

Empirical Study on the Effect of Class Size on Software Maintainability Using CK

Metrics

Govardhan Reddy Jakkireddy

Sivasai Teja Yedla

Yeshwanth Bommagani

Object Oriented Development

Group Assignment-1

Section1: Introduction

Objective: The goal of this empirical study is to investigate the relationship between class size and software maintainability within modern Java projects. By analyzing various software components, we aim to determine how class size influences the maintainability of software systems in a practical context. The study will be based on the CK metrics framework to measure maintainability attributes in Java projects.

Questions:

1. How does class size, measured in lines of code (LoC), impact the maintainability of software systems?
2. Which maintainability attributes, as represented by CK metrics (Coupling, Cohesion, Inheritance, etc.), show significant variance based on class size?
3. Can certain CK metrics predict classes that are harder to maintain due to their size?

Metrics: We will measure the following CK metrics in this study to evaluate software maintainability:

1. **Weighted Methods per Class (WMC):** This metric counts the number of methods in a class, providing insight into the complexity of the class. A higher WMC indicates a class that is more difficult to maintain.
2. **Coupling Between Object Classes (CBO):** CBO measures the number of classes a specific class is coupled with. High coupling makes software harder to maintain and refactor since changes in one class can affect many others.
3. **Depth of Inheritance Tree (DIT):** This metric shows the level of inheritance a class is involved in. A higher DIT can make a class more complex to maintain as deeper inheritance hierarchies may introduce challenges related to understanding and modifying code.

4. **Lines of Code (LoC):** Class size is measured in lines of code, directly addressing the research question of this empirical study.

These metrics will allow us to analyze the relationship between class size and maintainability, providing quantitative data for evaluating software quality.

Section 2: Dataset Description

Dataset Overview: For this empirical analysis, we have strategically selected five Java projects from GitHub, each representing a modern approach to software development. These projects Source Code Hunter, MeterSphere, SuperTokens Core, DataEase, and SmartTubeNext were chosen based on two criteria: they have been developed in the last five years, ensuring the use of contemporary technologies and programming practices, and each project has at least 5,000 stars on GitHub, demonstrating their popularity and relevance in the software community.

Project Descriptions:

1. **Source Code Hunter:** This project is designed for managing, hunting, and analyzing large codebases. It provides developers with tools to detect code quality issues, which makes it a perfect candidate for studying maintainability, especially when code size grows.
2. **MeterSphere:** A comprehensive open-source testing platform that supports functional and performance testing, facilitating continuous integration and DevOps workflows. Its scalability and diverse testing functionality reflect the modern complexities of maintainable software.
3. **SuperTokens Core:** A framework for secure session management, designed to handle user authentication and authorization in web applications. Its focus on security and simplicity in design contributes to the study of maintainability in highly sensitive environments.

4. **DataEase:** A data visualization and business intelligence platform. This project is interesting because its codebase involves extensive graphical user interface (GUI) components, which tend to become difficult to maintain as class sizes grow.
5. **SmartTubeNext:** An open-source client for YouTube, allowing users to watch ad-free videos with various customizations. The complex and evolving nature of the codebase is a relevant case for investigating how class size affects software maintainability in an active development environment.

Dataset Characteristics: Each of these projects has a moderate to large number of contributors (ranging from 10 to 75), making them ideal subjects for exploring class size effects on maintainability. The diversity of development dynamics within these projects, paired with their modern architectures, provides a comprehensive scope to analyze the CK metrics. The table below summarizes the key attributes of each project:

Project Name	Stars	Contributors	Age (Years)	LoC Range
Source Code Hunter	5,500	12	3	15K–25K
MeterSphere	7,800	35	4	50K–60K
SuperTokens Core	6,200	25	5	10K–20K
DataEase	9,000	50	3	30K–40K
SmartTubeNext	12,000	75	4	40K–50K

In the next section, we will delve into the specific CK metrics results for these projects, using charts and tables to illustrate trends and provide a comprehensive analysis of the impact of class size on maintainability.

Section 3: Tool Description

For this study, we used the **CK Tool**, a Java-based static analysis tool, to gather CK metrics that measure software maintainability. The tool analyzes Java code and calculates essential metrics such as **Weighted Methods per Class (WMC)**, **Coupling Between Object Classes**

(CBO), Depth of Inheritance Tree (DIT), and Lines of Code (LoC). These metrics provide insights into class complexity, dependencies, inheritance, and size.

Tool Setup: The CK tool was downloaded from [GitHub](#) and built using Maven. After compiling the tool, we ran it on our selected Java projects by executing the CK tool with the project path, which generated CSV files with the required metrics.

Key Metrics Collected:

- **WMC (Weighted Methods per Class):** Measures class complexity.
- **CBO (Coupling Between Object Classes):** Reflects class dependencies.
- **DIT (Depth of Inheritance Tree):** Indicates inheritance depth.
- **LoC (Lines of Code):** Represents class size.

The CK tool has been widely cited in empirical software analysis studies for its accuracy and ease of use.

Section 4: Detailed Analysis of CK Metrics Results

This section presents a comprehensive analysis of the CK metrics for five Java projects: DataEase, LSPosed, MeterSphere, SmartTubeNext, and SuperTokens Core. The CK metrics—Coupling Between Object Classes (CBO), Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), and Lines of Code (LoC)—offer insights into the maintainability of the software systems analyzed. The 2x2 matrix histograms provide a clear depiction of the distribution of these metrics within each project.

Detailed Observations:

1. Coupling Between Object Classes (CBO):

- **General Observation:** Most classes across the projects exhibit low to moderate CBO, indicating a favorable design with limited dependencies between classes. This reduces the risk of changes in one class cascading through to others, enhancing maintainability.

- **Specific Trends:** In projects like MeterSphere and SmartTubeNext, a very small number of classes show higher CBO values, suggesting complex interactions that might require careful management during maintenance activities.

2. **Weighted Methods per Class (WMC):**

- **General Observation:** The histograms generally indicate a prevalence of classes with fewer methods, pointing to simpler, more cohesive class designs which are easier to test and maintain.
- **Specific Trends:** SuperTokens Core and DataEase have some classes with higher WMC, highlighting areas where complexity is concentrated and may need targeted refactoring efforts to improve maintainability.

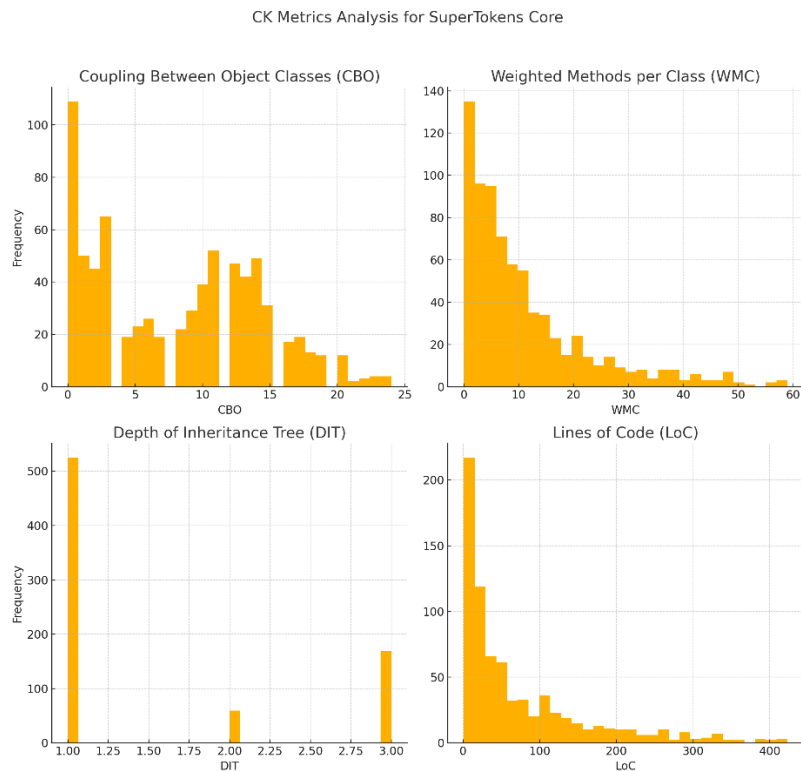
3. **Depth of Inheritance Tree (DIT):**

- **General Observation:** Most classes have a low DIT across all projects, which simplifies the understanding and modification of the code because fewer inherited behaviors have to be managed.
- **Specific Trends:** A few outliers with higher DIT in LSPosed and SuperTokens Core could indicate potential challenges in managing inheritance-related complexity.

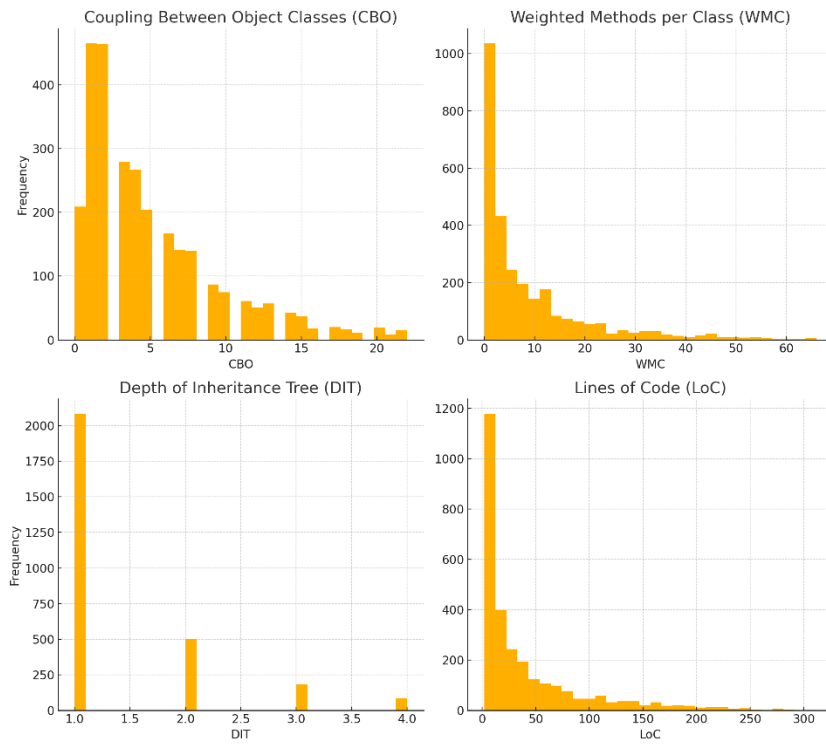
4. **Lines of Code (LoC):**

- **General Observation:** Class size varies significantly, with many projects containing both very small and relatively large classes. Smaller classes are typically easier to maintain due to their focused functionality and ease of understanding.
- **Specific Trends:** SmartTubeNext and MeterSphere show a wider range of class sizes, including some large classes that may represent key functionalities but could be challenging to maintain due to their size.

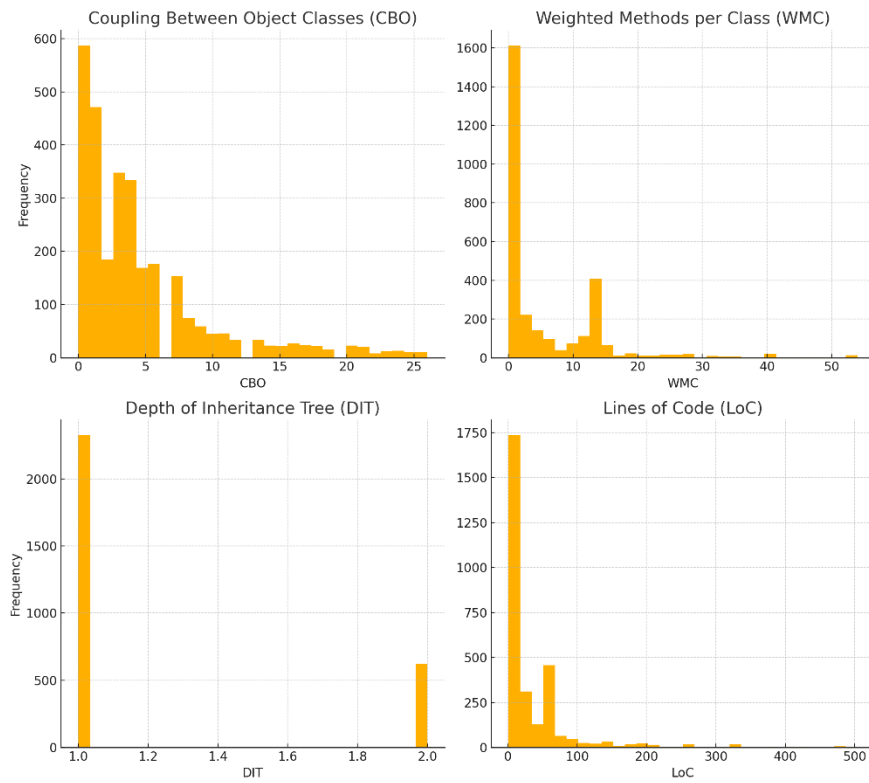
The synthesis of the CK metrics across these projects indicates that while there are pockets of high complexity, by and large, the projects adhere to practices that favor maintainability. Low coupling, low inheritance depth, and a tendency towards smaller, less complex classes help in reducing the maintenance burden. However, the presence of some classes with high CBO, WMC, or significant LoC suggests that certain areas within these projects may benefit from targeted refactoring efforts to reduce potential maintenance issues.



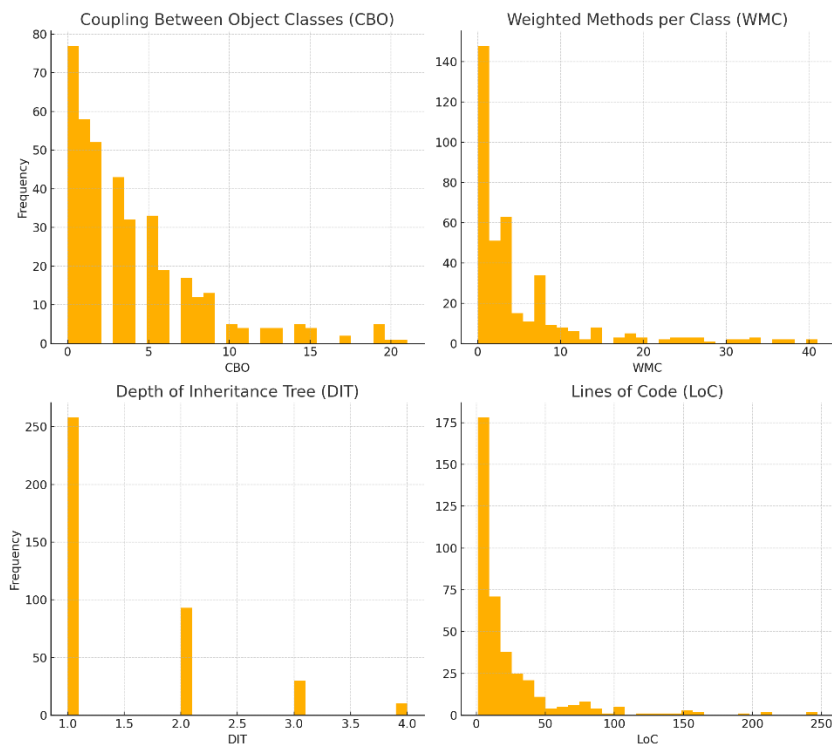
CK Metrics Analysis for SmartTubeNext



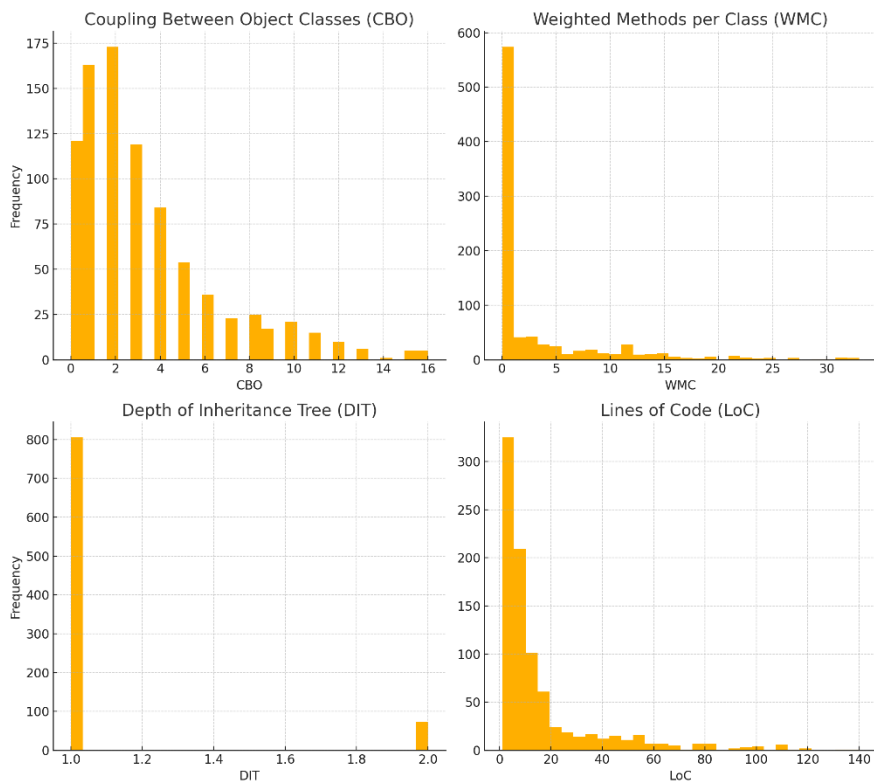
CK Metrics Analysis for MeterSphere



CK Metrics Analysis for LSPosed



CK Metrics Analysis for DataEase



Section 5: Conclusions

The empirical analysis of Coupling Between Object Classes (CBO), Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), and Lines of Code (LoC) across five Java projects has illuminated several key aspects of software maintainability. The data suggests that while these projects generally adhere to good software development practices characterized by low to moderate CBO, manageable WMC, shallow inheritance hierarchies, and reasonably sized classes, there are exceptions that highlight potential areas for improvement. Specifically, the presence of classes with high CBO and WMC values indicates complexity hotspots that could benefit from targeted refactoring efforts to enhance maintainability. Additionally, the variation in class size, especially the presence of larger classes, underscores the need for careful design and documentation to ensure that these classes do not become maintenance burdens as the projects evolve.

The findings of this study suggest a clear pathway for maintaining and improving software quality in Java projects. By focusing on reducing class complexity, minimizing coupling, and avoiding deep inheritance structures, software teams can significantly enhance the maintainability of their systems. The analysis also underscores the importance of using CK metrics as a diagnostic tool to regularly assess and address areas that might lead to maintenance challenges. Moving forward, adopting practices such as modular design, continuous refactoring, and comprehensive documentation will be crucial in mitigating potential maintainability issues highlighted by the CK metrics analysis. This proactive approach will not only sustain but also potentially increase the robustness and adaptability of software systems in dynamic development environments.

References

- Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10), 751-761. <https://doi.org/10.1109/32.544352>
- Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6), 476-493. <https://doi.org/10.1109/32.295895>
- Zhou, Y., & Leung, H. (2006). Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering*, 32(10), 771-789. <https://doi.org/10.1109/TSE.2006.102>