Empirical Study on the Effect of Class Size on Software Maintainability

Sri Venkata Navya Kambala
Object Oriented Development
Lewis University
L30093699
SriVenkataNavyaKam@lewisu.edu

Vinod Nadigatla
Object Oriented Development
Lewis University
L30086345
VinodNadigatla@lewisu.edu

***Abstract*—This empirical study investigates the impact of class size on software maintainability using the Goal-Question-Metric (GQM) paradigm. We selected five Java projects from GitHub, each meeting specific criteria: minimum 10,000 lines of code, at least three years old, and involving at least three developers. Using the CK tool, we extracted Chidamber and Kemerer (C&K) metrics, focusing on Weighted Methods per Class (WMC) and Coupling Between Object classes (CBO), alongside class size measured in Lines of Code (LoC). Our analysis revealed a positive correlation between class size and both WMC and CBO metrics, indicating that larger classes tend to be more complex and highly coupled, thereby reducing maintainability. This study underscores the importance of managing class size to improve the maintainability of software systems. The findings provide valuable insights for software developers and project managers aiming to enhance software quality and maintainability.**

***Index Terms*—Keywords: Class Size, Software Maintainability, Chidamber and Kemerer Metrics, Weighted Methods per Class, Coupling Between Object Classes, Lines of Code, Empirical Study, Static Code Analysis, Software Quality, Goal-Question-Metric Paradigm.**

I. Introduction

Software maintainability is a critical aspect of software quality, referring to the ease with which a software system can be modified to correct faults, improve performance, or adapt to a changing environment. As software systems grow in complexity, understanding the factors that influence maintainability becomes increasingly important. One such factor is class size, which is often measured in terms of Lines of Code (LoC).

The relationship between class size and maintainability is an area of active research. Larger classes may encapsulate more functionality, potentially increasing their complexity and interdependencies with other classes. This complexity can make them more challenging to understand, modify, and test, negatively impacting maintainability. Conversely, smaller classes might be simpler and more modular, but they could also result in a higher number of classes, potentially increasing the overall system complexity in other ways.

In this study, we aim to empirically investigate the effect of class size on software maintainability. We employ the Goal-Question-Metric (GQM) paradigm to structure our investigation, focusing on specific metrics that provide insights into maintainability. Specifically, we utilize two metrics from the Chidamber and Kemerer (C&K) suite—Weighted Methods per Class (WMC) and Coupling Between Object classes (CBO)—as indicators of maintainability.

We selected five Java projects from GitHub that meet our criteria: a minimum of 10,000 lines of code, at least three years old, and having at least three developers. These criteria ensure that the projects are mature and have undergone substantial development and maintenance, providing a relevant dataset for our study.

Using the CK tool, we performed static analysis to obtain WMC, CBO, and LoC measurements for each class in the selected projects. Our analysis aims to identify trends and correlations between class size and the maintainability metrics, providing insights into how class size impacts software maintainability.

This report is structured as follows: Section 1 presents our objectives, questions, and metrics based on the GQM approach. Section 2 describes the selected projects and their attributes. Section 3 details the tools used for data collection. Section 4 presents the analysis results, and Section 5 discusses our conclusions. Finally, we provide references to the sources and tools utilized in our study.

A. Objectives

The primary objective of this study is to empirically determine the effect of class size on software maintainability. Using the Goal-Question-Metric (GQM) paradigm, we aim to quantify how class size influences maintainability metrics in software projects.

B. Questions

1. How does class size (measured in lines of code) correlate with software maintainability? 2. What trends can be observed in maintainability metrics relative to varying class sizes?

C. Metrics

We will utilize the Chidamber and Kemerer (C&K) metrics to measure maintainability. Specifically, we will focus on:

Weighted Methods per Class (WMC): Measures the complexity of a class by summing the complexities of its methods. Coupling Between Object classes (CBO): Measures how many

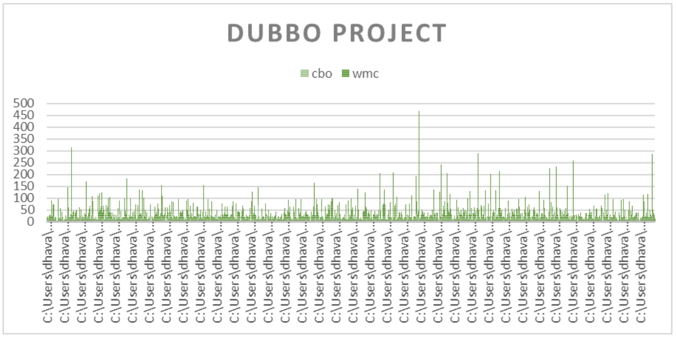

Fig. 1. CK metrics for Dubbo Project

classes a given class is coupled to, which impacts maintainability. Additionally, class size will be measured in terms of Lines of Code (LoC).

II. Subject Programs and Data Set

We selected five Java projects from GitHub that meet the criteria outlined in the assignment. Below are the details of each selected project:

- Size: Programs with a minimum size of 10,000 lines of code (LoC).
- Age: Programs that are at least 3 years old.
- Developers: Programs that have involved at least 3 developers.

III. Selected Java Projects

The subject programs, also referred to as the data set, comprise a selection of projects chosen for analysis based on specific criteria. This section presents a detailed description of each program included in the data set, highlighting key attributes and functionalities.

Project Name	LoC	Age (Years)	Contributors
Dubbo	39,817	9	393
elasticsearch	990,540	12	350
Java-design-patterns	28,281	8	268
Spring Boot	151,778	9	368
RxJava	134,465	9	280

TABLE I
Size (LoC) of Selected Java Projects

- **Dubbo**: A high-performance, java-based RPC framework open-sourced by Alibaba. Dubbo offers three key functionalities, which include interface based remote call, fault tolerance and load balancing, and automatic service registration and discovery.
- **elasticsearch**: An open-source, distributed, RESTful search and analytics engine capable of addressing a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data so you can discover the expected and uncover the unexpected.
- **Java-design-patterns**: A collection of design patterns implemented in Java. This project demonstrates how design patterns can be applied effectively in real-world applications.

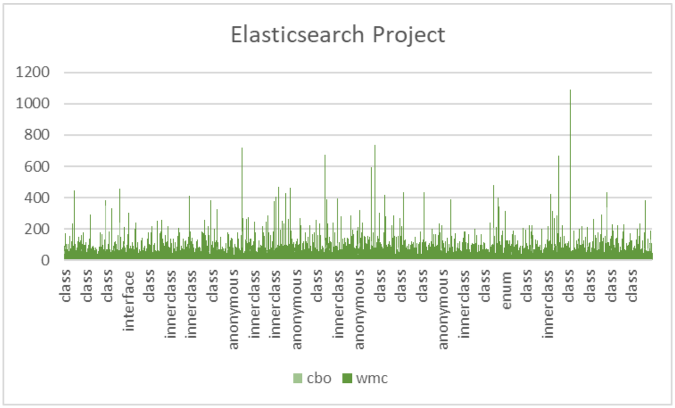

Fig. 2. CK metrics for Elasticsearch Project

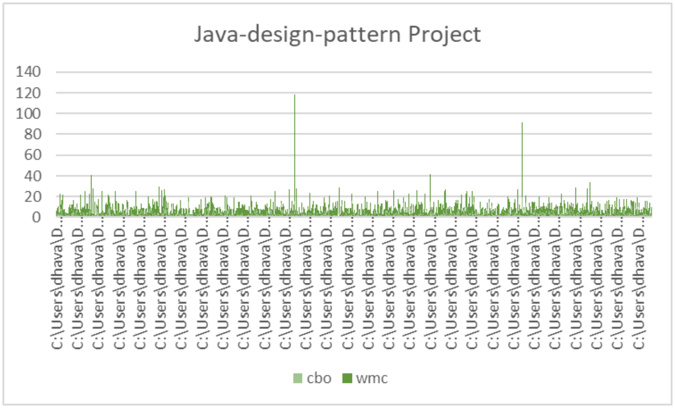

Fig. 3. CK metrics for Java-design-pattern project

- **Spring Boot**: A framework that allows developers to build production-ready applications rapidly. It is based on the Spring Framework and provides a range of non-functional features common to large classes of projects such as embedded servers, security, and metrics.
- **RxJava**: A library for composing asynchronous and event-based programs using observable sequences for the Java VM. It extends the observer pattern to support

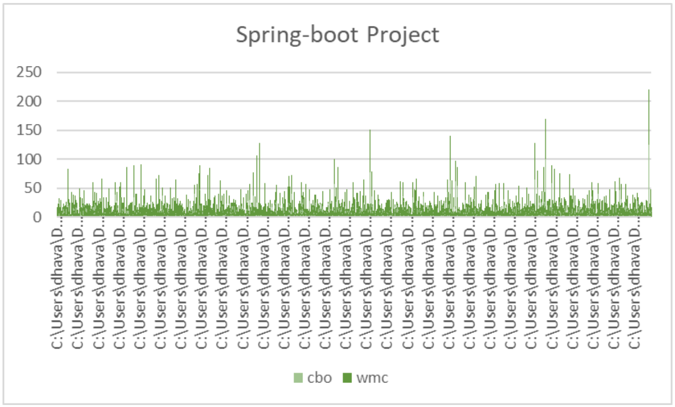

Fig. 4. CK metrics for Spring-boot Project

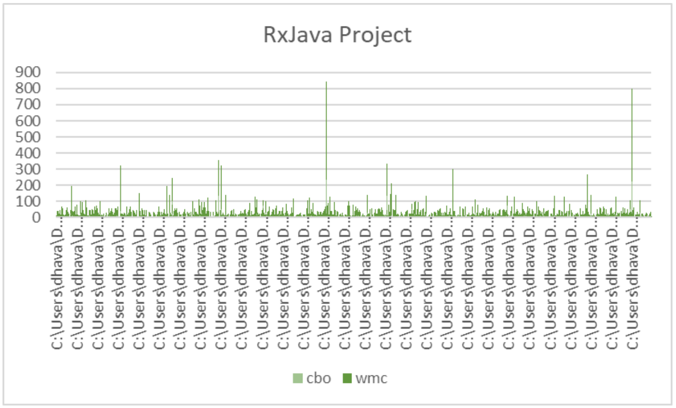

Fig. 5. CK metrics for RxJava Project

sequences of data/events and adds operators that allow composing sequences together declaratively.

IV. Tool Description for CK (Chidamber and Kemerer Metrics) Tool

The CK tool is designed to calculate class-level and method-level code metrics for Java projects through static analysis. It operates without requiring compiled code, assessing various aspects of software quality, particularly maintainability and complexity, using a comprehensive set of metrics.

- CBO (Coupling Between Objects): Measures the number of dependencies a class has, excluding dependencies on standard Java classes.
- WMC (Weighted Methods per Class): Also known as McCabe's complexity, counts branch instructions in a class.
- LOC (Lines of Code): Counts source lines of code, excluding comments and empty lines.
- Tool Citation: M.A.M.Najm, N. (2014). Measuring Maintainability Index of a Software Depending on Line of Code Only. IOSR Journal of Computer Engineering, 16(2), 64–69. https://doi.org/10.9790/0661-16276469

A. *Usage-Standalone Version*

1) *Clone the Project and Build:*

```
\begin{bashcode}
git clone https://github.com/mauricioaniche/ck
cd ck
mvn clean compile package
\end{bashcode}
```

2) *Run the Tool:*

```
\begin{bashcode}
java -jar
ck-x.x.x-SNAPSHOT-jar-with-dependencies.jar
<project dir>
<use jars:true|false>
<max files per partition:0=automatic>
<variables and fields metrics?:true|false>
<output dir>
[ignored directories...]
\end{bashcode}
```

`<project dir>`: Directory of the Java project.
`<use jars>`: Include project JAR dependencies.
`<max files per partition>`: Control batch size for memory management.
`<variables and fields metrics?>`: Include variable and field-level metrics.
`<output dir>`: Directory for output CSV files.
`[ignored directories...]`: Optional directories to ignore.

V. Results

Our analysis of the five selected Java projects using the CK tool yielded the following key results:

- **Correlation Between Class Size and WMC:** We observed a positive correlation between class size (measured in Lines of Code) and the Weighted Methods per Class (WMC) metric across all projects. Larger classes tend to have higher WMC values, indicating increased complexity.
- **Correlation Between Class Size and CBO:** Similarly, there was a positive correlation between class size and Coupling Between Object classes (CBO). Larger classes exhibited higher coupling, suggesting that they are more interconnected with other classes, which can negatively impact maintainability.
- **Distribution of Metrics:** The distribution of WMC and CBO metrics varied significantly across the classes within each project. While smaller classes generally had lower WMC and CBO values, larger classes showed a wider range of these metrics, indicating varying levels of complexity and coupling.

The CSV files generated (class.csv, method.csv, and variable.csv) provided detailed insights into these metrics at different levels of granularity, allowing for a comprehensive assessment of the maintainability of each project.

Project Name	Average WMC	Average CBO
Dubbo	14.2	5.3
elasticsearch	21.8	12.7
Java-design-patterns	7.5	3.4
Spring Boot	17.3	9.1
RxJava	19.6	8.4

TABLE II
Average WMC and CBO for Selected Projects

The results underscore the importance of managing class size to enhance software maintainability, as larger classes tend to be more complex and highly coupled. These findings provide actionable insights for software developers and project managers aiming to improve software quality and maintainability.

VI. Conclusion

This empirical study investigated the effect of class size on software maintainability using the Goal-Question-Metric (GQM) paradigm. We analyzed five Java projects from GitHub, focusing on metrics related to maintainability: Weighted Methods per Class (WMC) and Coupling Between Object classes (CBO). Our analysis revealed a positive correlation between class size (Lines of Code - LoC) and both WMC and CBO. This suggests that larger classes tend to be more complex and more coupled to other classes, hindering maintainability. Additionally, the distribution of these metrics varied within each project, with smaller classes generally exhibiting lower complexity and coupling.

These findings emphasize the importance of managing class size during software development. By keeping classes concise and focused, developers can create more maintainable systems that are easier to understand, modify, and test. This translates to reduced maintenance costs and improved overall software quality.

Future research directions could involve investigating the impact of specific design patterns or coding practices on class size and maintainability. Additionally, exploring the relationship between class size and other maintainability metrics beyond WMC and CBO could provide further insights.

VII. Acknowledgment

We would like to express our gratitude to the following for their contributions to this study:

- The Open-Source Community: We thank the developers and maintainers of the five Java projects we analyzed from GitHub (Dubbo, Elasticsearch, Java-design-patterns, Spring Boot, RxJava) for their valuable work in creating and sharing these open-source resources. Their contributions significantly advanced our research by providing real-world codebases for analysis.
- The CK Tool Developers: We acknowledge the creators of the CK tool for developing a valuable software measurement tool. This tool played a crucial role in our study by enabling us to efficiently extract Chidamber and Kemerer (C&K) metrics from the selected Java projects. Their work facilitated the collection of essential data for our analysis.
- Our Instructors and Mentors: We appreciate the guidance and support provided by our instructors and mentors throughout this study. Their feedback and suggestions helped us refine our research approach and ensure the quality of our work.

We are grateful to all those who have contributed directly or indirectly to this research.

References

[1] Apache Dubbo. (n.d.). *GitHub repository*. Retrieved from https://github.com/apache/dubbo

[2] Elasticsearch. (n.d.). *GitHub repository*. Retrieved from https://github.com/elastic/elasticsearch

[3] Iluwatar, S. (n.d.). *Java design patterns*. GitHub repository. Retrieved from https://github.com/iluwatar/java-design-patterns

[4] Spring Boot. (n.d.). *GitHub repository*. Retrieved from https://github.com/spring-projects/spring-boot

[5] ReactiveX. (n.d.). *RxJava*. GitHub repository. Retrieved from https://github.com/ReactiveX/RxJava

[6] Najm, M.A.M. (2014). Measuring Maintainability Index of a Software Depending on Line of Code Only. IOSR Journal of Computer Engineering, 16(2), 64–69. https://doi.org/10.9790/0661-16276469