Conversely, a low CBO value implies better modularity and potentially easier maintainability.

**WMC (Weighted Methods per Class):** WMC counts the number of methods in a class and weights them according to their complexity. A high WMC value can imply that a class has too many responsibilities or is too complex, making it harder to understand, test, and maintain.

The analysis will incorporate these observations and detail how the distribution of CBO and WMC values across the classes in each project correlates with the potential maintainability of the software. It will also be necessary to mention the mean, median, and standard deviation for each project to provide a statistical overview of the CBO and WMC values within each codebase.

In this section, we analyze the results obtained from the application of the CK metrics tool on the selected Java projects. The focus is on the Coupling Between Object classes (CBO) and Weighted Methods per Class (WMC) metrics, as these are indicative of the software's maintainability. We will discuss the implications of the metrics' distribution and point out notable trends and outliers that may influence the software's maintainability.

## 4.1 Analysis of Weighted Methods per Class (WMC)

The WMC metric reflects the complexity and responsibilities of a class. It is indicative of the potential effort required to maintain and understand the class. In our dataset, we observed a common pattern where a large proportion of classes have a low WMC value, suggesting that they are likely to be simpler and possibly more maintainable.

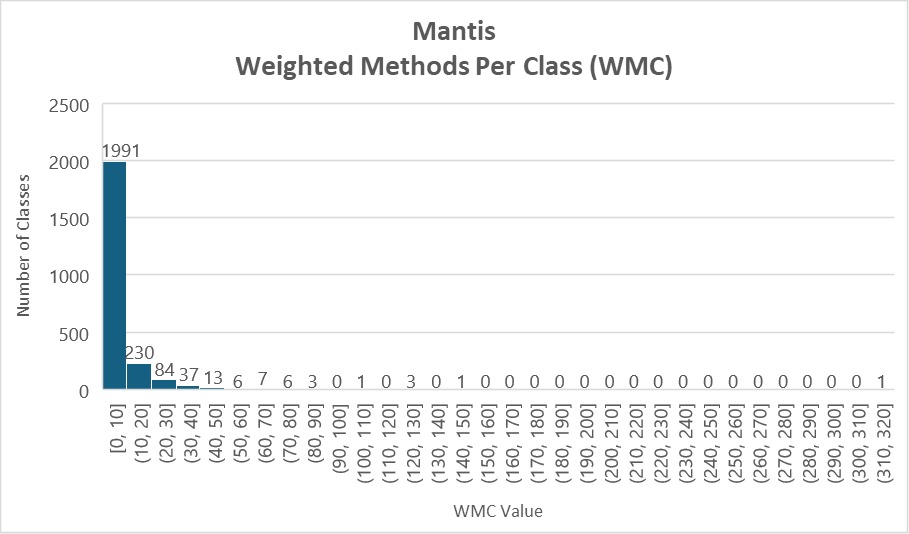


Figure 1: Mantis WMC graph

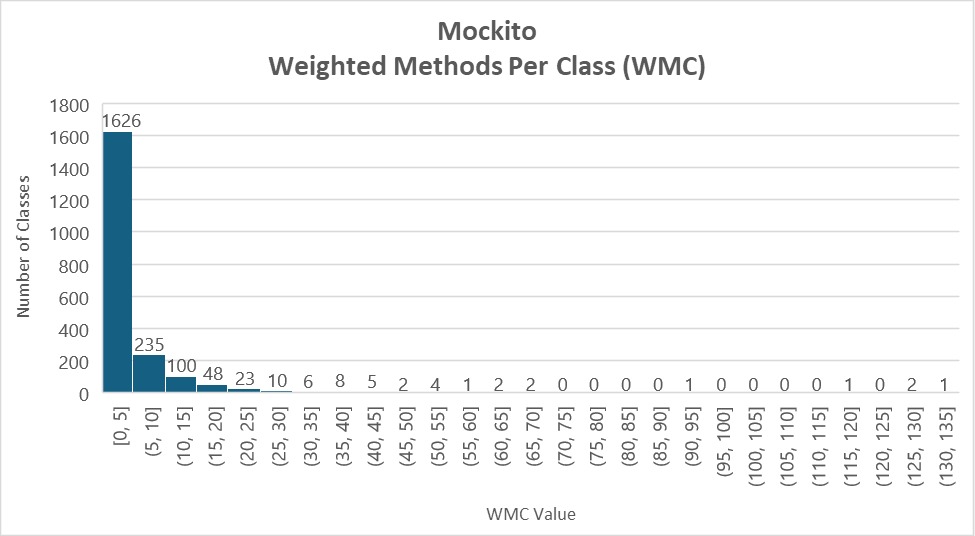


Figure 2: Mockito WMC Graph

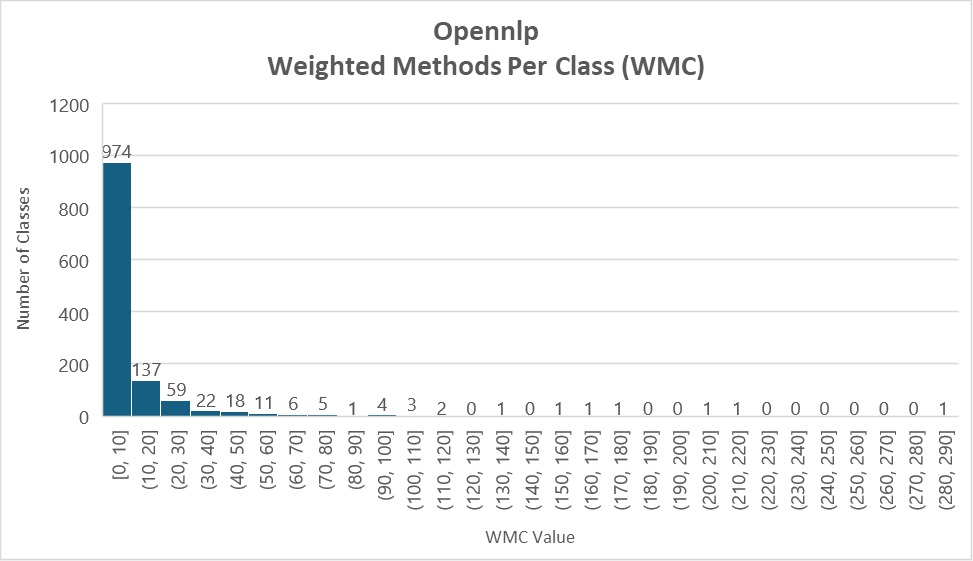


Figure 3:OpenNLP WMC Graph

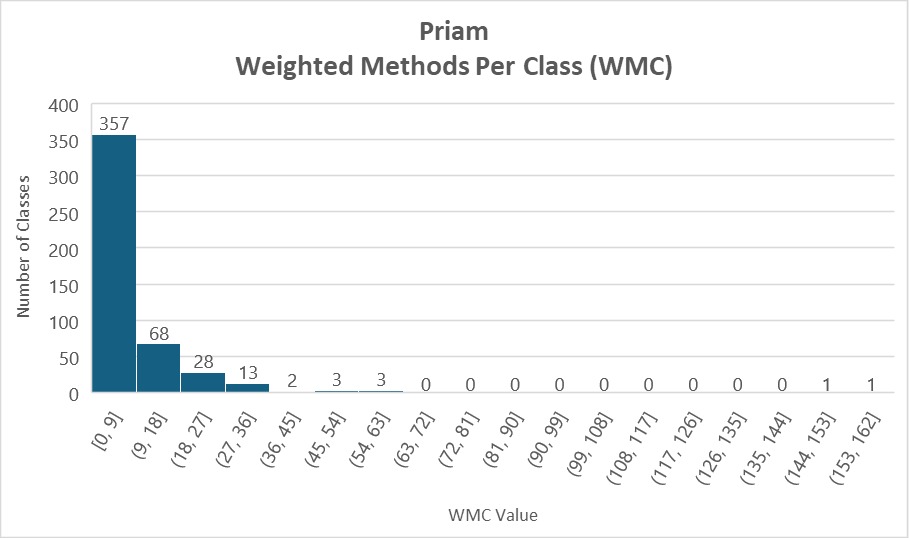


Figure 4:Priam WMC Graph

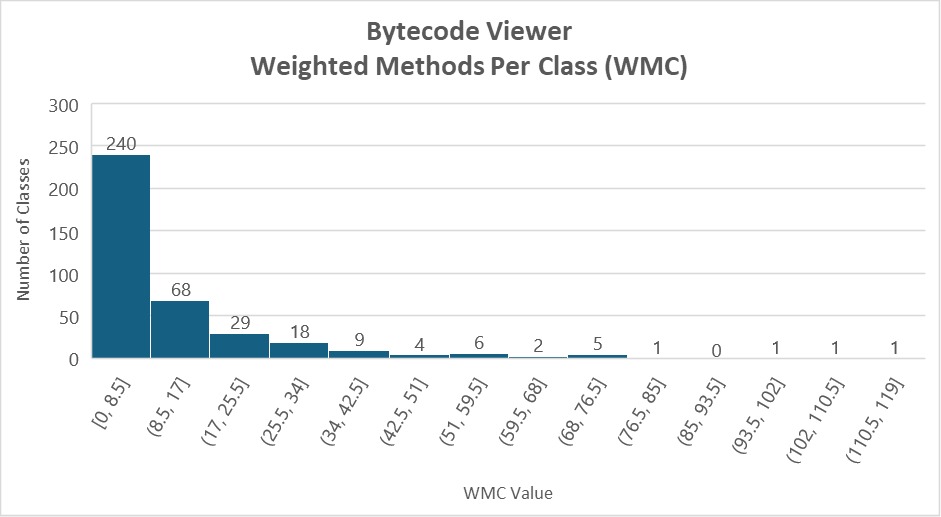


Figure 5: Bytecode Viewer WMC Graph

For each project, the graph showing the distribution of WMC values demonstrates that while most classes have a low WMC score, indicating lower complexity, there are several classes with notably high WMC scores. These outliers are indicative of classes that may be overly complex and thus harder to maintain. They may represent key areas in the system that are functionality-rich or central to the application logic. Such classes could be targets for refactoring, especially if they are critical to the application.

## 4.2 Analysis of Coupling Between Object Classes (CBO)

The CBO metric measures the degree to which a class is coupled with other classes. A lower CBO suggests better modularity, while a higher CBO can indicate a class's potential over-reliance on other parts of the software.

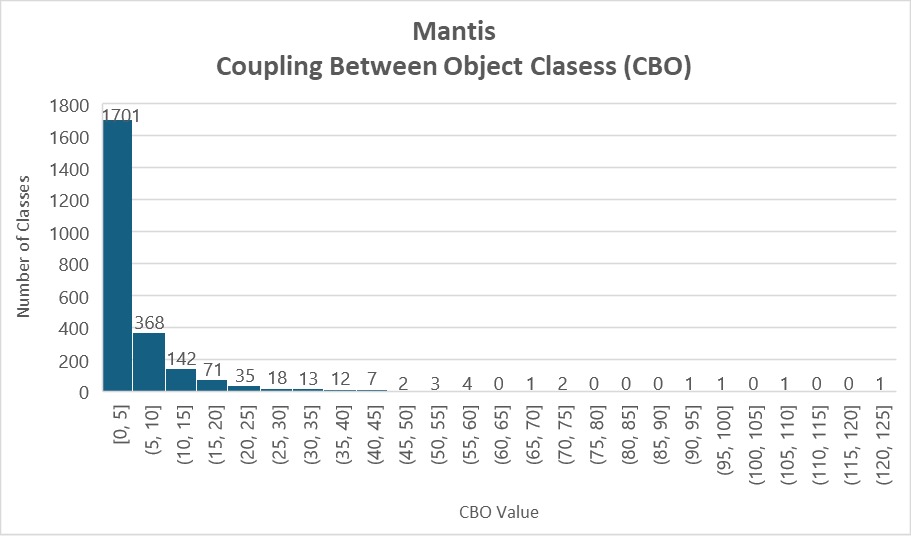


Figure 6:Mantis CBO Graph

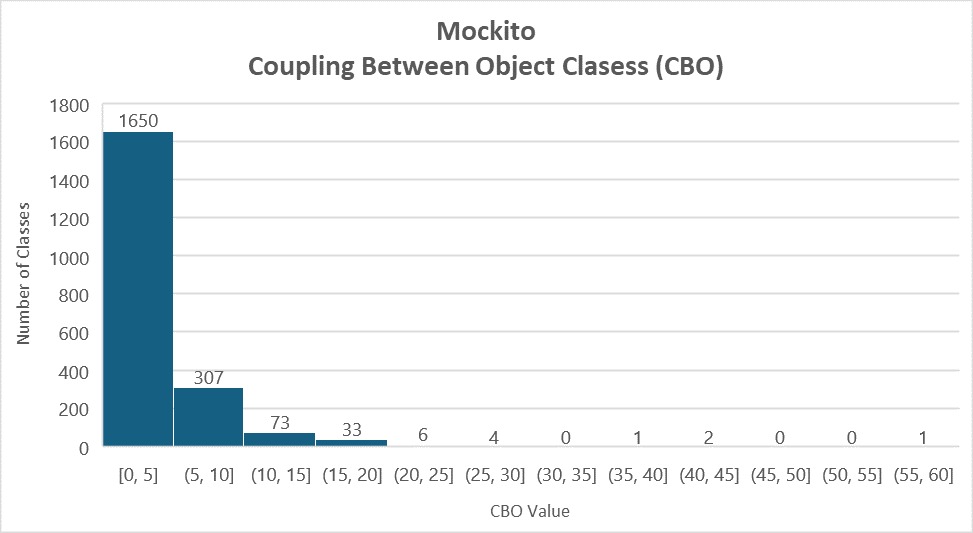


Figure 7: Mockito CBO Graph

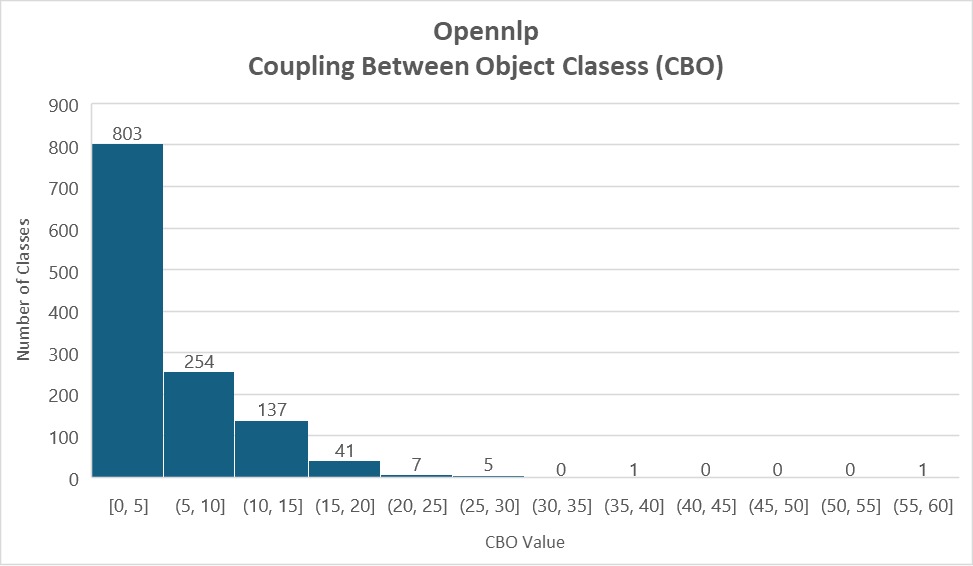


Figure 8:OpenNLP CBO Graph

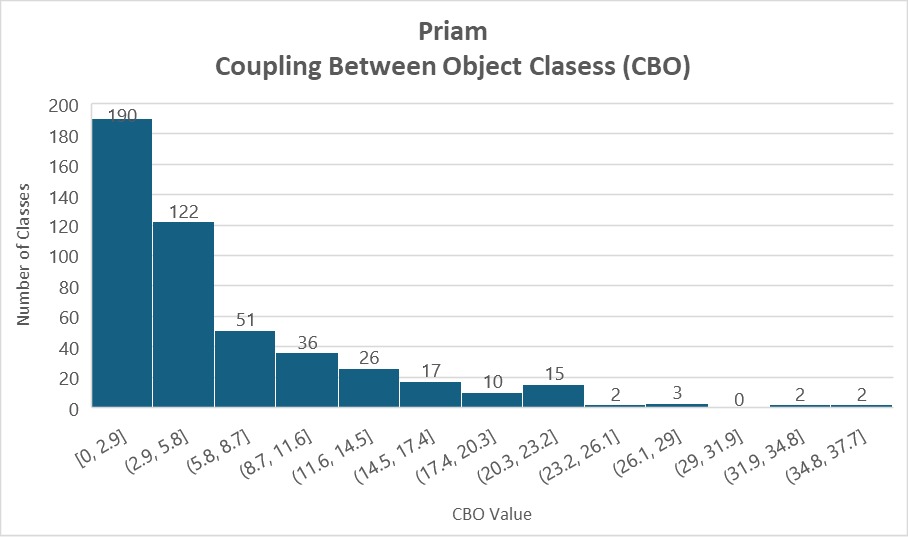


Figure 9: Priam CBO Graph

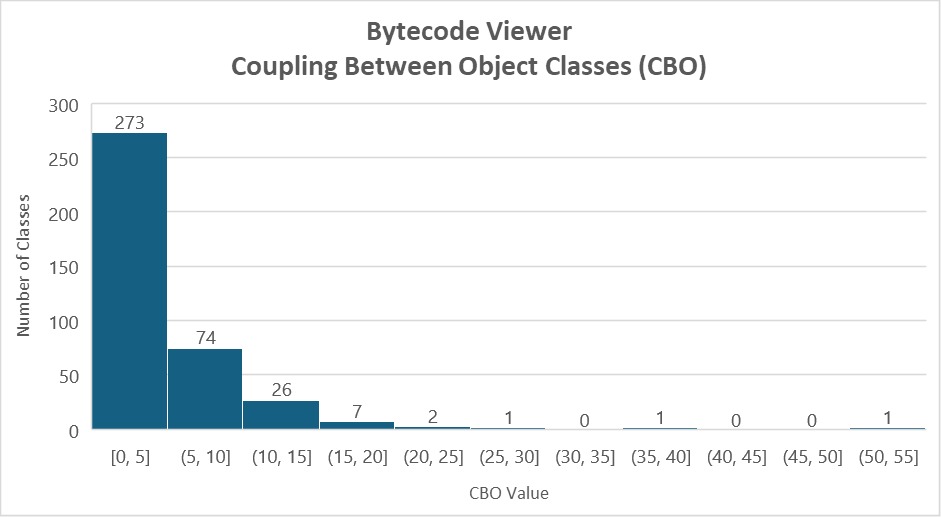


Figure 10:Bytecode Viewer CBO Graph

Across the projects, the CBO graphs reveal that a majority of classes have relatively low coupling, which is a positive sign for maintainability. However, some classes exhibit a high degree of coupling. These classes could be problematic from a maintenance standpoint because changes in one class could have ripple effects on others, increasing the risk of defects and the cost of changes.

## 4.3 Implications for Software Maintainability

The WMC and CBO metrics' values and their distributions have important implications for software maintainability. Generally, classes with lower WMC and CBO values are expected to be more maintainable. Classes that present as outliers, with high WMC and CBO values, may be critical for the maintainability of the software and could benefit from targeted refactoring efforts. Additionally, the concentration of classes in the lower end of the spectrum suggests that these projects, on the whole, adhere to good object-oriented design principles that favor maintainability.

## 4.4 Project-Specific Observations

For each project, we can make specific observations based on the distribution of CBO and WMC:

* **Bytecode Viewer:**
  + WMC: The majority of classes have low complexity, with a steep drop-off as complexity increases.
  + CBO: Most classes have a low to moderate coupling, with few classes having high coupling.
* **Mantis:**
  + WMC: A significant concentration of classes with low complexity, but also a noticeable presence of highly complex classes.
  + CBO: Predominantly low coupling, with a steep decline as coupling increases.
* **Mockito:**
  + WMC: Shows a large number of low complexity classes, but also outliers indicating some very complex classes.
  + CBO: Coupling is generally low, which is positive for maintainability.
* **OpenNLP:**
  + WMC: Similar to other projects with a high number of simple classes and a few complex ones.
  + CBO: The coupling shows a similar trend, with most classes having low coupling.
* **Priam:**
  + WMC: Demonstrates a majority of classes with low to moderate complexity.
  + CBO: Shows a high number of classes with minimal coupling.

## 4.5 Statistical Overview

The mean, median, and standard deviation for each project's CBO and WMC metrics were calculated to provide a statistical overview of the metrics distribution.

### 4.5.1 Bytecode Viewer

* **CBO:**
  + Maximum: 52
  + Mean: 4.480519
  + Standard Deviation: 5.269393
* **WMC:**
  + Maximum: 114
  + Mean: 11.48312
  + Standard Deviation: 16.74423

The Bytecode Viewer project has moderate coupling and complexity. The relatively low mean and standard deviation for CBO suggest a loosely coupled architecture, which is favorable for maintenance. However, the higher standard deviation for WMC implies that there are classes with varying levels of complexity, with some potentially requiring simplification.

### 4.5.2 Mantis

* **CBO:**
  + Maximum: 125
  + Mean: 5.408728
  + Standard Deviation: 8.469438
* **WMC:**
  + Maximum: 317
  + Mean: 6.426353
  + Standard Deviation: 12.71434

Mantis shows a greater range of coupling and complexity. The high maximum CBO value indicates some highly coupled classes, which may be critical to the application and could hinder maintenance efforts. The large spread in WMC values suggests that while many classes are simple, there are several highly complex classes that may be candidates for refactoring.

### 4.5.3 Mockito

* **CBO:**
  + Maximum: 57
  + Mean: 3.313539
  + Standard Deviation: 4.322487
* **WMC:**
  + Maximum: 135
  + Mean: 4.452576
  + Standard Deviation: 9.056092

Mockito demonstrates low mean values for both CBO and WMC, indicating a design that favors maintainability with low coupling and complexity. The standard deviations are moderate, suggesting that while there is some variability, it is not as pronounced as in other projects.

### 4.5.4 OpenNLP

* **CBO:**
  + Maximum: 57
  + Mean: 5.473179
  + Standard Deviation: 5.143058
* **WMC:**
  + Maximum: 290
  + Mean: 9.984788
  + Standard Deviation: 19.82656

OpenNLP's metrics reveal a project with classes that are, on average, moderately coupled and complex. The high maximum values for both CBO and WMC indicate the presence of certain classes that are both highly coupled and complex, which could become a challenge for maintainers.

### 4.5.5 Priam

* **CBO:**
  + Maximum: 36
  + Mean: 5.855042
  + Standard Deviation: 6.419999
* **WMC:**
  + Maximum: 158
  + Mean: 7.651261
  + Standard Deviation: 13.34057

For Priam, the metrics suggest a relatively uniform distribution with a moderate mean for both CBO and WMC. However, the standard deviation, especially for WMC, is somewhat high, pointing to a certain number of classes that are outliers in complexity and may benefit from optimization.

## 4.6 Discussion

The statistical data presented above provides insights into the nature of each project. Lower mean values for CBO and WMC are generally indicative of a codebase that is more maintainable. However, the presence of high maximum values and large standard deviations signal the existence of outlier classes that may disproportionately affect the overall maintainability of the project.

In conclusion, the analysis of the CBO and WMC metrics has highlighted both strengths and potential maintenance hotspots within the studied projects. The projects generally exhibit good practices with most classes having low coupling and complexity, but the identified outliers are critical areas for potential improvement to enhance overall maintainability.

# Section 5: Conclusions

This empirical study aimed to investigate the effect of class size, as measured by lines of code, on the maintainability of software, using two of the Chidamber and Kemerer (C&K) metrics: Coupling Between Object classes (CBO) and Weighted Methods per Class (WMC). Through the analysis of five Java-based projects, we sought to determine trends and correlations that would offer insights into the maintainability implications of class size.

## 5.1 Summary of Findings

Our analysis revealed several key findings:

* **Low Coupling and Complexity:** The majority of classes across all projects had low CBO and WMC values, indicating adherence to good object-oriented design principles that facilitate maintainability.
* **Presence of Outliers:** Each project had outliers with high values of CBO and WMC, suggesting classes that are complex and tightly coupled, potentially making them difficult to maintain.
* **Variability in Maintainability:** There was noticeable variability within projects, as reflected by the standard deviation values. Some projects showed greater variability, implying a more significant disparity in the maintainability of individual classes.

## 5.2 Implications for Software Engineering

The findings of this study have several implications for software engineering practices:

* **Refactoring for Maintainability:** Classes identified as outliers for their high CBO and WMC values should be primary candidates for refactoring. Focusing maintenance efforts on these classes could lead to a disproportionate improvement in the overall maintainability of the software.
* **Predictive Maintenance Planning:** The observed trends could inform predictive maintenance planning, where projects could be profiled early on for potential maintainability issues based on these metrics.
* **Balanced Design Considerations:** While striving for low CBO and WMC is beneficial, the presence of high-value outliers also indicates that complex functionalities might necessitate exceptions to the rule. These must be balanced carefully against the overall design and maintenance goals.

## 5.3 Study Limitations and Future Work

The conclusions drawn from this study come with the recognition of its limitations:

* **Scope of Metrics:** While CBO and WMC are indicative, they are not exhaustive measures of maintainability. Future studies could incorporate additional metrics to provide a more holistic view.
* **Qualitative Aspects:** Maintainability is also affected by qualitative aspects such as code readability and documentation, which were not assessed in this study.
* **Broader Sample:** The projects analyzed represent a snapshot, and broader samples could be used to generalize findings further.

## 5.4 Final Thoughts

In conclusion, the study confirms the value of using CBO and WMC metrics as indicators of software maintainability and underscores the importance of modular, low-complexity design. It also highlights the need to monitor for and address complexity and coupling outliers to maintain and improve software maintainability. As the field of software engineering continues to evolve, the continuous evaluation of these and other metrics will be crucial in managing the growing complexity of software projects and ensuring their long-term maintainability. This study contributes to that ongoing endeavor, offering both a methodology for evaluation and a basis for further research.

**References:**

1. S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design," IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476-493, Jun. 1994, doi: 10.1109/32.295895.
2. M. Aniche, "ck: CK metrics calculator," GitHub repository. [Online]. Available: <https://github.com/mauricioaniche/ck>
3. K. (n.d.). *GitHub - Konloch/bytecode-viewer: A Java 8+ Jar & Android APK Reverse Engineering Suite (Decompiler, Editor, Debugger & More)*. GitHub. <https://github.com/Konloch/bytecode-viewer>
4. N. (n.d.). *GitHub - Netflix/mantis: A platform that makes it easy for developers to build realtime, cost-effective, operations-focused applications*. GitHub. <https://github.com/Netflix/mantis>
5. M. (n.d.). *GitHub - mockito/mockito: Most popular Mocking framework for unit tests written in Java*. GitHub. <https://github.com/mockito/mockito>
6. A. (n.d.). *GitHub - apache/opennlp: Apache OpenNLP*. GitHub. <https://github.com/apache/opennlp>
7. N. (n.d.). *GitHub - Netflix/Priam: Co-Process for backup/recovery, Token Management, and Centralized Configuration management for Cassandra.* GitHub. <https://github.com/Netflix/Priam>